SAND2022-7261C

Sandia National Laboratories

Overview of the latest features and capabilities in the Dakota software

DAKOTA

Explore and predict with confidence

J. Adam Stephens, D. Thomas Seidl, Brian M. Adams, Gianluca Geraci

Sandia National Laboratories

2022 ECCOMAS Congress

June 8, 2022

Oslo, Norway

U.S. DEPARTMENT OF ENERGY NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

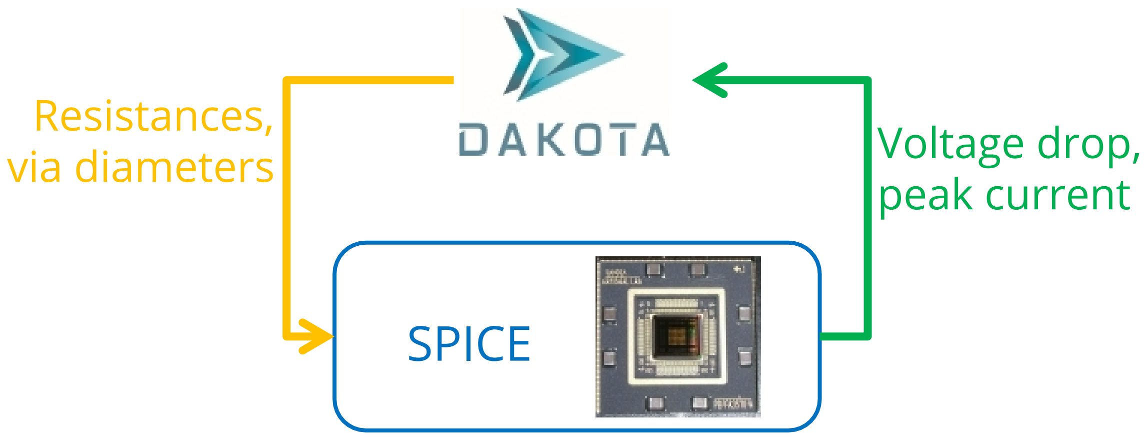
SAND2022-6778 C

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

What is Dakota?

Open-source software for black-box, ensemble analysis of computational simulations

- Suite of iterative mathematical and statistical methods and a convenient means of interfacing with (just about) any simulation software
- Provides scientists, engineers, and decision makers greater insight into the predictions of their models via:
  - Global Sensitivity Analysis
  - Uncertainty Quantification
  - Optimization
  - Calibration
- Production and Research focus
- Works on desktops and HPCs and the major operating systems (*nix, OS X, Windows)
- Command line interface, GUI, and API



Capabilities

Common interface to established and cutting edge algorithms

Sensitivity Analysis
- Designs: MC/LHS, DACE, sparse grid, one-at-a-time
- Analysis: correlations, scatter, Morris effects, Sobol indices

Uncertainty Quantification
- MC/LHS/Adaptive Sampling
- MF/ML sampling and surrogates
- Reliability
- Stochastic expansions
- Epistemic methods

Optimization
- Gradient-based local
- Derivative-free local
- Global/heuristics
- Surrogate-based

Calibration
- Tailored gradient-based
- Use any optimizer
- Bayesian inference

- Mixed aleatory/epistemic UQ
- Optimization under uncertainty

- Parallel execution
- HDF5 Output
- Direct Python interface

Develop simulation driver once; use in different kinds of studies

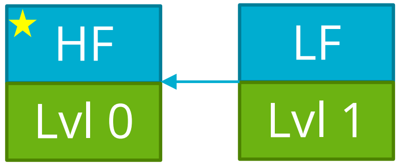
Multifidelity and Multilevel UQ Methods

Exploit hierarchies of models to compute lower variance estimates of moments at lower cost

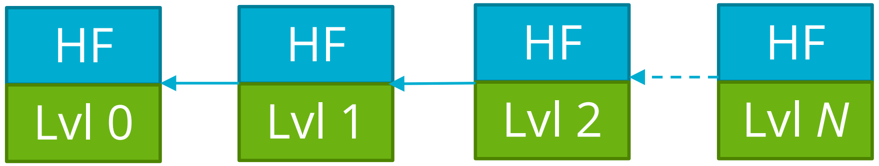
- Dakota 6.10 (May '19) included Control Variate Monte Carlo (CVMC), Multilevel (ML)MC, and MLCV MC.

Multifidelity Hierarchy Predictions of lower fidelity models are biased, regardless of model resolution	**Multilevel Hierarchy** Model predictions converge as resolution is improved

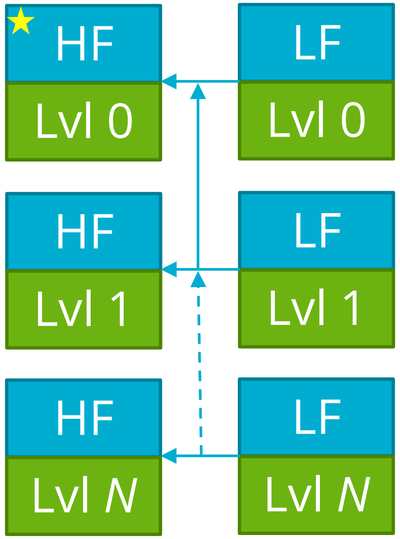
- Model costs, variance, and correlations are estimated by a pilot study, and a constrained optimization problem is solved to select a sample schedule that minimizes the cost for a desired estimator variance



CVMC
High and Low Fidelity
(LF prediction is biased)



MLMC
Multiple model resolutions
(Prediction converges as resolution is improved)

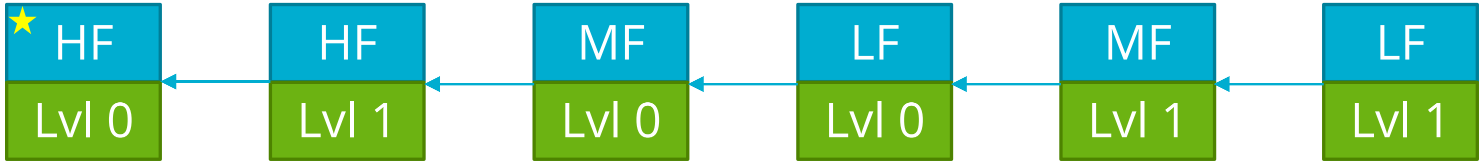


MLCV MC
(2D hierarchy)

Multifidelity and Multilevel UQ Methods

Lifting Restrictions on Model Hierarchies

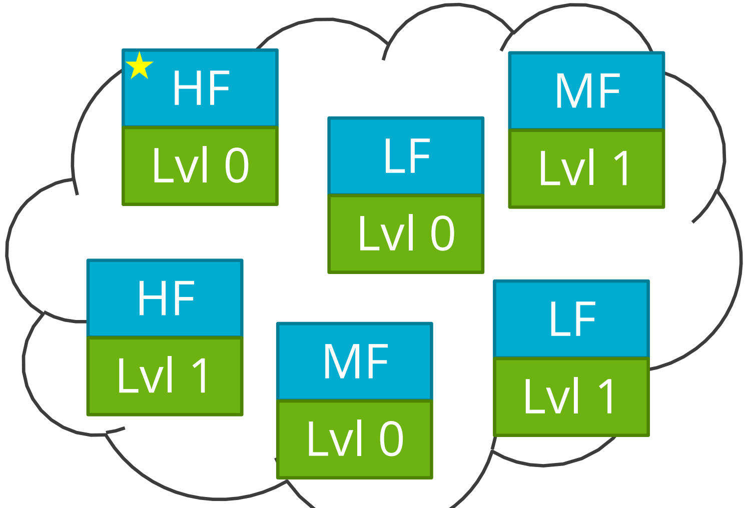
Multifidelity Monte Carlo - 6.15 (Nov ‘21)



1D hierarchy of models of unlimited depth; convergence requirement lifted

Approximate Control Variate – 6.15 (Nov ‘21)

- “Cloud” of models; no hierarchy assumed
- Recursion limit on variance reduction $(1-\rho^2)$ is lifted



Multifidelity and Multilevel UQ Methods

Other Recent Algorithmic and Usability Improvements

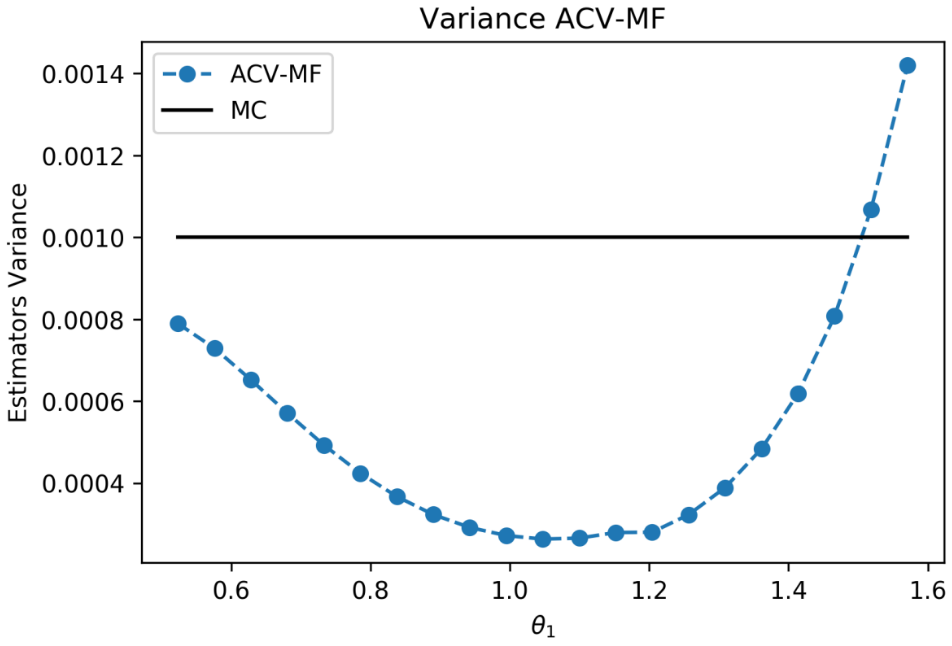
Pilot Projection - 6.16 (May '22)

Which estimator (CVMC, MLMC, ACV, etc) provides the greatest variance reduction at the least cost?

Geraci/Reuter's talk: *Multifidelity UQ workflows with Dakota's graphical user interface*

Model Tuning – 6.16 (May '22)

Tune solution level to achieve the best variance reduction and cost



Mike Eldred's talk: *Model tuning for multifidelity sampling in Dakota*

Batch Parallel Efficient Global Optimization (EGO)

Efficiently use parallel computing resources to perform global optimization

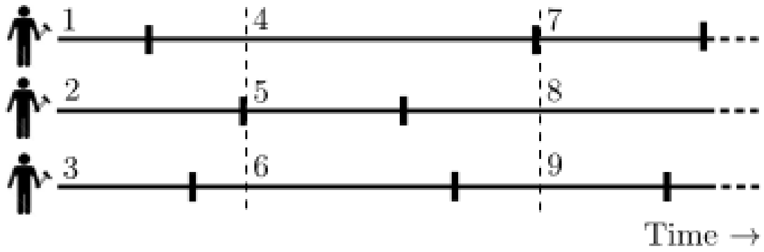
EGO uses the mean and variance prediction of a Gaussian process to balance exploration and exploitation

Original EGO Algorithm

1. Train a GP on an initial set of samples.
2. Identify candidate: $\underset{u}{\operatorname{argmax}} EI\left(\hat{G}(u)\right)$, where $EI\left(\hat{G}(u)\right) \equiv \mathbb{E}\left[\max\left(\hat{G}(u^*) - \hat{G}(u), 0\right)\right]$
3. Evaluate candidate using the truth model; incorporate into the GP's training set
4. If not converged, go to 2.

Improvement 1 - 6.12 (May '20)

Batch Sequential EGO. Instead of a single point, batches are added. The additional points are based on *hallucination.*



Improvement 2 - 6.14 (May '21)

Batch Parallel EGO. Evaluations and updates to the GP occur asynchronously.

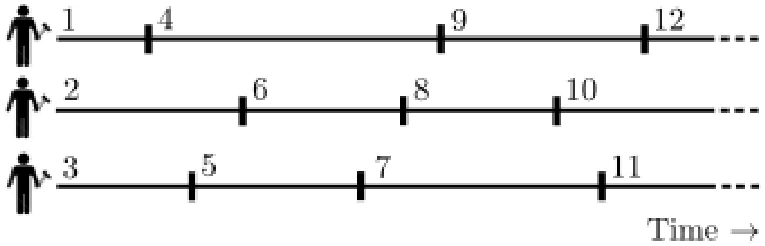


Figure courtesy Kandasamy et al. 2017

1. Tran, Anh, et al. https://doi.org/10.1007/s00158-021-03102-y
2. Tran, Anh, et al. https://doi.org/10.1016/j.cma.2018.12.033

Surrogates Library

Refresh and modularize Dakota's Surrogate Modeling Capabilities

Key Features of Dakota's new surrogate library:

- Export models from and import back into Dakota

```
model
  id_model = 'SurrogateModel'
  surrogate global
    dace_method_pointer = 'DesignMethod'
  experimental_gaussian_process
    export_model
      filename_prefix 'morris'
      formats binary_archive
    export_approx_variance = 'dak_gp_variances.dat'
```

- Python binding
- Rewritten Gaussian process and polynomial models

```
[4]: import dakota.surrogates as daksurr

nugget_opts = {"estimate nugget": True}
trend_opts = {
    "estimate trend": True,
    "Options": {"max degree": 2}
}

config_opts = {
    "kernel type": "squared exponential",
    "scaler name": "standardization",
    "Nugget": nugget_opts,
    "num restarts": 15,
    "Trend": trend_opts
}

gp = daksurr.GaussianProcess(xs, ys, config_opts)
gp_value = gp.value(ps)
gp_variance = gp.variance(ps)
gp_grad = gp.gradient(ps)
gp_hessian = [gp.hessian(p)[0, 0] for p in ps]
```

Examples Library

Dakota includes a large and growing collection of runnable examples

- Examples include
  - Dakota inputs
  - Drivers
  - Jupyter notebooks
  - Case studies
  - Tutorials
- Consistent Presentation
- Routinely Tested
- Included with Dakota downloads
- Soon to be a part of our unified documentation

Summary

Import a Python module into Dakota to use a decorated function it contains as a driver

Description

This example combines use of the Dakota direct python callback interface together with use of the `dakota.interfacing` Python module provided by Dakota to transparently convert from the incoming Python dictionary to Parameters and Response objects native to `dakota.interfacing`. This is done using an idiom supported in Python known as a decorator factory. More specific details for both the direct python callback in Dakota and the `dakota.interfacing` module can be found in the `linked` and `di` examples that are peers to this one.

Driver

The main function of the direct Python callback driver `driver.py` is:

```
@di.python_interface()
def decorated_driver(params, results):

    textbook_input = pack_textbook_parameters(params, results)
    fns, grads, hessians = textbook_list(textbook_input)
    results = pack_dakota_results(fns, grads, hessians, results)

    return results
```

Prior to this snippet, the driver imports the `dakota.interfacing` module as `di`, and the actual funxtion, gradient and hessian calculations are brought in from the `textbook` module.

The Python decorator is invoked by using the Python convention of the `@` followed by the

On the horizon..

Dakota's repositories will move to GitHub

- More easily explore and work with Dakota source
- Create and track feature requests and bug reports
- User support will move to GitHub Discussions

GitHub

New Documentation System