Creating and Evaluating Surrogate Models with Dakota and Surfpack

Dakota Development Team

November 7, 2017

This note offers a brief summary of ways to build and evaluate a Gaussian process surrogate model with both Dakota and Surfpack, including integrating the surrogate evaluation into another application via a C API. The discussion and examples here apply not only to Gaussian process models, but to the following Dakota global surrogate types, all provided by the Surfpack sub-package:

- `gaussian_process surfpack` (`kriging surfpack`)
- `mars`
- `moving_least_squares`
- `neural_network`
- `radial_basis`
- `polynomial`

The Dakota-only approaches described will also work for other Dakota surrogate models, but the Surfpack-specific approaches will not. All examples here use a two variable form of the textbook example function provided with Dakota, so are creating a surrogate model for the textbook function $f(x1, x2)$.

Requirements: The Dakota option to save a Surfpack model file and some of the standalone Surfpack capabilities require Dakota 5.4 or later. Saving (from Dakota or Surfpack) and later loading (and subsequently evaluating) surrogate models with the Surfpack executable require the Surfpack standalone and Boost serialization options to be enabled at compile time. Building and evaluating surrogate models directly in Dakota has no special requirements.

The example described here can be found in `dakota/examples/eval_surrogate`. It contains the following input files:

```
build_points.dat        Training data used to build the surrogate model
eval_points.dat         Points at which to query the surrogate model
dakota_surrogate.in     Dakota input to build/save/evaluate a surrogate
sp_build_surrogate.spk  Surfpack commands to build/save a surrogate
sp_eval_surrogate.spk   Surfpack commands to load/evaluate a surrogate
```

And the following example output files:

```
dak_gp_model.textbook.sps  Surfpack model file generated by Dakota
dak_surrogate_evals.dat    Dakota-generated evals of the surrogate model
sp_gp_model.textbook.sps   Surfpack model file generated by Surfpack
sp_surrogate_evals.dat     Surfpack-generated evals of the surrogate model
```

1 Building a Surrogate Model

1.1 Using Dakota

The examples B1 and B2 presented in this section demonstrate two ways to build a surrogate model, solely using Dakota. Later, examples E1 and E2 demonstrate evaluating the previously built models using Dakota. The Dakota build and evaluation examples are all based on running `dakota -input dakota_surrogate.in`. This input file is shown here.

```
# Dakota input file based on dakota/test/dakota_textbook_lhs_approx.in
# Build and evaluate a surrogate at a user-specified set of points

# Top-level controls
 environment
  method_pointer = 'EvalSurrogate'
  tabular_data
    tabular_data_file = 'dak_surrogate_evals.dat'
    custom_annotated header eval_id

# Method to perform evaluations of the surrogate model
method
  id_method = 'EvalSurrogate'
  model_pointer = 'SurrogateModel'

  # -----
  # Eval Option E1: Evaluate the surrogate model at 6 user-specified points
  list_parameter_study
    list_of_points
      0.90 1.00
      0.95 1.00
      1.00 1.00
      1.00 0.93
      1.00 0.98
      1.00 1.01
    # -- OR: Read the list of points from a file --
    #import_points_file = 'eval_points.dat'
    #  freeform
  # -----

  # -----
  # Eval Option E2: Perform Monte Carlo sampling on the surrogate model
  #sampling
  #  samples = 100 seed = 5
  #  sample_type lhs
  # -----


# Surrogate model specification
model
  id_model = 'SurrogateModel'
  surrogate global
```

```
    dace_method_pointer = 'DesignMethod'
    gaussian_process surfpack

  # -----
  # Build Data Option B2
  # Use this method with samples = 0 to build from a samples file
  #import_points_file = 'build_points.dat'
  #  freeform
  # -----

  # Save the model to a Surfpack model file for later evaluation in Surfpack
  # Model Save option S1
  export_model
   filename_prefix = 'dak_gp_model'
   formats
     text_archive

variables,
  uniform_uncertain =  2
    lower_bounds    =  0.9  0.9
    upper_bounds    =  1.1  1.1
    descriptors     =  'x1' 'x2'

responses
  response_functions = 1
    descriptors 'textbook'
  no_gradients
  no_hessians


# Build Data Option B1
# Method to generate a design to build the surrogate
method
  id_method = 'DesignMethod'
  model_pointer = 'SimulationModel'
  sampling
    seed = 50
    sample_type lhs

    # -----
    # Build Data Option B1
    samples = 10
    # -----

    # -----
    # Build Data Option B2
    #samples = 0
    # -----

# The true simulation model to evaluate to build the surrogate model
model
```

```
  id_model = 'SimulationModel'
  single
    interface_pointer = 'SimulationInterface'


interface,
  id_interface = 'SimulationInterface'
  fork
    analysis_driver = 'text_book'
#    analysis_driver = 'rosenbrock'
#    analysis_driver = 'herbie'
```

B1: Dakota-generated points design

Build data option B1 uses the LHS method in Dakota to generate the x1, x2 points at which to build the surrogate model. Dakota will generate 10 points at which to build the surrogate model and run the simulation interface at each point to get the response function value. It will then construct the surrogate model and write it to a Surfpack formatted model file `dak_gp_model.textbook.sps`.

B2: User-provided points file

With build data option B2, Dakota accepts a user-provided data file (here `build_points.dat`) in freeform data format (whitespace separated data with rows containing [x1, x2, function]). The surrogate model is constructed from these data, and again written to a Surfpack formatted model file `dak_gp_model.textbook.sps`.

Contents of the build data file `build_points.dat`:

```
1.0294891612e+00  9.5843978608e-01  3.7396199783e-06
1.0569654527e+00  9.4991259535e-01  1.6824249658e-05
1.0072406749e+00  9.9501931870e-01  3.3640259779e-09
1.0810233015e+00  1.0553102643e+00  5.2455135591e-05
1.0192158401e+00  1.0250915474e+00  5.3272267830e-07
9.0729758306e-01  1.0945758444e+00  1.5385803120e-04
9.8752353230e-01  9.8087858825e-01  1.5791485252e-07
9.5516488408e-01  1.0348640423e+00  5.5182980289e-06
9.3066849028e-01  9.2677547445e-01  5.1855119335e-05
9.6964108621e-01  1.0018724977e+00  8.4947617106e-07
```

Dakota also supports an annotated input data file format with a header row of labels and a leading column with point numbers. For this case the number of DesignMethod `samples = 0` in the Dakota input file, indicating that Dakota should not generate any additional DOE points beyond those provided in the user data file.

1.2 Using the Surfpack Executable

To build the same surrogate model using the standalone Surfpack executable `surfpack`, use the Surfpack instruction file `sp_build_surrogate.spk`, together with the data file `build_points.dat`:

```
Load[name = textbook_build, file = 'build_points.dat',
     n_predictors= 2, n_responses = 1]

CreateSurface[name = textbook_gp, data = textbook_build, type = kriging]

Save[surface = textbook_gp, file = 'sp_gp_model.textbook.sps']
```

To build, run `surfpack sp_build_surrogate.spk`, which will write the file `sp_gp_model.textbook.sps`. In this example the Surfpack-generated GP model will differ slightly from that generated by Dakota. This is being investigated.

2 Evaluating a Surrogate Model

This section discusses evaluating a surrogate model either in Dakota directly or in Surfpack after previously saving in Dakota or Surfpack.

2.1 Using Dakota

The examples in this section use the Dakota input file `dakota_surrogate.in` provided above. Dakota can use a constructed surrogate model in any of its iterative methods, for example to perform optimization on the surrogate model to find a best design. The simplest methods are to evaluate the surrogate at a user-specified set of points and to randomly sample the surrogate with a Monte Carlo method. The Dakota input file shown above demonstrates both.

To evaluate the surrogate model repeatedly at different sets of points without rerunning the expensive simulation, change the evaluation method specification (E1 or E2) and use Dakota's restart facility, e.g., `dakota -input dakota_surrogate.in -read_restart dakota.rst`

E1: User-provided list of points

The first example evaluates the surrogate model at the list of points provided in the Dakota input file. Dakota writes the surrogate evaluations to a file `dak_surrogate_evals.dat`:

```
%eval_id             x1             x2       textbook
       1            0.9              1 5.695149128e-05
       2           0.95              1 9.909938326e-06
       3              1              1 -1.681528987e-06
       4              1           0.93 3.257983696e-07
       5              1           0.98 -2.025243345e-07
       6              1           1.01 -3.01510746e-06
```

E2: Dakota-generated points design

In this example, Dakota generates a 100 sample LHS design and evaluates the surrogate model at the generated points. Dakota writes the surrogate evaluations to a file `dak_surrogate_evals.dat`:

```
%eval_id             x1             x2       textbook
       1    1.035667472   0.9359510107 2.782730535e-06
       2   0.9733842609   0.9604024011 5.039401052e-06
       3    1.065423859    1.092537917 8.668835233e-05
       4   0.9985319699    1.007958203 -2.912209131e-06
       5    0.900985511   0.9988335608 5.608757991e-05
...
     100    1.132894095    1.032154224 0.0001276646855
```

2.2 Using the Surfpack Executable

This example uses Surfpack to evaluate a saved surrogate at user provided points. Surfpack can also generate simple designs at which to evaluate the surrogate using the CreateSample command. The commands for evaluation are in `sp_eval_surrogate.spk`:

```
Load[name = eval_points, file = 'eval_points.dat',
     n_predictors = 2, n_responses = 0 ]

Load[name = textbook_gp, file = 'sp_gp_model.textbook.sps']

Evaluate[surface = textbook_gp, data = eval_points]
Save[data = eval_points, file = 'sp_surrogate_evals.dat']
```

Running `surfpack sp_eval_surrogate.spk` loads the surface from the file `sp_gp_model.sps`, loads a set of evaluation points from `eval_points.dat`:

```
0.90 1.00
0.95 1.00
1.00 1.00
1.00 0.93
1.00 0.98
1.00 1.01
```

and evaluates the model at them, writing `sp_surrogate_evals.dat`:

```
  9.000000e-01   1.000000e+00   3.324085e-05
  9.500000e-01   1.000000e+00   6.390078e-06
  1.000000e+00   1.000000e+00  -1.007914e-06
  1.000000e+00   9.300000e-01   2.761034e-05
  1.000000e+00   9.800000e-01  -2.456697e-07
  1.000000e+00   1.010000e+00  -2.285316e-06
```

The Surfpack phases for building and evaluating a surrogate model may be combined into a single Surfpack command file if desired.

2.3 Using Surfpack from a C Program

While the C++ interface to Surfpack offers richer functionality, a limited C interface is provided to load and evaluate a previously saved surrogate model from a C program.

Requirements: Linking to surfpack from C requires inclusion of `surfpack_c_interface.h` (which documents the API) and linking against (1) the surfpack libraries `surfpack`, `surfpack_fortran`, (2) TPLs directly used by surfpack `ncsuopt`, `conmin`, and (3) system libraries `boost_serialization`, `lapack`, `blas`, and the standard C++ libraries. The non-system dependencies are included in binary distributions of Surfpack and Dakota in `include/` or `lib/`.

Example: The source distribution example in `surfpack/examples/CInterface` demonstrates loading and evaluating a surfpack model from C. A representative `CMakeLists.txt` and `eval_model.c` are included. This example can also be obtained directly from
https://software.sandia.gov/trac/surfpack/browser/trunk/examples/CInterface, should a source distribution not be available.