



Power updates  
and plant life extension

CASL-U-2013-0213-000



Consortium for Advanced Simulation of LWRs

L3:VUQ.SAUQ.P7.01

# Separate Effects V&V for Hydra-TH

Brian Williams

Lori Pritchett-Sheats

Los Alamos National Laboratory

Kevin Copps

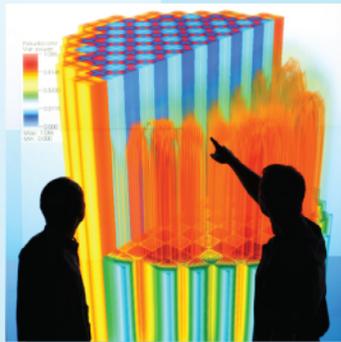
Laura Swiler

Sandia National Laboratories

Ernesto Prudencio

University of Texas Austin

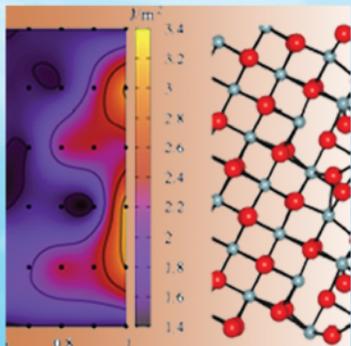
September 30, 2013



Engineering design  
and analysis



Science-enabling  
high performance  
computing



Fundamental science



Plant operational data



U.S. DEPARTMENT OF  
**ENERGY**

Nuclear Energy

# Separate Effects V&V for Hydra-TH

## CASL Milestone L3:VUQ.SAUQ.P7.01

Brian Williams<sup>1</sup>, Lori Pritchett-Sheats<sup>2</sup>, Kevin Copps<sup>3</sup>, Laura Swiler<sup>4</sup>, and Ernesto Prudencio<sup>5</sup>

<sup>1</sup>Statistical Sciences Group, Los Alamos National Laboratory

<sup>2</sup>Computational Physics and Methods Group, Los Alamos National Laboratory

<sup>3</sup>Validation and Uncertainty Quantification Processes Department, Sandia National Laboratories

<sup>4</sup>Optimization and Uncertainty Quantification Department, Sandia National Laboratories

<sup>5</sup>Institute for Computational Engineering and Sciences (ICES), UT-Austin

## 1. Introduction

Validation of computational models (referred to as *codes*) for specific applications involves quantitatively establishing their fidelity to representative observational data. Codes generally consist of multiple component models, each containing one or more input parameters. These inputs typically describe boundary and initial conditions, but may also represent uncertain quantities in closure laws needed to complete calculations of interest. Parameters in this latter category (referred to as *calibration* inputs) are often tuned to separate effects data representative of the phenomenon being modeled. Predictions of integral effects experiments tailored to the applications of interest are made based on these tuned parameters, and if they compare favorably to the observed data relative to its quantified observational error, the code is deemed validated for these applications.

The above approach to code validation typically ignores the fact that calibration inputs are inherently uncertain in an epistemic sense. Their optimal values are not known *a priori* to the analyst, a reality explicitly acknowledged by the act of tuning them to experimental data. In fact, uncertainty in the observational data used to tune such parameters should be formally transferred to their estimation. Furthermore, this implied uncertainty should be carried through the predictions upon which code validation decisions are made, and eventually accounted for in quantifying uncertainty in figures of merit used to make decisions in reactor applications. Unfortunately, these fundamental steps of rigorous uncertainty quantification are often not conducted in practice.

This fact is acknowledged by the Verification, Validation and Uncertainty Quantification (VUQ) focus area of the Consortium for Advanced Simulation of Light Water Reactors (CASL). VUQ has undertaken several projects over the last three years to bring mathematical rigor to the *probabilistic inference* of calibration inputs. For example, Markov chain Monte Carlo (MCMC) methods such as DiffeRential

Evolution Adaptive Metropolis (DREAM) and Delayed Rejection Adaptive Metropolis (DRAM) have been previously made available to CASL researchers for Bayesian calibration of model inputs through the Design Analysis Kit for Optimization and Terascale Applications (DAKOTA) developed and distributed by Sandia National Laboratories [1].

Bayesian calibration involves assuming an initial (*prior*) probability distribution for the calibration inputs, which often derives from subject matter expert opinion and analysis of available separate effects experiments, but may be diffuse or non-informative if little information about these parameters is available initially. Experimental data are then identified to cover integral effects of specific relevance to the application at hand. These data, in conjunction with their assumed or inferred uncertainties, are used to update the probability distribution of the calibration inputs. This updated distribution, referred to as the *posterior*, thus accounts for both the initial assessment of uncertainty in the parameter values and the knowledge gained through constraints imparted by the integral effects experimental data. Unfortunately this posterior distribution is generally not available analytically, but may be sampled via MCMC (for example) using modern computational tools.

In many applications, the code(s) needed to solve the problem at hand are sufficiently complex that only tens or hundreds of runs may be feasible within the allowable time horizon. Direct application of MCMC may require thousands of runs or more, limiting its usefulness. However, a small number of carefully planned code runs can be utilized to build a fast surrogate model, which can be used to predict code output at arbitrary parameter settings (within specified bounds) with quantified uncertainty. A validated surrogate can then be used in place of direct code runs in a MCMC implementation, allowing a good approximation to the calibration input posterior distribution to be sampled as necessary. Such a surrogate-based approach to Bayesian model calibration has been implemented in the Matlab package Gaussian Process Models for Simulation Analysis (GPMSA) developed at the Los Alamos National Laboratory (LANL).

GPMSA as currently implemented in Matlab is not compatible with the Virtual Environment for Reactor Analysis (VERA), the CASL platform for deployment of predictive, science-based simulation technology to support performance enhancement of light water reactors. To address this, VUQ has had an ongoing collaboration with the Institute for Computational Engineering and Sciences (ICES) at the University of Texas Austin to deploy GPMSA through their Quantification of Uncertainty for Estimation, Simulation, and Optimization (QUESO) library [2]. The QUESO library can be interfaced through the current release of DAKOTA, which is the designated VERA component supporting uncertainty analysis for CASL applications.

Prior to calibration of model parameters in codes, the codes should have been rigorously verified. The verification process involves several aspects including

software quality assurance (SQA), code verification and solution verification. Code verification typically involves comparison of code calculations against analytical or manufactured solutions, while solution verification analyzes numerical convergence of code solutions as meshes are successively refined. In the coming fiscal year, VUQ will integrate the *Percept* [3] software developed at Sandia National Laboratories for solution verification analysis into VERA.

This report has two primary objectives: (1) Demonstrate solution verification analysis tools in *Percept*, and (2) Provide an update on the status of GPMSA deployment in QUESO. Section 2 briefly describes the application used to demonstrate *Percept* and DAKOTA-QUESO-GPMSA. This application involves large-eddy simulation of a lid-driven cavity (LDC) flow, which serves as a verification and validation benchmark problem for Hydra-TH, a VERA component used for CFD calculations of thermal hydraulic phenomena. Section 3 provides an overview of *Percept* with a demonstration of solution verification analysis on the LDC flow application. Section 4 provides an overview of GPMSA and describes its implementation in DAKOTA-QUESO for the LDC flow application. Section 5 concludes with a discussion of the next steps planned for evolution of these capabilities in the upcoming fiscal year.

## 2. Lid-driven Cavity Flow Demonstration Problem

The LDC flow benchmark problem for Hydra-TH is described in detail in Section 3.1 of the Hydra-TH Verification and Validation (V&V) manual [4]. We provide a brief discussion here to orient the reader with basic elements of this benchmark.

This problem focuses on the large-eddy simulation of a LDC flow at a Reynolds number of  $Re = 10,000$ . Calculations are performed with Hydra-TH for each of two turbulence models, the Smagorinsky subgrid-scale (SSGS) model and the wall-adapted large eddy (WALE) subgrid-scale model. The solution verification study conducted with *Percept* considered three hexahedral grids, starting from a coarse mesh with  $32 \times 32 \times 16$  elements (reflecting cavity dimensions of 1 unit in the  $x$  and  $y$  directions, and 0.5 unit in the  $z$  direction) and continuing with two successive refinements constructed as described in Section 3. The grid spacing is geometrically stretched away from the wall in the stream-wise  $x$  and vertical  $y$  directions, and uniform in the span-wise  $z$  direction.

Both the SSGS and WALE turbulence models have three calibration inputs that are exposed to the user through the Hydra-TH control file. The *Percept* solution verification studies were conducted with these parameters set to their nominal values, while the demonstration of Bayesian input calibration was carried out on the three parameters of the SSGS model. Table 1 provides the nominal values for the SSGS and WALE inputs, along with bounds on the allowable variation of the SSGS inputs.

Table 1. Calibration inputs for SSGS and WALE turbulence models.

SSGS				WALE	
Parameter	Nominal	Minimum	Maximum	Parameter	Nominal
Cs	0.18	0	0.36	Cw	0.5
Prandtl	0.8889	0.8	1	Prandtl	0.8889
Schmidt	1	0.5	1.5	Schmidt	1

The outputs considered for calibration are mean velocity in the stream-wise  $x$ -direction ( $u$ ) computed along the vertical centerline, and mean velocity in the vertical  $y$ -direction ( $v$ ) computed along the horizontal centerline. The calculated kinetic energy profile at the nominal parameter settings was observed to reach a stationary state prior to the passage of 100 time units, and mean velocities computed from calculated velocities in the 100 to 500 time unit window constitute the  $u$  and  $v$  figures of merit. All calculations for the DAKOTA-QUESO-GPMSA demonstration were made using the medium resolution ( $64 \times 64 \times 32$  element) mesh constructed for the LDC verification study reported in the Hydra-TH V&V manual [4], as Section 3.1.3 of this manual indicates convergence of mean velocities was qualitatively observed at this grid spacing.

DAKOTA was run to generate 50 Latin hypercube samples in the three calibration inputs of the SSGS turbulence model, and Hydra-TH was run at each setting to obtain calculated mean  $u$  and  $v$  velocities. The Latin hypercube parameter settings along with the calculated mean  $u$  and  $v$  velocities were used to construct surrogate models for fast prediction of mean  $u$  and  $v$  velocity with quantified uncertainty, valid for arbitrary parameter settings within the prescribed ranges of Table 1. Specifically, the 50 calculated  $u$  velocities are centered and scaled, as are the 50 calculated  $v$  velocities. The transformed calculations are stacked and subjected to principal component analysis, and the three eigenvectors associated with the three largest singular values are selected to represent the observed joint variation in  $u$  and  $v$  velocities from the 50 runs. In GPMSA, the surrogate models have the following form,

$$\eta(t) = \mathbf{k}_1 w_1(t) + \mathbf{k}_2 w_2(t) + \mathbf{k}_3 w_3(t) + e$$

where  $t$  denotes the three calibration inputs,  $\eta(t)$  denotes the surrogate model evaluated at  $t$ ,  $\mathbf{k}_i$  denotes the  $i$ -th eigenvector having associated coefficient  $w_i(t)$ , and  $e$  denotes a Gaussian noise process collectively representing the contribution to output variation from unused eigenvectors. The coefficients  $w_i(t)$  are modeled as independent Gaussian processes. Additional details pertaining to how predictions with quantified uncertainties are generated from this representation are provided in [5].

The experimental data on  $u$  and  $v$  velocities published by [6] were used to calibrate the three SSGS inputs of Table 1. The prior distribution on the calibration inputs was assumed to be uniform on the ranges given in Table 1. Figure 1 plots the

calculated mean  $u$  and  $v$  velocities from the 50 Hydra-TH runs employing the SSGS turbulence model with the experimental data.

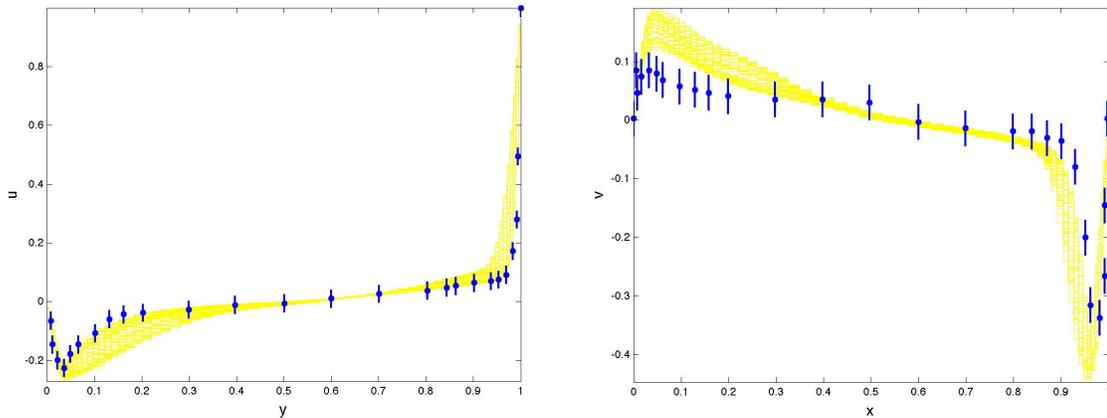


Figure 1. Plot of 50 Hydra-TH calculations from variation of SSGS inputs (yellow) and experimental data with  $3\text{-}\sigma$  uncertainty (blue).

The Percept verification study described in Section 3 considered four figures of merit in addition to the two mean velocities introduced above, namely elements of the Reynolds stress tensor viewed along particular slices of the LDC spatial domain.

The 50 Hydra-TH runs employing the SSGS turbulence model were automated using the DAKOTA fork simulation interface described in Section 13.3.3 of the DAKOTA users manual [1]. The DAKOTA input file used to set up the 50 Hydra-TH runs is provided in Appendix A. DAKOTA was run with this input deck by invoking the command

```
dakota -i <DAKOTA input file>
```

The keyword

```
analysis_driver = 'driver.sh --submit'
```

given in the input file causes DAKOTA to run the `driver.sh` script of Appendix B in submit mode. DAKOTA proceeds by running the `dprepro` script, which creates a work directory for each run (50 in this case) into which a Hydra-TH control file is copied that has input settings specified by the Latin hypercube sampling plan for that run. These run-specific Hydra-TH control files are created from the generic Hydra-TH control file provided in Appendix C, in which tags for each of the three SSGS inputs being varied are replaced by `dprepro` with actual input values for each run. The driver script also outputs the file `job.list`, which lists the 50 commands that are issued subsequently to run Hydra-TH and generate the necessary figures of merit from each run. The script `dprepro` itself does not actually run Hydra-TH, and so it completes its tasks by copying a dummy results file into each work directory so

DAKOTA (which expects results to have actually been generated from running a code as part of the fork interface) exits cleanly.

Once `job.list` has been created, the jobs were then submitted to the scheduler by running the `submit_all.sh` script of Appendix D with `job.list` as its argument. All Hydra-TH runs for this demonstration were conducted on the 154-node, 2464 CPU Pinto machine belonging to the LANL turquoise network.

Upon completion of each job, Hydra-TH writes a binary `plotstat` file in the Exodus II file format to each work directory. The standard use case for Hydra-TH is to manually extract the figures of merit from `plotstat` using the Kitware software ParaView. Given the number of jobs required to generate the needed figures of merit for surrogate construction, we developed a new Hydra-TH use case to automate the process of extracting figures of merit. The steps in ParaView required to extract mean  $u$  and  $v$  velocities were collected in an interactive session, and then exported by ParaView to the python script of Appendix E. The DAKOTA input deck of Appendix A was then invoked again, but now with a modified `analysis_driver` keyword:

```
analysis_driver = 'driver.sh --collect'
```

This runs the `driver.sh` script again, but now in `collect` mode. The python script of Appendix E is invoked in each work directory, resulting in the creation of two output files `u.csv` and `v.csv` containing the extracted  $u$  and  $v$  velocities at their corresponding  $y$  and  $x$  coordinate values, respectively, along with additional information not needed for our demonstration. The script `driver.sh` then parses these output files and creates new output files `u.txt` and `v.txt` for each run that contain only the  $u$  vs.  $y$  and  $v$  vs.  $x$  figures of merit required for our demonstration, and wraps up by concatenating these new files into a single `results.out` file for each run. Finally, the `make_output.sh` script of Appendix F is run to collate the `results.out` files from each run into the single matrix of stacked  $u$  and  $v$  velocities required by DAKOTA for subsequent invocation of QUESO-GPMSA.

### 3. Percept Verification Study

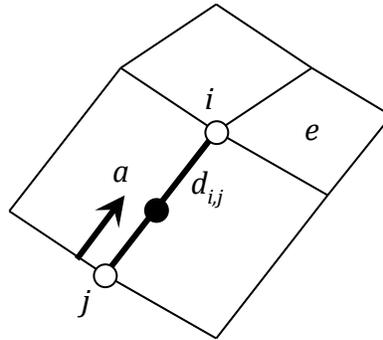
Percept is a software package that provides tools for code and solution verification [3]. Percept was used in this study to:

1. produce a uniformly refined sequence of grids,
2. extrapolate the value of the figures of merit as the mesh size approached zero, including the mean velocity and Reynolds stress tensor,
3. and compute the convergence rates of these figures of merit with respect to the mesh size.

Percept is a small set of libraries written in the C++ and Python language. It is based on the Sierra Toolkit and developed in concert with the Sierra computational simulation package at Sandia National Laboratories [7]. In addition, Percept is currently distributed with an open source license as part of Sandia's Trilinos package [8].

Because meshes used to model this physics work best with a strong grading near boundaries, a special type of mesh refinement implemented in Percept was used to place the new nodes, or vertices. The refined meshes start from an existing coarse mesh, and the desired grading was created in the initial coarse mesh.

The algorithm in Percept for respecting the grading calculates the position of a new node on mid-edges is as follows. It is assumed that the spacing of new nodes varies linearly on the edges, and the spacing is integrated along the edge to get a new midpoint position.



Let us define a *spacing* local to the node  $i$  of an element  $e$  at one end of an edge  $ij$ . Letting  $d_{ij}$  be the distance between the nodes  $i$  and  $j$ , the spacing associated with the edge and element is,

$$s_{ij}^{i,e} = \frac{1}{\|J_e^{-1} d_{i,j}\|}$$

The element Jacobian matrix  $J_e$  relates the physical coordinate derivatives to the reference coordinate derivatives.

$$\frac{\delta}{\delta \xi} = J_e \frac{\delta}{\delta x} \quad \frac{\delta}{\delta x} = J_e^{-1} \frac{\delta}{\delta \xi}$$

We define the spacing at a node to be the average of the spacings of each of the elements connected to that node,

$$s_{ij}^i = \text{avg } s_{ij}^{i,e}$$

A new midpoint node position on that edge, relative to node  $j$ , is then computed by integrating this linear variation in spacing. The distance  $a$ , of the new node from node  $j$ , is

$$a = \frac{s_{ij}^i + 3s_{ij}^j}{4(s_{ij}^i + s_{ij}^j)}$$

A similar method is used to place midface, centroid, nodes on quadrilateral faces from the edges to respect similar spacing. And a similar method is used again for centroid mid-element nodes of hexahedral elements.

Figure 2 illustrates this respect spacing algorithm on meshes constructed for the Hydra-TH LDC calculations.

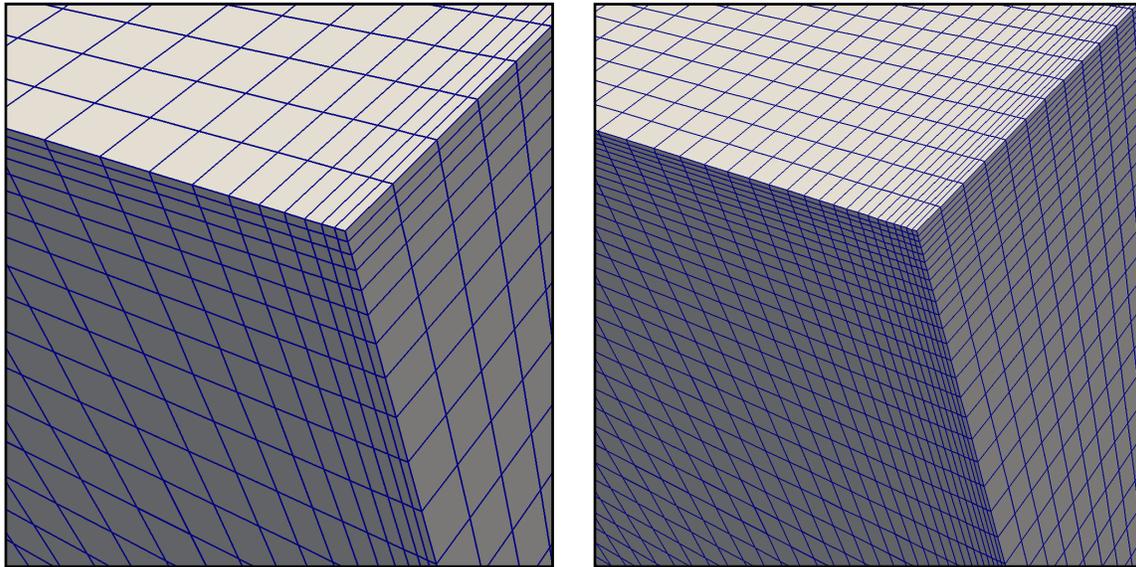


Figure 2. One corner of the coarse mesh, of  $32 \times 32 \times 16$  elements, exhibiting the initial grading in two of the axis aligned directions (left). The same corner of the first refined mesh, of  $64 \times 64 \times 32$  elements, produced by Percept using the respect spacing algorithm (right).

We analyzed the convergence of the fields in this simulation, varying in the  $x$  and  $y$  directions.

- $x$ -direction:
  - the mean velocity component  $v(x)$
  - elements of the Reynolds stress tensor:  $\langle v'v' \rangle(x)$  and  $\langle u'v' \rangle(x)$
- $y$ -direction:
  - the mean velocity component  $u(y)$
  - elements of the Reynolds stress tensor:  $\langle u'u' \rangle(y)$  and  $\langle u'v' \rangle(y)$

At any point at a specific  $x$  or  $y$  coordinate, as the mesh size  $h$  is refined,  $h$ ,  $\frac{1}{2}h$ ,  $\frac{1}{4}h$ , we see that the value of the Reynolds stress tensor may or may not converge monotonically. Points with non-monotonic convergence are more often present in the SSGS model, but are also noticeable in the WALE model. There may be a small number of non-monotonic points in the mean velocity results, but this is not obvious looking at the plots. This is not a serious issue in analyzing the convergence and convergence rates. It is most often the case with finite element models or other discretizations of partial differential equations that field values at points do not converge monotonically.

Instead of analyzing the convergence of these fields pointwise, we instead integrate the fields over the domain and then analyze the convergence of these global norms. For this study we approximate the L2 norm of the velocity, and L2 norms of the components of the Reynolds stress tensor over the range of  $x$  and  $y$  in the domain. For example, we use the trapezoid quadrature to approximate the L2 norm of the mean velocity  $u(x)$ , integrated along the range of  $x$ ,

$$\|u(x)\|_{L^2} = \int_0^1 u^2 dx$$

To compute convergence rates, and extrapolate the value of the norms, we avoid variability of different forms of regression by using a procedure called “Robust Multi-Regression,” or RMR, described in [9]. The RMR procedure produces a set of constrained regressions of the data over the sequence of meshes using several regression techniques, L1, L2,  $L^\infty$ , weighted L2,  $p_{\text{theo}}$  L2,  $p_{\text{lower}}$  L2, and  $p_{\text{upper}}$  L2. In addition, the method may use L4, L8, Tikhonov, LASSO, and weighted variants of each. We used a mixture of 12 regressions in the procedure employed for this report. We will not go into the details of these various regression algorithms, but we report the median, mean and standard deviation of the convergence rates output by this procedure.

Figure 3 presents a L2 regression performed under the RMR for the SSGS model. The L2 norm of the mean velocity component,  $\|u(x)\|_{L^2}$ , is regressed on mesh size,  $h$ , with mesh size plotted on a log-scale.

Tables 2 and 3 present convergence rates of the L2 norms of the six fields computed using the SSGS and WALE models, respectively, integrated along the corresponding range of  $x$  or  $y$  using the RMR (robust method of regression), and using the three meshes in the sequence with mesh sizes  $h$ ,  $\frac{1}{2}h$ ,  $\frac{1}{4}h$ .

Tables 4 and 5 present L2 norms of the six fields computed using the SSGS and WALE models, respectively, for the three meshes in the sequence. The norms are approximated using the trapezoidal rule.

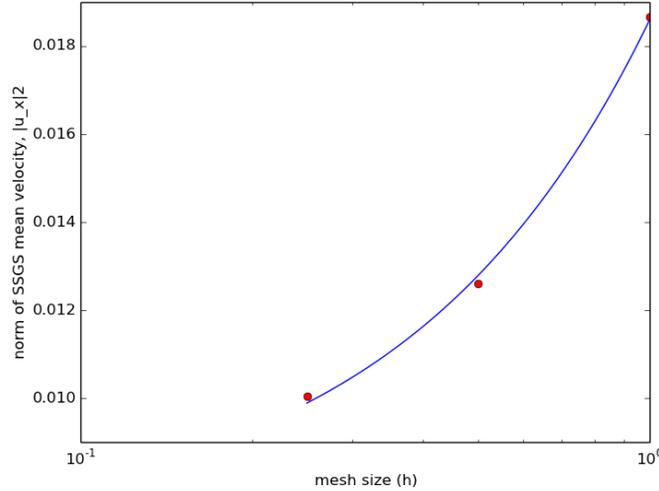


Figure 3: L2 regression of the values of  $\|u(x)\|_{L^2}$  on  $h$  for the SSGS model.

Table 2: Convergence rates of the L2 norms of the six fields for the SSGS model. N/A entries indicate quantities exhibiting non-monotonic behavior.

SSGS convergence rates	$v(x)$	$\langle v'v' \rangle(x)$	$\langle u'v' \rangle(x)$	$u(y)$	$\langle u'u' \rangle(y)$	$\langle u'v' \rangle(y)$
Median	1.0028	1.0056	1.0002	1.0002	N/A	N/A
Mean	1.0061	0.93699	0.75574	0.99905	.	.
Std. deviation	0.0081	0.32836	0.35614	0.011849	.	.

Table 3: Convergence rates of the L2 norms of the six fields for the WALE model. N/A entries indicate quantities exhibiting non-monotonic behavior.

WALE convergence rates	$v(x)$	$\langle v'v' \rangle(x)$	$\langle u'v' \rangle(x)$	$u(y)$	$\langle u'u' \rangle(y)$	$\langle u'v' \rangle(y)$
Median	1.0007	N/A	1.0029	0.99965	N/A	N/A
Mean	1.1677	.	0.88577	0.75059	.	.
Std. deviation	0.3722	.	0.28467	0.35247	.	.

Table 4. L2 norms of the six fields for the SSGS model. Non-monotonic behavior is highlighted in red.

SSGS L2 norms mesh size	$\ \bullet\ _{L^2}$					
	$v(x)$	$\langle v'v' \rangle(x)$	$\langle u'v' \rangle(x)$	$u(y)$	$\langle u'u' \rangle(y)$	$\langle u'v' \rangle(y)$
$h$	0.018665	0.065560	0.030541	0.025155	0.033246	0.006287
$\frac{1}{2}h$	0.012596	0.064237	0.018513	0.017049	0.045295	0.012929
$\frac{1}{4}h$	0.010043	0.033335	0.003693	0.012755	0.044755	0.005896

Table 5. L2 norms of the six fields for the WALE model. Non-monotonic behavior is highlighted in red.

WALE L2 norms mesh size	$v(x)$	$\langle v'v' \rangle(x)$	$\langle u'v' \rangle(x)$	$u(y)$	$\langle u'u' \rangle(y)$	$\langle u'v' \rangle(y)$
$h$	0.013202	0.045993	0.019010	0.009378	0.012933	0.000375
$\frac{1}{2}h$	0.009515	0.049642	0.015989	0.010031	0.033860	0.007982
$\frac{1}{4}h$	0.008970	0.020846	0.001710	0.010648	0.028179	0.006516

Figure 4 plots the calculated mean  $v$  and  $u$  velocities used in this mesh convergence study for the SSGS and WALE models.

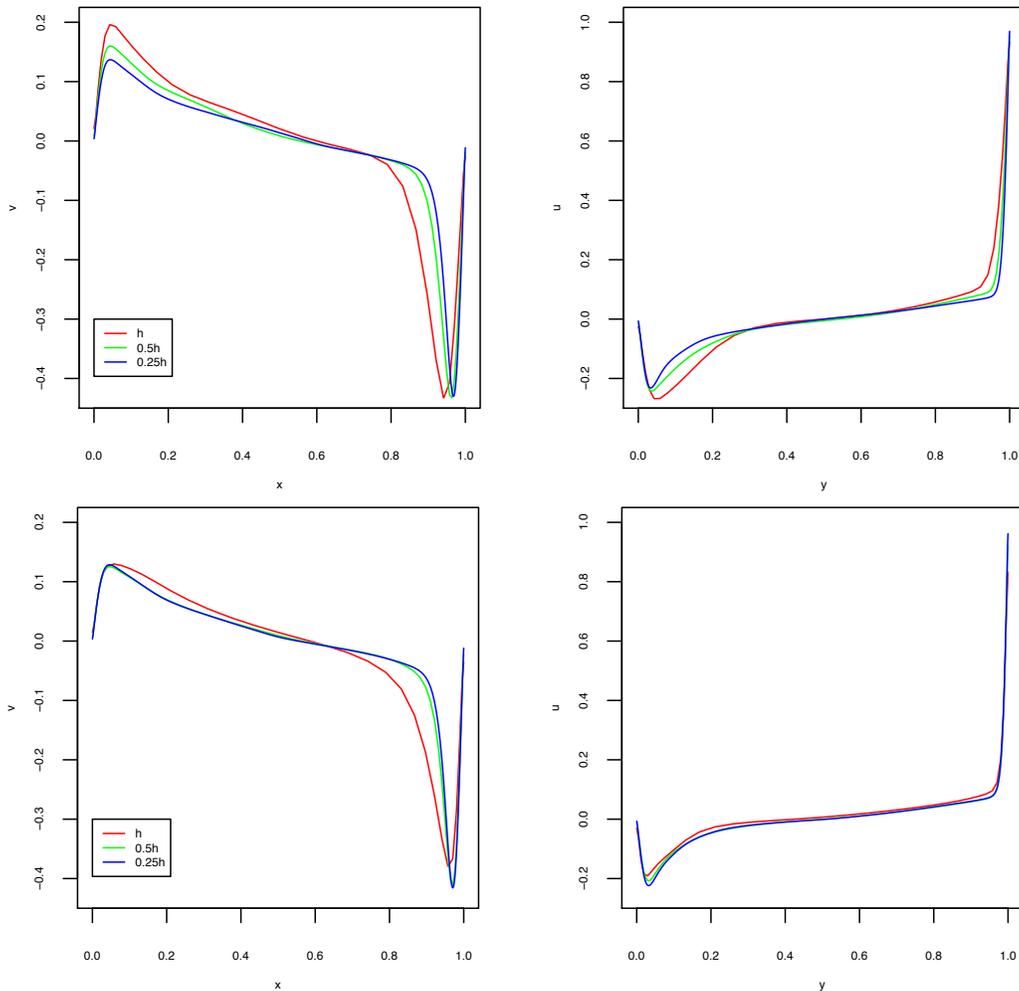


Figure 4. Calculated mean velocities for the SSGS (top) and WALE (bottom) models.

Figures 5 and 6 plot the two calculated Reynolds stress tensor components in the stream-wise  $x$  and vertical  $y$  directions, respectively, used in this mesh convergence study for the SSGS and WALE models.

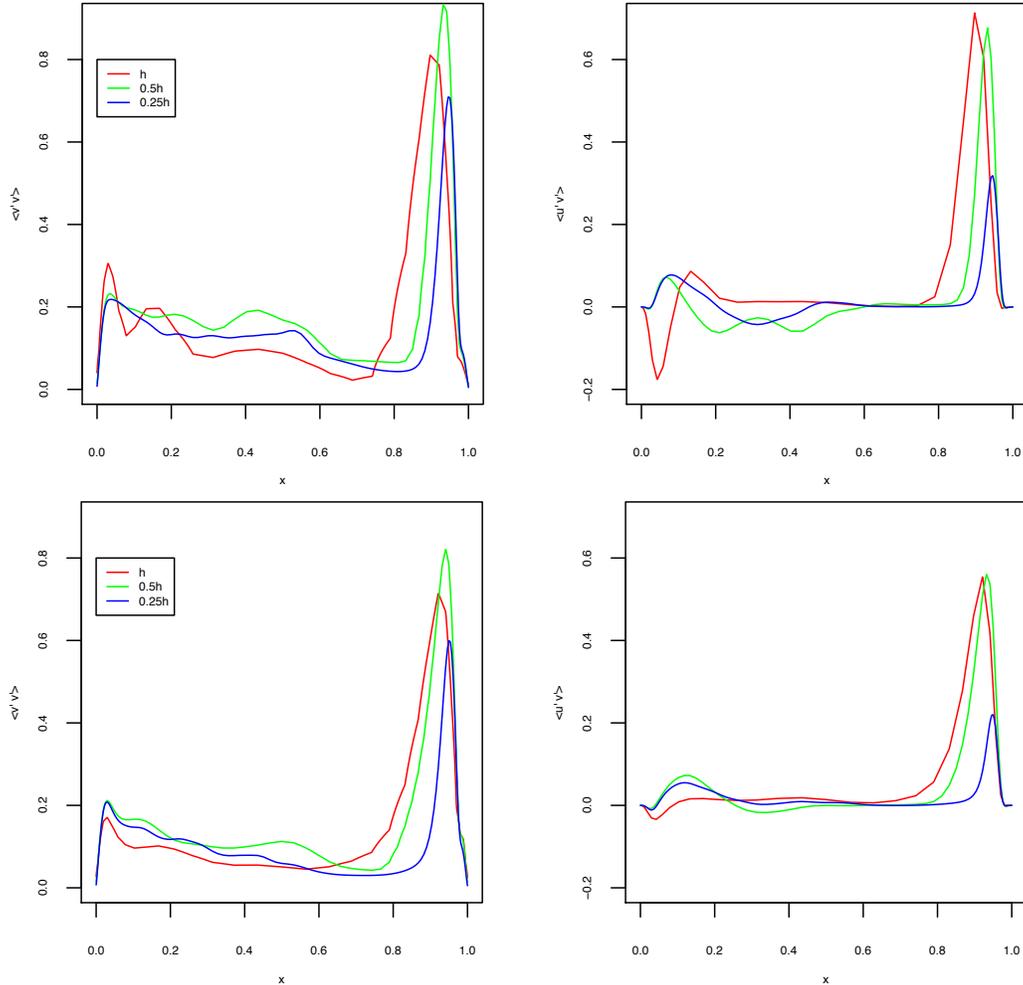


Figure 5. Calculated Reynolds stress tensor components in the stream-wise  $x$  direction for the SSGS (top) and WALE (bottom) models.

The first thing to note about the convergence behavior of the SSGS model fields from Table 2 and Figure 4 is that the mean  $v(x)$  and  $u(y)$  velocities seem to converge relatively well, at a rate of nearly one, i.e.,  $O(h)$ . Hydra-TH developers report  $O(h^2)$  convergence in the global kinetic energy norm. This difference may be due in part to interpolation errors in our analysis introduced by obtaining our figures of merit from lineouts in ParaView. The components of the Reynolds stress tensor in the SSGS model do not fare nearly as well, and as seen in Table 4 and Figure 6 the two components varying in the  $y$ -direction clearly do not behave monotonically. Because of the wide standard deviation in the rates from the regression analysis, we would also hesitate to say these converge at a specific asymptotic rate.

Regarding the WALE model, we also see from Table 3 and Figure 4 that the convergence of the mean velocities fares relatively well, but the asymptotic rate is not nearly as solid as we see with the SSGS model. This is reflected in the large standard deviations in the convergence rate over the family of regressions we

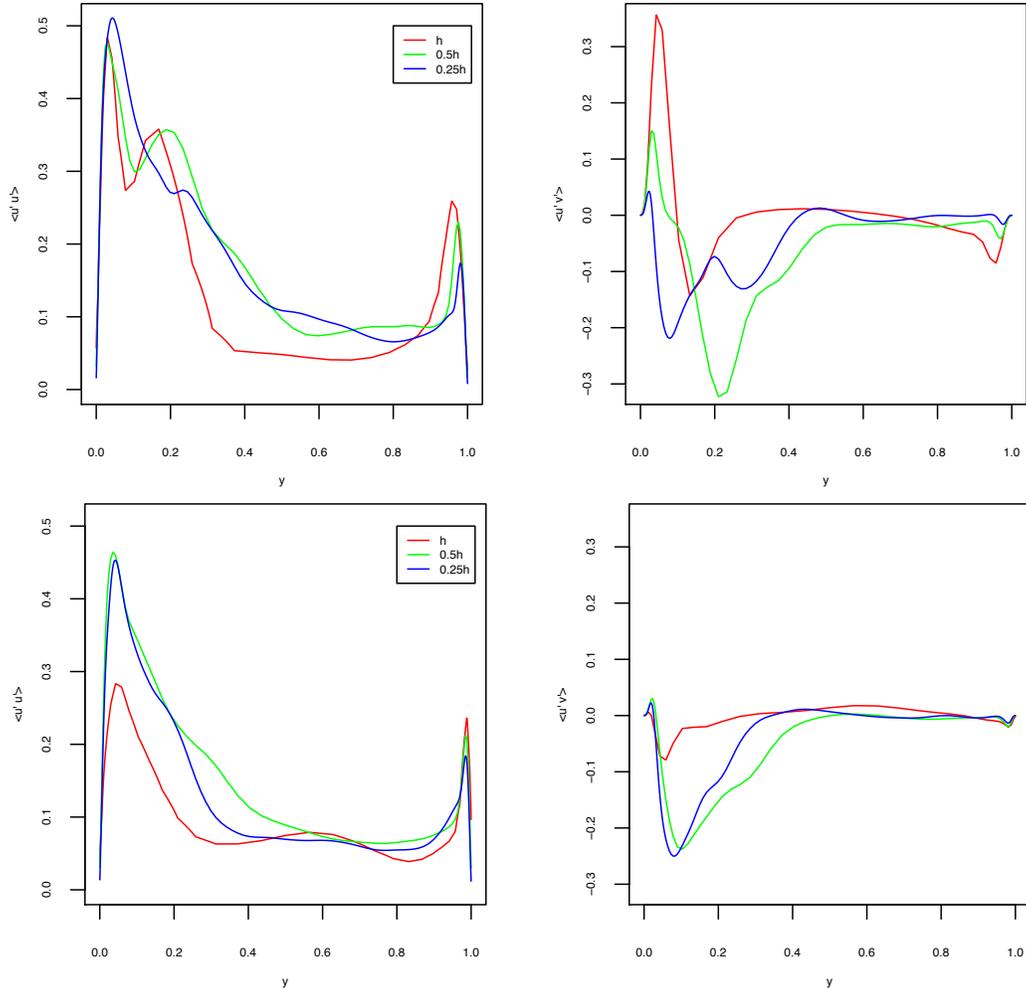


Figure 6. Calculated Reynolds stress tensor components in the vertical  $y$  direction for the SSGS (top) and WALE (bottom) models.

performed. Further, with the WALE model, we see from Table 5 and Figures 5 and 6 that all but one of the components of the Reynolds stress tensor either does not behave monotonically, or has a relatively higher standard deviation in the regression.

The convergence behavior of the calculated mean velocities and Reynolds stress tensor depends on the adopted time averaging, which was 100 to 500 time units in the calculations reported on here. Additional analysis on the sensitivity of these calculations to factors such as time averaging will be necessary before more definitive conclusions on the convergence behavior of these calculations can be reached. However, if further computations at finer mesh resolutions were affordable we may have seen the Reynolds stress tensor components enter a more asymptotic region with clear monotonic behavior.

## 4. DAKOTA-QUESO-GPMSA Demonstration of Bayesian Calibration

The statistical model of the experimental data employed in GPMSA for Bayesian calibration combines the Gaussian process surrogate model introduced in Section 2 with empirical bias correction and observational error terms,

$$\mathbf{y} = \eta(\theta) + \delta + \epsilon$$

where  $\theta$  is the unknown, best value of the calibration inputs  $t$  for fitting experimental data  $\mathbf{y}$  consisting of stacked  $u$  and  $v$  velocities,  $\delta$  is the bias correction term often referred to as *model discrepancy*, and  $\epsilon$  models observational error. The model discrepancy term provides an empirical representation of model form uncertainty, in that it attempts to model observed differences between the “best” parametric code representation of the stacked  $u$  and  $v$  velocities as approximated by the fast surrogate,  $\eta(\theta)$ , and the experimental data  $\mathbf{y}$  within bounds established by the observational error process  $\epsilon$ .

A nonparametric regression model is utilized by GPMSA to model  $\delta$  for our demonstration. For each of the mean  $u$  and  $v$  velocities, 7 Gaussian kernels with specified standard deviations are centered at locations in index space (i.e., the  $y$  or  $x$  coordinate dimension, respectively) of interest for capturing potential model form uncertainty. For example, the discrepancy model  $\delta_u$  for the  $u$  velocity takes the form

$$\delta_u = \mathbf{d}_1 v_1 + \cdots + \mathbf{d}_7 v_7$$

Here, the  $i$ -th kernel is evaluated on the grid of index coordinates corresponding to the locations at which experimental data on  $u$  velocity is observed to generate vectors  $\mathbf{d}_i$ , and the corresponding coefficients are modeled as independent zero-mean Gaussian given an assumed common variance. This model is joined with a corresponding discrepancy representation for  $v$  velocity,  $\delta_v$ , to construct the  $\delta$  term in the above Bayesian calibration model. Figure 7 plots the 7 kernels utilized in the discrepancy model for  $u$  velocity ( $\delta_u$ ). This set of kernels is also used in the discrepancy model for  $v$  velocity ( $\delta_v$ ).

For both  $u$  and  $v$  velocities, the best model fit does not represent the experimental data well for small values of the index variables ( $y$  and  $x$ , respectively). As shown in Figure 7, the statistical model utilizes closer and tighter kernels to model this localized behavior, while wider and more spread out kernels suffice for the much larger region of index space in which the model/data difference remains near zero with little variation.

As indicated previously in Section 2, the unknown “best” calibration input value  $\theta$  is assigned a uniform prior on the ranges of Table 1. The experimental data of Figure

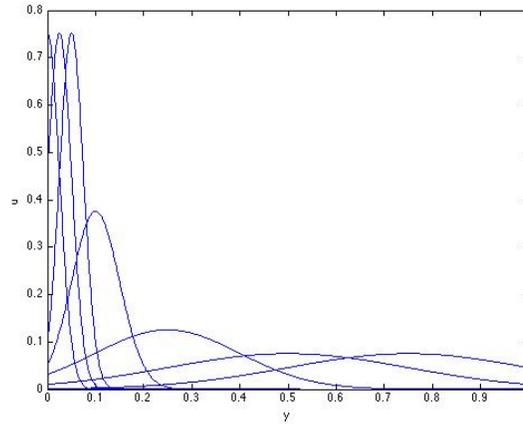


Figure 7. Kernel basis functions used in discrepancy models for the  $u$  velocity.

1 are combined with the fast surrogate for Hydra-TH calculations of mean  $u$  and  $v$  velocities introduced in Section 2 according to the statistical model representation above to arrive at the posterior distribution of  $\theta$ , which is the critical quantity of interest for follow-on uncertainty quantification studies that properly account for parametric uncertainty. In other words, the posterior distribution of  $\theta$  represents our most up-to-date knowledge about uncertainty in the “best” value of the calibration inputs after accounting for both our prior knowledge about more likely values for these inputs and the constraints on these values imposed by the experimental data. A more detailed description of the technical methods underlying this approach to Bayesian calibration is provided in [5].

As the posterior distribution of  $\theta$  is not available analytically, DAKOTA invokes QUESO-GPMSA with the input file of Appendix G to sample this posterior using DRAM. The DRAM method combines two essentially distinct but related methods for improving mixing of the Markov chain so that it covers the posterior distribution more effectively for any given number of iterations.

1. Delayed Rejection (DR). When a sample is rejected by the Metropolis sampler, instead of being immediately discarded, a second stage proposal sample that is allowed to depend on the previous rejected sample is generated with an acceptance probability that is specially calculated to guarantee convergence to the posterior density [10].
2. Adaptive Metropolis (AM). This method relies on global adaptation of the proposal covariance based on previously accepted samples in the chain. At specified intervals the proposal covariance is updated to reflect information gained from the previous samples drawn by the chain. Unlike DR, the adaptations persist through subsequent steps of the chain [11].

QUESO implements another MCMC algorithm referred to as multilevel [12]. This method works by sampling successively from a collection of target distributions, starting from the prior distribution and ending at the posterior distribution. This

sequence of target distributions is established to sample efficiently from multimodal posterior distributions, and may be user-specified or automatically generated by QUESO. Multilevel also produces an estimate of the model *evidence*. The model evidence, also known as the marginal likelihood, is often used for Bayesian model selection (e.g. in the computation of Bayes factors). Models with larger evidence values with respect to predetermined reference data are generally preferred (subject to possible modification by prior model preferences), with the strength of preference for a particular model determined by the relative size of its evidence value compared with the evidence values of its competitors [13].

The DAKOTA implementation of QUESO-GPMSA involves a DAKOTA class that instantiates the QUESO-GPMSA application, provides the correct experimental and simulation data structures to QUESO-GPMSA, initializes the proposal covariance matrix, and asks QUESO-GPMSA to perform the DRAM MCMC. The DAKOTA class is called `NonDGPMSABayesCalibration`. Details about the class structure can be found in the DAKOTA developer's manual at the following website, <http://dakota.sandia.gov/docs/dakota/stable/html-dev/index.html>.

From the DAKOTA user perspective, the DAKOTA user sees an input file that is similar to other DAKOTA input files. The user must define the variables (e.g. the ranges on the calibration variables as well as any "configuration" or state variables), the responses from the simulation (in this case, the velocity fields as shown in Figure 1), the interface to the simulation, the experimental data, and the method used (in this case, Bayesian calibration using GPMSA). An example DAKOTA input file is shown in Appendix G.

Stepping briefly through each section of the DAKOTA input in Appendix G: the first section (strategy) states that we are running a single method (in this case, Bayesian calibration). The second section (method) defines the particular method with the keywords

```
bayes_calibration gpmsa
```

The number of emulator samples defines the number of times we run the Hydra-TH simulation code to generate data with which to build the emulator. In this case, we indicate that we will perform 50 runs of Hydra-TH with the specification

```
emulator_samples = 50
```

In this example, we ran the Hydra-TH runs previously and have produced a text file with all of the results, so we can simply read that into DAKOTA and not execute Hydra-TH, although we could run Hydra-TH inline if desired. The specification

```
import_points_file = 'williams9.txt' freeform
```

indicates to DAKOTA that the results should be read in from the file `williams9.txt`. Finally,

```
samples = 50000
```

indicates that there are 50,000 samples to be taken in the MCMC chain.

The next section (variables) states that we have 3 uncertain uniform variables. These are the variables we wish to calibrate. We also have one configuration variable: there is only one configuration in these experiments. Note that the variables are all scaled before presenting them to DAKOTA. This is not strictly necessary, but we scaled the variables in advance for this example.

The next section (interface) defines an interface to the simulation code. In this case, the simulation driver is called `hydra3`. Note that we are not invoking `hydra3` in this particular example because we are reading in the simulation data, but it is a necessary part of the DAKOTA specification and certainly can be used to run the actual simulations in other examples.

The final section (responses) defines the number of calibration terms. We have 1001 terms for both velocity components of the response, so a total of 2002 calibration terms. There is only one set of experimental data,

```
num_experiments = 1
```

and the data is found in the file called `cas12.dat`:

```
calibration_data_file = 'cas12.dat'
```

Note that Appendix G contains the current specification: we expect that the DAKOTA specification will evolve as we add functionality to the way DAKOTA handles experimental data.

Figure 8 compares three sets of calibration results: one from the Matlab implementation of GPMSA, which utilizes a Metropolis-within-Gibbs MCMC algorithm, and the other two from QUESO-GPMSA invoking DRAM and the multilevel algorithms. The marginal posterior distributions for the elements of  $\theta$  presented in this plot indicate that the three distinct MCMC methods are producing consistent results. This is confirmed in a qualitative sense by the summary statistics presented in Tables 5 and 6.

Comparisons between summary statistics such as those presented in Tables 5 and 6 will be made more formal in a quantitative sense as this demonstration is further developed into a verification test for QUESO-GPMSA. For example, although all the correlation coefficients between parameters in Table 6 are close to zero in a practical sense, the posterior correlations between  $C_s$  and the other two parameters

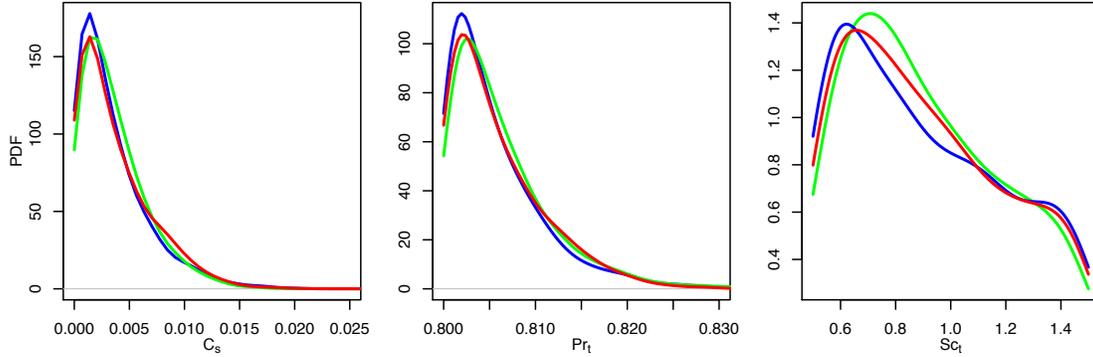


Figure 8. Marginal posterior densities for the three SSGS inputs of Table 1 from Matlab-GPMSA (blue), and QUESO-GPMSA DRAM (green) and multilevel (red).

Table 5. Posterior means and standard deviations for three SSGS inputs of Table 1.

Posterior Mean				Posterior Standard Deviation			
Parameter	Matlab-GPMSA	QUESO-GPMSA		Parameter	Matlab-GPMSA	QUESO-GPMSA	
		DRAM	Multilevel			DRAM	Multilevel
Cs	0.00337	0.00360	0.00360	Cs	0.00315	0.00292	0.00327
Prandtl	0.80557	0.80629	0.80581	Prandtl	0.00533	0.00568	0.00523
Schmidt	0.90054	0.90574	0.90460	Schmidt	0.29541	0.27117	0.28432

Table 6. Posterior correlations for three SSGS inputs of Table 1.

	Matlab-GPMSA			QUESO-GPMSA					
				DRAM			Multilevel		
	Cs	Prandtl	Schmidt	Cs	Prandtl	Schmidt	Cs	Prandtl	Schmidt
Cs	1	-0.032	-0.056	1	-0.077	-0.012	1	-0.147	-0.172
Prandtl		1	-0.017		1	0.052		1	-0.077
Schmidt			1			1			1

appear to be larger (in an absolute sense) in the multilevel results than for Metropolis-within-Gibbs or DRAM. These sorts of observations are easily amenable to quantitative examination, and will be useful as verification tests to establish confidence in the implementations of MCMC algorithms.

Velocity predictions resulting from the Matlab-GPMSA analyses are presented in Figure 9 for the  $u$  velocity and Figure 10 for the  $v$  velocity. In these figures, the Hydra-TH runs conducted to generate the surrogate model of Section 2 are shown in yellow and the experimental data with three standard deviation errors are shown in blue. In the left panel, a 95% prediction interval for prediction of calibrated Hydra-TH velocity is shown in green. Calibrated code predictions are obtained by running posterior samples of the calibration inputs  $\theta$  through the code surrogate model (or the code directly, if computational resources allow). In the right panel, a 95% prediction interval for prediction of discrepancy  $\delta$  is shown in cyan. This result indicates difficulty with predicting  $u$  and  $v$  velocity data for coordinate locations less than about 0.25. In the center panel, a 95% prediction interval in shown in black for discrepancy-adjusted predictions, i.e. calibrated code predictions (left panel) adjusted for inferred discrepancy (right panel). In the upcoming fiscal year,

capability for generating the predictions in Figures 9 and 10 will be incorporated into QUESO-GPMSA.

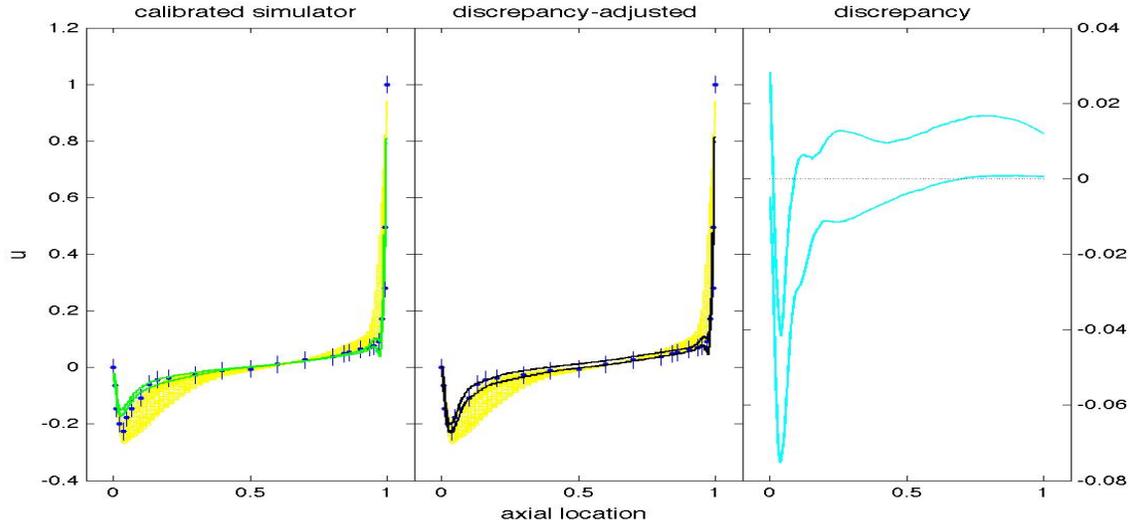


Figure 9. Calibrated Hydra-TH  $u$  velocity predictions (left), discrepancy predictions (right), and discrepancy-adjusted predictions (center).

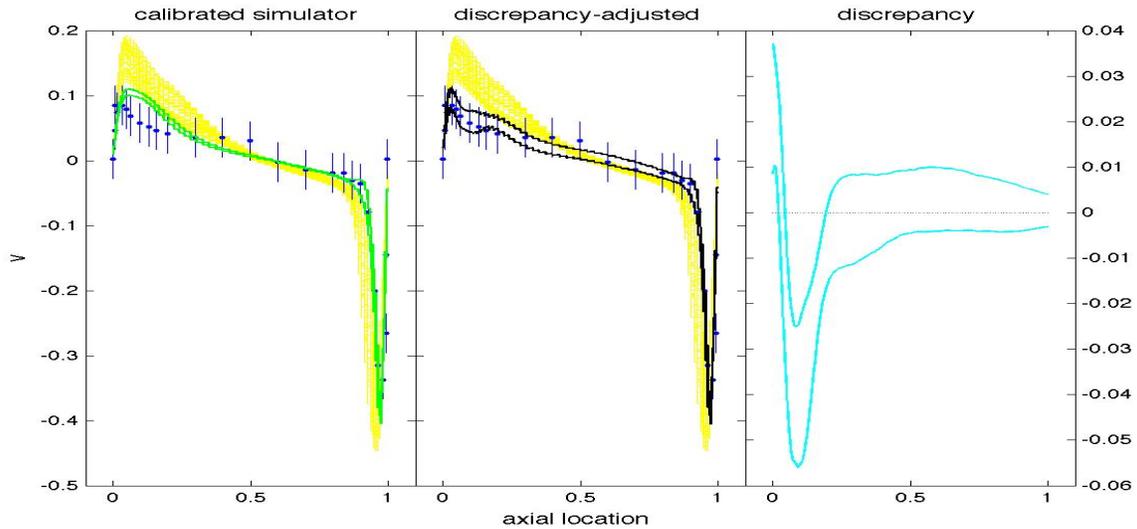


Figure 10. Calibrated Hydra-TH  $v$  velocity predictions (left), discrepancy predictions (right), and discrepancy-adjusted predictions (center).

Figure 11 provides a check on the surrogate-based calibration results. In both panels, the Hydra-TH runs conducted to generate the surrogate model of Section 2 are shown in yellow and the experimental data with three standard deviation errors are shown in blue. Ten samples from the posterior (calibrated) distribution of the SSGS inputs  $\theta$  were drawn from the QUESO-GPMSA DRAM results and run through Hydra-TH directly. The resulting  $u$  and  $v$  velocities are shown in magenta. Comparing against the left panels of Figures 9 and 10, these calibrated predictions are consistent with the surrogate-based calibrated predictions. The nominal

calculation (i.e.,  $u$  and  $v$  velocities computed at the nominal values of the SSGS inputs of Table 1) is shown in green.

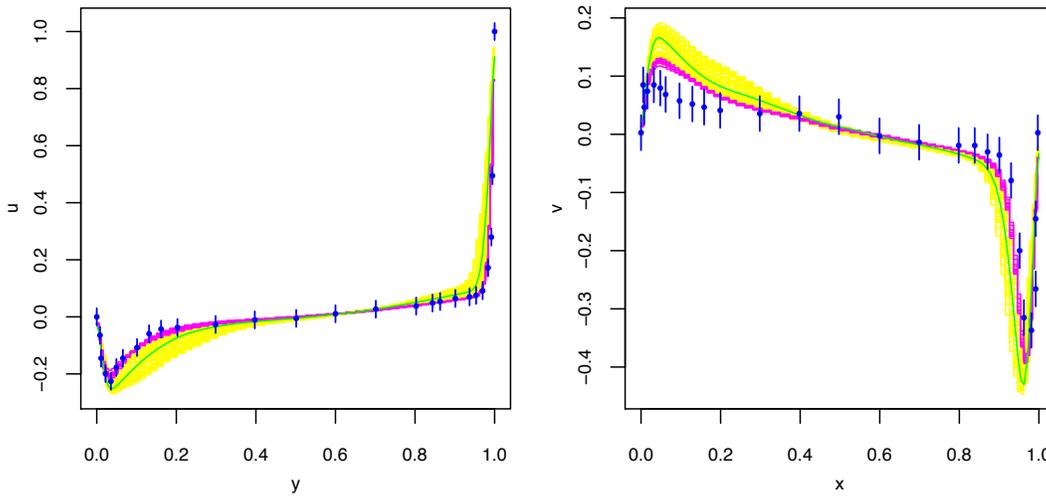


Figure 11. Comparison of calibrated predictions computed directly from Hydra-TH (magenta) with experimental data (blue) and nominal Hydra-TH calculation (green).

The calibration process results in velocity predictions that are generally more consistent with the experimental data than the nominal calculation. Of course, the nominal SSGS inputs were not derived for the LDC application specifically. The sensitivity of calibration results to specific applications depends on the robustness of the deployed physics models and on the quality of the experimental data selected for calibration purposes. Ideally, discrepancies observed between calibration results for different applications should naturally diminish as the predictive maturity of physics models and the quality, quantity and relevance of available experimental data improves.

The log model evidence value for SSGS is 191645. Although meaningless as a stand-alone statistic, in principle this figure could be compared with evidence values from WALE and other turbulence models for this LDC setting (or, more generally, to settings involving alternative reference experimental data) to quantitatively determine which model performs best.

## 5. Discussion

This milestone was concerned primarily with two demonstrations of verification and validation methodology utilizing Hydra-TH calculations of a lid-driven cavity flow problem. In the first, the verification toolkit Percept was utilized to generate a successively refined set of meshes to study the convergence behavior of Hydra-TH. Quantitative methods based on the RMR (Robust Method of Regression) were employed by Percept to analyze the convergence rates of field quantities. Because pointwise values of the field quantities often do not converge monotonically, semi-

global norms of those fields were analyzed in the RMR procedure. In the second, the Bayesian calibration software GPMSA was utilized to illustrate probabilistic calibration of code input parameters to experimental data. Implementations of GPMSA in Matlab and QUESO were compared.

With regard to Percept, we learned that it was valuable for the mesh refinement package to report on the histograms of elements sizes, and also to provide the ability to report on edge length normal to the boundary for fluids simulation codes. The new capability in Percept to respect existing mesh spacing (gradients in element size near boundaries) provides a push button solution to creating a uniformly (or quasi-uniformly) refined mesh sequence. This respect spacing capability can also be used in conjunction with Percept's ability to conform to curved CAD geometry, along with an extensive mesh smoothing and element shape improvement algorithm. These other capabilities were not necessary for this study because the computational domain for the lid driven cavity geometry is an axis-aligned cube.

The RMR procedure provides more confidence on possible regressions during convergence analysis. The RMR procedure eliminates the use of an empirical safety factor, like the GCI (Grid Convergence Index of [14]), and relies instead upon the diversity of estimates and the use of statistics (mean, median, std. deviation, etc.) of those estimates to provide safety.

The analyses of Section 4 showed that the Matlab and QUESO implementations of GPMSA are producing consistent results. It should be noted that both the three calibration inputs  $\theta$  and 22 statistical model parameters were calibrated by Matlab-GPMSA, while in QUESO-GPMSA the statistical model parameters were fixed. In the upcoming fiscal year, this demonstration will be extended so that QUESO-GPMSA is verified on the full 25-dimensional parameter space and quantitative metrics will be employed to more formally compare posterior summary statistics between the distinct MCMC implementations.

QUESO-GPMSA is in the nascent stage of development. The VUQ focus area has plans to extend QUESO-GPMSA capabilities in the upcoming fiscal year in the following directions:

1. Integrate all Matlab GPMSA use cases into QUESO-GPMSA.
2. Integrate Metropolis-within-Gibbs MCMC capability into QUESO.
3. Integrate Matlab GPMSA prediction capabilities into QUESO-GPMSA.
4. Integrate Matlab GPMSA sensitivity analysis capabilities into QUESO-GPMSA.
5. Integrate Matlab GPMSA leave-one-out cross-validation capability into QUESO-GPMSA.

Currently, QUESO-GPMSA only implements Bayesian calibration for functional output settings, in which the figures of merit are functions defined on a refined set of index values such as temporal or spatial coordinates. The purpose of item #1 is to

expand this use case in two directions, to cover scalar and multivariate output settings. The multivariate output setting covers the situation in which the user wishes to calibrate inputs to multiple scalar figures of merit simultaneously.

Item #2 will expand the suite of MCMC methods employed in QUESO to include the single-site scan method employed by the Matlab version of GPMSA. Item #3 will allow predictions of calibrated code output and discrepancy to be made based on the Gaussian process surrogate models employed in GPMSA, with full quantification of uncertainty. Item #4 will allow estimation of global sensitivity indices based on ANOVA-based variance decomposition methods tailored to the Gaussian process code surrogate models of GPMSA. Finally, item #5 provides a goodness-of-fit diagnostic based on the leave-one-out cross-validation method that facilitates user evaluation of the predictive capability of the Gaussian process-based code surrogate model.

In the upcoming fiscal year, rigorous verification tests of QUESO-GPMSA will be developed that cover all three main use cases: functional, scalar, and multivariate output settings. These tests will involve comparison of QUESO-GPMSA calibration results against both manufactured solutions and solutions obtained by the Matlab implementation of GPMSA in the absence of analytical solutions.

## Acknowledgements

The authors would like to thank J. Adam Stephens and Brian Adams of Sandia National Laboratories for providing the `driver.sh` and `submit_all.sh` scripts and instructions for automating Hydra-TH runs under the DAKOTA fork simulation interface. Greg Weirs of Sandia National Laboratories provided a tutorial on generating Python scripts from ParaView to support automation of figure of merit extraction from Hydra-TH output. Mark Christon and Jozsef Bakosi of Los Alamos National Laboratory provided the experimental data and Hydra-TH control file and meshes necessary to run the LDC validation problem, as well as considerable advice on compiling and running Hydra-TH and interactively analyzing Hydra-TH output with ParaView. Mark Christon generously exposed SSGS turbulence model parameters in Hydra-TH to allow this Bayesian calibration study to move forward. This research was supported by the Consortium for Advanced Simulation of Light Water Reactors (<http://www.casl.gov>), an Energy Innovation Hub (<http://www.energy.gov/hubs>) for Modeling and Simulation of Nuclear Reactors under U.S. Department of Energy Contract No. DE-AC05-00OR22725. Los Alamos National Laboratory (LANL) and Sandia National Laboratories (SNL) are core CASL partners.

## References

- [1] Adams, B.M., Bohnhoff, W.J., Dalbey, K.R., Eddy, J.P., Eldred, M.S., Gay, D.M., Haskell, K., Hough, P.D., Lefantzi, S., and Swiler, L.P. (2010). DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 5.1 User's Manual. Technical Report SAND2010-2183, Sandia National Laboratories.
- [2] Estacio-Hiroms, K.C. and Prudencio, E.E. (2012). Quantification of Uncertainty for Estimation, Simulation, and Optimization (QUESO): User's Manual 0.46.0. The University of Texas at Austin Center for Predictive Engineering and Computational Sciences, Austin, TX.
- [3] Cops, K.D. (2012). Percept: Tools for Verification, Presentation to DOE/CASL roundtable, June 11, 2012. Technical Report SAND2012-4730 C, Sandia National Laboratories.
- [4] Bakosi, J., Christon, M.A., Pritchett-Sheats, L., Luo, H., Xia, T., and Nourgaliev, R. (2013). Hydra-TH Verification, Validation and Thermal-Hydraulics Benchmark Problems. Technical Report LA-UR-13-22017, Los Alamos National Laboratory.
- [5] Higdon, D., Gattiker, J., Williams, B., and Rightley, M. (2008). Computer model calibration using high-dimensional output. *Journal of the American Statistical Association*, 103 (482), 570-583.
- [6] Prasad, A.K. and Koseff, J.R. (1989). Reynolds number and end-wall effects on a lid-driven cavity flow. *Physics of Fluids A*, 1(2), 208-218.
- [7] Edwards, H. C., Williams, A.B., Sjaardema, G.D., Baur, D.G., and Cochran, W.K. (2010). SIERRA Toolkit Computational Mesh Conceptual Model. Technical Report SAND2010-1192, Sandia National Laboratories.
- [8] Heroux, M., Bartlett, R., *et al.* (2003). An Overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories.
- [9] Rider, W.J., Wildey, T.M., and Weirs, V. G. (2013), Does Solution Adaptivity Help or Hinder Solution Verification? Technical Report SAND2013-7920 (to be released), Sandia National Laboratories.
- [10] Tierney, L. and Mira, A. (1999). Some adaptive Monte Carlo methods for Bayesian inference. *Statistics in Medicine*, 18, 2507-2515.
- [11] Haario, H., Laine, M., Mira, A., and Saksman, E. (2006). DRAM: Efficient adaptive MCMC. *Statistics and Computing*, 16(4), 339-354.

- [12] Prudencio, E.E. and Cheung, S.H. (2012). Parallel adaptive multilevel sampling algorithms for the Bayesian analysis of mathematical models. *International Journal for Uncertainty Quantification*, 2(3), 215–237.
- [13] Wasserman, L. (2000). Bayesian model selection and model averaging. *Journal of Mathematical Psychology*, 44, 92-107.
- [14] Roache, P. (2009). *Fundamentals of Verification and Validation*. Hermosa Publishers, Albuquerque.

## Appendix A. DAKOTA Input File for Hydra-TH Runs

```
strategy,
  single_method
  tabular_graphics_data

method,
  nond_sampling
  samples = 50 seed = 241
  sample_type lhs

model,
  single

variables,
  uniform_uncertain = 3
  lower_bounds 0      0.8      0.5
  upper_bounds 0.36  1        1.5
  descriptors 'c_s' 'prandtl' 'schmidt'

interface,
  fork
  analysis_driver = 'driver.sh --submit'
  allow_existing_results
  parameters_file = 'params.in'
  results_file = 'results.out'
  file_save
  work_directory
  directory_tag
  directory_save
  named 'workdir'

responses,
  num_response_functions = 5
  descriptors = 'Step' 'Time-Step' 'Time' 'RMS Div' 'KE'
  no_gradients
  no_hessians
```

## Appendix B. driver.sh Script

```
#!/bin/tcsh

# The path/name of the hydra input template.
set HYDRA_TEMPLATE = ldc_Rele4_ssgs.cntl
# The path/name of the parameterized hydra input file.
set HYDRA_INPUT = ldc_Rele4_ssgs.cntl
# The path/name of the job file to be submitted from each working directory.
set JOB = ../run_hydra.sh
# The path/name of the file copied (and renamed) by this script in order to
# supply Dakota with responses during the submission phase.
set DUMMY_RESPONSE = dummy_responses.out
# The path/name of the python script to extract required FOMs.
set PV_PY = ../pv-ldc-out.py

# If the script is being run in collect mode.
# This portion of the script is also a placeholder for any post-processing
# that may be desired.
if ( $1 == "--collect" ) then
    setenv MODULEPATH ${MODULEPATH}:/usr/projects/views/modulefiles
    module load paraview/4.0.1-mu
    /usr/projects/views/ParaView/4.0.1/mu/bin/pvbatch $PV_PY
    awk -F '\",\"*' 'BEGIN { getline; } { print $8 " " $1 }' u.csv > u.txt
    awk -F '\",\"*' 'BEGIN { getline; } { print $7 " " $2 }' v.csv > v.txt
    cat u.txt v.txt > results.out
endif

# If the script is being run in submit mode.
# Do some basic checking to ensure the validity of the input.
if ( $1 == "--submit" ) then
    if ( $# != 3 ) then
        echo "Syntax is $0 <dakota params file> <dakota response file>"
        exit -1
    endif
    if ( ! -f $2 ) then
        echo "Unable to find specified Dakota parameters file: $2"
        exit -1
    endif
    if ( ! -e ../$HYDRA_TEMPLATE ) then
        echo "Unable to find hydra template: $HYDRA_TEMPLATE"
        exit -1
    endif

    # Create the hydra input file
    dprepro $2 ../$HYDRA_TEMPLATE $HYDRA_INPUT
    # Append this job to the list of those to be submitted
    echo "${PWD}:$JOB" >> ../job.list

    # Create the dummy responses for dakota.
    cp ../$DUMMY_RESPONSE $3
endif
```

## Appendix C. Generic Hydra-TH Control File

```

title
3-D Lid-Driven Cavity Re=10000

cc_navierstokes

  nsteps 100000
  deltat 0.01
  term 500.0

  time_integration
    type fixed_cfl
    CFLinit 1.0
    CFLmax 10.0
    dtmax 0.2
    dtscale 1.025
    thetaa 1.0
    thetak 1.0
    thetaf 1.0
  end

  # Output options
  pltype exodusii
  filetype serial
  plti 1000
  ttyi 200
  dump 0

  # Turbulence model
  turbulence smagorinsky
  c_s {c_s}
  prandtl {prandtl}
  schmidt {schmidt}
end

# Material model setup
# & assignment to sets
material
  id 1
  rho 1.0
  mu 1.0e-4
end

materialset
  id 10
  material 1
  block 1
end

plotvar
  elem vel
  elem div
  node vel
  node pressure
  node vorticity
  node helicity
end

statistics
  starttime 100.0
  endtime 500.0
  plotwinsize 20.0
end

plotstatvar
  elem <velocity>
  elem <pressure>
  elem <vorticity>
  elem tke
  elem reynoldsstress

  node <velocity>
  node <pressure>
  node <vorticity>
  node tke
  node reynoldsstress
end

# Simple IC's
initial
  velx 0.0
  vely 0.0
  velz 0.0
end

hydrostat
  nodeset 1 -1 0.0
end

distance
  sideset 1 -1 0.0 #lid
  sideset 2 -1 0.0 #walls
end

velocity
  velx sideset 1 -1 1.0 #lid
  vely sideset 1 -1 0.0
  velz sideset 1 -1 0.0
  velx sideset 2 -1 0.0 #walls
  vely sideset 2 -1 0.0
  velz sideset 2 -1 0.0
end

ppesolver
  type AMG
  amgpc hypre
  itmax 250
  itchk 1
  solver cg
  smoother ICC
  coarse_size 1000
  diagnostics off
  convergence off
  eps 1.0e-5
end

momentumsolver
  type ILUFGMRES
  itmax 50
  itchk 2
  restart 15
  diagnostics off
  convergence off
  eps 1.0e-5
end

transportsolver
  type ILUFGMRES
  itmax 50
  itchk 2
  restart 15
  diagnostics off
  convergence off
  eps 1.0e-5
end

end

exit

```

## Appendix D. submit\_all.sh Script

```
#!/bin/bash

# Submission command:
SUBMIT="msub"

if [ $# != 1 ]
then
    echo "Syntax is $0 <list of jobs>."
    exit -1
fi
if [ ! -f $1 ]
then
    echo "$1 not found."
    exit -1
fi

CWD=$PWD
for line in `cat $1`
do
    WD=`echo $line | cut -d":" -f1`
    JOB=`echo $line | cut -d":" -f2`
    cd $WD
    $SUBMIT $JOB
done
```

## Appendix E. ParaView Python Script pv-1dc-out .py

```
try: paraview.simple
except: from paraview.simple import *
paraview.simple._DisableFirstRenderCameraReset()

plotstat = ExodusIIReader( FileName=['plotstat'] )

AnimationScene3 = GetAnimationScene()
plotstat.NodeMapArrayStatus = []
plotstat.FaceVariables = []
plotstat.ElementVariables = []
plotstat.XMLFileName = 'Invalid result'
plotstat.FaceSetResultArrayStatus = []
plotstat.PointVariables = []
plotstat.FaceSetArrayStatus = []
plotstat.FaceMapArrayStatus = []
plotstat.FileRange = [0, 0]
plotstat.SideSetResultArrayStatus = []
plotstat.ElementSetArrayStatus = []
plotstat.EdgeVariables = []
plotstat.FilePrefix = 'plotstat'
plotstat.FilePattern = '%s'
plotstat.EdgeSetArrayStatus = []
plotstat.SideSetArrayStatus = []
plotstat.GlobalVariables = []
plotstat.NodeSetArrayStatus = []
plotstat.NodeSetResultArrayStatus = []
plotstat.ElementMapArrayStatus = []
plotstat.EdgeSetResultArrayStatus = []
plotstat.ModeShape = 0
plotstat.EdgeMapArrayStatus = []
plotstat.ElementSetResultArrayStatus = []

tsteps=plotstat.TimestepValues

AnimationScene3.EndTime = tsteps[-1]
AnimationScene3.PlayMode = 'Snap To TimeSteps'

RenderView3 = GetRenderView()

plotstat.FaceBlocks = []
plotstat.EdgeBlocks = []
plotstat.ElementVariables = ['<velocity>']
plotstat.ElementBlocks = ['Unnamed block ID: 1 Type: HEX']

RenderView3.CenterOfRotation = [0.5, 0.5, 0.25]

PlotOverLine2 = PlotOverLine( Source="High Resolution Line Source" )

AnimationScene3.AnimationTime = tsteps[-1]

RenderView3.CacheKey = tsteps[-1]
RenderView3.CameraPosition = [0.5, 0.5, 3.147777478867205]
RenderView3.CameraClippingRange = [2.3712997040785329, 3.5674941410502132]
RenderView3.ViewTime = tsteps[-1]
RenderView3.UseCache = 0
RenderView3.CameraFocalPoint = [0.5, 0.5, 0.25]
RenderView3.CameraParallelScale = 0.75

PlotOverLine2.Source.Point2 = [1.0, 1.0, 0.5]

active_objects.source.SMProxy.InvokeEvent('UserEvent', 'ShowWidget')

XYChartView2 = CreateXYPlotView()

RenderView3.CameraClippingRange = [2.3712997040785329, 3.5674941410502132]

PlotOverLine2.Source.Resolution = 1000
PlotOverLine2.Source.Point1 = [0.0, 0.5, 0.25]
PlotOverLine2.Source.Point2 = [1.0, 0.5, 0.25]
```

```

active_objects.source.SMProxy.InvokeEvent('UserEvent', 'HideWidget')

DataRepresentation4 = Show()
DataRepresentation4.XArrayName = 'arc_length'
DataRepresentation4.SeriesVisibility = ['<velocity> (0)', '0', '<velocity> (1)', '0',
'<velocity> (2)', '0', 'ObjectId', '0', 'Points (0)', '0', 'Points (1)', '0', 'Points
(2)', '0', 'Points (Magnitude)', '0', 'arc_length', '0', 'vtkOriginalIndices', '0',
'vtkValidPointMask', '0']
DataRepresentation4.UseIndexForXAxis = 0

AnimationScene3.ViewModules = [ RenderView3, XYChartView2 ]

XYChartView2.ViewTime = tsteps[-1]
XYChartView2.LeftAxisRange = [0.0, 0.40000000000000002]
XYChartView2.RightAxisRange = [0.0, 6.6600000000000001]
XYChartView2.TopAxisRange = [0.0, 6.6600000000000001]

DataRepresentation4.XArrayName = 'Points (0)'
DataRepresentation4.SeriesColor = ['<velocity> (Magnitude)', '0.301961', '0.686275',
'0.290196', '<velocity> (1)', '0.894118', '0.101961', '0.109804']
DataRepresentation4.SeriesVisibility = ['<velocity> (0)', '0', '<velocity> (1)', '1',
'<velocity> (2)', '0', 'ObjectId', '0', 'Points (0)', '0', 'Points (1)', '0', 'Points
(2)', '0', 'Points (Magnitude)', '0', 'arc_length', '0', 'vtkOriginalIndices', '0',
'vtkValidPointMask', '0', '<velocity> (Magnitude)', '0']

SetActiveSource(plotstat)
PlotOverLine3 = PlotOverLine( Source="High Resolution Line Source" )

PlotOverLine3.Source.Point2 = [1.0, 1.0, 0.5]

DataRepresentation5 = Show()
DataRepresentation5.XArrayName = 'arc_length'
DataRepresentation5.SeriesVisibility = ['<velocity> (0)', '0', '<velocity> (1)', '0',
'<velocity> (2)', '0', 'ObjectId', '0', 'Points (0)', '0', 'Points (1)', '0', 'Points
(2)', '0', 'Points (Magnitude)', '0', 'arc_length', '0', 'vtkOriginalIndices', '0',
'vtkValidPointMask', '0']
DataRepresentation5.UseIndexForXAxis = 0

PlotOverLine3.Source.Resolution = 1000
PlotOverLine3.Source.Point1 = [0.5, 0.0, 0.25]
PlotOverLine3.Source.Point2 = [0.5, 1.0, 0.25]

DataRepresentation5.XArrayName = 'Points (1)'
DataRepresentation5.SeriesColor = ['<velocity> (Magnitude)', '0.301961', '0.686275',
'0.290196', '<velocity> (0)', '0', '0', '0']
DataRepresentation5.SeriesVisibility = ['<velocity> (0)', '1', '<velocity> (1)', '0',
'<velocity> (2)', '0', 'ObjectId', '0', 'Points (0)', '0', 'Points (1)', '0', 'Points
(2)', '0', 'Points (Magnitude)', '0', 'arc_length', '0', 'vtkOriginalIndices', '0',
'vtkValidPointMask', '0', '<velocity> (Magnitude)', '0']

Render()

writervy = CreateWriter("v.csv", PlotOverLine2)
writervy.FieldAssociation = "Points" # or "Cells"
writervy.Precision = 10
writervy.UpdatePipeline()

writervx = CreateWriter("u.csv", PlotOverLine3)
writervx.FieldAssociation = "Points" # or "Cells"
writervx.Precision = 10
writervx.UpdatePipeline()

```

## Appendix F. make\_output.sh Script

```
#!/bin/bash

ndirs=$(find . -type d | grep 'workdir' | wc -l)

file1=workdir.1/results.out

for (( i=2; i <= $ndirs; i++ ))
do
    dir=workdir.$i
    join $file1 $dir/results.out > .tmp
    mv .tmp results.out
    file1=results.out
done
```

## Appendix G. DAKOTA Input File for QUESO-GPMSA Analysis

```
# DAKOTA INPUT FILE - dakota_bayes.in

strategy,
    single_method
    tabular_graphics_data

method,
    bayes_calibration gpmsa,
    emulator_samples = 50
    seed = 241
    import_points_file = 'williams9.txt' freeform
    samples = 20000

variables,
    active uncertain
    uniform_uncertain = 3
    lower_bounds = 0.0 0.0 0.0
    upper_bounds = 1.0 1.0 1.0
    continuous_state = 1
    lower_bounds = 0.
    upper_bounds = 1.

interface,
    system
    analysis_driver = 'hydra3'
    parameters_file = 'params.in'
    results_file = 'results.out'
    file_save file_tag

responses,
    calibration_terms = 2002
    calibration_data_file = 'casl2.dat'
    freeform
    num_experiments = 1
    num_config_variables = 1
    no_gradients
    no_hessians
```