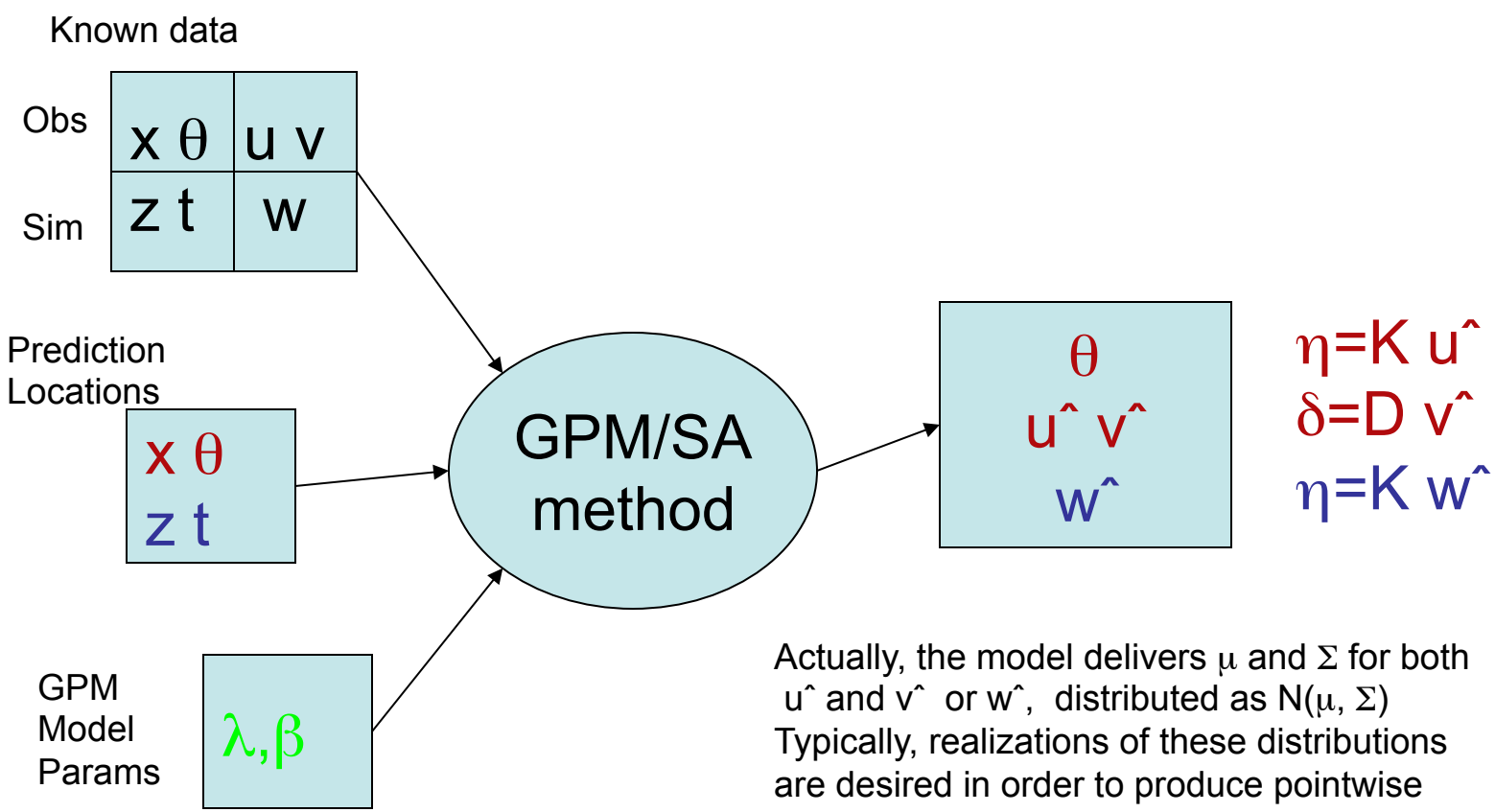
GPM/SA Code Operation

March, 2010
LA-UR-10-01395

Brian Williams, CCS-6

Jim Gattiker, CCS-6

Predicting Models with GPM/SA [in principle]



Actually, the model delivers μ and Σ for both $\hat{u}$ and $\hat{v}$ or $\hat{w}$, distributed as $N(\mu, \Sigma)$
Typically, realizations of these distributions are desired in order to produce pointwise prediction intervals.
The mean may also be used as an estimate.
All of these are produced by the **gPred** routine

Detail on Input Data

Observed data:

measured variables *x* correspond to a y_{obs}

there may be many experiments, y_{obs} may be different sizes

Simulation data:

input variables (z t) generate y_{sim}

there will be multiple simulations, y_{sim} is constant size

z and x represent the controllable system parameters. t represents values of the tuning parameters at which the code is run; θ represents the unknown best value of these parameters for matching code to experiment.

In the basis representation, the y responses will be modeled as

$Kw=y_{sim}$

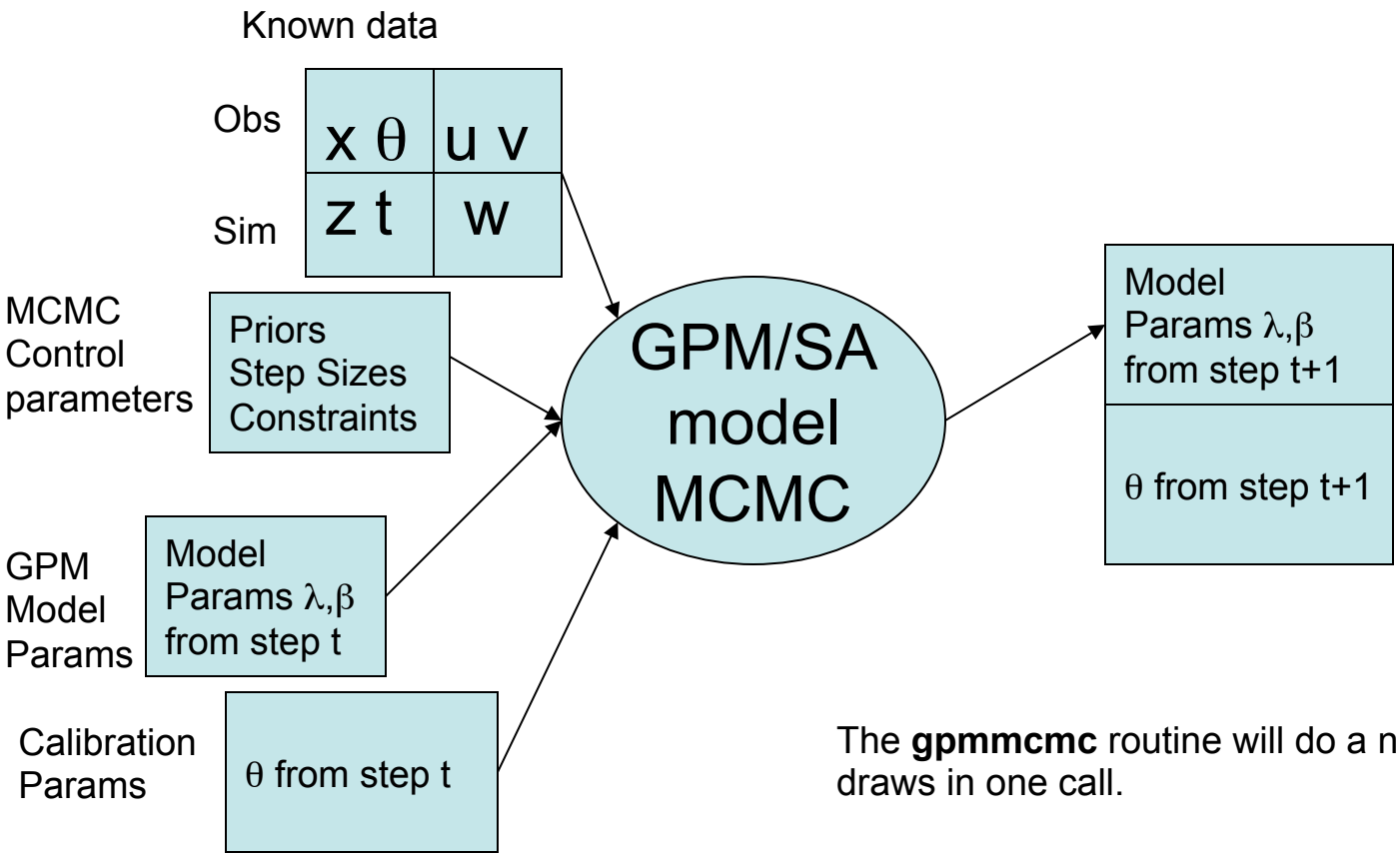
$Ku+Dv=y_{obs}$

The user supplies y and K and D matrices.

GPM Model Parameters TBD

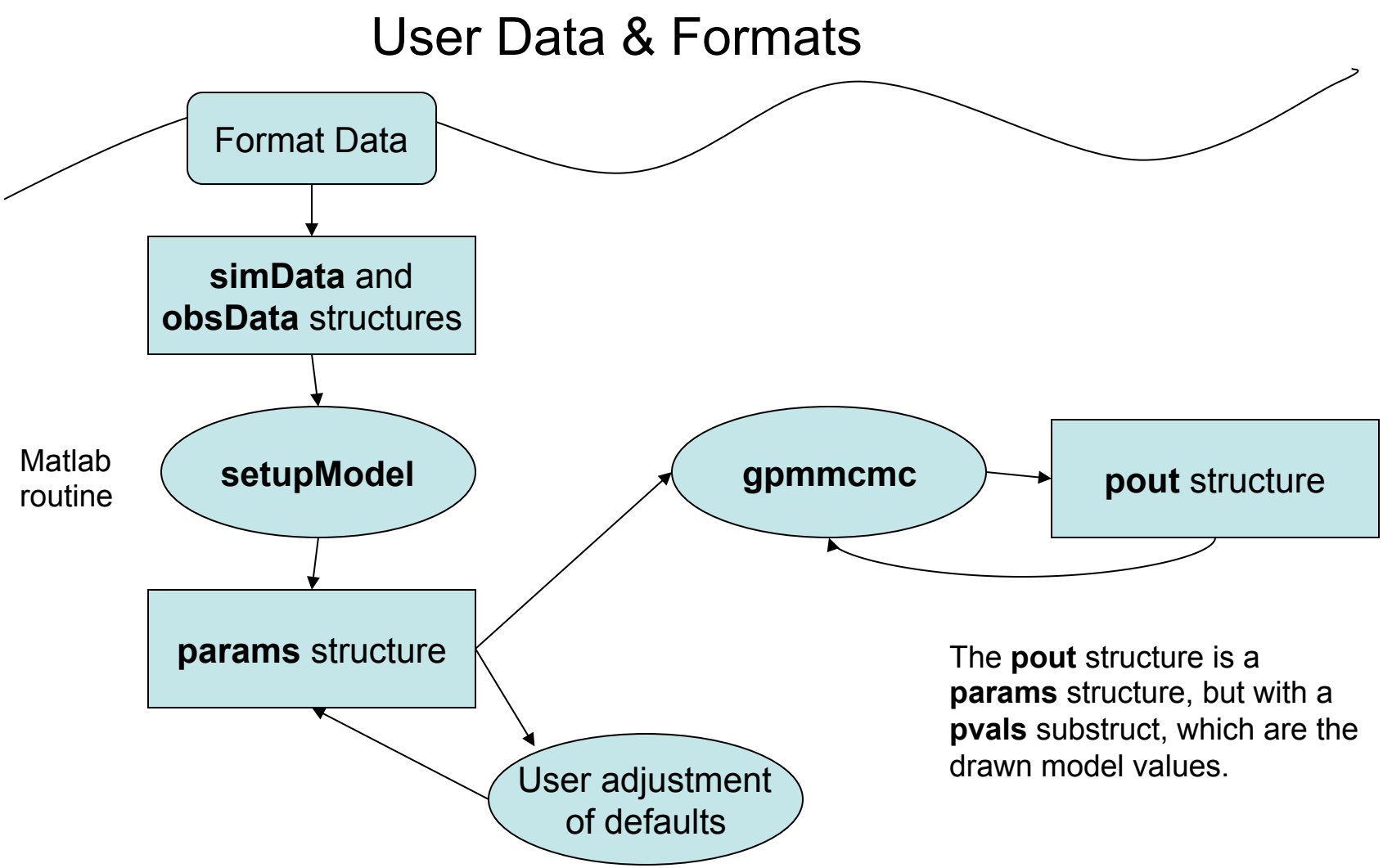
parameter	initial value	description
theta	0.5	theta parameters vector
betaV	0.1	discrepancy scaling vector
lamVz	20	marginal discrepancy precision
betaU	0.1	simulation response scaling vectors
lamUz	1	marginal simulation precision
lamWs	1000	simulations data precision
lamWOs	mean of prior, min value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

GPM/SA model building [in principle]



The **gpmmcmc** routine will do a number of draws in one call.

GPM/SA modeling process [in practice]



The **pout** structure is a **params** structure, but with a **pvals** substruct, which are the drawn model values.

Data naming conventions

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 observed
θ	variables corresponding to t, but not observed in data (i.e., to be calibrated)
q	length of t and θ vectors
n	number of observed data examples
m	number of simulation examples
y	original space, dependent variable (either obs or sim)
u	transformed dependent variable
p_u	length of transformed dependent variables (e.g., number of principle components)
v	transformed discrepancy dependent variable
p_v	length of transformed discrepancy (e.g., number of basis functions)
w	same as u, but where u denotes obs examples, w denotes sim examples

Structured data format: obsData and simData

Table 2: Input: `obsData` fields description. `obsData` is an array with one element per experiment. Experiments may have different data sizes.

x	independent variable(s)	vector, p
yStd	standardized (mean 0 var 1) response data	vector, $len_{y_{obs}}$
Kobs	response transform matrix	$len_{y_{obs}}$ by p_u
Dobs	discrepancy transform matrix	$len_{y_{obs}}$ by p_v
Sigy	Optional; covariance of observed data	square matrix $len_{y_{obs}}$

Table 3: Input: `simData` fields description. `simData` is not an array; simulation output is presumed to be of a constant size and so fields are matrices of data, where applicable.

x	independent variable(s)	m by $p+q$
yStd	standardized (mean 0 var 1) response data	$len_{y_{sim}}$ by m
Ksim	response transform matrix	$len_{y_{sim}}$ by p_u

Any other data may be stored for convenience, recommend using a "**.orig**" substruct. This would contain, e.g., (un)standardizing information [x,θ ranges; response mean and sd], plotting support, labels, design ID, raw data, etc.

Optional setupModel Parameter Structures

obsData and simData are passed to setupModel.m, which structures data for the GPM MCMC process (grouping, precomputing), and assigns defaults for priors, and MCMC parameters.

setupModel also accepts some additional data; named substructs of the optional argument exercise options:

Table 6: Allowed fields of optional `setupModel` parameter struct

field	type	description
lamVzGroup	vector of size p_v	integers from 1 to the number of groups; each v is associated with the corresponding group; each group has an independent lamVz parameter. (initial value is then a vector of appropriate length)
thetaConstraints	cell array of strings	each string will be evaluated when a theta is drawn, and if all are not true (i.e., constraints not satisfied) then the draw is rejected. See further description in Section 2.4.3.

MCMC params priors and range constraints

substruct variable name	distribution	substruct names and initial values	bounds substruct names and initial values
lamVz	Γ	.a=1 .b=10^{-5}	.bLower=0.3 .bUpper=∞
lamUz	Γ	.a=5 .b=5	.bLower=0.3 .bUpper=∞
lamWOs	Γ	.a=5 .b=5×10^{-3}	.bLower=60 .bUpper=10^5
lamWs	Γ	.a=3 .b=3×10^{-3}	.bLower=60 .bUpper=10^5
lamOs	Γ	.a=1 .b=10^{-3}	.bLower=0 .bUpper=∞
rhoU	β	$\alpha = 1$, fixed .b=0.2	calculated as beta
rhoV	β	$\alpha = 1$, fixed .b=0.2	calculated as beta
betaU	calculated as rho		.bLower=0 .bUpper=∞
betaV	calculated as rho		.bLower=0 .bUpper=∞
theta	Normal	.mean=0.5 .std=10	.bLower=0 .bUpper=1 .constraints={}

These parameters can be overridden by the modeler, though these defaults are often sufficient.

MCMC step sizes

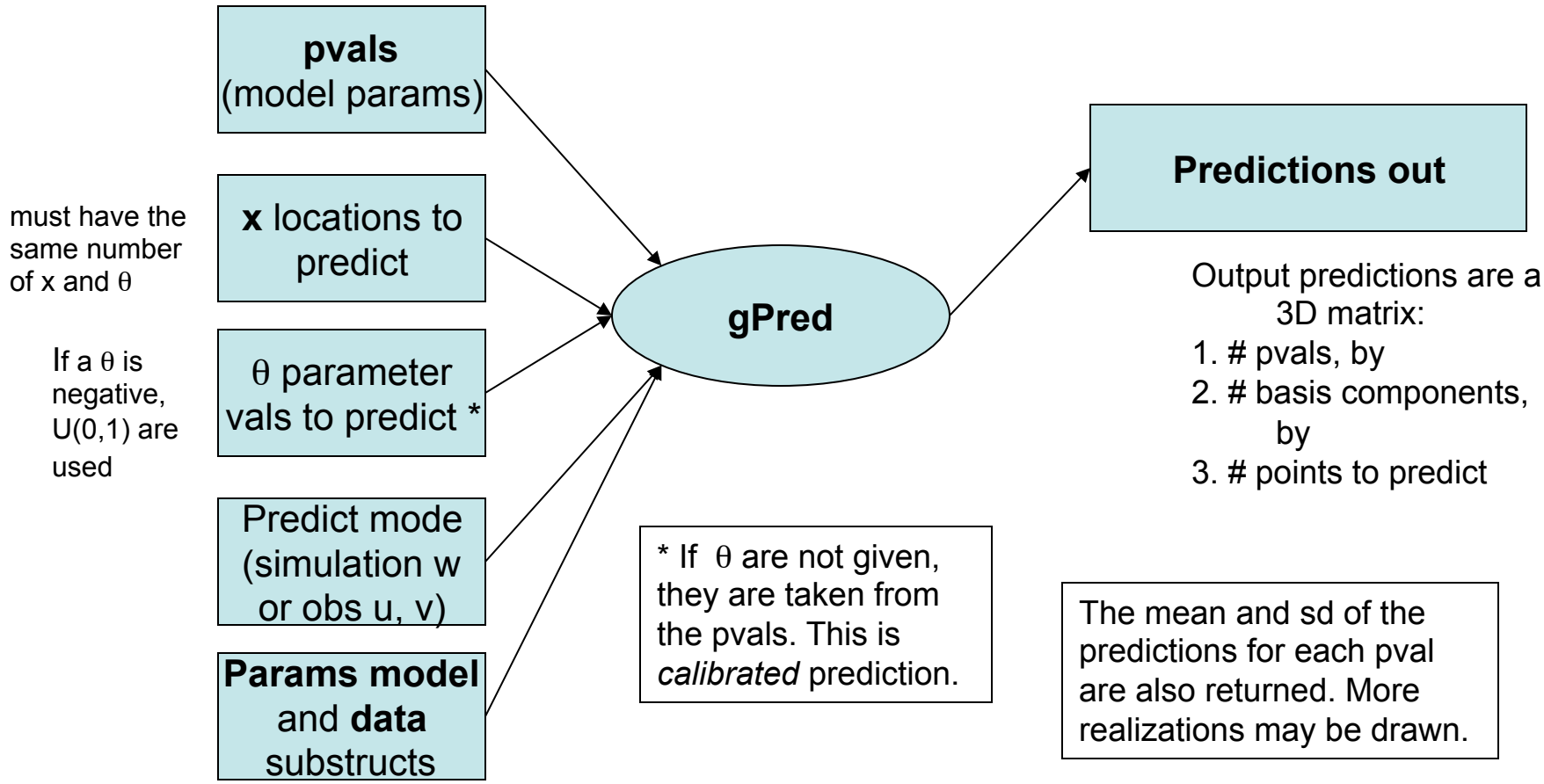
Default scheme is:

lamVz	adaptive
lamUz	adaptive
lamWOs	adaptive
lamWs	adaptive
lamOs	adaptive
rhoU	0.1
rhoV	0.1
theta	0.2

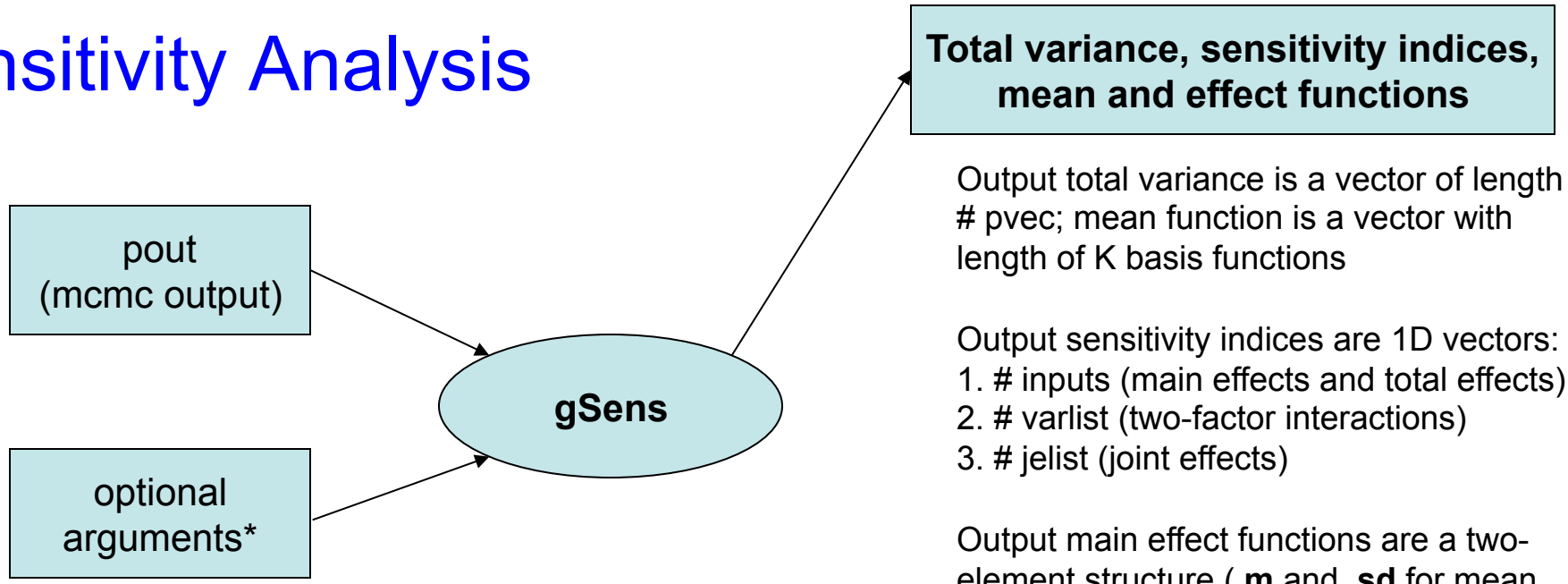
Adaptive indicates the range is +/- 1/3 of current value. These parameters can wander over orders of magnitude, so constant step sizes can cause sample chain problems.

A new feature is the adaptive selection of step size based on a sample run of the chain. Acceptance rates are noted for a collection of candidate step sizes, and a logistic regression model is fit to estimate the step size corresponding to a target acceptance rate (Graves, LA-UR-05-2359).

Prediction is more complex, since there are different motivations for predicting output



Sensitivity Analysis



*Optional arguments are:
pvec: subset of MCMC iterates (default all)
ngrid: number of grid locations for function evaluation (default 21)
varlist: pairs of variables for which joint effects are to be evaluated (default empty); 'all' – indicates all pairs
jelist: cell array with row vectors indicating variables for which joint effects are to be evaluated (default empty)
rg: matrix of ranges over which variables are to be integrated for computation of sensitivity indices and functions (default empty)
option: sensitivity calculations use (default empty) –
'mean' – posterior mean GP parameters
'median' – posterior median GP parameters
params – user provided GP parameters (betaU, lamUz, lamWs fields required)

Output total variance is a vector of length # pvec; mean function is a vector with length of K basis functions

Output sensitivity indices are 1D vectors:
1. # inputs (main effects and total effects)
2. # varlist (two-factor interactions)
3. # jelist (joint effects)

Output main effect functions are a two-element structure (.**m** and .**sd** for mean and standard deviation); each element is a 3D matrix:
1. # inputs, by
2. length of K basis functions, by
3. # grid

Output two-factor joint effect functions for each input pair in varlist are a two-element structure (.**m** and .**sd**); each element is a 3D matrix:
1. # varlist, by
2. # grid × length of K basis functions, by
3. # grid

Additional outputs available to extract uncertainties in indices and functions

Example

0. (generate data)
1. create required fields for simData and obsData
    1. Scale z,t to [0,1], standardize response vars to ~N(0,1)
    2. Create the K and D bases
2. Call setupModel to create the params structure
3. Run the model, pout=gpmmcmc(params,1000)
4. Examine the drawn model parameters
    1. Diagnostic impressions
    2. Variable significance from Rho
    3. Theta calibration
5. Generate predictions from the model for the η (simulator; u or Ku) and δ (discrepancy; v or Dv) models
6. Examine variable calibration, sensitivity, etc.
7. Calibration of k-L turbulent mix model…

Runfile

```
%commands to run the RT-Spike mix example...
addpath('../../matlab')

%read data
dat = fd(1,'pcpct',0.99975);

%initial set-up
params = setupModel(dat.obsData,dat.simData);

%modifications to defaults
params.priors.lamVz.params = repmat([1 0.0001],params.model.lamVzGnum,1);
params.priors.lamWs.params = repmat([1 0.0001],params.model.pu,1);

%step size
nburn=500; nlev=21;
params=stepsize(params,nburn,nlev);

%mcmc
nmcmc=10000;
pout=gpmmcmc(params,nmcmc,'step',1);
save pout pout;

nmcmc=nmcmc+nburn*nlev;
pvec=floor(linspace(nburn*nlev+1,nmcmc,500));
```

Add path to main GPM subroutines

Run main input subroutine for pre-processing

Run setupModel. Currently, changes to the specification of prior distribution parameters for lamWOs and lamOs must be made in setupModel. All other changes to set-up can be made in the main runfile.

Prior parameters for lamVz and lamWs. Others set to defaults defined in setupModel.

Runs automated step-size selection algorithm
nlev: number of candidate step sizes for each parameter
nburn: number of MCMC iterations at each candidate step size

Production run of MCMC algorithm

Runfile (cont' d)

```
%plots
%load pout;
fdPlots(pout,pvec,1:4);

%calibration parameters
tmp = [pout.pvals.theta]';
save 'theta' tmp '-ascii';

%sensitivity analysis
rn=[.5 .5;0 1;0 1;0 1];
sens=gSens(pout,'pvec',pvec,'varlist','all','rg',rn);

pout.sens=sens;
save pout pout;

tmp=[sens.smePm;sens.stePm]';
save 'sa_pm' tmp '-ascii';
tmp=sens.siePm';
save 'sa_ie' tmp '-ascii';

fdPlots(pout,pvec,5);
```

Run main output subroutine for generating plots

Save posterior samples of calibration parameters to file

Plot mean of main effect functions

Code for variance-based sensitivity analysis
sens: structure storing all sensitivity output
gSens: computes main and total effect sensitivity indices (two-factor interaction effect indices optional), and means/standard deviations for main effect and two-factor joint effect functions (optional)
smePm: holds mean of main effect sensitivity indices (weighted average over basis components)
stePm: holds mean of total effect sensitivity indices (weighted average over basis components)
siePm: holds mean of two-factor interaction effect sensitivity indices (weighted average over basis components)

Main Input File

pcpct: proportion of total variance explained by principal components
nkern: number of kernels used in discrepancy model

design: # runs by dimension of parameter space; standardize inputs to unit hypercube

simdata: first column is "time," remaining colunms are model outputs
ysim: "time" by "space" collection of model runs

obsdata: first column is "time", second column is experimental data

Standardize sims: subtract row means ("time"), divide by overall standard deviation

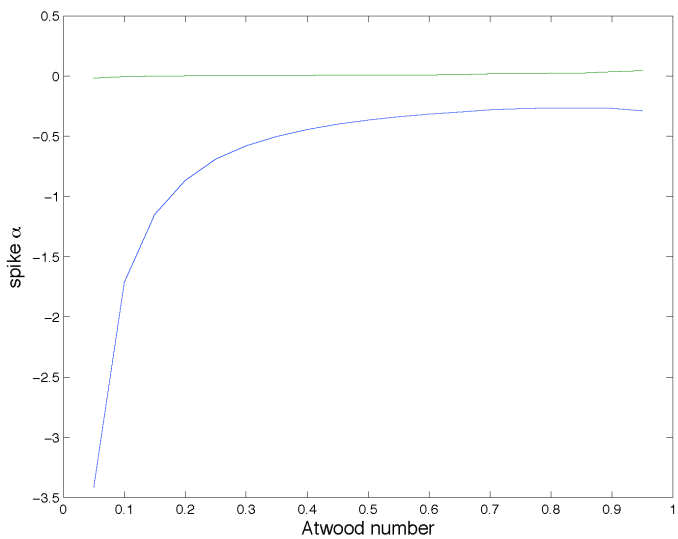
```
function params=fd(doPlot,varargin);

if ~exist('doPlot'); doPlot=0; else; doPlot=1; end

pcpct = 0.99; nkern = 11;
parseAssignVarargs({'pcpct','nkern'});

% read in design
design = textread('design.txt'); m=size(design,1);
% scale design
xmin = min(design); xrange = max(design)-xmin;
design=(design-repmat(xmin,m,1))./repmat(xrange,m,1);

% read in sim data
simdata = textread('sim_outputs');
ysim = simdata(:,2:end);
tsim = simdata(:,1);

% read in obs data
n = 1; % number of experiments
for ii=1:n
  inf = ['obs_outputs' int2str(ii)];
  obsdata{ii} = textread(inf);
  obsdata{ii} = obsdata{ii}(1:end-2,:);
  ydat{ii} = obsdata{ii}(:,2);
  tdat{ii} = obsdata{ii}(:,1);
  Sigy{ii} = 1.0^2 .* eye(length(ydat{ii}));
end

% summary stats from sims
ysimmean = mean(ysim,2);
ysimStd = ysim-repmat(ysimmean,1,m);
ysimsd = std(ysimStd(:));
ysimStd = ysimStd/ysimsd;
```

Main Input File (cont'd)

Interpolate simulator mean on "time points" corresponding to data grid; standardize data consistent with sims standardization above

```
% structures for field data
% interpolate to data grid and standardize experimental data
for ii=1:n
  yobs(ii).y = ydat{ii}; yobs(ii).t = tdat{ii};
  yobs(ii).ymean = interp1(tsim,ysimmean,yobs(ii).t,...
                           'linear','extrap');
  yobs(ii).yStd = (yobs(ii).y-yobs(ii).ymean)/ysimsd;
%  yobs(ii).Sigy = Sigy{ii}./(ysimsd.^2);
  yobs(ii).Sigy = Sigy{ii};
end
```

Simulator basis: eigenvectors computed from SVD



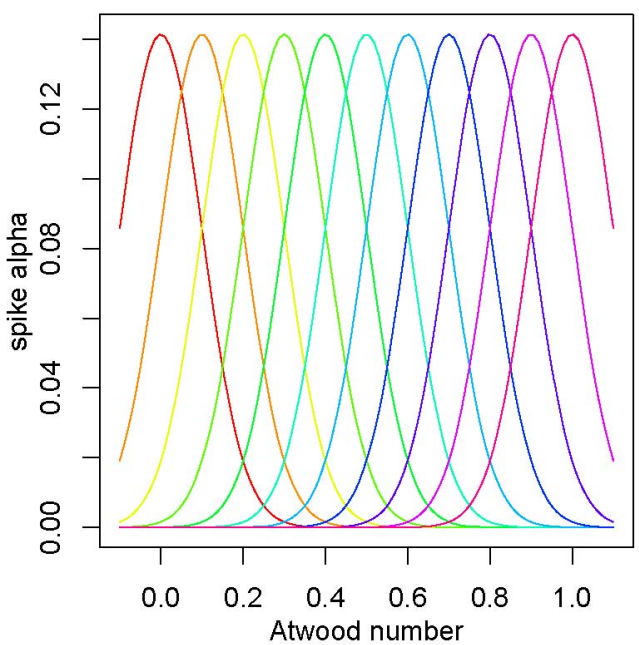

```
% K basis
% compute on simulations
[U,S,V]=svd(ysimStd,0);
lam = diag(S).^2/sum(diag(S).^2); lam = cumsum(lam);
pu = sum(lam<pcpct)+1;
Ksim=U(:,1:pu)*S(1:pu,1:pu)./sqrt(m);

if(doPlot)
  figure(1);
  plot(tsim,Ksim);
  xlabel('Atwood number','FontSize',14);
  ylabel('spike \alpha','FontSize',14);
  figure(1); print -depsc2 fdPc; close;
end
```

Main Input File (cont'd)

Linear interpolation of eigenvectors constructed from simulator runs to obtain their representation on the data grid

```
% interpolate K onto data grids
for ii=1:n
  yobs(ii).Kobs = zeros(length(yobs(ii).yStd),pu);
  for jj=1:pu
    yobs(ii).Kobs(:,jj) = interp1(tsim,Ksim(:,jj),...
                          yobs(ii).t,'linear','extrap');
  end
end
```

Construct kernel basis for discrepancy: 11 equally-spaced kernels with bandwidth equal to separation between centers

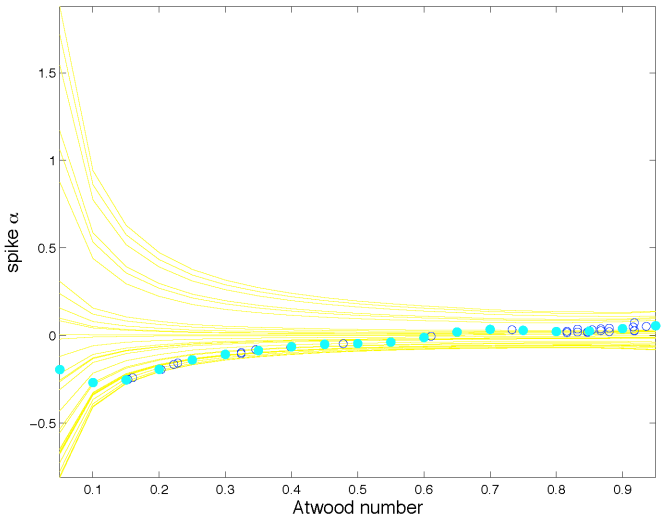



```
% D basis
% lay it out, and record decomposition on sim and data grids
% kernel centers and widths
Dgrid = linspace(min(tsim),max(tsim),nkern);
Dwidth = Dgrid(2)-Dgrid(1);
Dgrid = [Dgrid(1)-Dwidth Dgrid Dgrid(nkern)+Dwidth];
pv=length(Dgrid);
% compute the kernel function map, for each kernel
Dsim=zeros(size(ysimStd,1),pv);
for ii=1:n; yobs(ii).Dobs=zeros(length(yobs(ii).yStd),pv); end
for jj=1:pv
% first the obs
  for ii=1:n
    yobs(ii).Dobs(:,jj)=normpdf(yobs(ii).t,...
                                Dgrid(jj),Dwidth);
  end
% now the sim
  Dsim(:,jj)=normpdf(tsim,Dgrid(jj),Dwidth);
end
```

Main Input File (cont'd)

Normalize discrepancy kernel basis for numerical stability

```
% normalize the D maps
Dmax=max(max(Dsim*Dsim'));
Dsim=Dsim/sqrt(Dmax);
for ii=1:n; yobs(ii).Dobs=yobs(ii).Dobs/sqrt(Dmax); end

% plot sims and data
if (doPlot)
  Xp=[Ksim Dsim];
  for ii=1:n
    X=[yobs(ii).Kobs yobs(ii).Dobs]; Lamy=inv(yobs(ii).Sigy);
    uvhat=inv(X'*Lamy*X+1e-4*eye(pu+pv))*X'*Lamy*yobs(ii).yStd;
    yhat{ii}=Xp*uvhat; yhat{ii}=yhat{ii}*ysimsd+ysimmean;
  end
  figure(2);
  for ii=1:m, cysim(:,ii)=ysim(:,ii)-ysimmean; end
  plot(tsim,cysim,'y'); hold on;
  xlabel('Atwood number','FontSize',14);
  ylabel('spike \alpha','FontSize',14);
  for ii=1:n
    plot(tdat{ii},yobs(ii).y-yobs(ii).ymean,'bo'); hold on;
    plot(tsim,yhat{ii}-ysimmean,'c.','MarkerSize',20); hold on;
  end
  xlim([min(tsim) max(tsim)]);
  ylim([min([cysim(:);[yobs.y]]) max([cysim(:);[yobs.y]])]);
  figure(2); print -depsc2 fdDat; close;
end
```



Main Input File (cont'd)

Populate simData; orig structure contains elements needed for post-processing

Populate obsData; orig structure contains elements needed for post-processing

```
% record the data into the obsData and simData structures
% first simData
% required fields
% must include dummy x if no x parameters in design
 simData.x = [.5*ones([m 1]) design];
 simData.yStd=ysimStd;
 simData.Ksim=Ksim;
% extra fields: original data and transform stuff
 simData.orig.y=ysim;
 simData.orig.ymean=ysimmean;
 simData.orig.ysd=ysimsd;
 simData.orig.Dsim=Dsim;
 simData.orig.t=tsim;
 simData.orig.xmin=[0.5 xmin];
 simData.orig.xrange=[0 xrange];
% obsData
% set the x values
% use dummy x if no x parameters in design
 x = [.5];
 for ii=1:n
% required fields
   obsData(ii).x = x(ii,:);
   obsData(ii).yStd=yobs(ii).yStd;
   obsData(ii).Kobs=yobs(ii).Kobs;
   obsData(ii).Dobs=yobs(ii).Dobs;
   obsData(ii).Sigy=yobs(ii).Sigy;
% extra fields
   obsData(ii).orig.y=yobs(ii).y;
   obsData(ii).orig.ymean=yobs(ii).ymean;
   obsData(ii).orig.t =yobs(ii).t;
 end
% pack up and leave
 params.simData=simData; params.obsData=obsData;
end
```

Main Output File

pout: contains everything from set-up and MCMC results
pvec: indicates subset of MCMC runs to be used for prediction
plotnum: indicates which plots are to be drawn

parameter labels for plotting

pu: number of principal components
pv: number of kernel basis functions
p: number of x parameters (control)
q: number of θ parameters (calibration)

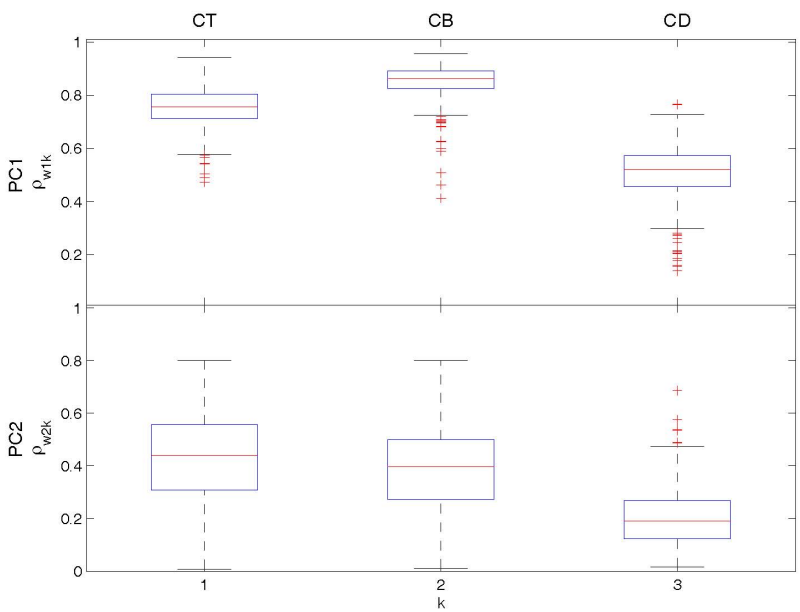
```
function fdPlots(pout,pvec,plotnum,varargin)

% Set up parameter labels
xlabs = {};
thlabs = {'CT','CB','CD'};
labs=[xlabs thlabs];
nxlabs=length(xlabs); nlabs=length(labs);

model=pout.model;
data=pout.data;
pvals=pout.pvals(pvec);
nreal=length(pvals);

pu=model.pu; pv=model.pv;
p=model.p; q=model.q;

doPlot(1:5)=0;
if exist('plotnum'); doPlot(plotnum)=1; end

% process input arguments
ngrid=21; subset=1:nlabs;
parseAssignVarargs({'ngrid','subset'});
lsub=length(subset);
if nxlabs, thsub=subset(subset>p)-p;
else thsub=subset; end
```

Main Output File (cont'd)

grid for main effect sensitivity plot

Plot 1: boxplots of correlation lengths for each principal component

Transformation from β to ρ




```
% Set up prediction grid
grid=linspace(0,1,ngrid);

if doPlot(1)
   % plot the betaU response in box plots
   figure(1); clf; colormap([0 0 0]);
   bu=[pvals.betaU]';
   ru=exp(-bu/4);
   complexity=[]; sparsity=[];
   for ii=1:pu
      if nxlabs, b=bu(:,(ii-1)*(p+q)+1:ii*(p+q));
      else b=bu(:,(ii-1)*(p+q)+p+1:ii*(p+q)); end
      complexity=[complexity sum(b,2)];
      sparsity=[sparsity sum(b.^2,2)];
      if nxlabs, r=ru(:,(ii-1)*(p+q)+1:ii*(p+q));
      else r=ru(:,(ii-1)*(p+q)+p+1:ii*(p+q)); end
      gPackSubplot(pu,1,ii,1);
      boxplot(r);
      ylab = ['\rho_{w',num2str(ii),'k}'];
      ylabel(ylab,'FontSize',12); xlabel('k');
      if ii==pu, ylim([0 1.01]);
      else ylim([0.01 1.01]); end
      if ii==1
         text(1:nlabs,ones([1 nlabs])+0.08,labs, ...
              'FontSize',12,'HorizontalAlignment','center');
      end
      text(0.2,0.5,['PC' int2str(ii)],'FontSize',12,...
           'HorizontalAlignment','center','Rotation',90);
   end
   figure(1); print -depsc2 fdRhoBox; close;
   save 'complexity' complexity '-ascii';
   save 'sparsity' sparsity '-ascii';
end
```

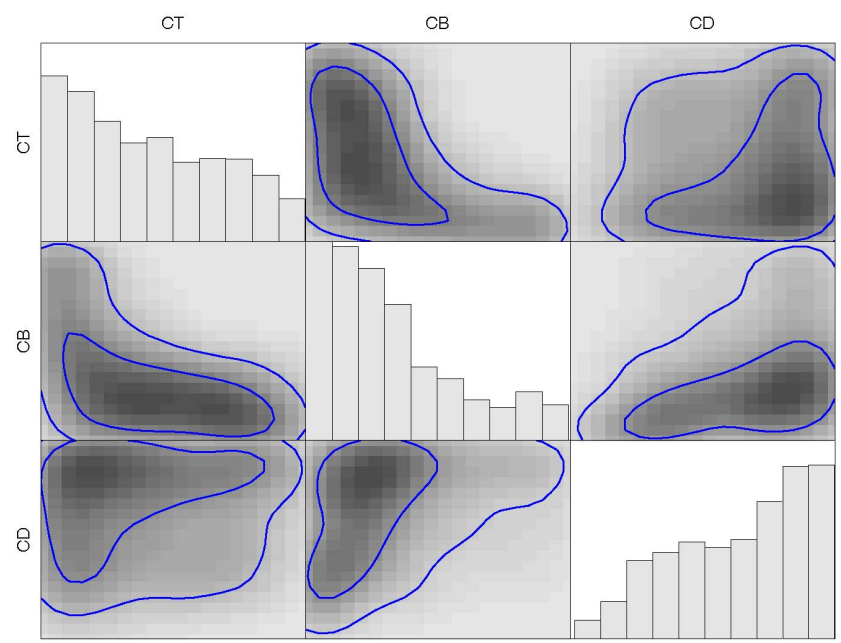
Main Output File (cont'd)

Plot 2: density estimates for univariate and bivariate marginal posterior distributions of θ

```
if doPlot(2)
   figure(2); clf;

   if length(pvec)>1000,
      pvec2=pvec(floor(linspace(1,length(pvec),1000)));
   else pvec2=pvec; end

   t=[pout.pvals(pvec2).theta]'; t=t(:,thsub);
   gPlotMatrix(t,'Pcontours',[0.5 0.9], ...
               'ustyle','imcont','lstyle','imcont', ...
               'ngrid',ngrid,'ksd',0.1, ...
               'labels',thlabs(thsub));
   figure(2); print -depsc2 fdPost; close;
end
```

Histogram of marginal distributions on diagonal

Density esitmates of bivariate marginals off-diagonal



Main Output File (cont'd)

Plot 3: prediction plots
(i) calibrated simulator,
(ii) discrepancy
(iii) discrepancy-adjusted calibrated simulator

```
if doPlot(3)
  h=[];
  ctr=0;
  if ctr,
    for ii=1:pout.model.m
      ysim(:,ii)=pout.simData.orig.y(:,ii)-pout.simData.orig.ymean;
    end
    for ii=1:pout.model.n
      yobs{ii}=pout.obsData(ii).orig.y-pout.obsData(ii).orig.ymean;
    end
  else
    ysim=pout.simData.orig.y;
    for ii=1:pout.model.n, yobs{ii}=pout.obsData(ii).orig.y; end
  end
  reps=0;
  if reps
    ind=1;
    nind=length(ind);
  else ind=1:pout.model.n; nind=pout.model.n; end
  tr=[min(pout.simData.orig.t) max(pout.simData.orig.t)];
  rtr=range(tr);

  for ii=1:nind
    jj=ind(ii);
    figure(3); clf;
    outF=strcat('fdPreds',int2str(ii));
```

Main Output File (cont'd)

Calibrated Simulator

pred.u: (# mcmc x # components) array of posterior realizations from simulator model weight processes evaluated at calibrated values of θ

eta: emulator (statistical representation of simulator) at each θ, on original scale

etabounds: 5% and 95% pointwise quantiles from eta

meanmat: matrix with columns equal to the mean of the original simulations across "space"

Discrepancy

pred.v: (# mcmc x # kernels) array of posterior realizations from discrepancy model weight processes

deltaR: discrepancy realizations on original scale

deltaRbounds: 5% and 95% pointwise quantiles from deltaR

```
% calibrated eta
pred=gPred(pout.obsData(jj).x,pvals,model,data,'uvpred');
eta = pout.simData.Ksim*pred.u'.*pout.simData.orig.ysd;
save 'etapred' eta '-ascii';
etabounds = prctile(eta,[5 95],2);
if ctr, meanmat=0;
else meanmat = repmat(pout.simData.orig.ymean,[1 2]); end

% now delta
deltaR = pout.simData.orig.Dsim*pred.v'.*pout.simData.orig.ysd;
save 'deltapred' deltaR '-ascii';
deltaRbounds = prctile(deltaR,[5 95],2);

% now a discrepancy-adjusted prediction
yhat = deltaR+eta;
save 'zetapred' yhat '-ascii';
yhatbounds = prctile(yhat,[5 95],2);

% plot
h(1)=gPackSubplot(1,3,1,1);
plot(pout.simData.orig.t,ysim,'y');
hold on;
if ii==nind, ll=pout.model.n; else ll=ind(ii+1)-1; end
for kk=jj:ll
  plot(pout.obsData(kk).orig.t,yobs{kk},'bo'); hold on;
end
plot(pout.simData.orig.t,etabounds+meanmat,'g','LineWidth',1);
ylabel('spike \alpha','FontSize',12);
title('calibrated simulator','Fontsize',12);
```

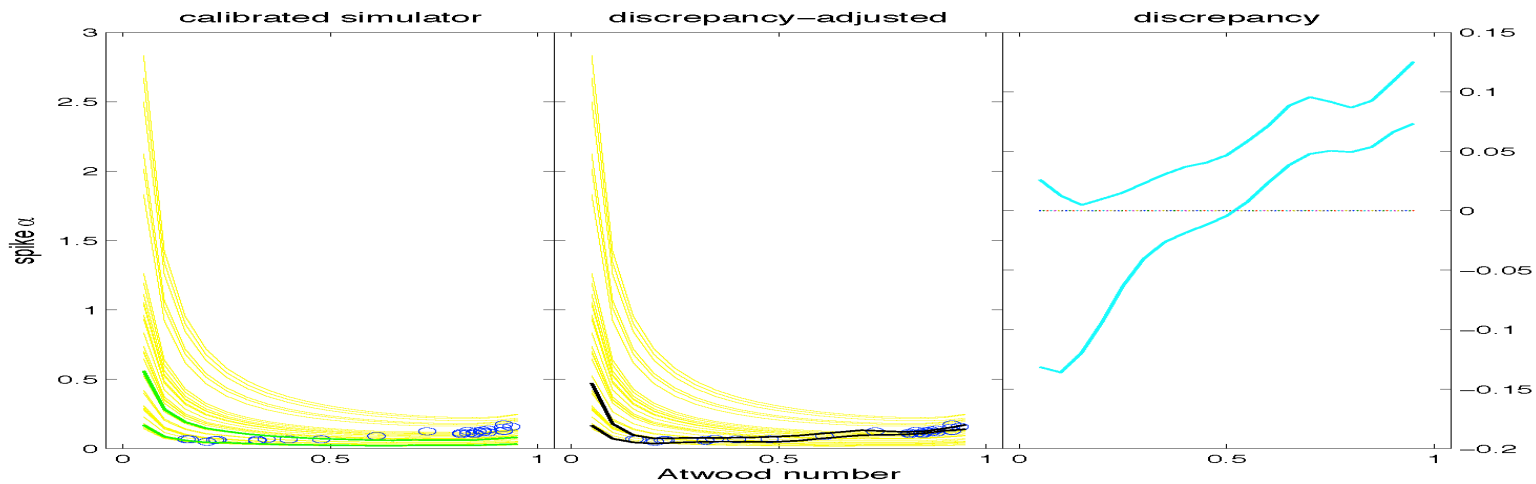
Discrepancy-adjusted Calibrated Simulator

yhat: calibrated simulator (eta) + discrepancy (deltaR)

yhatbounds: 5% and 95% pointwise quantiles from yhat

Main Output File (cont'd)

```
    h(2)=gPackSubplot(1,3,1,2);
    plot(pout.simData.orig.t,ysim,'y');
    hold on;
    for kk=jj:ll, plot(pout.obsData(kk).orig.t,yobs{kk},'bo'); hold on; end
    plot(pout.simData.orig.t,yhatbounds+meanmat,'k','LineWidth',1);
    set(gca,'YtickLabel','');
    xlabel('Atwood number','FontSize',12);
    title('discrepancy-adjusted','Fontsize',12);

    h(3)=gPackSubplot(1,3,1,3);
    plot(pout.simData.orig.t,deltaRbounds,'c','LineWidth',1); hold on;
    line(tr(1):rtr/100:tr(2),0,'LineStyle','--');
    set(gca,'YaxisLocation','right');
    title('discrepancy','Fontsize',12);

    axisNorm(h(1:2),'ymax'); axisNorm(h(3),'ymax');
    axisNorm(h(1:3),'x',[tr(1)-0.1*rtr tr(2)+0.1*rtr]);
    figure(3); print('-depsc2',outF); close;
  end
end
```



Main Output File (cont'd)

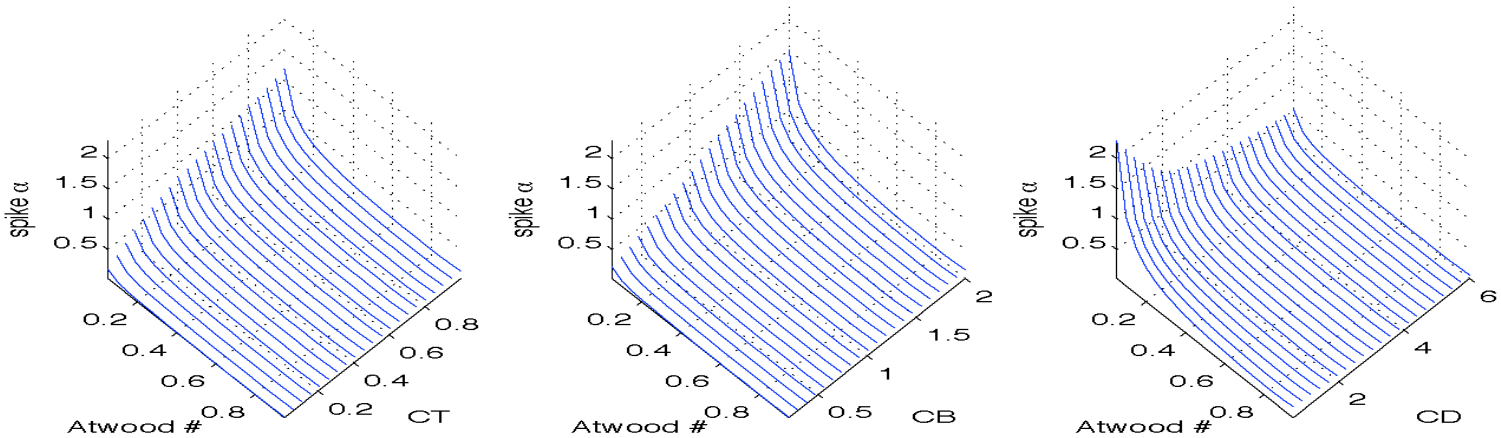
Plot 4: "baselline" sensitivity plots (each parameter individually varied; others held at nominal)

```
if doPlot(4)
   h=[];
   ctr=0;
   % Now the realizations over each theta
   figure(4); clf; colormap('copper');
   npred=ngrid;
   AzEl=[45 55];
   np=min(lsub,4);
   for ii=1:lsub
     jj=subset(ii);
     des=ones(npred,p+q)*0.5;
     if nxlabs
       des(:,jj)=grid';
       tt=pout.simData.orig.xrange(jj)*grid+ ...
          pout.simData.orig.xmin(jj);
     else
       des(:,p+jj)=grid';
       tt=pout.simData.orig.xrange(p+jj)*grid+ ...
          pout.simData.orig.xmin(p+jj);
     end
     xpred=des(:,1:p); theta=des(:,1+p:end);
     pred=gPred(xpred,pvals,model,data,'wpred',theta);
     pw=zeros(length(pvals),pu,npred);
     for kk=1:pu
       pw(:,kk,:)=pred.Myhat(:,(kk-1)*npred+1:kk*npred);
     end
     pm=reshape(squeeze(mean(pw,1)),pu,[]);
     r=(pout.simData.Ksim*pm)'*pout.simData.orig.ysd;
```

pred: (# mcmc x # components x # prediction sites) array of realizations from simulator model weight processes
pm: realizations averaged over MCMC samples
r: emulator (statistical representation of simulator) at each prediction site, on original scale

Main Output File (cont'd)

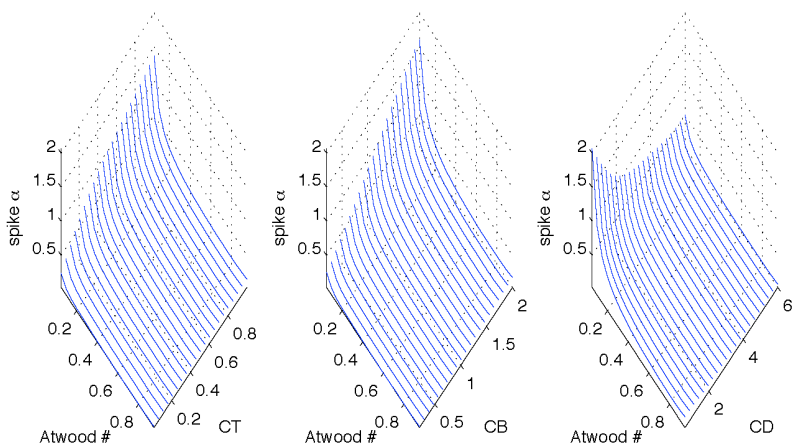
```
    if ~ctr
      for kk=1:npred
        r(kk,:)=r(kk,:)+pout.simData.orig.ymean';
      end
    end

    h(ii)=gPackSubplot(ceil(lsub/np),np,ceil(ii/np),...
                       mod(ii-1,np)+1,0.6);
    plot3(repmat(pout.simData.orig.t,size(grid)),...
      repmat(tt,size(pout.simData.orig.t)),r','b');
      view(AzEl);
      xlabel('Atwood #'); ylabel(labs(jj));
      zlabel('spike \alpha');
      alpha(0.25);
      axis tight;
      set(gca,'Xgrid','on','Ygrid','on','Zgrid','on');
  end
  axisNorm(h,'xzmax');
  figure(4); print -depsc2 fdNomSens; close;
end
```



Main Output File (cont'd)

Plot 5: main effect sensitivity plots (each parameter individually varied; average over others)

me: (# θ x length of basis vectors x # prediction sites) array with mean of main effect functions




```
if doPlot(5)
  h=[];
  ctr=0;
  % Now the marginalizations over each theta
  me=pout.sens.tmef.m; npred=size(me,3);
  if ctr, meanmat=repmat(pout.simData.orig.ymean,[1 npred]);
  else meanmat=0; end
  figure(5); clf; colormap('copper');
  tdat=0:1.0/(npred-1):1.0; AzEl=[45 55]; np=min(lsub,4);
  for ii=1:lsub
    jj=subset(ii);
    if nxlabs
      tt=pout.simData.orig.xrange(jj)*tdat+ ...
         pout.simData.orig.xmin(jj);
    else
      tt=pout.simData.orig.xrange(p+jj)*tdat+ ...
         pout.simData.orig.xmin(p+jj);
    end
    r=squeeze(me(jj,:,:))-meanmat;
    h(ii)=gPackSubplot(ceil(lsub/np),np,ceil(ii/np), ...
                       mod(ii-1,np)+1,0.6);
    plot3(repmat(pout.simData.orig.t,size(tdat)),...
          repmat(tt,size(pout.simData.orig.t)),r,'b');
    view(AzEl);
    xlabel('Atwood #'); ylabel(labs(jj));
    zlabel('spike \alpha');
    alpha(0.25); axis tight;
    set(gca,'Xgrid','on','Ygrid','on','Zgrid','on');
  end
  axisNorm(h,'xzmax');
  figure(5); print -depsc2 fdMeSens; close;
 end
end
```