Gaussian Process Models for Simulation Analysis (GPM/SA) Command, Function, and Data Structure Reference

LA-UR-08-08057

James R. Gattiker
Los Alamos National Laboratory
gatt@lanl.gov

1 Introduction

This document describes the commands (functions) of the GPM/SA code package, the data structures that these functions accept and generate, details on the technical underpinnings, and some comments on command flow and usage. For details about how and why GPM/SA is used, please refer to the associated document that goes through an example in detail [1], and publications describing the modeling technology [2, 3].

The general motivation for this project is the analysis of computer models, and the analysis of systems that have both computer simulations and observed data. The underlying assumption is that experimental results are expensive or difficult to collect, though it is possible to get simulation studies of the experimental problem in comparatively large, but still restricted, quantities. Therefore a statistical model of system responses is to be constructed, which is called a surrogate or *emulator*.

The emulator is constructed from an ensemble of simulation responses collected at various settings of input parameters. The statistical model may be extended to incorporate system observations to improve system predictions and constrain, or calibrate, unknown parameters. The statistical model accounts for the discrepancy between the system observations and the simulation output. Parameters of the statistical model are determined through sampling their posterior probability with MCMC. The statistical model can then be used in several ways. Statistical prediction of simulation and observation response is relatively fast and includes uncertainty. Examination and analysis of the model parameters can be used for calibration, screening, and analysis of model properties. The emulator may be further used for sensitivity analysis, and other system diagnostics.

Before using the GPMSA package, the analysis problem will have been defined, including: collecting system observations; determining uncertain simulation parameters; establishing a

design over the simulation parameters and running an ensemble of simulations; considering an appropriate model for dimension reduction of the observation and simulation response; considering an appropriate model for the discrepancy between simulation and observation; and considering prior model parameter settings related to these issues. These general issues are covered in the publication and tutorial references in the bibliography.

The first sections of this manual will describe the interface to functions a user might be concerned with, along with some comments about the data and parameters involved. Following sections describe the concepts and some details of the model.

2 Simulation Analysis Framework (SAF) Project: Obtaining, Installing, and Development

2.1 Project

The GPM/SA code is developed and maintained by the CCS-6 group at Los Alamos National Laboratory. Version 2.0 of GPM/SA is LANL computer code release LA-CC-06-079, which is open source under (C-06, 144).

The project is maintained in LANL's SourceForge system, which hosts documentation and latest release bundles of code, trackers for issues and future requirements, as well as a CVS repository of the source code. Development of GPM/SA is currently supported by a small number of LANL CCS-6 group members working closely in physical proximity (therefore style and coding standards are a matter for direct discussion by the development team).

The chief requirement for the GPM/SA code is to implement the technology described in [2]. The system can be best thought of as a set of tools extending the Matlab interactive data analysis environment. As such, the expectation is that the tools run on a desktop system in an interactive mode. Using the GPM/SA tools will require a fair amount of customized user setup, and one would expect customized post-processing to derive results relevant to a specific application problem.

2.2 Testing

The system has been vetted through use in a number of application problems, most of which contain proprietary data. Internally, we maintain a suite of examples that are suitable for examining and exercising various aspects of the code package. The results of the code are stochastic, so acceptance of result correctness is not a matter of duplicating results indentically, but rather generating results that are consistent with previous runs and internally consistent with respect to the problem. Thus, acceptance of results is a matter of judgement of an expert in this technology domain.

2.3 Platforms and Requirements

GPM/SA is a set of Matlab functions, requiring no toolboxes. Matlab is an interactive data analysis system that runs on several platforms including Windows, Mac OS X, Linux, and other versions of Unix. A C++ program is also available, that allows the computationally intensive MCMC parameter sampling to execute on a system not hosting Matlab. The C++ code depends on the GNU Scientific Library (GSL) and the GSL's C templates to interface to a BLAS library.

2.4 Obtaining

Users with access to LANL's SourceForge server can download the latest source code and documentation from the SF4 server. Other users should contact the development team.

2.5 Installing

The installation of the basic package is straightforward: download and expand the directory containing the functions, and add the directory to the Matlab path.

To use the C++ version of the MCMC sampling function, the code must be compiled on the platform. This facility is a stand-alone piece of code interfacing through text files for input and output, and does not need to be compiled with matlab. The installation is to download and expand the directory containing the code files, and then compile them as, e.g. on the LANL Flash environment:

```
g++ -lgsl -lgslcblas -I <directory> -o gpmsa main.cpp
```

As above, the compilation will require the GNU Scientific Library (GSL) and a BLAS.

3 Using the GPM/SA Code

The GPM/SA code is implemented in Matlab, and requires Release 14. This description assumes a basic familiarity with Matlab.

There are two basic functions of the code: calculation of parameters of the statistical model, and obtaining predictions from the model. The required input data structures will be described.

3.1 Overview of Flow and Commands for GPM Modeling

The first step in using the GPM/SA code is to format the required observation and computer code data. This data specification is described in detail in Section 4. This input is summarized in two Matlab structures: `simData` , containing and describing the simulation runs; and `obsData` , containing and describing the empirical observations. The `obsData` and `simData` structures are parsed by `setupModel` , and an expanded structure is returned that contains various substructures used in processing. This command would be executed as:

```
> params=setupModel(obsData,simData);
```

After this setup is complete, the model parameters are sampled (determined) with the `gpmmcmc` function, which accepts the `params` output from `setupModel` . It returns that same structure plus the mcmc draws of the statistical model parameters and the unknown simulation parameters. This structure may then be passed to `gpmmcmc` again to append new draws. This command would be executed as:

```
> pout=gpmmcmc(params,1000)
```

to perform 1000 mcmc draws of the model parameters. Another 500 draws could be appended as:

```
> pout=gpmmcmc(pout,500)
```

In that example, the structure `pout` is used to hold the entire model including the added draws, although there is no reason necessarily to keep a separate structure for `params` , as they contain the same information.

The utility `showPvals` is a quick way to graph and examine the MCMC parameter draws, which are stored in a substructure array of the `pout` struct called `pvals` . The command:

```
> showPvals(pout.pvals)
```

displays all the parameters. This substruct array can be indexed to examine specific values, as in:

```
> showPvals(pout.pvals(1000:end))
```

This provides a visual diagnostic of MCMC convergence and stability, as well as a look at parameter values. Default parameters for the MCMC are supplied, though it is of course possible that these will not suit a particular problem. MCMC parameters are discussed in the Section 4.

The `gPredict` function produces predictions from the model, given a set of MCMC draws and data locations. This will be described in more detail in Section 5, and in the detailed example problem. Other uses of the calibrated statistical model include diagnostics such as cross-validation such as with `gXval` and sensitivity analysis with `gSens` .

3.2 Using the C++ Code for MCMC

The GPMSA package can optionally use a C++ implementation of the MCMC procedure for generating posterior draws from the GP model's parameter distribution. First, the model is written from the Matlab environment to a text file using the Matlab command `writeModel` . Then the draws are executed by running the command-line `gpmsa` , with arguments of the model datafile, the output draws datafile, and the number of draws. The draws (parameter values) are read back into the Matlab environment for post-processing using the Matlab command `readPvals` .

4 Variables and Data Structures

This section describes the contents of common data structures used in the GPM/SA code package. These structs represent encapsulated information that will be passed as arguments or received as results. At a minimum, the dataset structures must be understood in order to be populated by the user.

The following are definitions of variables and sizes used in GPM/SA and this manual:

x	independent variable(s), associated with observed data
z	same type and size as x, but associated with simulations
p	length x and z vectors
t	independent vars associated with simulations but not observations
θ	variables corresponding to t, to be calibrated for observations
q	length of t and θ vectors
n	number of observed data
m	number of simulations
y	native space, dependent variable (may be designated obs_i or sim)
$yStd$	normalized dependent variable (may be designated obs_i or sim)
l	length of y (may be designated obs_i or sim)
K	response linear mapping matrix (may be designated obs or sim)
u	transformed dependent variable for observations (through K matrix)
w	same as u, but where u denotes observations, w denotes simulations
p_u	length of transformed simulation response (e.g., number linear basis functions)
D	discrepancy linear mapping matrix (may be designated obs or sim)
v	transformed discrepancy dependent variable
p_v	length of transformed discrepancy (e.g., number of linear basis functions)

4.1 obsData

Contains the experimental observations and related basis transformation matrices. `obsData` is a struct array, one element for each empirical/observed data example. It is a struct array rather than matrices because observations may not be on the same grid, and so may be of different sizes. Each element of the struct array contains these fields:

field	description	size
x	independent variable(s) in range [0,1]	vector, p
yStd	standardized (mean 0 var 1) response data	vector, l_{obs_i}
Kobs	response transform matrix	l_{obs_i} by p_u
Dobs	discrepancy transform matrix	l_{obs_i} by p_v
Sigy	covariance of observed data	square matrix l_{obs_i}

The data in `obsData` are scaled and standardized so that x are in the interval [0,1] and $yStd$ are distributed $N(0,1)$. An x variable is required; if there are no x-type variables in the problem it should be set to a constant value, e.g. 0.5, for the observation example(s). `obsData` and `simData` (below) should have the same scaling.

The Sigy matrix is the covariance of the observed data examples. If it is not supplied, it is assumed to be identity. This is typically a diagonal matrix of representing observation error.

Postprocessing of model predictions requires additional information to return to native space, so the following additional fields in the substruct `orig` are suggested in order to return to the original problem domain:

field	description	size
y	original (raw) response data	vector, l_{obs_i}
ymean	mean of y used to calculate yStd	vector, l_{obs_i}
ysd	sd of y used to calculate yStd	vector, l_{obs_i} or scalar
Dsim	discrepancy transform for predictions on the sim grid	l_{sim} by p_v

4.2 simData

`simData` contains the simulation dataset, and is a non-array struct. It is assumed that the simulation data conforms to a single consistent grid, and so the examples are expected to be collected into a simpler matrix form, rather than requiring the more complex struct array of the observed data. Required fields are:

field	description	size
x	independent variable(s) in range [0,1]	m by $p+q$
yStd	standardized (mean 0 var 1) response data	l_{sim} by m
Ksim	response transform matrix	l_{sim} by p_u

The data in `simData` is standardized so that x are in the interval [0,1] and $yStd$ are distributed $N(0,1)$. An x variable is required; if there are no x-type variables in the problem it should be set to a constant value, e.g. 0.5, for the simulations. Postprocessing of model predictions requires additional information to return to native space, so the following additional fields in the substruct `orig` are suggested in order to return to the original problem domain:

field	description	size
y	response data	$len_{y_{sim}}$ by m
ymean	vector mean of y used to calculate yStd	l_{sim}
ysd	sd of y used to calculate yStd	l_{sim} or scalar

4.2.1 K and D matrices

The standardized y data is transformed by the supplied K and D matrices by the `setupModel` function. These linear transforms of the data are intended to be used to reduce the data cardinality. In typical GPM/SA operation, the K matrix is composed of principal component analysis (PCA) or canonical correlations derived variation vectors, and the D matrix is a smooth gaussian kernel decomposition (or, e.g., *spline*). The discussion below assumes this choice of models, though any linear transformation is possible.

The K matrix is computed as the principal component weights on the standardized simulation data, where only the first p_u eigenvectors are retained. p_u is chosen to reduce data size as much as possible while retaining as much of the data variance as required. The resulting transform matrix is K_{sim}. K_{obs} is produced from K_{sim} by interpolation between the simulation and observation grids.

The transform matrix D_{obs} models the residuals of the observations, i.e. the *discrepancy*, as that data is modeled by the K_{sim} matrix. A Gaussian kernel model is constructed over the data range. The key features of this are kernel centers that cover the range, and kernel widths that are appropriate for the kernel density (e.g., approximately 50% overlap of the kernel functions). A corresponding D_{sim}, produced by the same kernel densities over the appropriate data grid, can be produced and may be useful in some predictions and model output, though it is not required for the model – in principle there is no role for a D_{sim}.

$$
\begin{aligned}
y_{obs} &= Ku + Dv \\
y_{sim} &= Kw
\end{aligned}
$$

The K and D matrices should be scaled so that the variance of the w and v are approximately 1.

4.3 params

Contains all information necessary to evaluate the model, and sample the posterior distributions of parameters. params will contain the following substructs: data, model, priors, mcmc, `pvals` , `obsData` , `simData` . `obsData` and `simData` are recorded exactly as passed to `setupModel` .

4.3.1 params.data

The `data` struct holds processed information related to the input data, specifically the fields: x, v, u, z, w, t. These are defined above.

4.3.2 params.model

The `model` struct holds information related to the current state of the model in the MCMC chain, as well as precomputed data products. Most interesting components of this are recorded in the `pvals` struct. There are two reasons to be interested in the `model` struct: to both validate the values of model parameter values and data size parameters, and to change these parameters. Interesting data size parameters include: n, m, p, q, pu, and pv, defined above. Other fields should have self-explanatory naming. One reason to change these may be to start the MCMC chain at a particular setting.

4.3.3 params.priors

The `priors` struct has substructures, one for each variable named in the `mcmc` struct `svars` , i.e., for each variable to be sampled, as discussed in the `mcmc` struct description.

Each struct contains a function name for the prior call, a params struct, and fields `bLower` and `bUpper` that are hard boundaries for the value of the corresponding variable. There are two exceptions: 1)the `theta` field contains the special field `constraints` , which is the cell array of constraint strings discussed in the setupModel function description; 2) the dual-domain field `rhoU` $=e^{-\frac{1}{4}}$`betaU` are both listed. The `bLower` and `bUpper` fields are defined under `betaU` , and the prior function and params are defined under `rhoU` .

If another prior function is to be supplied, the details of the corresponding parameter-named field must be understood. As an example, the required fields (`fname` , `params` , `bLower` , and `bUpper`) for `lamWs` (for a problem where p_u=3) are:

```
 fname: 'gLogGammaPrior'
params:    3        0.003
           3        0.003
           3        0.003
bLower: 60
bUpper: 100000
```

The prior functions accept a value and a parameters array, which will be the corresponding row of the `params` field; in the case of `gLogGammaPrior` these parameters are the shape and scale parameters. Thus each of the three values of `lamWs` will be sampled, to compute the prior, each will in this example called the function `gLogGammaPrior` with the parameter array [0 0.003]. Also, if the value exceeds the bounds the prior will be taken to be -Inf.

4.3.4 params.mcmc

The `mcmc` struct contains fields which define the variables to be sampled and recorded, as well as proposal width definitions.

The `svars` struct is a cell array naming the variables to be sampled. Corresponding to this is a cell array with the field name `wvars` , which name the fields within `mcmc` that are the widths of the proposal distribution. However, in the adaptive operation mode (as opposed to the stepsize operation mode), adaptive choices are made for parameters that vary in scale, specifically all other than `theta` , `betaV` , and `betaU` . There is a correspoding array of integers in the field called `svarSize` , which describes how many elements there are in each sampled variable, e.g. `betaU` has $p_u(p+q)$ elements.

The `pvars` struct contains the names of variables to be recorded in the `pvals` struct.

4.3.5 params.pvals

The `pvals` struct is an array holding recorded MCMC draws of parameters specified in the `priors` struct's `wvars` field. When a GPM/SA component requires draws, they are taken from the `pvals` struct, which may be indexed to not include all draws for either skipping burn-in MCMC transient draws and/or sample the draws.

By default the fields in `pvals` will consist of one of two possibilities. An eta-only model (including no observation data and thus no model discrepancy or valibration parameters) will contain `betaU` , `lamUz` , `lamWs` , `lamWOs` , and `lamOs` , as well as the fields `logLik` , `logPrior` , and `logPost` . A full model will also contain `theta` , `betaV` , and `lamVz` .

4.4 hierParams

Contains all information necessary to augment the basic model into a hierarchical model, evaluate the model, and sample the posterior distributions of hierarchical parameters. This corresponds closely to the `params` struct, as should be evident from inspection. The function `setupDefaultHierParams` provides an example of hierarchical parameters setup to copy and modify.

4.5 pred

Contains predictions generated from the model by the `gPredict` function. There are two formats for this struct, corresponding to whether the prediction was from `gPredict` invoked in 'uvpred' mode or 'wpred'mode .
Returned from gPredict invoked with 'wpred' mode, the pred struct has the following fields:

field	description
w	a 3-d matrix of realizations, the first dimension indexes the number of realizations, which is one per pval supplied; the second dimension indexes the number of linear bases p_u, the third dimension indexes the number of points in xpred.
Myhat	a matrix of the means of the models, of size (number of `pvals`) x (sizeof-xpred x p_u)
Syhat	a cell array of covariance matrices corresponding to rows of Myhat.

For predictions from gPredict invoked in 'uvpred' mode, the fields are:

field	description
u	a 3-d matrix of realizations from the eta component of the model, the first dimension indexes the number of realizations, which is one per pval supplied; the second dimension indexes the number of linear bases p_u, the third dimension indexes the number of points in xpred.
v	a 3-d matrix of realizations from the discrepancy component of the model, the first dimension indexes the number of realizations, which is one per pval supplied; the second dimension indexes the number of linear bases p_v, the third dimension indexes the number of points in xpred.
Myhat	a matrix of the means of the models, of size (number of `pvals`) x (sizeof-xpred x (p_v+p_u)
Syhat	a cell array of covariance matrices corresponding to rows of Myhat.

4.6 predProcess

Contains processed predictions. Field names should indicate context from inspection of the struct. There are 4 substructs possibly present, `pc` for basis response statistics, `scaled` for native space scaled response statistics, `mean0` for native space mean removed statistics, and native for native response statistics. These will contain a field called `eta` for modeled simulation response and `delta` for modeled discrepancy response. Under each of these fields are fields named for a range of percentiles and mean.

4.7 sens

Contains sensitivity analysis results. The struct contains the following fields:

field	description
totalMean	overall output mean (posterior mean)
totalVar	total output variance (posterior samples)
smePm	main effect sensitivity indices (posterior mean)
stePm	two-factor interaction effect sensitivity indices (posterior mean)
siePm	two-factor interaction effect sensitivity indices (posterior mean)
sjePm	joint effect sensitivity indices (posterior mean)
mef	main effect functions by basis component (posterior mean and standard deviation)
tmef	main effect functions (posterior mean and standard deviation)
jef	two-factor joint effect functions by basis component (posterior mean and standard deviation)
tjef	two-factor joint effect functions (posterior mean and standard deviation)
sa	structure with sensitivity analysis information by basis coefficient

e0	overall output mean (posterior samples)
vt	total output variance (posterior samples)
sme	main effect sensitivity indices (posterior samples)
ste	total effect sensitivity indices (posterior samples)
sie	two-factor interaction effect sensitivity indices (posterior samples)
sje	joint effect sensitivity indices (posterior samples)
mef	main effect functions (mean and variance by posterior sample)
jef	two-factor joint effect functions (mean and variance by posterior sample)

5 Command Reference

This section details the calling interface to the matlab functions supplied with the GPM/SA code package.

5.1 User Commands

This section details commands that are normally of interest to a user of the GPM/SA package.

5.1.1 gAnalyzePCA.m

Examine properties of the principal components analysis of a dataset.

Definition:
`a=gAnalyzePCA(y,y1)`

Input Arguments:

y	dataset used for finding the principal components. This would typically be the simulation response data
y1	(optional) a second dataset to also be analyzed. This would typically be the observation response data

Output Arguments:

a	the cumulative proportion of variance explained by the principal components

Description: Produces two plots.
The first plot has two panels. The first is cumulative variance explained for all principal components, the second is a zoom on the first 10 principal components. If the optional dataset $y1$ is supplied, these panels are overlayed by the proportion of absolute residual explained for both y and $y1$ datasets, using a given number of principal components; this is a diagnostic of whether the observation data will be modeled well by the principal components. The second plot is the first 5 principal components.

5.1.2 diagPlots.m

Some diagnostic plots that may be of general use. These may also be used as templates for customization.

Definition:
`result = diagPlots(params,pvec,plotType,optional-arguments)`

Input Arguments:

params	params struct, see the data structures section
pvec	indices of which drawn MCMC samples to use in computation
plotType	which plot option to perform, one of {1, 101, 2,3,4}, described below

Optional Arguments: Passed as tag/value pairs

'labels'	input variable names for plots
'labels2'	output variable names for plots
'figNum'	which matlab figure window to plot in (default is the plot number)
'evenWeight'	where relevant, a value of true indicates to not weight results by PC weight
'ngrid'	where relevant, a grid size to pass to gPlotMatrix, when estimating contours
'ksd'	where relevant, a kernel sd to pass to gPlotMatrix
'standardized'	where relevant, variables remain on the standardized scale

Output Arguments:

result	in some options, some information is returned to the user. Examine code for details.

Effect: This will give a brief description by plot type.
Plot type 1 is a boxplot of the $\rho_U = e^{-\frac{1}{4}\beta_U}$ from the `pvals` specified, for each principal component. The ρ values provide a diagnostic of model fit and variable importance. A ρ near 1 indicates little response activity (though the variable may still be important), where a value of ρ approaching zero indicates that the statistical model may not be adequately modeling the simulation response.
Plot type 101 integrates the ρ values over the principal components, resulting in a single ρ for each input variable x to the model. There is no perfect way to do this, the default is to take a weighted mean by proportion of variance explained by the each PC, but if the optional argument 'evenWeight' is true then it is an unweighted mean.
Plot type 2 plots the distributions of the calibrated parameters θ, by forwarding the values to gPlotMatrix. The plot shows a histogram of each distribution down the diagonal of a 2D grid. The joint effects are shown in the off diagonal entries. Un the upper triangle, the data are plotted as points, color coded from blue to red with bluer being earlier in the chain.

The lower triangle contains an estimated density plot, with contours of estimated 50 and 90 percentile.
Plot type 3 plots combined λ_{OS} and λ_{V_z} values, as these together indicate how much the simulation model is regularized, and gives an indication of how important the simulation data is.
Plot type 4 plots the calibrated discrepancy of the model. Don't expect this plot to always work (in particular, for observations on different grids), it requires cognizance of the native response domain to work properly.

5.1.3 gpmmcmc.m

Generate MCMC draws of the parameters of a GPM/SA model.

Definition:
`[pout Hpout]=gpmmcmc(params, numDraws, args)`

Input Arguments:

params	GPM/SA parameters structure (see *data structures* definitions). This may be an array argument for linked models evaluation.
numDraws	number of parameter set samples to draw

Optional Arguments: Passed as tag/value pairs

'noCounter'	True causes progress output to be suppressed
'step'	True involes step size mode (default is adaptive mode)
'initOnly'	Precompute data products, but execute no draws
'noInit'	execute draws, but don't initialize and recompute data products. Used when draws will be added to a previously sampled model.
'clist'	definition for shared calibration parameters across models. This is a matrix, where each row defines a parameter (θ) shared between models. The number of columns is the number of models, i.e., the length of the params array. A zero in a column indicates the θ is not in the corresponding model, a nonzero entry indicates the index of the θ in the corresponding model.
'hierParams'	data structure for constructing and drawing from a hierarchical model on a parameter to be calibrated. Refer to the section on data structure descriptions

Output Arguments:

pout	parameters structure augmented with draws in the `pvals` substructure, and updated internal data structures
Hpout	(if hierParams optional argument supplied) hierarchical model parameters structure augmented with draws

Effect: This function does the specified number of draws of the model parameters, including, e.g., length scales β, calibration parameters θ, and process variance parameters λ, as well as hierarchical model parameters, if defined. One draw with `gpmmcmc` performs one draw on each parameter.

The output structures are the same as the corresponding input structures, except: 1) the draws are added to the `pvals` structure, and 2) the internal state of the computational products are updated. A new params structure is augmented with several computational

products (in the `model` substructure), unless specified otherwise with the 'noinit' optional argument, which is useful when sequential calls for 1 or a small number of draws may be requested.

Adaptive step sizes are used for the λ variables. These steps are chosen according to their location, with the acceptance criteria modified for non-symmetric moves. The step size is $\frac{1}{3}$, or at a minimum 1. That is:

$$ss = \max(1, \frac{\lambda}{3}) \tag{1}$$

$$\lambda' \sim U(\lambda \pm ss) \tag{2}$$

To compute the acceptance modification, we need the step size from the new location

$$ss' = \max(1, \frac{\lambda'}{3})$$

The step correction is a, where the acceptance criterion for a draw is then modified to:

$$U(0,1) < \frac{p}{p'}a$$

where

$$a = \begin{cases} 0 & : \quad \lambda > \lambda' + ss' \\ \frac{ss}{ss'} & : \quad otherwise \end{cases}$$

The operation is intended to correct for asymmetry where a candidate λ is drawn so low in its interval that is is impossible for the next draw to again reach up to λ. If this case is met, the point is rejected, i.e., $a = 0$ guarantees rejection of the draw. Otherwise the value of a is the ratio of step size, i.e., the probability that this case did not occur.

5.1.4 gPredict.m

Generate predictions from the posterior Gaussian process model.

Definition:
`pred=gPredict(xpred,pvals,model,data,args)`

Input Arguments:

xpred	x locations at which to predict the model response
pvals	an array of MCMC draws; taken from the `pvals` substruct
model	the model substruct from the params struct
data	the data substruct from the params struct

Optional Arguments: Passed as tag/value pairs

'mode'	one of: **'wpred'** predict from the simulation model only **'uvpred'** generate predictions of both the simulation and discrepancy response **default:** 'uvpred' if observation data is present, 'wpred' for an eta-only model
'theta'	contains specific θ values to be used for prediction. If not present, calibrated prediction is done: the theta used is the theta from the `pvals` .
'addResidVar'	value of true directs to add the residual variance on the simulation predictions. Default is false.
'returnRealization'	value of true directs that a realization will be computed and returned for each prediction. Default is true.
'returnMuSigma'	value of true directs to return the mean vector and covariance matrix in the predict struct. default is false.

Output Arguments:

pred	a structure containing a number of prediction products appropriate to the prediction mode, see section on data structures.

Effect: Perform predictions from the posterior Gaussian process model, in a number of modes, as specified in the input argument description. The predictions from `gPredict` are the linear basis weights; to achieve system predictions the output of `gPredict` must be returned back to the scaled and unscaled native space (as desired).

In general, the process is to make a GP posterior prediction model for the locations in $[x\theta]$, and to produce a realization from that model for each parameter set in the supplied `pvals` . As described, the θ variables may either be supplied, or not supplied and therefore taken from the `pvals` structure for calibrated prediction. Also included in the output are the model mean and covariance for each instance of `pvals` .

The predictions can be one of two modes, 'wpred', or 'uvpred', with output fields appropriate to the particular selection.

'wpred' mode is used to get predictions from only the simulation model, but in the case where a fill model with simulation and observation data has been calibrated.

'uvpred' returns predictions from both the simulation and discrepancy model components, using the supplied θ or calibrated θ.

5.1.5 gPred.m

A deprecated interface to `gPredict` , to generate predictions from the posterior Gaussian process model. It should not be used in new code implementations.

Definition:
`pred=gPred(xpred,pvals,model,data,mode,theta)`

Input Arguments:

xpred	x locations at which to predict the model response
pvals	an array of MCMC draws; taken from the `pvals` substruct
model	the `model` substruct from the `params` struct
data	the `data` substruct from the `params` struct
mode	'uvpred', 'wpred', or 'etamod'
theta	optional, if specified these are theta values to predict, if not calibrated prediction is performed using theta values from supplied `pvals` .

Output Arguments:

pred	a structure containing a number of prediction products appropriate to the prediction mode, see section on data structures.

Effect: Consistent with `gPredict` .

5.1.6 gPredProcess.m

Produce several products of typical interest in model predictions.

Definition:
`predProcessed=gPredProcess(params,pred)`

Input Arguments:

params	parameters structure for the model
pred	predictions from gPredict

Output Arguments:

predProcess	a structure containing a number of processed products of raw predictions

Effect: The linear basis weights over prediction locations is not usually the final answer of interest. `gPredProcess` uses the raw predictions to produce 5,10,20,50,80,90,95 percentile and mean of the PC loadings and the native-scaled space, and if possible the native-mean-zero-unscaled space, and native space. The latter are possible if the params structure optional fields are set up in the suggested manner so that the mean and standard deviations are available. (also, they must be either points or vectors).
Refer to the data structures section for details on the output structure.

5.1.7 gSens.m

Compute sensitivity functions and indices

Definition:
`sens=gSens(pout, optional-arguments)`

Input Arguments:

params	params structure for a calibrated model

Optional Arguments: Passed as tag/value pairs

'pvec'	the indices of the `pvals` to be used in analysis (default = all)
'ngrid'	number of grid points in each dimension for calculation of main/joint effect functions (default 21)
'varlist'	matrix with pairs of variables in each row for which joint effects are desired; value of 'all' indicates to compute joint effects for all pairs of variables; (default - empty matrix)
'jelist'	cell array with row vectors indicating variables for which joint sensitivities are desired (default - empty)
'rg'	matrix with one row for each variable, giving min/max values for sensitivity calculations assuming unit hypercube scaling (default - unit hypercube)
'option'	sensitivity calculations based on • 'mean' - posterior mean GP parameters • 'median' - posterior median GP parameters • params - user provided GP parameters; must have betaU, lamUz and lamWs fields (row vectors) • (default - empty)

Output Arguments:

sens	sensitivity results structure, refer to data structures section for detail

Effect: By sampling the posterior models for the given parameter draws, do a classical sensitivity analysis on main and pairwise effects.

5.1.8 gXval.m

Some options for cross validation; and at least a template for coding customized cross-validation. Note that there are limited general options, since native space cross-validation is generally not defined.

Definition:
`h=gXval(params, pvec,optional-arguments)`

Input Arguments:

params	a `params` struct with drawn `pvals`
pvec	`pvals` to be used in the cross validation analysis

Optional Arguments: Passed as tag/value pairs

'mode'	what type of analysis and plot to perform, one of: • 'PCplot' - (default) plot each PC response • PCplotOrder' - plot of each PC response in canonical order in a boxplot • 'residErr' - prediction accuracy of a multivariate response (which will be a problem if the result is highly multivariate) • 'residSummary' - residual summary from each holdout validation run, integrated over all multivariate responses.
'numSamp'	number of the points to be drawn for the cross validation (default - size of dataset)
'figNum'	figure number to plot in
'standardized'	in native space modes, a value of true indicates to work on the standardized scale, otherwise it's the unscaled response (default false). Note that unscaled computation requires mean and standard deviation to be in their suggested optional location.
'labels'	in some cases, allows a cell array of supplied labels to be applied to a plot (sorry, you'll have to examine the code for details).

Output Arguments:

h	see code; in some cases this is handles to the plots, in other cases it's the error measure of the cross-validated points.

Effect: Generally, this performs a number of predictions, holding out each specified sample (default is all points in the dataset, but this may be limited by the option described above).

Then, it analyzes and plots the results in different ways according to the specified mode.

5.1.9 readPvals.m

Read in parameter draws to the Matlab environment, from a text file generated by the C++ MCMC sampling routine. See the section on the C++ code for details on file structure.

Definition:
`pvals=readPvals(filename)`

Input Arguments:

filename	the filename of the text file containing the parameter draws

Output Arguments:

pvals	structure containing the parameter draws, (refer to the data structures section)

Effect: Read in `pvals` structure from a text file generated by the C++ version of MCMC sampling.

5.1.10 setupDefaultHierParams.m

Example of building a hierarchical parameters structure.

Definition:
`hParams=setupDefaultHierParams()`

Input Arguments:
none

Output Arguments:

hParams	a hierarchical parameters structure (refer to data structures section)

Effect: An example of setting up hierarchical parameters. Sets up variable designations between models, initial parameters for the hierarchical distribution (normal distribution parameters mean and precision), priors for the meta-parameters (normal for the mean parameter, gamma for the lambda parameter), and MCMC sampling control parameters.

5.1.11 setupModel.m

Accepts a data descriptor and sets up a full params structure for the GPM/SA model.

Definition:
`params=setupModel(obsData, simData,optParms)`

Input Arguments:

`obsData`	observation data struct, efer to data structures section for detail. This parameter is optional, if it is an empty matrix an eta-only parameters structure is generated
`simData`	simulation data struct, refer to data structures section for detail.
optParms	a structure containing optional information (this parameter is optional) in substructs: • `scalarOutput` - must be present if the response is scalar. • `lamVzGroup` - indicates that the v responses are not governed by a single λ_{V_z}, but are instead composed of groups of parameters, each of which has a separate parameters. If supplied, the value of this is a vector of length p_v, composed of integers from 1 to the number of groups. A group is the v responses corresponding to positions with the same integer value in `lamVzGroup` . • `priors` - this may contain substructs `lamWOs.params` and `lamOs.params`. The priors for these variables will take these values, if supplied. See the data structures section for details on the prior parameters substructures. • `thetaConstraints` - this is a cell array of strings, each of which will be evaluated in an envirionment where the vector named `theta` exists. All of these strings must evaluate to true in order for a draw on the θ calibration parameters to be accepted. This is a way to allow a general constraint on the joint values of theta. Initialization of the θ parameters will be randomly drawn until these are all true.

Output Arguments:

params	a parameters structure, refer to the data structures section.

Effect: Accept the observation and simulation data structures, transform through the linear basis specification supplied in those structures, and set up the model parameters struc-

ture, including these operations:

- map the simulation data y_{sim} into w with the supplied K_{sim}.
- map the observation data y_{obs} into u and v with the supplied K_{obs} and D_{obs}.
- set up initial parameter values
- precompute some model products, primarily distances that will be used repeatedly
- compute Λ_{obs}, the u and v spatial correlation
- set the default prior distribution types and parameters, including modification of λ_{Os} and λ_{WOs}, due to the basis transformations.
- set up prior bounds on parameter ranges
- set supplied theta constraints, if supplied, and draw initial values satisfying constraints
- set MCMC parameters: variables to be sampled, variables to be recorded, and default step sizes.

The default prior parameter values are:

parameter	initial value	description
theta	0.5	theta parameters vector
betaV	0.1	discrepancy response basis scaling vector
lamVz	20	marginal discrepancy precision (possibly of groups)
betaU	0.1	simulation response basis scaling vectors
lamUz	1	marginal simulation precision
lamWs	1000	simulations data precision
lamWOs	mean of prior, min value 100	simulations noise precision
lamOs	mean of prior, min value 20	observed data noise precision

The default prior distributions (see the data structures section for details on the location of these specifications) are:

substruct variable name	distribution function `fname`	`params` and initial values	bounds substruct names and initial values
lamVz	`gLogGammaPrior`	.params(:,1)=1 .params(:,2)=10^{-5}	.bLower=0.3 .bUpper=∞
lamUz	`gLogGammaPrior`	.params(:,1)=5 . params(:,2)=5	.bLower=0.3 .bUpper=∞
lamWOs	`gLogGammaPrior`	.params(:,1)=5 .params(:,2)=5 $\times$ 10^{-3}	.bLower=60 .bUpper=10^5
lamWs	`gLogGammaPrior`	.params(:,1)=3 .params(:,2)=3 $\times$ 10^{-3}	.bLower=60 .bUpper=10^5
lamOs	`gLogGammaPrior`	.params(:,1)=1 .params(:,2)=10^{-3}	.bLower=0 .bUpper=∞
rhoU	`gLogBetaPrior`	.params(:,1)=1 .params(:,2)=0.2	calculated as beta
rhoV	`gLogBetaPrior`	.params(:,1)=1 .params(:,2)==0.2	calculated as beta
betaU	calculated as rho		.bLower=0 .bUpper=∞
betaV	calculated as rho		.bLower=0 .bUpper=∞
theta	gLogNormalPrior	.params(:,1)=0.5 .params(:,2)=10	.bLower=0 .bUpper=1 .constraints={}

Basis Transformation Prior Parameter Value Modification in `setupModel` Output The basis transformation affects what the values of some of the parameters should be, because of the relative rescaling. This section outlines which variables are changed, and how.

lamOs Prior Modification The Γ parameter 1 is modified from its default by adding to it half of the difference between the original observation response elements and the number of elements in the u and v basis space. That is:

$$aCorr = \frac{1}{2} \left(\sum_{i=1}^{n} |Y_{obs}^{i}| \right) - np_u p_v \Bigg)$$

using the cardinality (length) operator $|\cdot|$. The second parameter is corrected by adding to it the half sum square residual between the scaled data observations and their reconstructions from the transformed space, modified appropriately by $\Lambda_y = \Sigma_y^{-1}$ (which is I if not specified). That is:

$$bCorr = \frac{1}{2} \sum_{i=1}^{n} (DK_i \cdot vu_i - y)^T \Lambda_y (DK_i \cdot vu_i - y)$$

lamWOs Prior Modification The lamWOs priors are modified in a similar way to the lamOs priors, except using the simulation data rather than the observed data. Because the simulation data is of uniform size, the computation is easier:

$$aCorr = \frac{1}{2}(|Y_{sim}| - p_u)m$$

$$bCorr = \frac{1}{2}\sum_{j=1}^{m}(KW_j - Y_{sim}^j)^T(KW_j - Y_{sim}^j)$$

Once these are calculated, the initial values are set to the mean of the priors.

5.1.12 showPvals.m

Plot specified MCMC draws, for quick diagnostic.

Definition:
`showPvals(pvals)`

Input Arguments:

pvals	a `pvals` array struct; substruct of `params` . See section on data structures for details.

Effect: Plots all the fields in the supplied `pvals` .

5.1.13 stepsize.m

Perform computations to optimize step size draws in the MCMC.

Definition:
`[params hierParams]=stepsize(params,nBurn,nLev,optional-arguments)`

Input Arguments:

params	a params structure, refer to the data structures section.
nBurn	number of step size analysis MCMC samples to draw
nLev	number of levels to sample, to be used in fitting the step size optimization model

Optional Arguments: Passed as tag/value pairs

'clist'	same as the clist argument to gpmmcmc
'hierParams'	same as the hierParams argument to gpmmcmc

Output Arguments:

params	params struct augmented with step sizes computed
hierParams	hierParams struct augmented with step sizes computed

Effect: Draws a number of MCMC samples from the model at different step sizes, and uses the collected data on acceptance rates to estimate an optimal step size for each model parameter being drawn with MCMC.
Performing step size optimization takes some time to draw samples, but the resulting optimized step sizes should improve the quality of sampling.

5.1.14 writeModel.m

Writes the GPMSA model to a text file, to be used by the C++ version of the MCMC sampling code. See the section on the C++ code for details of the text file structure.

Definition:
`writeModel(params,filename)`

Input Arguments:

params	the `params` structure in the Matlab environment containing the model. (see section on data structures for details)
filename	the filename of the text file generated

Output Arguments:

none	

Effect: Write a text file containing a complete model description, for use by the C++ version of MCMC sampling.

5.2 Internal and Utility Functions

This section describes briefly the purpose of functions supplied with the GPM/SA package, but will in normal operation nor be directly invoked by a user. However, some of these are utility routines that may be helpful. They will be defined as narrative rather than with a detailed specification.

5.2.1 axisNorm.m

Called with an array of axis handles and a normalization mode; makes specified axis ranges the same across plots.

5.2.2 computeLogLik.m

Computes the log likelihood for a given model.

5.2.3 computeLogPrior.m

Computes the log prior for a given model.

5.2.4 counter.m

Provides a standardized way to report progress through loops.

5.2.5 diagInds.m

Indices of the diagonal of a matrix (1-D form).

5.2.6 gBoxPlot.m

A substitute for the statistics toolbox's boxplot command.

5.2.7 gCovMat.m

Generate a covariance matrix.

5.2.8 gGMICDF.m

Compute the inverse CDF of a Gaussian mixture.

5.2.9 genDist.m

Generate all manhattan distances between points in a given dataset.

5.2.10 genDist2.m

Generate all manhattan distances between points in two given datasets.

5.2.11 gLogBetaPrior.m

Compute a log beta pdf for given value and parameters.

5.2.12 gLogGammaPrior.m

Compute a log gamma pdf for given value and parameters.

5.2.13 gLogNormalPrior.m

Compute a log normal pdf for a given value and parameters.

5.2.14 gMvnrnd.m

Generate draw from a multivariate normal distribution (substitute for statistics toolbox's mvnrnd).

5.2.15 gNormpdf.m

Generate a draw from a normal pdf (substitute for statistics toolbox's normpdf).

5.2.16 gPackSubplot.m

Place an axis on a figure window, with somewhat more flexibility than the subplot command.

5.2.17 gPlotMatrix.m

A replacement for matlab's plotmatrix command, with additional options.

5.2.18 gQuantile.m

A substitute for the statistics toolbox's quantile command.

5.2.19 gSample.m

Generates a random sample of size m without replacement from the integers 1:n.

5.2.20 parseAssignVarargs.m

Provides a simplified way for dealing with optional argument lists, as tag-value pairs. It assigns the default-initialized variable with the name ¡tag¿ to the value ¡value¿.

6 C++ MCMC Sampling Operation Details

The core of the GPMSA code is determining the parameters of the GP emulator. Parameters are sampled from their posterior distribution using MCMC. This requires repeated computation of a posterior distribution, which requires the computation of parameter priors and the likelihood of the data given the parameter values. Determining prior probabilities is straightforward. Determining the likelihood is a computation involving solving (in the sense of inversion of) the model covariance matrix for a given set of parameters. This is a computationally intensive operation that may be computed in the Matlab environment using the function `gpmmcmc` , or it can be computed using the C++ program `gpmsa` .

6.1 Execution

The command is invoked as:

```
gpmsa <input-model-filename> <output-pvals-filename> <numDraws>
```

The argument `input-model-filename` is the name of the text file containing the model data. This will have been generated by the `writeModel` command from the Matlab environment.

The argument `output-pvals-filename` is the name of the text file containing the resulting parameter draws. This will be read in to the Matlab environment with the `readPvals` command.

The argument `numDraws` is the number of MCMC parameter draws to perform.

6.2 Text File Description

The interface to the C++ code is the two text files described above. The contents of these text files can determined in detail by looking at the simple read and write matlab functions `readPvals` and `writeModel` . In overview, each text file begins with the model size parameters n, m, p, q, pv and pu, which determine the sizes of all vectors and matrices.

The `pvals` text file further contains lines containing the complete model parameter set. Complete detail is easily read in the `readPvals` function.

The `model` text file follows the size parameters with the data, the current state of the MCMC chain (model parameters), the parameter prior parameters, and the MCMC control parameters. The Matlab `writeModel` function may be examined for complete detail.

7 GPM/SA Model Details

The simplest Gaussian process (GP) models problem is: Given a dataset (X, Y), where $X = x_1, ..., x_n$, each x possible vectors and $Y = y_1, ..., y_n$, determine an accurate model $y = f(x)$, so that for a new dataset X^* one can determine $\hat{Y} = f(X^*)$.

An additional level of complexity comes when there are two related independent variables In our applications, these are intended to be empirical observations of the independent variable and predictions from simulations, and so will be labeled X_{obs} and X_{sim}, with corresponding Y_{obs} and Y_{sim}. The weak assumption that there is more data from simulations than from empirical observations leads to a two part model structure: the first part models principally the shape of the response surface and is driven by the simulation data, the second models the discrepancy between the simulation model and the empirical observations, thus:

$$Y_{sim} = f(X_{sim})$$
$$Y_{obs} = f(X_{obs}) + g(X_{obs})$$

In modeling simulation and empirical data, we face the issue of uncalibrated parameters. These are known independent variables input to the simulations, but which have no observable (or even necessarily real) counterpart in the empirical observations. In observations, they are unknown constants. These θ parameters must be determined. They are inputs to functions of *sim*, but they are not present in the functions only of *obs*. The final model is thus:

$$Y_{sim} = f(X_{sim}, \theta)$$
$$Y_{obs} = f(X_{obs}, \theta_{obs}) + g(X_{obs})$$

Parameters to be determined are the parameterization of the functions f and g (as GP models), and the value(s) of θ_{obs}.

7.1 Covariance Specification

The basic specification for gaussian process models is a covariance function, parameterized but highly constrained. The covariance is $Cov(x, \beta, \lambda_z, \lambda_s)$. The covariance is $n \times n$, where n is the number of examples in X, with elements:

$$Cov_{ij} = \frac{1}{\lambda_z} e^{-d(x_i, x_j, \beta)^2} + I \cdot \frac{1}{\lambda_s}$$

d is a distance function, here taken to be the scaled Euclidean distance:

$$d(x_i, x_j, \beta) = \sqrt{\beta \cdot (x_i - x_j)^2}$$

This covariance specifies a dependence relationship on neighboring points, where the correlation falls off as a gaussian distribution. The precisions λ_z and λ_s characterize the

data variance (as precisions) as the amount of data variance captured by the model, and the amount of data variance captured by the residuals, respectively. In some cases the λ_s parameter may not be specified, in which case the final term is ignored.

We also use in notation what we call the cross-covariance $CCov(x1, x2, \beta, \lambda_z)$, which provides the covariance between two different datasets:

$$CCov_{ij} = \frac{1}{\lambda_z} e^{-d(x1_i, x2_j, \beta)^2}$$

7.2 Linear Basis Variable Reduction

In many problems the dependent variable will be multidimensional, and further in a space that is prohibitively large to compute the model described. The solution is to reduce the variables by a linear basis transformation, and compute the GP model on the basis coefficients. The linear basis is theoretically arbitrary, though our convention is to choose a principle component basis for the simulation response and a kernel decomposition basis for the discrepancy response.

The notation for this spatial mapping is:

$$Y_{sim} \approx K \cdot w$$
$$Y_{obs} \approx K \cdot u + D \cdot v$$

From the perspective of running the code, the matrices K and D are inputs specified by the user.

The nature of the solution in the transformed space is biased compared to the original space. The K and D matrices are needed in the modeling in order to correct this bias, appearing as additional correlation terms on the gaussian matrix.

Model uv and w are generated by the least-square fit of the data to the basis.

The Σ_y term modifies the solution for the fit. In the absence of a Σ_y term:

$$vu = (DK' \cdot DK + I \times 10^{-4})^{-1} DK' \cdot y$$

where DK is the concatenation of the Dobs and Kobs matrices. When the Σ_y term is included, it's inverse, the associated precision Λ_y is used as:

$$vu = (DK' \cdot \Lambda_y \cdot DK + I \times 10^{-4})^{-1} DK' \cdot \Lambda_y \cdot y$$

This somewhat extended formulation is used in order to incorporate the regularization "nugget" term for numeric stability.

This basis transformation bias extends also the modification of the initial values and prior distribution parameters of certain model variables described in Section 5.1.11, as well as a correction term Λ_{obs}, describing the modifications to the covariance that is necessary to model the changed correlation structure of observed data the original space compared to the results of the DK basis transformation. Λ_{sim} describes the corresponding correction for the simulation examples due to the K transformation.

7.3 The GPM/SA Optimization Problem

The above describes elements of gaussian process models. The model calculated by this code incorporates the extensions discussed, including the two-part solution of: a base model for the simulation examples, and a discrepancy model that maps the simulation model to the experimental data.

Constructing the statistical model is sampling the posterior distribution for the parameters. The posterior is the priors on the parameters (as discussed) times the likelihood:

$$L = \frac{1}{|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(vuw)^T \Sigma^{-1} (vuw)}$$

where Σ is a matrix of spatial covariances and dependencies:

$$\Sigma = \Sigma_S + \Sigma_D$$

$$\Sigma_S = \begin{pmatrix} \Sigma_v & 0 & 0 \\ 0 & \Sigma_u & \Sigma_{uw} \\ 0 & \Sigma_{uw}^T & \Sigma_w \end{pmatrix}$$

Each of these components of Σ_S are block diagonal, the blocks are the covariance of the components of the multidimensional v, u, and w.

$$\Sigma_v = \begin{pmatrix} \Sigma_{v*} & & & \\ & \Sigma_{v*} & & \\ & & \ddots & \\ & & & \Sigma_{v*} \end{pmatrix}$$

where

$$\Sigma_{v*} = Cov(X_{dat}, \beta_v, \lambda_{vz})$$

The number of blocks in the Σ_v matrix is p_v, the number of basis components in v.

$$\Sigma_u = \begin{pmatrix} \Sigma_{u_1} & & & \\ & \Sigma_{u_2} & & \\ & & \ddots & \\ & & & \Sigma_{u_{p_u}} \end{pmatrix}$$

Where

$$\Sigma_{u_i} = Cov(\begin{pmatrix} x & \theta \end{pmatrix}, \beta_{u_i}, \lambda_{uz_i}) + I \begin{pmatrix} \frac{1}{\lambda_{ws_i}} \end{pmatrix}$$

$$\Sigma_w = \begin{pmatrix} \Sigma_{w_1} & & & \\ & \Sigma_{w_2} & & \\ & & \ddots & \\ & & & \Sigma_{w_{p_u}} \end{pmatrix}$$

Where

$$\Sigma_{w_i} = Cov(\begin{pmatrix} z & t \end{pmatrix}, \beta_{u_i}, \lambda_{uz_i}) + I\left(\frac{1}{\lambda_{ws_i}}\right)$$

$$\Sigma_{uw} = \begin{pmatrix} \Sigma_{uw_1} & & & \\ & \Sigma_{uw_2} & & \\ & & \ddots & \\ & & & \Sigma_{uw_{p_u}} \end{pmatrix}$$

Where

$$\Sigma_{uw_i} = CCov(\begin{pmatrix} x & \theta \end{pmatrix}, \begin{pmatrix} z & t \end{pmatrix}, \beta_{u_i}, \lambda_{uz_i})$$

$$\Sigma_D = \begin{pmatrix} \Sigma_{obs} & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \quad \Sigma_{sim}$$

$$\Sigma_{sim} = I\left(\frac{1}{\Lambda_{sim_i} * \lambda_{WOs}}\right)$$

Σ_{obs} characterizes the dependence introduced by the transformation from the original Y space into the $\{u, v\}$ space, mentioned above as the precision matrix Λ_{obs}, and is scaled by λ_{Os} as:

$$\Sigma_{obs} = \Lambda_{Os}^{-1} \frac{1}{\lambda_{Os}}$$

Our approach to generating the posterior statistical model is to sample parameter sets from the posterior with Gibbs sampling. Compared to directly optimizing the likelihood, the MCMC sampling approach has an advantage in establishing uncertainty on parameters and then on predicted results.

7.4 Model Predictions

This model can be thought of an interpolating missing data from a set of X's which are the model and the prediction datapoints. This yields a converged covariariance:

$$Cov\begin{pmatrix} X \\ X^* \end{pmatrix} = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix} \Bigg($$

The predictive model is then:

$$\begin{aligned} \hat{y} &\sim N(\mu_{\hat{y}}, \Sigma_{\hat{y}}) \\ \mu_{\hat{y}} &= \mu_y + \Sigma_{21}\Sigma_{11}^{-1} \cdot (y - \mu_y) \\ \Sigma_{\hat{y}} &= \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12} \end{aligned}$$

References

[1] K. Myers, D.Higdon, J. Gattiker, E. Lawrence "Using the Gaussian Process Model for Simulation Analysis Code: a tutorial example", Los Alamos technical report xxx, 2008.

[2] Dave Higdon, Jim Gattiker, Brian Williams, Maria Rightley, "Computer Model Calibration using High Dimensional Output", Los Alamos Technical report LA-UR-07-1444, to appear in the Journal of the American Statistical Association, 2007.

[3] Brian Williams, David Higdon, James Gattiker, Leslie Moore, Michael McKay, Sallie Keller-McNulty, "Combining Experimental Data and Computer Simulations, With an Application to Flyer Plate Experiments", Journal of Bayesian Analysis 1, no. 4, pp. 765-792, 2006.