Optimization and Uncertainty Quantification with CACTUS: DAKOTA Interface

Jonathan C. Murray
Aerosciences Department
Sandia National Laboratories
Albuquerque, NM 87185-0825

1 Introduction

An interface has been developed to couple CACTUS [1] to the Sandia-developed software package DAKOTA [2] for use in performing general purpose optimization and uncertainty quantification (UQ) analysis on arbitrary turbine designs. CACTUS provides a flexible, reasonably inexpensive, mid-fidelity model for integrated turbine performance and blade load distributions, and as such is an ideal candidate for use in optimization and UQ studies. CACTUS may be used as the single source of turbine performance information, or as the low-fidelity component of a multi-fidelity surrogate-based analysis utilizing Navier-Stokes CFD as the high-fidelity component. An example optimization analysis using CACTUS as the single source of turbine performance information is provided in this document.

The CACTUS source code, DAKOTA interface scripts, and DAKOTA input file for the example optimization analysis are available in revision 64 of the CACTUS repository.

2 CACTUS Analysis Driver

DAKOTA is built to interface with an arbitrary stand-alone engineering code through the use of non-interactive scripts called analysis drivers. The analysis driver provides the value of the cost function (for optimization) or quantity of interest (for UQ) as a function of all the relevant design parameters exposed to DAKOTA. The driver script can be written in any scripting language. This script reads the input parameter structure in a DAKOTA generated input file, conducts the appropriate analysis, and writes a file containing the analysis outputs in the DAKOTA output structure format.

A general purpose analysis driver has been written for CACTUS using Python. The Python scripting language [3] can be installed from a package on virtually all Linux distributions. This driver is available in the CACTUS repository as *DAKOTA/Driver/CACTUS_Driver.py*. The driver is designed to modify values in a nominal CACTUS input file, and optionally a nominal CACTUS geometry creation script, regenerate the geometry if necessary, and execute the CACTUS calculation on the modified geometry using the modified CACTUS inputs. The driver is written to be sufficiently general that little or no modification should be necessary for each individual analysis, attempting to confine the details of a particular analysis to the DAKOTA input file, the nominal CACTUS input file, and the nominal CACTUS geometry creation script. The user may need to modify the portion of the driver that generates the values of the analysis quantities of interest from the data in the CACTUS output files. This portion of the driver has been structured to allow the user to easily create or modify the analysis outputs available for return to DAKOTA.

The *CreateGeom* folder of the CACTUS repository contains a set of MATLAB functions that can be used to generate the CACTUS turbine geometry input file for an analysis. The user can use these functions to create script that generates geometry for an arbitrary turbine and writes a geometry file for CACTUS. These scripts are designed to be compatible with GNU Octave [4] in non-interactive mode (plotting functions not used or turned off in the script). GNU Octave is lightweight, open-source software that clones the basic MATLAB syntax, and can be installed from a package on most Linux distributions. Most optimization and UQ methods available in DAKOTA are capable of performing many function evaluations in parallel. As such, the

CACTUS analysis driver makes use of Octave instead of MATLAB to re-run the nominal geometry generation script (if required) to avoid licensing issues with associated with spawning multiple instances of MATLAB in parallel. Note that the nominal geometry creation script must define the output geometry filename in a variable called *FN* (the driver needs to modify this on each evaluation).

2.1 Driver Interface in DAKOTA Input File

The interface to the CACTUS driver is specified in the *interface* section of the DAKOTA input file (refer to Appendix A for an example). The *analysis_driver* input specifies the path to the CACTUS driver script. The *analysis_components* input supplies additional inputs to the driver script. The analysis components expected by the driver are given below.

1. Path to CACTUS executable
2. Nominal CACTUS input file to modify on each call
3. If a nominal geometry file is to be modified, should be set to 'Y', otherwise 'N'
4. Path to template geometry generation script (if component 3 set to 'Y').

There are other generic analysis components that can be specified as well (see driver script for details).

The variable names defined in the *variables* section of DAKOTA’s input file propagate to the input file passed to the CACTUS driver. The variable names (“descriptors” in DAKOTA) are first checked against any input specifications defined in the inputs section of the driver (see driver script for details). If a match exists, the input value is handled accordingly. If no match exists, the driver assumes that the variable name has the form *‘TAG:DES’*, where *TAG* matches the name of a variable in either the nominal CACTUS input file or the nominal CACTUS geometry creation script, and *DES* specifies an operation to be performed on the matched variable. *DES* can be either *‘F’*, *‘D’*, or *‘V’*, indicating that the input value should multiply (factor) the nominal value, be added to the nominal value (delta), or be used directly in place of the nominal value (value). The *‘V’* designator is the default if nothing is input (no *‘:’* in *DES*). For example, if the nominal tip speed ratio is to be modified with an additive delta, the DAKOTA descriptor should be *‘ut:D’*.

The response names defined in the *responses* section of DAKOTA’s input file propagate to the input file passed to the CACTUS driver. The response names (“descriptors” in DAKOTA) are checked against the output designations defined in the outputs section of the driver (see driver script for details), and the corresponding output is returned.

3 DAKOTA Optimization Methods

DAKOTA provides a variety of gradient-based and direct search optimization methods. Gradient-based methods make use of some amount of cost function gradient information, and are best for precisely identifying local optimum points given a good initial condition. Direct search methods can be useful when many local optimum points exist, or when cost function gradient information is unreliable. DAKOTA is capable of applying these methods individually, or in sequential or collaborative hybrid approaches to blend the global search capability of direct methods with the speed and precision of gradient-based local search methods.

DAKOTA provides gradient-based optimization methods from the OPT++ and DOT library. Most of these methods can be run using a quasi-newton update on each iteration, eliminating the need to calculate a full Hessian (second derivative) matrix for the cost function. All of the methods provided are stabilized with either a line search or a trust region method. The DOT BFGS quasi-newton method has proven to be a robust and somewhat efficient method for many analyses using CACTUS output.

The optimization cost function can be directly evaluated, or represented with a variety of surrogate model formulations for the purpose of speeding up searching within a local “trust region” for the surrogate model. In addition to local search performance gains, surrogate models can also be useful in cases where gradient-based local optimization methods are generally applicable, but where gradients derived from local finite differencing of the cost function can be inaccurate. This is often the case if the cost function involves the output of an iterative calculation with some residual non-convergence error, or slight discontinuities in the output.

4 DAKOTA Uncertainty Quantification Methods

DAKOTA provides a variety of UQ methods for ranging from direct sampling (Monte Carlo) to approximate surrogate-based stochastic expansion methods. The particular method used depends on the type of analysis being performed (aleatory, epistemic) and the type of output (continuous, discrete), and the approximate order of variation between the driving inputs and the output quantity of interest. When the quantities of interest vary somewhat linearly with the driving inputs, stochastic expansion methods can be much more efficient than the direct sampling approaches, but may be inaccurate for highly non-linear or discontinuous functional variations. See the DAKOTA User’s Manual [2] for a complete description of the UQ methods provided by DAKOTA.

5 Example Optimization Analysis

The example analysis described in this section illustrates the use of DAKOTA in performing optimization on CACTUS output. The analysis calculates the maximum value of the power coefficient for a reference turbine configuration as a function of the operating tip speed ratio. The files for this example analysis are located in the *DAKOTA/Example1* directory of the CACTUS repository. The DAKOTA input file (*OptTSR.dak*) is also shown in Appendix A.

The analysis uses the DOT BFGS gradient-based optimization method to find the point of maximum power coefficient and the corresponding tip speed ratio. The gradient-based method is executed on a surrogate model rather than the raw CACTUS output. A globally-fitted, local surrogate model is used in which the model is deemed valid only within a local trust region in state space, and is fit to the raw CACTUS output using all CACTUS evaluations (from the current and all previous iterations) that remain within the current trust region. Fitting the model to output generated across the trust region (globally-fitted) eliminates the need to calculate local gradients of CACTUS output through finite differencing, which can be inaccurate due to the residual non-convergence error in each run. The kriging (Gaussian process) model is used as the form of the surrogate model.

The nominal CACTUS input file (*TestVAWTNom.in*) defines a calculation done on a generic vertical axis wind turbine. The nominal geometry file (*TestVAWT.geom*, which is unchanged in this analysis) is given in the *Test/TestGeom* directory of the CACTUS repository.

The DAKOTA analysis can be run from the terminal (in the *Example1* folder) as

```
dakota OptTSR.dak
```

The output of the calculation can be compared against the expected output given in the *OptTSR_CL.out* (DAKOTA command line output) and the *Opt_Data_Out.dat* (DAKOTA tabular output) files located in the *Example1* directory.

6 References

1. Murray, J., Barone, M., “CACTUS User’s Manual,” Version 1.0, Sandia National Laboratories, Albuquerque, NM.
2. *The DAKOTA Project.* Retrieved June 30, 2013, from http://dakota.sandia.gov
3. *The Python Programming Language.* Retrieved June 30, 2013, from http://www.python.org
4. *GNU Octave.* Retrieved June 30, 2013, from http://www.gnu.org/software/octave

7 Appendix A: Example DAKOTA Input File

File: *OptTSR.dak*. The *‘…’* indicates a line continuation.

```
# Runs local surrogate based optimization on CACTUS. The local surrogate is
…fit to the CACTUS cost function using "global" sampling within a trust
# region defined for the surrogate function. This keeps the method from
# having to use the slightly noisy CACTUS results in gradient calculations.
# This method is faster (especially when using async parallel CACTUS
# evalutations in the fitting of the surrogate function) and more robust than
# direct gradient optimization on CACTUS output.
# Note: Currently, surrogate optimization methods can only minimize a cost
# function (their trust region method fails for maximization)...

strategy
    single_method
    #tabular_graphics_data
      #tabular_graphics_file = 'Opt_Data_Out.dat'
    method_pointer = 'SBLO'

method
    id_method = 'SBLO'
    surrogate_based_local
    max_iterations = 50
    convergence_tolerance = 1e-02
    soft_convergence_limit = 3 # terminate after this number of surrogate
…model iterations with improvement less than tolerance
    trust_region
      initial_size = 0.5
      minimum_size = 1.0e-2
    output silent
    # Surrogate model and minimization method pointers
    model_pointer = 'SURROGATE'
    approx_method_pointer = 'NLP'

method
    id_method = 'NLP'
    output silent
    dot_bfgs
      max_iterations = 50
      convergence_tolerance = 1e-8

model
    id_model = 'SURROGATE'
    surrogate
      global # Surrogate fit with "global" sampling within trust region
      # Surrogate function
      #polynomial quadratic # simple
      gaussian_process surfpack # kriging
      # Total samples to take in fitting the surrogate
      total_points = 5 # oversample (more than recommended point count) to
…smooth noisy cost function output
      #recommended_points
      reuse_samples region # reuse samples that are still in the current
…trust region
      # Sampling method and responses pointers
      responses_pointer = 'SURROGATE_RESP'
      dace_method_pointer = 'SAMPLING'
      # Surrogate model fit diagnostics
      #diagnostics

variables
    # There appears to be a bug in DAKOTA's (version <= 5.2)
…surrogate_based_local method that makes it
    # impossible to use state variables, at least when doing global sampling
…to fit the surrogate model
    # to the cost function... Set all non-optimized parameters directly in
…the CACTUS input file...
    continuous_design = 1
      initial_point = 6
      lower_bounds = 2
      upper_bounds = 10
      descriptors = 'Ut'

responses
    id_responses = 'SURROGATE_RESP'
    num_objective_functions = 1
    # Must apply CACTUS descriptors to the surrogate responses as well as the
…true responses as these descriptors
```

```
    # somehow propagate to, and overwrite, the true response descriptors in
…the input files passed to the analysis driver (probably a bug)...
    descriptors = 'CostFunc_MaxCP'
    numerical_gradients
      method_source dakota
      interval_type central
      fd_gradient_step_size = 1.e-6
    no_hessians

method
    id_method = 'SAMPLING'
    model_pointer = 'TRUTH'
    dace lhs
      # Set samples to minimum number of new truth model samples to be added
…to the
      # surrogate model basis on each surrogate model iteration. Do a couple
…per dimension
      # to hedge against directional bias potentially introduced by reusing
…samples...
      samples = 2

model
    id_model = 'TRUTH'
    single
      interface_pointer = 'TRUE_FN'
      responses_pointer = 'TRUE_RESP'

interface
    id_interface = 'TRUE_FN'
    analysis_drivers = '../Driver/CACTUS_Driver.py'
      analysis_components = '/home/jmurray/Project/CACTUS/stable/cactus'
…'TestVAWTNom.in' 'N'
      fork
      parameters_file = 'Inputs.in'
      results_file = 'Outputs.out'
      # need to tag files for async calculation
      file_tag
    asynchronous
      evaluation_concurrency = 10

responses
    id_responses = 'TRUE_RESP'
    num_objective_functions = 1
    descriptors = 'CostFunc_MaxCP'
    no_gradients
    no_hessians
```