

Package: impala (via r-universe)

September 21, 2026

Type Package

Title Bayesian Model Calibration

Version 0.1.4.9000

Description Package provides tools for modular Bayesian model calibration. these tools allow for posterior exploration with sampling methods including tempering and adaptive Markov Chain Monte Carlo (MCMC). Allows for pooled calibration or hierarchal calibration of parameters. For more information see Francom et al., 2025 <[DOI:10.1137/24M1644092](https://doi.org/10.1137/24M1644092)>.

License MIT + file LICENSE

Encoding UTF-8

Imports cli, einsum, methods, parallel, progress, stats, utils

Suggests fdasrvf, BASS, mvBayes, bayesplot, gridExtra, hexbin, knitr, rmarkdown, testthat (>= 3.0.0)

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

URL <https://github.com/sandialabs/rImpala>

BugReports <https://github.com/sandialabs/rImpala/issues>

VignetteBuilder knitr

Repository <https://sandialabs.r-universe.dev>

Date/Publication 2026-09-21 16:54:12 UTC

RemoteUrl <https://github.com/sandialabs/rimpala>

RemoteRef HEAD

RemoteSha d76251149d52b7153342af77cc4e7adcd57e1562

Contents

addJointThetaPrior	2
addThetaPrior	3

addVecExperiments	4
calibPool	6
CalibSetup	6
cf_bounds	7
discrep_sample	8
evalm	8
fixTheta	9
gen_cand	10
lik_cov_inv	10
llik	11
ModelBassPca_func	11
ModelmvBayes	12
ModelmvBayes_elastic	13
ModelmvBayes_elastic_GP	14
ModelmvBayes_GP	15
setMCMC	15
setTemperatureLadder	16
step_m	17
tran_unif	17
unfixTheta	18
update_m	18
update_tau	19

Index	20
--------------	-----------

addJointThetaPrior	<i>Add joint theta prior</i>
--------------------	------------------------------

Description

This method adds a joint prior over multiple theta parameters

Usage

```
addJointThetaPrior(obj, pnames, log_density_fn)
```

Arguments

obj	‘CalibSetup’ Object
pnames	character vector of parameter names to apply joint prior to
log_density_fn	function that takes named list of parameter values and returns log-density

Details

‘log_density_fn’ receives a named list of parameter *vectors*, one element per temperature, and must return a vector of log densities of that same length. Write it vectorised, or wrap a scalar density in [mapply()].

Value

An object of class ‘CalibSetup’

Examples

```
# Joint prior on t_1 and t_2 correlating them through a bivariate normal.
# params arrives as list(t_1 = <vector>, t_2 = <vector>), one entry per
# temperature, so the result must be a vector of the same length.
my_joint_prior <- function(params) {
  rho <- 0.5
  z <- (params$t_1^2 - 2 * rho * params$t_1 * params$t_2 + params$t_2^2) /
    (1 - rho^2)
  -0.5 * z - log(2 * pi * sqrt(1 - rho^2))
}

bounds = list()
bounds[['t_1']] = c(0, 1)
bounds[['t_2']] = c(0, 1)
setup <- CalibSetup(bounds, cf_bounds)
setup <- addJointThetaPrior(setup, c("t_1", "t_2"), my_joint_prior)

# returns one log density per temperature
my_joint_prior(list(t_1 = c(0.2, 0.5), t_2 = c(0.3, 0.6)))
```

addThetaPrior

Add theta prior

Description

This method adds a prior for the specified theta

Usage

```
addThetaPrior(
  obj,
  dist = "uniform",
  params = list(min = 0, max = 1),
  pname = NULL
)
```

Arguments

obj	‘CalibSetup’ Object
dist	string defining distribution e.g., ‘normal’, ‘beta’, etc.
params	list of parameters e.g., if dist=‘normal’, params is a list with elements ‘mean’ and ‘sd’
pname	name of parameter

Value

An object of class ‘CalibSetup’

addVecExperiments	<i>Add vector experiments</i>
-------------------	-------------------------------

Description

This method adds vector experiments to calibration object

Usage

```
addVecExperiments(
  obj,
  yobs,
  model,
  sd_est,
  s2_df,
  s2_ind,
  sd_lower = NULL,
  sd_upper = NULL,
  meas_error_cor = NULL,
  theta_ind = NULL,
  D = NULL,
  discrep_tau = 1
)
```

Arguments

obj	‘CalibSetup’ Object
yobs	a vector of the experiment or observation
model	emulator (currently expecting a object of class ‘ModelBassPca_func’ or ‘ModelmvBayes’)
sd_est	vector of initial standard deviation estimates, one per separately estimated measurement error variance. ‘length(sd_est)’ sets the number of variance groups.
s2_df	vector of inverse gamma prior degrees of freedom, one per variance group, so ‘length(s2_df)’ must equal ‘length(sd_est)’. A value of zero selects a half-Cauchy prior instead; if any entry is zero the half-Cauchy is used for every group.
s2_ind	vector of 1-based variance group indices, one per element of ‘yobs’, mapping each observation to the variance it is drawn with. Must satisfy ‘length(s2_ind) == length(yobs)’ and ‘max(s2_ind) <= length(sd_est)’.
sd_lower	optional vector of lower bounds on the learned standard deviations, one per variance group (default: ‘NULL’, meaning no lower bound)

sd_upper	optional vector of upper bounds on the learned standard deviations, one per variance group (default: 'NULL', meaning no upper bound)
meas_error_cor	measurement error correlation (default: 'NULL')
theta_ind	indices of theta (default: 'NULL')
D	discrepancy basis (matrix of columns of basis, default: 'NaN')
discrep_tau	discrepancy sampling tau

Details

'sd_est', 's2_df', 'sd_lower' and 'sd_upper' are all indexed by variance group, while 's2_ind' is indexed by observation. A scalar 'sd_est' with 's2_ind = rep(1, length(yobs))' gives the usual single measurement error variance for the whole vector.

Any grouping of the observations is allowed. Passing 's2_ind = seq_along(yobs)' with 'sd_est' of that same length learns a separate standard deviation for every component of 'yobs'. Note that each variance is then informed by a single observation, so the prior set through 's2_df' and 'sd_est' does most of the work; use a larger 's2_df', or 'sd_lower'/'sd_upper', when the per component variances need to be constrained.

Value

An object of class 'CalibSetup'

Examples

```
set.seed(1)
ny <- 20
A <- matrix(stats::rnorm(ny * 2), ny, 2)
yobs <- as.numeric(A %*% c(0.45, 0.55)) + stats::rnorm(ny, 0, 0.05)

# a minimal linear emulator standing in for a fitted surrogate
mod <- structure(list(s2 = "gibbs", nd = 0, stochastic = FALSE,
                     discrep_cov = diag(ny) * 1e-12), class = "ExampleModel")

setup <- CalibSetup(list(t_1 = c(0, 1), t_2 = c(0, 1)), cf_bounds)

# two error groups: the first half of yobs shares one variance, the rest another
two_group <- addVecExperiments(setup, yobs, mod,
                              sd_est = c(0.05, 0.05),
                              s2_df = c(2, 2),
                              s2_ind = rep(1:2, each = ny / 2))

two_group$ns2[[1]]

# one standard deviation per component of yobs, bounded away from zero
per_component <- addVecExperiments(setup, yobs, mod,
                                   sd_est = rep(0.05, ny),
                                   s2_df = rep(10, ny),
                                   s2_ind = seq_along(yobs),
                                   sd_lower = rep(0.01, ny),
                                   sd_upper = rep(0.50, ny))

per_component$ns2[[1]]
```

calibPool

Pool Calibration

Description

This function runs pooled Bayesian Model Calibration with adaptive MCMC, tempering, and decorrelation steps. input theta will be normalized to 0-1 and sampled from uniform priors

Usage

```
calibPool(setup)
```

Arguments

setup an object of class ‘CalibSetup’

Details

Parameters fixed with [fixTheta()] are held at their given values throughout: they keep their column in ‘theta’ and ‘theta_native’, constant at the fixed value, while the Metropolis proposal and the decorrelation step act on the free parameters only.

Value

a list with the following elements - theta: mcmc samples of variables - s2: mcmc samples of error variance - count: number of counts of acceptance - count_s2: number of counts of acceptance on error variance - count_decor: number of times decorrelation occurred - cov_theta_cand: final theta covariance - cov_ls2_cand: final error covariance - pred_curr: current emulator predictions - discrep_vars: discrepancy coefficients - llik: log likelihood - theta_native: mcmc samples of variables in native scale - theta_fixed: named vector of parameters held fixed, or ‘NULL’ if none

CalibSetup

Pooled Bayesian Model Calibration

Description

This function setups up calibration object

Usage

```
CalibSetup(bounds, constraint_func)
```

Arguments

- bounds a list with fields of variable names with values of two dimensional arrays where first element is lower bound and second element is upper bound.
- constraint_func function handle to constraint function on variables. Defaults to function on checking bounds

Value

An object of class 'CalibSetup' which is a list with the following components:

- 'nexp': number of experiments
- 'theta_fixed': named vector of parameters held fixed during calibration, 'NULL' until [fixTheta()] is called

cf_bounds

Compare to bounds

Description

This function compares variables to bounds

Usage

```
cf_bounds(x, bounds)
```

Arguments

- x list of of parameters
- bounds list of bounds for each parameter that is a two parameter vector with high and low

Value

a vector of 'TRUE' or 'FALSE' if the values are within the bounds

discrep_sample	<i>Sample discrepancy coefficients</i>
----------------	--

Description

Generic that draws discrepancy basis coefficients for an emulator model with a discrepancy basis attached.

Usage

```
discrep_sample(obj, ...)
```

Arguments

obj	an emulator model object
...	additional arguments passed to the method, typically the observations, the prediction, the marginal covariance, and the inverse temperature

Value

a vector of sampled discrepancy coefficients

evalm	<i>evalm constructor</i>
-------	--------------------------

Description

Default constructor for evalm class

Usage

```
evalm(obj, ...)
```

Arguments

obj	evalm object
...	additional arguments passed to method

Value

An object of class ‘evalm’

fixTheta	<i>Fix calibration parameters</i>
----------	-----------------------------------

Description

This method holds one or more calibration parameters at fixed values, so that [calibPool()] samples the conditional posterior of the remaining parameters. This supports conditional inference and cut-Bayes style analyses, where some inputs are treated as known, or as determined by a separate module, rather than learned from the data at hand.

Usage

```
fixTheta(obj, pname, value)
```

Arguments

obj	‘CalibSetup’ Object
pname	character vector of parameter names to fix; each must appear in ‘names(obj\$bounds)’
value	numeric vector of values to fix the parameters at, on the native scale of ‘obj\$bounds’ and the same length as ‘pname’

Details

Fixed parameters keep their column in the ‘theta’ and ‘theta_native’ output of [calibPool()]; that column is simply constant at the fixed value. The adaptive Metropolis proposal and the decorrelation step both act on the free parameters only, so fixing a parameter does not degrade the mixing of the others.

Calling ‘fixTheta()’ again for a parameter that is already fixed replaces its value. Use [unfixTheta()] to return a parameter to being sampled.

A prior added with [addThetaPrior()] or [addJointThetaPrior()] for a parameter that is fixed contributes a constant to the log posterior and so has no effect on the samples; a warning is issued when that is detected.

Fixing every parameter is allowed: [calibPool()] then samples the error variances and any discrepancy terms with ‘theta’ held at the fixed values.

Value

An object of class ‘CalibSetup’

Examples

```
bounds = list()
bounds[['t_1']] = c(0, 1)
bounds[['t_2']] = c(-2, 2)
setup <- CalibSetup(bounds, cf_bounds)
```

```
# hold t_2 at 0.5 and calibrate t_1 conditional on it
setup <- fixTheta(setup, "t_2", 0.5)
setup$theta_fixed

# several at once
setup <- fixTheta(setup, c("t_1", "t_2"), c(0.25, -1))
setup$theta_fixed

# and back to sampling both
setup <- unfixTheta(setup)
setup$theta_fixed
```

gen_cand	<i>Generate a proposal</i>
----------	----------------------------

Description

Generic that draws a Metropolis candidate from an adaptive proposal covariance.

Usage

```
gen_cand(obj, ...)
```

Arguments

obj	an adaptive covariance object, e.g. from ‘AMcov_pool’
...	additional arguments passed to the method, typically the chain of samples and the current iteration

Value

a matrix of candidate values, one row per temperature

lik_cov_inv	<i>Likelihood covariance inverse</i>
-------------	--------------------------------------

Description

Generic that builds the inverse and log determinant of the marginal likelihood covariance for an emulator model.

Usage

```
lik_cov_inv(obj, ...)
```

Arguments

`obj` an emulator model object

`...` additional arguments passed to the method, typically the vector of error variances

Value

a list with elements ‘inv’ (the inverse covariance) and ‘ldet’ (its log determinant)

llik	<i>Log likelihood</i>
------	-----------------------

Description

Generic for the log likelihood of an emulator model. Methods are provided for each supported emulator class; supply your own to calibrate with a custom emulator.

Usage

```
llik(obj, ...)
```

Arguments

`obj` an emulator model object

`...` additional arguments passed to the method, typically the observations, the emulator prediction, and the marginal covariance returned by `[lik_cov_inv()]`

Value

the log likelihood, a scalar

ModelBassPca_func	<i>PCA Based Model Emulator using BASS</i>
-------------------	--

Description

This function setups up emulator object

Usage

```
ModelBassPca_func(bmod, input_names, exp_ind = NULL, s2 = "MH")
```

Arguments

bmod	a object of the type 'bassBasis'
input_names	cell array of strings of input variable names
exp_ind	experiment indices (default: NULL)
s2	how to sample error variance (default: 'MH')

Value

An object of class 'ModelBassPca_func'

ModelmvBayes	<i>mvBayes Emulator for Functional Outputs (can use different BASS/BPPR type emulators)</i>
--------------	---

Description

ModelmvBayes Handles larger-dimensional functional responses (e.g., on large spatial fields) using various inversion tricks. We require any other covariance e.g., from discrepancy, measurement error, and basis truncation error) to be diagonal. This function setups up emulator object.

Usage

```
ModelmvBayes(bmod, input_names, exp_ind = NULL, s2 = "MH", h = FALSE)
```

Arguments

bmod	a object of the type 'mvBayes'
input_names	cell array of strings of input variable names
exp_ind	experiment indices (default: NULL)
s2	how to sample error variance (default: 'MH')
h	h representation of warping function (default: FALSE)

Value

An object of class 'ModelmvBayes'

ModelmvBayes_elastic	<i>mvBayes elastic Emulator for Functional Outputs (can use different BASS/BPPR type emulators)</i>
----------------------	---

Description

ModelmvBayes_elastic Handles larger-dimensional functional responses (e.g., on large spatial fields) using various inversion tricks. We require any other covariance e.g., from discrepancy, measurement error, and basis truncation error) to be diagonal. Contains a change on the likelihood

Usage

```
ModelmvBayes_elastic(
  bmod,
  bmod_warp,
  input_names,
  exp_ind = NULL,
  s2 = "MH",
  h = FALSE
)
```

Arguments

bmod	a object of the type ‘mvBayes’ of aligned functions
bmod_warp	a object of the type ‘mvBayes’ of warping functions
input_names	cell array of strings of input variable names
exp_ind	experiment indices (default: NULL)
s2	how to sample error variance (default: ‘MH’)
h	h representation of warping function (default: FALSE)

Details

This function setups up emulator object.

Value

An object of class ‘ModelmvBayes_elastic’

ModelmvBayes_elastic_GP

mvBayes elastic Emulator for Functional Outputs (can use different BASS/BPPR type emulators)

Description

mvBayes_elastic Handles larger-dimensional functional responses (e.g., on large spatial fields) using various inversion tricks. We require any other covariance e.g., from discrepancy, measurement error, and basis truncation error) to be diagonal. Contains a change on the likelihood and built for a GP emulator. This function setups up emulator object.

Usage

```
ModelmvBayes_elastic_GP(
  bmod,
  bmod_warp,
  input_names,
  exp_ind = NULL,
  s2 = "MH",
  h = FALSE
)
```

Arguments

bmod	a object of the type 'mvBayes' of aligned functions
bmod_warp	a object of the type 'mvBayes' of warping functions
input_names	cell array of strings of input variable names
exp_ind	experiment indices (default: NULL)
s2	how to sample error variance (default: 'MH')
h	h representation of warping function (default: FALSE)

Value

An object of class 'ModelmvBayes_elastic_GP'

ModelmvBayes_GP	<i>mvBayes Emulator for Functional Outputs (GP)</i>
-----------------	---

Description

ModelmvBayes_GP Handles larger-dimensional functional responses (e.g., on large spatial fields) using various inversion tricks. We require any other covariance e.g., from discrepancy, measurement error, and basis truncation error) to be diagonal. Built for a GP emulator.

Usage

```
ModelmvBayes_GP(bmod, input_names, exp_ind = NULL, s2 = "MH", h = FALSE)
```

Arguments

bmod	a object of the type 'mvBayes'
input_names	cell array of strings of input variable names
exp_ind	experiment indices (default: NULL)
s2	how to sample error variance (default: 'MH')
h	h representation of warping function (default: FALSE)

Details

This function setups up emulator object.

Value

An object of class 'ModelmvBayes_GP'

setMCMC	<i>Set MCMC Parameters</i>
---------	----------------------------

Description

This function setups up MCMC parameters for adaptive MCMC, also includes tempering and decorrelation steps

Usage

```
setMCMC(
  obj,
  nmcmc,
  nburn = 0,
  thin = 1,
  decor = 100,
  start_var_theta = 1e-08,
  start_tau_theta = 0,
  start_var_ls2 = 1e-05,
  start_tau_ls2 = 0,
  start_adapt_iter = 300
)
```

Arguments

obj	‘calibPool‘ object
nmcmc	number of mcmc iterations
nburn	number of mcmc burn in iterations (default: 0)
thin	number of samples to thin (default: 1)
decor	number of mcmc iterations before decorrelation step (default: 100)
start_var_theta	start variance of theta proposal (default: 1e-8)
start_tau_theta	start tau of theta (default: 0)
start_var_ls2	start variance of sigma proposal (default: 1e-5)
start_tau_ls2	start tau of sigma (default: 0)
start_adapt_iter	number of iterations before adaption (default: 300)

Value

An object of class ‘CalibSetup‘

setTemperatureLadder *Set Temperature Ladder*

Description

This function setups up temperature ladder for tempering

Usage

```
setTemperatureLadder(obj, temperature_ladder, start_temper = 1000)
```

Arguments

obj CalibSetup Object
 temperature_ladder array of temperatures
 start_temper what MCMC sample to start tempering (default: 1000)

Value

An object of class ‘CalibSetup’

step_m	<i>Advance a stochastic emulator</i>
--------	--------------------------------------

Description

Generic that advances an emulator one MCMC step, used for emulators that carry their own posterior samples.

Usage

```
step_m(obj, ...)
```

Arguments

obj an emulator model object
 ... additional arguments passed to the method

Value

the updated emulator model object

tran_unif	<i>Transform parameters</i>
-----------	-----------------------------

Description

Transforms variables from the scale 0 to 1 to variable provided by bounds

Usage

```
tran_unif(th, bounds, names)
```

Arguments

th	list of of parameters
bounds	list of bounds for each parameter that is a two parameter vector with high and low
names	vector of variable names

Value

a list of parameters

unfixTheta	<i>Unfix calibration parameters</i>
------------	-------------------------------------

Description

This method releases parameters fixed by [fixTheta()] so that [calibPool()] samples them again.

Usage

```
unfixTheta(obj, pname = NULL)
```

Arguments

obj	‘CalibSetup‘ Object
pname	character vector of parameter names to release, or ‘NULL‘ (the default) to release every fixed parameter

Value

An object of class ‘CalibSetup‘

update_m	<i>Update adaptive covariance</i>
----------	-----------------------------------

Description

Generic that updates the running mean and covariance of an adaptive Metropolis proposal.

Usage

```
update_m(obj, ...)
```

Arguments

obj	an adaptive covariance object, e.g. from ‘AMcov_pool’
...	additional arguments passed to the method, typically the chain of samples and the current iteration

Value

the updated adaptive covariance object

update_tau	<i>Update proposal scaling</i>
------------	--------------------------------

Description

Generic that applies diminishing adaptation to the proposal scale based on the recent acceptance rate.

Usage

```
update_tau(obj, ...)
```

Arguments

obj	an adaptive covariance object, e.g. from ‘AMcov_pool’
...	additional arguments passed to the method, typically the current iteration

Value

the updated adaptive covariance object

Index

`addJointThetaPrior`, [2](#)
`addThetaPrior`, [3](#)
`addVecExperiments`, [4](#)

`calibPool`, [6](#)
`CalibSetup`, [6](#)
`cf_bounds`, [7](#)

`discrep_sample`, [8](#)

`evalm`, [8](#)

`fixTheta`, [9](#)

`gen_cand`, [10](#)

`lik_cov_inv`, [10](#)
`llik`, [11](#)

`ModelBassPca_func`, [11](#)
`ModelmvBayes`, [12](#)
`ModelmvBayes_elastic`, [13](#)
`ModelmvBayes_elastic_GP`, [14](#)
`ModelmvBayes_GP`, [15](#)

`setMCMC`, [15](#)
`setTemperatureLadder`, [16](#)
`step_m`, [17](#)

`tran_unif`, [17](#)

`unfixTheta`, [18](#)
`update_m`, [18](#)
`update_tau`, [19](#)