Package 'laGP'

December 8, 2018

Title Local Approximate Gaussian Process Regression

Version 1.5-3

Date 2018-12-06

Author Robert B. Gramacy <rbg@vt.edu>, Furong Sun <furongs@vt.edu>

Depends R (>= 2.14)

Imports tgp, parallel

Suggests mvtnorm, MASS, akima, lhs, crs, DiceOptim

Description Performs approximate GP regression for large computer experiments and spatial datasets. The approximation is based on finding small local designs for prediction (independently) at particular inputs. OpenMP and SNOW parallelization are supported for prediction over a vast out-of-sample testing set; GPU acceleration is also supported for an important subroutine. OpenMP and GPU features may require special compilation. An interface to lower-level (full) GP inference and prediction is provided. Wrapper routines for black-box optimization under mixed equality and inequality constraints via an augmented Lagrangian scheme, and for large scale computer model calibration, are also provided.

Maintainer Robert B. Gramacy <rbg@vt.edu>

License LGPL

URL http://bobby.gramacy.com/r_packages/laGP

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-12-07 23:40:03 UTC

R topics documented:

---

aGP *Localized Approximate GP Regression For Many Predictive Locations*

Description

Facilitates localized Gaussian process inference and prediction at a large set of predictive locations, by (essentially) calling `laGP` at each location, and returning the moments of the predictive equations, and indices into the design, thus obtained

Usage

```
aGP(X, Z, XX, start = 6, end = 50, d = NULL, g = 1/10000,
    method = c("alc", "alcray", "mspe", "nn", "fish"), Xi.ret = TRUE,
    close = min((1000+end)*if(method[1] == "alcray") 10 else 1, nrow(X)),
    numrays = ncol(X), num.gpus = 0, gpu.threads = num.gpus,
    omp.threads = if (num.gpus > 0) 0 else 1,
    nn.gpu = if (num.gpus > 0) nrow(XX) else 0, verb = 1)
aGP.parallel(cls, XX, chunks = length(cls), X, Z, start = 6, end = 50,
    d = NULL, g = 1/10000, method = c("alc", "alcray", "mspe", "nn", "fish"),
    Xi.ret = TRUE,
    close = min((1000+end)*if(method[1] == "alcray") 10 else 1, nrow(X)),
    numrays = ncol(X), num.gpus = 0, gpu.threads = num.gpus,
    omp.threads = if (num.gpus > 0) 0 else 1,
    nn.gpu = if (num.gpus > 0) nrow(XX) else 0, verb = 1)
aGP.R(X, Z, XX, start = 6, end = 50, d = NULL, g = 1/10000,
    method = c("alc", "alcray", "mspe", "nn", "fish"), Xi.ret = TRUE,
    close = min((1000+end) *if(method[1] == "alcray") 10 else 1, nrow(X)),
    numrays = ncol(X), laGP=laGP.R, verb = 1)
aGPsep(X, Z, XX, start = 6, end = 50, d = NULL, g = 1/10000,
    method = c("alc", "alcray", "nn"), Xi.ret = TRUE,
    close = min((1000+end)*if(method[1] == "alcray") 10 else 1, nrow(X)),
    numrays = ncol(X),  omp.threads = 1, verb = 1)
aGPsep.R(X, Z, XX, start = 6, end = 50, d = NULL, g = 1/10000,
    method = c("alc", "alcray", "nn"), Xi.ret = TRUE,
    close = min((1000+end)*if(method[1] == "alcray") 10 else 1, nrow(X)),
    numrays = ncol(X), laGPsep=laGPsep.R, verb = 1)
aGP.seq(X, Z, XX, d, methods=rep("alc", 2), M=NULL, ncalib=0, ...)
```

Arguments

X	a matrix or data.frame containing the full (large) design matrix of input locations
Z	a vector of responses/dependent values with length(Z) = nrow(X)
XX	a matrix or data.frame of out-of-sample predictive locations with ncol(XX) = ncol(X); aGP calls laGP for each row of XX as a value of Xref, independently
start	the number of Nearest Neighbor (NN) locations to start each independent call to laGP with; must have start >= 6
end	the total size of the local designs; start < end
d	a prior or initial setting for the lengthscale parameter in a Gaussian correlation function; a (default) NULL value triggers a sensible regularization (prior) and initial setting to be generated via darg; a scalar specifies an initial value, causing darg to only generate the prior; otherwise, a list or partial list matching the output of darg can be used to specify a custom prior. In the case of a partial list, only the missing entries will be generated. Note that a default/generated list specifies MLE/MAP inference for this parameter. When specifying initial values, a vector of length nrow(XX) can be provided, giving a different initial value for each predictive location. With aGPsep, the starting values can be an ncol(X)-by-nrow(XX) matrix or an ncol(X) vector
g	a prior or initial setting for the nugget parameter; a NULL value causes a sensible regularization (prior) and initial setting to be generated via garg; a scalar (default g = 1/10000) specifies an initial value, causing garg to only generate the prior; otherwise, a list or partial list matching the output of garg can be used to specify a custom prior. In the case of a partial list, only the missing entries will be generated. Note that a default/generated list specifies *no* inference for this parameter; i.e., it is fixed at its starting or default value, which may be appropriate for emulating deterministic computer code output. In such situations a value much smaller than the default may work even better (i.e., yield better out-of-sample predictive performance). The default was chosen conservatively. When specifying non-default initial values, a vector of length nrow(XX) can be provided, giving a different initial value for each predictive location
method	specifies the method by which end-start candidates from X are chosen in order to predict at each row XX independently. In brief, ALC ("alc", default) minimizes predictive variance; ALCRAY ("alcray") executes a thrifty search focused on rays emanating from the reference location(s); MSPE ("mspe") augments ALC with extra derivative information to minimize mean-squared prediction error (requires extra computation); NN ("nn") uses nearest neighbor; and ("fish") uses the expected Fisher information - essentially 1/G from Gramacy & Apley (2015) - which is global heuristic, i.e., not localized to each row of XX
methods	for aGP.seq this is a vectorized method argument, containing a list of valid methods to perform in sequence. When methods = FALSE a call to M is invoked instead; see below for more details
Xi.ret	a scalar logical indicating whether or not a matrix of indices into X, describing the chosen sub-design for each of the predictive locations in XX, should be returned on output

close — a non-negative integer `end < close <= nrow(X)` specifying the number of NNs (to each row `XX`) in `X` to consider when searching for the sub-design; `close = 0` specifies all. For `method="alcray"` this specifies the scope used to snap ray-based solutions back to elements of `X`, otherwise there are no restrictions on that search

numrays — a scalar integer indicating the number of rays for each greedy search; only relevant when `method="alcray"`. More rays leads to a more thorough, but more computationally intensive search

laGP — applicable only to the R-version `aGP.R`, this is a function providing the local design implementation to be used. Either `laGP` or `laGP.R` can be provided, or a bespoke routine providing similar outputs

laGPsep — applicable only to the R-version `aGPsep.R`, this is a function providing the local design implementation to be used. Either `laGPsep` or `laGPsep.R` can be provided, or a bespoke routine providing similar outputs

num.gpus — applicable only to the C-version `aGP`, this is a scalar positive integer indicating the number of GPUs available for calculating ALC (see `alcGP`); the package must be compiled for CUDA support; see README/INSTALL in the package source for more details

gpu.threads — applicable only to the C-version `aGP`; this is a scalar positive integer indicating the number of SMP (i.e., CPU) threads queuing ALC jobs on a GPU; the package must be compiled for CUDA support. If `gpu.threads >= 2` then the package must *also* be compiled for OpenMP support; see README/INSTALL in the package source for more details. We recommend setting `gpu.threads` to up to two-times the sum of the number of GPU devices and CPU cores. Only `method = "alc"` is supported when using CUDA. If the sum of `omp.threads` and `gpu.threads` is bigger than the max allowed by your system, then that max is used instead (giving `gpu.threads` preference)

omp.threads — applicable only to the C-version `aGP`; this is a scalar positive integer indicating the number of threads to use for SMP parallel processing; the package must be compiled for OpenMP support; see README/INSTALL in the package source for more details. For most Intel-based machines, we recommend setting `omp.threads` to up to two-times the number of hyperthreaded cores. When using GPUs (`num.gpu > 0`), a good default is `omp.threads=0`, otherwise load balancing could be required; see `nn.gpu` below. If the sum of `omp.threads` and `gpu.threads` is bigger than the max allowed by your system, then that max is used instead (giving `gpu.threads` preference)

nn.gpu — a scalar non-negative integer between `0` and `nrow(XX)` indicating the number of predictive locations utilizing GPU ALC calculations. Note this argument is only useful when both `gpu.threads` and `omp.threads` are non-zero, whereby it acts as a load balancing mechanism

verb — a non-negative integer specifying the verbosity level; `verb = 0` is quiet, and larger values cause more progress information to be printed to the screen. The value `min(0,verb-1)` is provided to each `laGP` call

cls — a cluster object created by `makeCluster` from the **parallel** or **snow** packages

chunks — a scalar integer indicating the number of chunks to break `XX` into for **parallel** evaluation on cluster `cls`. Usually `chunks = length(cl)` is appropriate. How-

	ever specifying more chunks can be useful when the nodes of the cluster are not homogeneous
M	an optional function taking two matrix inputs, of ncol(X)-ncalib and ncalib columns respectively, which is applied in lieu of aGP. This can be useful for calibration where the computer model is cheap, i.e., does not require emulation; more details below
ncalib	an integer between 1 and ncol(X) indicating how to partition X and XX inputs into the two matrices required for M
...	other arguments passed from aGP.sep to aGP

Details

This function invokes laGP with argument Xref = XX[i,] for each i=1:nrow(XX), building up local designs, inferring local correlation parameters, and obtaining predictive locations independently for each location. For more details see laGP.

The function aGP.R is a prototype R-only version for debugging and transparency purposes. It is slower than aGP, which is primarily in C. However it may be useful for developing new programs that involve similar subroutines. Note that aGP.R may provide different output than aGP due to differing library subroutines deployed in R and C.

The function aGP.parallel allows aGP to be called on segments of the XX matrix distributed to a cluster created by **parallel**. It breaks XX into chunks which are sent to aGP workers pointed to by the entries of cls. The aGP.parallel function collects the outputs from each chunk before returning an object almost identical to what would have been returned from a single aGP call. On a single (SMP) node, this represents is a poor-man's version of the OpenMP version described below. On multiple nodes both can be used.

If compiled with OpenMP flags, the independent calls to laGP will be farmed out to threads allowing them to proceed in parallel - obtaining nearly linear speed-ups. At this time aGP.R does not facilitate parallel computation, although a future version may exploit the **parallel** functionality for clustered parallel execution.

If num.gpus > 0 then the ALC part of the independent calculations performed by each thread will be offloaded to a GPU. If both gpu.threads >= 1 and omp.threads >= 1, some of the ALC calculations will be done on the GPUs, and some on the CPUs. In our own experimentation we have not found this to lead to large speedups relative to omp.threads = 0 when using GPUs. For more details, see Gramacy, Niemi, & Weiss (2014).

The aGP.sep function is provided primarily for use in calibration exercises, see Gramacy, et al. (2015). It automates a sequence of aGP calls, each with a potentially different method, successively feeding the previous estimate of local lengthscale (d) in as an initial set of values for the next call. It also allows the use of aGP to be bypassed, feeding the inputs into a user-supplied M function instead. This feature is enabled when methods = FALSE. The M function takes two matrices (same number of rows) as inputs, where the first ncol(X) - ncalib columns represent "field data" inputs shared by the physical and computer model (in the calibration context), and the remaining ncalib are the extra tuning or calibration parameters required to evalue the computer model. For examples illustrating aGP.seq please see the documentation file for discrep.est and demo("calib")

Value

The output is a list with the following components.

mean	a vector of predictive means of length nrow(XX)
var	a vector of predictive variances of length nrow(Xref)
llik	a vector indicating the log likelihood/posterior probability of the data/parameter(s) under the chosen sub-design for each predictive location in XX; provided up to an additive constant
time	a scalar giving the passage of wall-clock time elapsed for (substantive parts of) the calculation
method	a copy of the method argument
d	a full-list version of the d argument, possibly completed by darg
g	a full-list version of the g argument, possibly completed by garg
mle	if d$mle and/or g$mle are TRUE, then mle is a data.frame containing the values found for these parameters, and the number of required iterations, for each predictive location in XX
Xi	when Xi.ret = TRUE, this field contains a matrix of indices of length end into X indicating the sub-design chosen for each predictive location in XX
close	a copy of the input argument

The aGP.seq function only returns the output from the final aGP call. When methods = FALSE and M is supplied, the returned object is a data frame with a mean column indicating the output of the computer model run, and a var column, which at this time is zero

Note

aGPsep provides the same functionality as aGP but deploys a separable covariance function. Criteria (methods) EFI and MSPE are not supported by aGPsep at this time.

Note that using method="NN" gives the same result as specifying start=end, however at some extra computational expense.

At this time, this function provides no facility to find local designs for the subset of predictive locations XX jointly, i.e., providing a matrix Xref to laGP. See laGP for more details/support for this alternative.

The use of OpenMP threads with aGPsep is not as efficient as with aGP when calculating MLEs with respect to the lengthscale (i.e., d=list(mle=TRUE, ...)). The reason is that the lbfgsb C entry point uses static variables, and is therefore not thread safe. To circumvent this problem, an OpenMP critical pragma is used, which can create a small bottle neck

Author(s)

Robert B. Gramacy <rbg@vt.edu>

References

R.B. Gramacy (2016). **laGP***: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in* R., Journal of Statistical Software, 72(1), 1-46; or see vignette("laGP")

R.B. Gramacy and D.W. Apley (2015). *Local Gaussian process approximation for large computer experiments.* Journal of Computational and Graphical Statistics, 24(2), pp. 561-678; preprint on arXiv:1303.0383; http://arxiv.org/abs/1303.0383

R.B. Gramacy, J. Niemi, R.M. Weiss (2014). *Massively parallel approximate Gaussian process regression.* SIAM/ASA Journal on Uncertainty Quantification, 2(1), pp. 568-584; preprint on arXiv:1310.5182; http://arxiv.org/abs/1310.5182

R.B. Gramacy and B. Haaland (2016). *Speeding up neighborhood search in local Gaussian process prediction.* Technometrics, 58(3), pp. 294-303; preprint on arXiv:1409.0074 http://arxiv.org/abs/1409.0074

See Also

vignette("laGP"), laGP, alcGP, mspeGP, alcrayGP, makeCluster, clusterApply

Examples

```
## first, a "computer experiment"

## Simple 2-d test function used in Gramacy & Apley (2014);
## thanks to Lee, Gramacy, Taddy, and others who have used it before
f2d <- function(x, y=NULL)
  {
    if(is.null(y)){
      if(!is.matrix(x) && !is.data.frame(x)) x <- matrix(x, ncol=2)
      y <- x[,2]; x <- x[,1]
    }
    g <- function(z)
      return(exp(-(z-1)^2) + exp(-0.8*(z+1)^2) - 0.05*sin(8*(z+0.1)))
    z <- -g(x)*g(y)
  }

## build up a design with N=~40K locations
x <- seq(-2, 2, by=0.02)
X <- expand.grid(x, x)
Z <- f2d(X)

## predictive grid with NN=400 locations,
## change NN to 10K (length=100) to mimic setup in Gramacy & Apley (2014)
## the low NN set here is for fast CRAN checks
xx <- seq(-1.975, 1.975, length=10)
XX <- expand.grid(xx, xx)
ZZ <- f2d(XX)

## get the predictive equations, first based on Nearest Neighbor
out <- aGP(X, Z, XX, method="nn", verb=0)
## RMSE
sqrt(mean((out$mean - ZZ)^2))

## Not run:
## refine with ALC
out2 <- aGP(X, Z, XX, method="alc", d=out$mle$d)
## RMSE
sqrt(mean((out2$mean - ZZ)^2))

## visualize the results
```

```
par(mfrow=c(1,3))
image(xx, xx, matrix(out2$mean, nrow=length(xx)), col=heat.colors(128),
      xlab="x1", ylab="x2", main="predictive mean")
image(xx, xx, matrix(out2$mean-ZZ, nrow=length(xx)), col=heat.colors(128),
      xlab="x1", ylab="x2", main="bias")
image(xx, xx, matrix(sqrt(out2$var), nrow=length(xx)), col=heat.colors(128),
      xlab="x1", ylab="x2", main="sd")

## refine with MSPE
out3 <- aGP(X, Z, XX, method="mspe", d=out2$mle$d)
## RMSE
sqrt(mean((out3$mean - ZZ)^2))

## End(Not run)

## version with ALC-ray which is much faster than the ones not
## run above
out2r <- aGP(X, Z, XX, method="alcray", d=out$mle$d, verb=0)
sqrt(mean((out2r$mean - ZZ)^2))

## a simple example with estimated nugget
library(MASS)

## motorcycle data and predictive locations
X <- matrix(mcycle[,1], ncol=1)
Z <- mcycle[,2]
XX <- matrix(seq(min(X), max(X), length=100), ncol=1)

## first stage
out <- aGP(X=X, Z=Z, XX=XX, end=30, g=list(mle=TRUE), verb=0)

## plot smoothed versions of the estimated parameters
par(mfrow=c(2,1))
df <- data.frame(y=log(out$mle$d), XX)
lo <- loess(y~., data=df, span=0.25)
plot(XX, log(out$mle$d), type="l")
lines(XX, lo$fitted, col=2)
dfnug <- data.frame(y=log(out$mle$g), XX)
lonug <- loess(y~., data=dfnug, span=0.25)
plot(XX, log(out$mle$g), type="l")
lines(XX, lonug$fitted, col=2)

## second stage design
out2 <- aGP(X=X, Z=Z, XX=XX, end=30, verb=0,
d=list(start=exp(lo$fitted), mle=FALSE),
g=list(start=exp(lonug$fitted)))

## plot the estimated surface
par(mfrow=c(1,1))
plot(X,Z)
df <- 20
s2 <- out2$var*(df-2)/df
q1 <- qt(0.05, df)*sqrt(s2) + out2$mean
```

```
q2 <- qt(0.95, df)*sqrt(s2) + out2$mean
lines(XX, out2$mean)
lines(XX, q1, col=1, lty=2)
lines(XX, q2, col=1, lty=2)

## compare to the single-GP result provided in the mleGP documentation
```

---

alcGP *Improvement statistics for sequential or local design*

Description

Calculate the active learning Cohn (ALC) statistic, mean-squared predictive error (MSPE) or expected Fisher information (fish) for a Gaussian process (GP) predictor relative to a set of reference locations, towards sequential design or local search for Gaussian process regression

Usage

```
alcGP(gpi, Xcand, Xref = Xcand, parallel = c("none", "omp", "gpu"),
      verb = 0)
alcGPsep(gpsepi, Xcand, Xref = Xcand, parallel = c("none", "omp", "gpu"),
      verb = 0)
alcrayGP(gpi, Xref, Xstart, Xend, verb = 0)
alcrayGPsep(gpsepi, Xref, Xstart, Xend, verb = 0)
ieciGP(gpi, Xcand, fmin, Xref = Xcand, w = NULL, nonug = FALSE, verb = 0)
ieciGPsep(gpsepi, Xcand, fmin, Xref = Xcand, w = NULL, nonug = FALSE, verb = 0)
mspeGP(gpi, Xcand, Xref = Xcand, fi = TRUE, verb = 0)
fishGP(gpi, Xcand)
alcoptGP(gpi, Xref, start, lower, upper, maxit = 100, verb = 0)
alcoptGPsep(gpsepi, Xref, start, lower, upper, maxit = 100, verb = 0)
dalcGP(gpi, Xcand, Xref = Xcand, verb = 0)
dalcGPsep(gpsepi, Xcand, Xref = Xcand, verb = 0)
```

Arguments

gpi	a C-side GP object identifier (positive integer); e.g., as returned by newGP
gpsepi	a C-side separable GP object identifier (positive integer); e.g., as returned by newGPsep
Xcand	a matrix or data.frame containing a design of candidate predictive locations at which the ALC (or other) criteria is (are) evaluated. In the context of laGP, these are the possible locations for adding into the current local design
fmin	for ieci* only: a scalar value indicating the value of the best minimum found so far. This is usually set to the minimum of the Z-values stored in the gpi or gpsepi reference (for deterministic/low nugget settings), or otherwise the predicted mean value at the X locations

Xref	a matrix or data.frame containing a design of reference locations for ALC or MSPE. I.e., these are the locations at which the reduction in variance, or mean squared predictive error, are calculated. In the context of laGP, this is the single location, or set of reference locations, around which a local design (for accurate prediction) is sought. For alcrayGP and alcrayGPsep the matrix may only have one row, i.e., one reference location
parallel	a switch indicating if any parallel calculation of the criteria (method) is desired. For parallel = "omp", the package must be compiled with OpenMP flags; for parallel = "gpu", the package must be compiled with CUDA flags (only the ALC criteria is supported on the GPU); see README/INSTALL in the package source for more details
Xstart	a 1-by-ncol(Xref) starting location for a search along a ray between Xstart and Xend
Xend	a 1-by-ncol(Xref) ending location for a search along a ray between Xstart and Xend
fi	a scalar logical indicating if the expected Fisher information portion of the expression (MSPE is essentially ALC + c(x)*EFI) should be calculated (TRUE) or set to zero (FALSE). This flag is mostly for error checking against the other functions, alcGP and fishGP, since the constituent parts are separately available via those functions
w	weights on the reference locations Xref for IECI calculations; IECI, which stands for Integrated Expected Conditional Improvement, is not fully documented at this time. See Gramacy & Lee (2010) for more details.
nonug	a scalar logical indicating if a (nonzero) nugget should be used in the predictive equations behind IECI calculations; this allows the user to toggle improvement via predictive mean uncertainty versus full predictive uncertainty. The latter (default nonug = FALSE) is the standard approach, but the former may work better (citation forthcoming)
verb	a non-negative integer specifying the verbosity level; verb = 0 is quiet, and larger values cause more progress information to be printed to the screen
start	initial values to the derivative-based search via "L-BFGS-B" within alcoptGP and alcoptGPsep; a nearest neighbor often represents a sensible initialization
lower, upper	bounds on the derivative-based search via "L-BFGS-B" within alcoptGP and alcoptGPsep
maxit	the maximum number of iterations (default maxit=100) in "L-BFGS-B" search within alcoptGP and alcoptGPsep

Details

The best way to see how these functions are used in the context of local approximation is to inspect the code in the laGP.R function.

Otherwise they are pretty self-explanatory. They evaluate the ALC, MSPE, and EFI quantities outlined in Gramacy & Apley (2015). ALC is originally due to Seo, et al. (2000). The ray-based search is described by Gramacy & Haaland (2015).

MSPE and EFI calculations are not supported for separable GP models, i.e., there are no mspeGPsep or fishGPsep functions.

alcrayGP and alcrayGPsep allow only one reference location (nrow(Xref) = 1). alcoptGP and alcoptGPsep allow multiple reference locations. These optimize a continuous ALC analog in its natural logarithm using the starting locations, bounding boxes and (stored) GP provided by gpi or gpisep, and finally snaps the solution back to the candidate grid. For details, see Sun, et al. (2017).

Note that ieciGP and ieciGPsep, which are for optimization via integrated expected conditional improvement (Gramacy & Lee, 2011) are "alpha" functionality and are not fully documented at this time.

Value

Except for alcoptGP, alcoptGPsep, dalcGP, and dalcGPsep, a vector of length nrow(Xcand) is returned filled with values corresponding to the desired statistic

par	the best set of parameters found
its	a two-element integer vector giving the number of calls to the object function and the gradient respectively.
msg	a character string giving any additional information returned by the optimizer, or NULL
convergence	An integer code. 0 indicates successful completion. For the other error codes, see the documentation for optim
alcs	reduced predictive variance averaged over the reference locations
dalcs	the derivative of alcs with respect to the new location

Author(s)

Robert B. Gramacy <rbg@vt.edu> and Furong Sun <furongs@vt.edu>

References

F. Sun, R.B. Gramacy, B. Haaland, E. Lawrence, and A. Walker (2017) *Emulating satellite drag from large simulation experiments.*; preprint on arXiv:1712.00182. http://arxiv.org/abs/1712.00182

R.B. Gramacy (2016). **laGP***: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in* R., Journal of Statistical Software, 72(1), 1-46; or see vignette("laGP")

R.B. Gramacy and B. Haaland (2016). *Speeding up neighborhood search in local Gaussian process prediction.* Technometrics, 58(3), pp. 294-303; preprint on arXiv:1409.0074 http://arxiv.org/abs/1409.0074

R.B. Gramacy and D.W. Apley (2015). *Local Gaussian process approximation for large computer experiments.* Journal of Computational and Graphical Statistics, 24(2), pp. 561-678; preprint on arXiv:1303.0383; http://arxiv.org/abs/1303.0383

R.B. Gramacy, J. Niemi, R.M. Weiss (2014). *Massively parallel approximate Gaussian process regression.* SIAM/ASA Journal on Uncertainty Quantification, 2(1), pp. 568-584; preprint on arXiv:1310.5182; http://arxiv.org/abs/1310.5182

R.B. Gramacy, H.K.H. Lee (2011). *Optimization under unknown constraints.*, Valencia discussion paper, in Bayesian Statistics 9. Oxford University Press; preprint on arXiv:1004.4027; http://arxiv.org/abs/1004.4027

Seo, S., Wallat, M., Graepel, T., Obermayer, K. (2000). *Gaussian Process Regression: Active Data Selection and Test Point Rejection.* In Proceedings of the International Joint Conference on Neural Networks, vol. III, 241-246. IEEE

See Also

laGP, aGP, predGP

Examples

```
## this follows the example in predGP, but only evaluates
## information statistics documented here

## Simple 2-d test function used in Gramacy & Apley (2015);
## thanks to Lee, Gramacy, Taddy, and others who have used it before
f2d <- function(x, y=NULL)
  {
    if(is.null(y)) {
      if(!is.matrix(x) && !is.data.frame(x)) x <- matrix(x, ncol=2)
      y <- x[,2]; x <- x[,1]
    }
    g <- function(z)
      return(exp(-(z-1)^2) + exp(-0.8*(z+1)^2) - 0.05*sin(8*(z+0.1)))
    z <- -g(x)*g(y)
  }

## design with N=441
x <- seq(-2, 2, length=11)
X <- expand.grid(x, x)
Z <- f2d(X)

## fit a GP
gpi <- newGP(X, Z, d=0.35, g=1/1000, dK=TRUE)

## predictive grid with NN=400
xx <- seq(-1.9, 1.9, length=20)
XX <- expand.grid(xx, xx)

## predict
alc <- alcGP(gpi, XX)
mspe <- mspeGP(gpi, XX)
fish <- fishGP(gpi, XX)

## visualize the result
par(mfrow=c(1,3))
image(xx, xx, matrix(sqrt(alc), nrow=length(xx)), col=heat.colors(128),
      xlab="x1", ylab="x2", main="sqrt ALC")
image(xx, xx, matrix(sqrt(mspe), nrow=length(xx)), col=heat.colors(128),
      xlab="x1", ylab="x2", main="sqrt MSPE")
image(xx, xx, matrix(log(fish), nrow=length(xx)), col=heat.colors(128),
      xlab="x1", ylab="x2", main="log fish")
```

```
## clean up
deleteGP(gpi)


##
## Illustrating some of the other functions in a sequential design context,
## using X and XX above
##

## new, much bigger design
x <- seq(-2, 2, by=0.02)
X <- expand.grid(x, x)
Z <- f2d(X)

## first build a local design of size 25, see laGP documentation
out <- laGP.R(XX, start=6, end=25, X, Z, method="alc", close=10000)

## extract that design and fit GP
XC <- X[out$Xi,] ## inputs
ZC <- Z[out$Xi]  ## outputs
gpi <- newGP(XC, ZC, d=out$mle$d, g=out$g$start)

## calculate the ideal "next" location via continuous ALC optimization
alco <- alcoptGP(gpi=gpi, Xref=XX, start=c(0,0), lower=range(x)[1], upper=range(x)[2])

## alco$par is the "new" location; calculate distances between candidates (remaining
## unchosen X locations) and this solution
Xcan <- X[-out$Xi,]
D <- distance(Xcan, matrix(alco$par, ncol=ncol(Xcan)))

## snap the new location back to the candidate set
lab <- which.min(D)
xnew <- Xcan[lab,]
## add xnew to the local design, remove it from Xcan, and repeat

## evaluate the derivative at this new location
dalc <- dalcGP(gpi=gpi, Xcand=matrix(xnew, nrow=1), Xref=XX)

## clean up
deleteGP(gpi)
```

`blhs` *Bootstrapped block Latin hypercube subsampling*

Description

Provides bootstrapped block Latin hypercube subsampling under a given data set to aid in consistent estimation of a global separable lengthscale parameter

Usage

```
blhs(y, X, m)
blhs.loop(y, X, m, K, da, g = 1e-3, maxit = 100, verb = 0, plot.it = FALSE)
```

Arguments

y	a vector of responses/dependent values with length(y) = nrow(X)
X	a matrix or data.frame containing the full (large) design matrix of input locations
m	a positive scalar integer giving the number of divisions on each coordinate of input space defining the block structure
K	a positive scalar integer specifying the number of Bootstrap replicates desired
da	a lengthscale prior, say as generated by darg
g	a positive scalar giving the fixed nugget value of the nugget parameter; by default g = 1e-3
maxit	a positive scalar integer giving the maximum number of iterations for MLE calculations via "L-BFGS-B"; see mleGPsep for more details
verb	a non-negative integer specifying the verbosity level; verb = 0 (by default) is quiet, and larger values cause more progress information to be printed to the screen
plot.it	plot.it = FALSE by default; if plot.it = TRUE, then each of the K lengthscale estimates from bootstrap iterations will be shown via boxplot

Details

Bootstrapped block Latin hypercube subsampling (BLHS) yields a global lengthscale estimator which is asymptotically consistent with the MLE calculated on the full data set. However, since it works on data subsets, it comes at a much reduced computational cost. Intuitively, the BLHS guarantees a good mix of short and long pairwise distances. A single bootstrap LH subsample may be obtained by dividing each dimension of the input space equally into m intervals, yielding m^d mutually exclusive hypercubes. It is easy to show that the average number of observations in each hypercube is Nm^{-d} if there are N samples in the original design. From each of these hypercubes, m blocks are randomly selected following the LH paradigm, i.e., so that only one interval is chosen from each of the m segments. The average number of observations in the subsample, combining the m randomly selected blocks, is Nm^{-d+1}.

Ensuring a subsample size of at least one requires having $m \leq N^{\frac{1}{d-1}}$, thereby linking the parameter m to computational effort. Smaller m is preferred so long as GP inference on data of that size remains tractable. Since the blocks follow an LH structure, the resulting sub-design inherits the usual LHS properties, e.g., retaining marginal properties like univariate stratification modulo features present in the original, large N, design.

For more details, see Liu (2014), Zhao, et al. (2017) and Sun, et al. (2017).

blhs returns the subsampled input space and the corresponding responses.

blhs.loop returns the median of the K lengthscale maximum likelihood estimates, the subsampled data size to which that corresponds, and the subsampled data, including the input space and the responses, from the bootstrap iterations

Value

`blhs` returns

`xs`	the subsampled input space
`ys`	the subsampled responses, `length(ys) = nrow(xs)`

`blhs.loop` returns

`that`	the lengthscale estimate (median), `length(that) = ncol(X)`
`ly`	the subsampled data size (median)
`xm`	the subsampled input space (median)
`ym`	the subsampled responses (median)

Note

The global input space should be relatively homogeneous. A space-filling design (input) `X` is ideal, but not required

Author(s)

Robert B. Gramacy <rbg@vt.edu> and Furong Sun <furongs@vt.edu>

References

F. Sun, R.B. Gramacy, B. Haaland, E. Lawrence, and A. Walker (2017) *Emulating satellite drag from large simulation experiments*; preprint on arXiv:1712.00182. http://arxiv.org/abs/1712.00182

Y. Zhao, Y. Hung, and Y. Amemiya (2017). *Efficient Gaussian Process Modeling using Experimental Design-Based Subagging.* To appear in Statistica Sinica;

Yufan Liu (2014) *Recent Advances in Computer Experiment Modeling*. Ph.D. Thesis at Rutgers, The State University of New Jersey. http://dx.doi.org/doi:10.7282/T38G8J1H

Examples

```
# input space based on latin-hypercube sampling (not required)
library(lhs)
## two dimensional example with N=216 sized sample
X <- randomLHS(216, 2)
# pseudo responses, not important for visualizing design
Y <- runif(216)

## BLHS sample with m=6 divisions in each coordinate
sub <- blhs(y=Y, X=X, m=6)
Xsub <- sub$xs # the bootstrapped subsample

# visualization
plot(X, xaxt="n", yaxt="n", xlim=c(0,1), ylim=c(0,1), xlab="factor 1",
  ylab="factor 2", col="cyan", main="BLHS")
b <- seq(0, 1, by=1/6)
```

```
abline(h=b, v=b, col="black", lty=2)
axis(1, at=seq (0, 1, by=1/6), cex.axis=0.8,
  labels=expression(0, 1/6, 2/6, 3/6, 4/6, 5/6, 1))
axis(2, at=seq (0, 1, by=1/6), cex.axis=0.8,
  labels=expression(0, 1/6, 2/6, 3/6, 4/6, 5/6, 1), las=1)
points(Xsub, col="red", pch=19, cex=1.25)

## Comparing global lengthscale MLE based on BLHS and random subsampling
## Not run:
  # famous borehole function
  borehole <- function(x){
    rw <- x[1] * (0.15 - 0.05) + 0.05
    r <-  x[2] * (50000 - 100) + 100
    Tu <- x[3] * (115600 - 63070) + 63070
    Tl <- x[5] * (116 - 63.1) + 63.1
    Hu <- x[4] * (1110 - 990) + 990
    Hl <- x[6] * (820 - 700) + 700
    L <-  x[7] * (1680 - 1120) + 1120
    Kw <- x[8] * (12045 - 9855) + 9855
    m1 <- 2 * pi * Tu * (Hu - Hl)
    m2 <- log(r / rw)
    m3 <- 1 + 2*L*Tu/(m2*rw^2*Kw) + Tu/Tl
    return(m1/m2/m3)
  }

  N <- 100000                   # number of observations
  xt <- randomLHS(N, 8)         # input space
  yt <- apply(xt, 1, borehole)  # response
  colnames(xt) <- c("rw", "r", "Tu", "Tl", "Hu", "Hl", "L", "Kw")

  ## prior on the GP lengthscale parameter
  da <- darg(list(mle=TRUE, max=100), xt)

  ## make space for two sets of boxplots
  par(mfrow=c(1,2))

  # BLHS calculating with visualization of the K MLE lengthscale estimates
  K <- 10  # number of Bootstrap samples; Sun, et al (2017) uses K <- 31
  sub_blhs <- blhs.loop(y=yt, X=xt, K=K, m=2, da=da, maxit=200, plot.it=TRUE)

  # a random subsampling analog for comparison
  sn <- sub_blhs$ly # extract a size that is consistent with the BLHS
  that.rand <- matrix(NA, ncol=8, nrow=K)
  for(i in 1:K){
    sub <- sample(1:nrow(xt), sn)
    gpsepi <- newGPsep(xt[sub,], yt[sub], d=da$start, g=1e-3, dK=TRUE)
    mle <- mleGPsep(gpsepi, tmin=da$min, tmax=10*da$max, ab=da$ab, maxit=200)
    deleteGPsep(gpsepi)
    that.rand[i,] <- mle$d
  }

  ## put random boxplots next to BLHS ones
  boxplot(that.rand, xlab="input", ylab="theta-hat", col=2,
```

```
      main="random subsampling")

## End(Not run)
```

---

darg *Generate Priors for GP correlation*

Description

Generate empirical Bayes regularization (priors) and choose initial values and ranges for (isotropic) lengthscale and nugget parameters to a Gaussian correlation function for a GP regression model

Usage

```
darg(d, X, samp.size = 1000)
garg(g, y)
```

Arguments

d	can be NULL, or a scalar indicating an initial value or a partial list whose format matches the one described in the Value section below
g	can be NULL, or a scalar indicating an initial value or a partial list whose format matches the one described in the Value section below
X	a matrix or data.frame containing the full (large) design matrix of input locations
y	a vector of responses/dependent values
samp.size	a scalar integer indicating a subset size of X to use for calculations; this is important for very large X matrices since the calculations are quadratic in nrow(X)

Details

These functions use aspects of the data, either X or y, to form weakly informative default priors and choose initial values for a lengthscale and nugget parameter. This is useful since the likelihood can sometimes be very flat, and even with proper priors inference can be sensitive to the specification of those priors and any initial search values. The focus here is on avoiding pathologies while otherwise remaining true to the spirit of MLE calculation.

darg output specifies MLE inference (out$mle = TRUE) by default, whereas garg instead fixes the nugget at the starting value, which may be sensible for emulating deterministic computer simulation data; when out$mle = FALSE the calculated range outputs c(outmin, outmax) are set to dummy values that are ignored in other parts of the **laGP** package.

darg calculates a Gaussian distance matrix between all pairs of X rows, or a subsample of rows of size samp.size. From those distances it chooses the range and start values from the range of (non-zero) distances and the 0.1 quantile, respectively. The Gamma prior values have a shape of out$a = 3/2 and a rate out$b chosen by the incomplete Gamma inverse function to put 0.95 probability below out$max.

garg is similar except that it works with (y- mean(y))^2 instead of the pairwise distances of darg. The only difference is that the starting value is chosen as the 2.5% quantile.

Value

Both functions return a list containing the following entries. If the input object (d or g) specifies one of the values then that value is copied to the same list entry on output. See the Details section for how these values are calculated

mle	by default, TRUE for darg and FALSE for garg
start	starting value chosen from the quantiles of distance(X) or (y - mean(y))^2
min	minimum value in allowable range for the parameter - for future inference purposes
max	maximum value in allowable range for the parameter - for future inference purposes
ab	shape and rate parameters specifying a Gamma prior for the parameter

Author(s)

Robert B. Gramacy <rbg@vt.edu>

See Also

vignette("laGP"), laGP, aGP, mleGP, distance, llikGP

Examples

```
## motorcycle data
library(MASS)
X <- matrix(mcycle[,1], ncol=1)
Z <- mcycle[,2]

## get darg and garg
darg(NULL, X)
garg(list(mle=TRUE), Z)
```

---

deleteGP *Delete C-side Gaussian Process Objects*

Description

Frees memory allocated by a particular C-side Gaussian process object, or all GP objects currently allocated

Usage

```
deleteGP(gpi)
deleteGPsep(gpsepi)
deleteGPs()
deleteGPseps()
```

Arguments

gpi	a scalar positive integer specifying an allocated isotropic GP object
gpsepi	similar to gpi but indicating a separable GP object

Details

Any function calling newGP or newGPsep will require destruction via these functions or there will be a memory leak

Value

Nothing is returned

Author(s)

Robert B. Gramacy <rbg@vt.edu>

See Also

vignette("laGP"), newGP, predGP, mleGP

Examples

```
## see examples for newGP, predGP, or mleGP
```

---

discrep.est *Estimate Discrepancy in Calibration Model*

Description

Estimates the Gaussian process discrepancy/bias and/or noise term in a modularized calibration of a computer model (emulator) to field data, and returns the log likelihood or posterior probability

Usage

```
discrep.est(X, Y, Yhat, d, g, bias = TRUE, clean = TRUE)
```

Arguments

X	a matrix or data.frame containing a design matrix of input locations for field data sites. Any columns of X without at least three unique input settings are dropped in a pre-processing step
Y	a vector of values with length(Y) = ncol(X) containing the response from field data observations at X. A Y-vector with length(Y) = k*ncol(X), for positive integer k, can be supplied in which case the multiple code Y-values will be treated as replicates at the X-values

Yhat — a vector with length(Yhat) = length(Y) containing predictions at X from an emulator of a computer simulation

d — a prior or initial setting for the (single/isotropic) lengthscale parameter in a Gaussian correlation function; a (default) NULL value triggers a sensible regularization (prior) and initial setting to be generated via darg; a scalar specifies an initial value, causing darg to only generate the prior; otherwise, a list or partial list matching the output of darg can be used to specify a custom prior. In the case of a partial list, the only the missing entries will be generated. Note that a default/generated list specifies MLE/MAP inference for this parameter. When specifying initial values, a vector of length nrow(XX) can be provided, giving a different initial value for each predictive location.

g — a prior or initial setting for the nugget parameter; a NULL value causes a sensible regularization (prior) and initial setting to be generated via garg; a scalar (default g = 1/1000) specifies an initial value, causing garg to only generate the prior; otherwise, a list or partial list matching the output of garg can be used to specify a custom prior. In the case of a partial list, only the missing entries will be generated. Note that a default/generated list specifies *no* inference for this parameter; i.e., it is fixed at its starting value, which may be appropriate for emulating deterministic computer code output

bias — a scalar logical indicating if a (isotropic) GP discrepancy should be estimated (TRUE) or a Gaussian noise term only (FALSE)

clean — a scalar logical indicating if the C-side GP object should be freed before returning.

Details

Estimates an isotropic Gaussian correlation Gaussian process (GP) discrepancy term for the difference between a computer model output (Yhat) and field data observations (Y) at locations X. The computer model predictions would typically come from a GP emulation from simulation data, possibly via aGP if the computer experiment is large.

This function is used primarily as a subroutine by fcalib which defines an objective function for optimization in order to solve the calibration problem via the method described by Gramacy, et al. (2015), designed for large computer experiments. However, once calibration is performed this function can be useful for making comparisons to other methods. Examples are provided in the fcalib documentation.

When bias=FALSE no discrepancy is estimated; only a zero-mean Gaussian error distribution is assumed

Value

The output object is comprised of the output of jmleGP, applied to a GP object built with responses Y - Yhat. That object is augmented with a log likelihood, in $ll, and with a GP index $gpi when clean=FALSE. When bias = FALSE the output object retains the same form as above, except with dummy zero-values since calling jmleGP is not required

Note

Note that in principle a separable correlation function could be used (e.g, via newGPsep and mleGPsep), however this is not implemented at this time

Author(s)

Robert B. Gramacy <rbg@vt.edu>

References

R.B. Gramacy (2016). **laGP***: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in* R., Journal of Statistical Software, 72(1), 1-46; or see vignette("laGP")

R.B. Gramacy, D. Bingham, JP. Holloway, M.J. Grosskopf, C.C. Kuranz, E. Rutter, M. Trantham, P.R. Drake (2015). *Calibrating a large computer experiment simulating radiative shock hydrodynamics.* Annals of Applied Statistics, 9(3) 1141-1168; preprint on arXiv:1410.3293 http://arxiv.org/abs/1410.3293

F. Liu, M. Bayarri and J. Berger (2009). *Modularization in Bayesian analysis, with emphasis on analysis of computer models.* Bayesian Analysis, 4(1) 119-150.

See Also

vignette("laGP"), jmleGP, newGP, aGP.seq, fcalib

Examples

```
## the example here combines aGP.seq and discrep.est functions;
## it is comprised of snippets from demo("calib"), which contains
## code from the Calibration Section of vignette("laGP")

## Here we generate calibration data using a true calibration
## parameter, u, and then evaluate log posterior probabilities
## and out-of-sample RMSEs for that u value;  the fcalib
## documentation repeats this with a single call to fcalib rather
## than first aGP.seq and then discrep.est

## begin data-generation code identical to aGP.seq, discrep.est, fcalib
## example sections and demo("calib")

## M: computer model test functon used in Goh et al, 2013 (Technometrics)
## an elaboration of one from Bastos and O'Hagan, 2009 (Technometrics)
M <- function(x,u)
  {
    x <- as.matrix(x)
    u <- as.matrix(u)
    out <- (1-exp(-1/(2*x[,2])))
    out <- out * (1000*u[,1]*x[,1]^3+1900*x[,1]^2+2092*x[,1]+60)
    out <- out / (100*u[,2]*x[,1]^3+500*x[,1]^2+4*x[,1]+20)
    return(out)
  }
```

```
## bias: discrepancy function from Goh et al, 2013
bias <- function(x)
  {
    x<-as.matrix(x)
    out<- 2*(10*x[,1]^2+4*x[,2]^2) / (50*x[,1]*x[,2]+10)
    return(out)
  }

## beta.prior: marginal beta prior for u,
## defaults to a mode at 1/2 in hypercube
beta.prior <- function(u, a=2, b=2, log=TRUE)
{
  if(length(a) == 1) a <- rep(a, length(u))
  else if(length(a) != length(u)) stop("length(a) must be 1 or length(u)")
  if(length(b) == 1) b <- rep(b, length(u))
  else if(length(b) != length(u)) stop("length(b) must be 1 or length(u)")
  if(log) return(sum(dbeta(u, a, b, log=TRUE)))
  else return(prod(dbeta(u, a, b, log=FALSE)))
}

## tgp for LHS sampling
library(tgp)
rect <- matrix(rep(0:1, 4), ncol=2, byrow=2)

## training and testing inputs
ny <- 50; nny <- 1000
X <- lhs(ny, rect[1:2,])    ## computer model train
XX <- lhs(nny, rect[1:2,],) ## test

## true (but unknown) setting of the calibration parameter
## for the computer model
u <- c(0.2, 0.1)
Zu <- M(X, matrix(u, nrow=1))
ZZu <- M(XX, matrix(u, nrow=1))

## field data response, biased and replicated
sd <- 0.5
## Y <- computer output + bias + noise
reps <- 2 ## example from paper uses reps <- 10
Y <- rep(Zu,reps) + rep(bias(X),reps) + rnorm(reps*length(Zu), sd=sd)
YYtrue <- ZZu + bias(XX)
## variations: remove the bias or change 2 to 1 to remove replicates

## computer model design
nz <- 10000
XT <- lhs(nz, rect)
nth <- 1 ## number of threads to use in emulation, demo uses 8

## augment with physical model design points
## with various u settings
XT2 <- matrix(NA, nrow=10*ny, ncol=4)
for(i in 1:10) {
  I <- ((i-1)*ny+1):(ny*i)
```

```
  XT2[I,1:2] <- X
}
XT2[,3:4] <- lhs(10*ny, rect[3:4,])
XT <- rbind(XT, XT2)

## evaluate the computer model
Z <- M(XT[,1:2], XT[,3:4])

## flag indicating if estimating bias/discrepancy or not
bias.est <- TRUE
## two passes of ALC with MLE calculations for aGP.seq
methods <- rep("alcray", 2) ## demo uses rep("alc", 2)

## set up priors
da <- d <- darg(NULL, XT)
g <- garg(list(mle=TRUE), Y)

## end identical data generation code

## now calculate log posterior and do out-of-sample RMSE calculation
## for true calibration parameter value (u).  You could repeat
## this for an estimate value from demo("calib"), for example
## u.hat <- c(0.8236673, 0.1406989)

## first log posterior

## emulate at field-data locations Xu
Xu <- cbind(X, matrix(rep(u, ny), ncol=2, byrow=TRUE))
ehat.u <- aGP.seq(XT, Z, Xu, da, methods, ncalib=2, omp.threads=nth, verb=0)

## estimate discrepancy from the residual
cmle.u <- discrep.est(X, Y, ehat.u$mean, d, g, bias.est, FALSE)
cmle.u$ll <- cmle.u$ll + beta.prior(u)
print(cmle.u$ll)
## compare to same calculation with u.hat above

## now RMSE
## Not run:
## predictive design with true calibration parameter
XXu <- cbind(XX, matrix(rep(u, nny), ncol=2, byrow=TRUE))

## emulate at predictive design
ehat.oos.u <- aGP.seq(XT, Z, XXu, da, methods, ncalib=2,
  omp.threads=nth, verb=0)

## predict via discrepency
YYm.pred.u <- predGP(cmle.u$gp, XX)

## add in emulation
YY.pred.u <- YYm.pred.u$mean + ehat.oos.u$mean

## calculate RMSE
rmse.u <- sqrt(mean((YY.pred.u - YYtrue)^2))
```

```
print(rmse.u)
## compare to same calculation with u.hat above

## clean up
deleteGP(cmle.u$gp)

## End(Not run)
```

---

distance *Calculate the squared Euclidean distance between pairs of points*

Description

Calculate the squared Euclidean distance between pairs of points and return a distance matrix

Usage

```
distance(X1, X2 = NULL)
```

Arguments

X1	a matrix or data.frame containing real-valued numbers
X2	an optional matrix or data.frame containing real-valued numbers; must have ncol(X2) = ncol(X1)

Details

If X2 = NULL distances between X1 and itself are calculated, resulting in an nrow(X1)-by-nrow(X1) distance matrix. Otherwise the result is nrow(X1)-by-nrow(X2) and contains distances between X1 and X2.

Calling distance(X) is the same as distance(X,X)

Value

The output is a matrix, whose dimensions are described in the Details section above

Author(s)

Robert B. Gramacy <rbg@vt.edu>

See Also

darg

Examples

```
x <- seq(-2, 2, length=11)
X <- as.matrix(expand.grid(x, x))
## predictive grid with NN=400
xx <- seq(-1.9, 1.9, length=20)
XX <- as.matrix(expand.grid(xx, xx))

D <- distance(X)
DD <- distance(X, XX)
```

`fcalib` *Objective function for performing large scale computer model calibration via optimization*

Description

Defines an objective function for performing blackbox optimization towards solving a modularized calibration of large computer model simulation to field data

Usage

```
fcalib(u, XU, Z, X, Y, da, d, g, uprior = NULL, methods = rep("alc", 2),
  M = NULL, bias = TRUE, omp.threads = 1, save.global = FALSE, verb = 1)
```

Arguments

`u`	a vector of length `ncol(XU) - ncol(X)` containing a setting of the calibration parameter
`XU`	a `matrix` or `data.frame` containing the full (large) design matrix of input locations to a computer simulator whose final `ncol(XU) - ncol(X)` columns contain settings of a calibration or tuning parameter like `u`
`Z`	a vector of responses/dependent values with `length(Z) = ncol(XU)` of computer model outputs at `XU`
`X`	a `matrix` or `data.frame` containing the full (large) design matrix of input locations
`Y`	a vector of values with `length(Y) = ncol(X)` containing the response from field data observations at `X`. A `Y`-vector with `length(Y) = k*ncol(X)`, for positive integer `k`, can be supplied in which case the multiple `Y`-values will be treated as replicates at the `X`-values
`da`	for emulating `Z` at `XU`: a prior or initial setting for the (single/isotropic) length-scale parameter in a Gaussian correlation function; a (default) `NULL` value triggers a sensible regularization (prior) and initial setting to be generated via `darg`; a scalar specifies an initial value, causing `darg` to only generate the prior; otherwise, a list or partial list matching the output of `darg` can be used to specify a custom prior. In the case of a partial list, the only the missing entries will be generated. Note that a default/generated list specifies MLE/MAP inference for

this parameter. When specifying initial values, a vector of length `nrow(XX)` can be provided, giving a different initial value for each predictive location

`d` for the discrepancy between emulations `Yhat` at `X`, based on `Z` at `XU`, and the oputs `Y` observed at `X`. Otherwise, same description as `da` above

`g` for the nugget in the GP model for the discrepancy between emulation `Yhat` at `X`, based on `Z` at `XU`, and the outputs `Y` observed at `X`: a prior or initial setting for the nugget parameter; a `NULL` value causes a sensible regularization (prior) and initial setting to be generated via `garg`; a scalar (default `g = 1/1000`) specifies an initial value, causing `garg` to only generate the prior; otherwise, a list or partial list matching the output of `garg` can be used to specify a custom prior. In the case of a partial list, only the missing entries will be generated. Note that a default/generated list specifies *no* inference for this parameter; i.e., it is fixed at its starting value, which may be appropriate for emulating deterministic computer code output. At this time, estimating a nugget for the computer model emulator is not supported by `fcalib`

`uprior` an optional function taking `u` arguments which returns a log prior density value for the calibration parameter.

`methods` a sequence of local search methods to be deployed when emulating `Z` at `XU` via `aGP`; see `aGP.seq` for more details; provide `methods = FALSE` to use the computer model `M` directly

`M` a computer model "simulation" function taking two matrices as inputs, to be used in lieu of emulation; see `aGP.seq` for mode details

`bias` a scalar logical indicating whether a GP discrepancy or bias term should be estimated via `discrep.est`, as opposed to only a Gaussian (zero-mean) variance; see `discrep.est` for more details

`omp.threads` a scalar positive integer indicating the number of threads to use for SMP parallel processing; see `aGP` for more details

`save.global` an environment, e.g., `.GlobalEnv` if each evaluation of `fcalib`, say as called by a wrapper or optimization routine, should be saved. The variable used in that environment will be `fcalib.save`. Otherwise `save.global = FALSE` will skip saving the information

`verb` a non-negative integer specifying the verbosity level; `verb = 0` is quiet, whereas a larger value causes each evaluation to be printed to the screen

Details

Gramacy, et al. (2015) defined an objective function which, when optimized, returns a setting of calibration parameters under a setup akin to the modularized calibration method of Liu, et al., (2009). The `fcalib` function returns a log density (likelihood or posterior probability) value obtained by performing emulation at a set of inputs `X` augmented with a value of the calibration parameter, `u`. The emulator is trained on `XU` and `Z`, presumed to be very large relative to the size of the field data set `X` and `Y`, necessitating the use of approximate methods like `aGP`, via `aGP.seq`. The emulated values, call them `Yhat` are fed along with `X` and `Y` into the `discrep.est` function, whose likelihood or posterior calculation serves as a measure of merit for the value `u`.

The `fcalib` function is deterministic but, as Gramacy, et al. (2015) described, can result is a rugged objective surface for optimizing, meaning that conventional methods, like those in `optim`

are unlikely to work well. They instead recommend using a blackbox derivative-free method, like NOMAD (Le Digabel, 2011). In our example below we use the implementation in the **crs** package, which provides an R wrapper around the underlying C library.

Note that while `fcalib` automates a call first to `aGP.seq` and then to `discrep.est`, it does not return enough information to complete, say, an out-of-sample prediction exercise like the one demonstrated in the `discrep.est` documentation. Therefore, after `fcalib` is used in an optimization to find the best setting of the calibration parameter, `u`, those functions must then be used in post-processing to complete a prediction exercise. See `demo("calib")` or `vignette("laGP")` for more details

Value

Returns a scalar measuring the negative log likelihood or posterior density of the calibration parameter `u` given the other inputs, for the purpose of optimization over `u`

Note

Note that in principle a separable correlation function could be used (e.g, via `newGPsep` and `mleGPsep`), however this is not implemented at this time

Author(s)

Robert B. Gramacy <rbg@vt.edu>

References

R.B. Gramacy (2016). **laGP***: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in* R., Journal of Statistical Software, 72(1), 1-46; or see `vignette("laGP")`

R.B. Gramacy, D. Bingham, JP. Holloway, M.J. Grosskopf, C.C. Kuranz, E. Rutter, M. Trantham, and P.R. Drake (2015). *Calibrating a large computer experiment simulating radiative shock hydrodynamics.* Annals of Applied Statistics, 9(3) 1141-1168; preprint on arXiv:1410.3293 http://arxiv.org/abs/1410.3293

F. Liu, M. Bayarri, and J. Berger (2009). *Modularization in Bayesian analysis, with emphasis on analysis of computer models.* Bayesian Analysis, 4(1) 119-150.

S. Le Digabel (2011). *Algorithm 909: NOMAD: Nonlinear Optimization with the MADS algorithm.* ACM Transactions on Mathematical Software, 37, 4, 44:1-44:15.

J.S. Racine, Z. and Nie (2012). **crs**: *Categorical regression splines*. R package version 0.15-18.

See Also

`vignette("laGP")`, `jmleGP`, `newGP`, `aGP.seq`, `discrep.est`, `snomadr`

Examples

```
## the example here illustrates how fcalib combines aGP.seq and
## discrep.est functions, duplicating the example in the discrep.est
## documentation file.  It is comprised of snippets from demo("calib"),
## which contains code from the Calibration Section of vignette("laGP")
```

```
## Here we generate calibration data using a true calibration
## parameter, u, and then evaluate log posterior probabilities;
## the discrep.est documentation repeats this with by first calling
## aGP.seq and then discrep.est.  The answers should be identical, however
## note that a call first to example("fcalib") and then
## example("discrep.est") will generate two random data sets, causing
## the results not to match

## begin data-generation code identical to aGP.seq, discrep.est, fcalib
## example sections and demo("calib")

## M: computer model test function used in Goh et al, 2013 (Technometrics)
## an elaboration of one from Bastos and O'Hagan, 2009 (Technometrics)
M <- function(x,u)
  {
    x <- as.matrix(x)
    u <- as.matrix(u)
    out <- (1-exp(-1/(2*x[,2])))
    out <- out * (1000*u[,1]*x[,1]^3+1900*x[,1]^2+2092*x[,1]+60)
    out <- out / (100*u[,2]*x[,1]^3+500*x[,1]^2+4*x[,1]+20)
    return(out)
  }

## bias: discrepancy function from Goh et al, 2013
bias <- function(x)
  {
    x<-as.matrix(x)
    out<- 2*(10*x[,1]^2+4*x[,2]^2) / (50*x[,1]*x[,2]+10)
    return(out)
  }

## beta.prior: marginal beta prior for u,
## defaults to a mode at 1/2 in hypercube
beta.prior <- function(u, a=2, b=2, log=TRUE)
{
  if(length(a) == 1) a <- rep(a, length(u))
  else if(length(a) != length(u)) stop("length(a) must be 1 or length(u)")
  if(length(b) == 1) b <- rep(b, length(u))
  else if(length(b) != length(u)) stop("length(b) must be 1 or length(u)")
  if(log) return(sum(dbeta(u, a, b, log=TRUE)))
  else return(prod(dbeta(u, a, b, log=FALSE)))
}

## tgp for LHS sampling
library(tgp)
rect <- matrix(rep(0:1, 4), ncol=2, byrow=2)

## training inputs
ny <- 50;
X <- lhs(ny, rect[1:2,])    ## computer model train

## true (but unknown) setting of the calibration parameter
## for the computer model
```

```
u <- c(0.2, 0.1)
Zu <- M(X, matrix(u, nrow=1))

## field data response, biased and replicated
sd <- 0.5
## Y <- computer output + bias + noise
reps <- 2 ## example from paper uses reps <- 10
Y <- rep(Zu,reps) + rep(bias(X),reps) + rnorm(reps*length(Zu), sd=sd)
## variations: remove the bias or change 2 to 1 to remove replicates

## computer model design
nz <- 10000
XU <- lhs(nz, rect)
nth <- 1 ## number of threads to use in emulation, demo uses 8

## augment with physical model design points
## with various u settings
XU2 <- matrix(NA, nrow=10*ny, ncol=4)
for(i in 1:10) {
  I <- ((i-1)*ny+1):(ny*i)
  XU2[I,1:2] <- X
}
XU2[,3:4] <- lhs(10*ny, rect[3:4,])
XU <- rbind(XU, XU2)

## evaluate the computer model
Z <- M(XU[,1:2], XU[,3:4])

## flag indicating if estimating bias/discrepancy or not
bias.est <- TRUE
## two passes of ALC with MLE calculations for aGP.seq
methods <- rep("alcray", 2) ## demo uses rep("alc", 2)

## set up priors
da <- d <- darg(NULL, XU)
g <- garg(list(mle=TRUE), Y)

## end identical data generation code

## now calculate log posterior for true calibration parameter
## value (u).  You could repeat this for an estimate value
## from demo("calib"), for example u.hat <- c(0.8236673, 0.1406989)

fcalib(u, XU, Z, X, Y, da, d, g, beta.prior, methods, M, bias.est, nth)
```

Description

Build a sub-design of X of size end, and infer parameters, for approximate Gaussian process prediction at reference location(s) Xref. Return the moments of those predictive equations, and indices into the local design

Usage

```
laGP(Xref, start, end, X, Z, d = NULL, g = 1/10000,
     method = c("alc", "alcopt", "alcray", "mspe", "nn", "fish"), Xi.ret = TRUE,
    close = min((1000+end)*if(method[1] %in% c("alcray", "alcopt")) 10 else 1, nrow(X)),
     alc.gpu = FALSE, numstart = if(method[1] == "alcray") ncol(X) else 1,
     rect = NULL, lite = TRUE, verb = 0)
laGP.R(Xref, start, end, X, Z, d = NULL, g = 1/10000,
     method = c("alc", "alcopt", "alcray", "mspe", "nn", "fish"),
     Xi.ret = TRUE, pall = FALSE,
    close = min((1000+end)*if(method[1] %in% c("alcray", "alcopt")) 10 else 1, nrow(X)),
     parallel = c("none", "omp", "gpu"),
     numstart = if(method[1] == "alcray") ncol(X) else 1,
     rect = NULL, lite = TRUE, verb = 0)
laGPsep(Xref, start, end, X, Z, d = NULL, g = 1/10000,
     method = c("alc", "alcopt", "alcray", "nn"), Xi.ret = TRUE,
    close = min((1000+end)*if(method[1] %in% c("alcray", "alcopt")) 10 else 1, nrow(X)),
     alc.gpu = FALSE, numstart = if(method[1] == "alcray") ncol(X) else 1,
     rect = NULL, lite = TRUE, verb=0)
laGPsep.R(Xref, start, end, X, Z, d = NULL, g = 1/10000,
     method = c("alc", "alcopt", "alcray", "nn"),
     Xi.ret = TRUE, pall = FALSE,
    close = min((1000+end)*if(method[1] %in% c("alcray", "alcopt")) 10 else 1, nrow(X)),
     parallel = c("none", "omp", "gpu"),
     numstart = if(method[1] == "alcray") ncol(X) else 1,
     rect = NULL, lite = TRUE, verb = 0)
```

Arguments

Xref	a vector of length ncol(X) containing a single reference location; or a matrix with ncol(Xref) = ncol(X) containing multiple reference locations (unless method = "alcray") for simultaneous sub-design and prediction
start	the number of Nearest Neighbor (NN) locations for initialization; must specify start >= 6
end	the total size of the local designs; must have start < end
X	a matrix or data.frame containing the full (large) design matrix of input locations
Z	a vector of responses/dependent values with length(Z) = nrow(X)
d	a prior or initial setting for the lengthscale parameter for a Gaussian correlation function; a (default) NULL value causes a sensible regularization (prior) and initial setting to be generated via darg; a scalar specifies an initial value, causing

darg to only generate the prior; otherwise, a list or partial list matching the output of darg can be used to specify a custom prior. In the case of a partial list, the only the missing entries will be generated. Note that a default/generated list specifies MLE/MAP inference for this parameter. With laGPsep, the starting values can be an ncol(X)-by-nrow(XX) matrix or ncol(X) vector

g — a prior or initial setting for the nugget parameter; a NULL value causes a sensible regularization (prior) and initial setting to be generated via garg; a scalar (default g = 1/10000) specifies an initial value, causing garg to only generate the prior; otherwise, a list or partial list matching the output of garg can be used to specify a custom prior. In the case of a partial list, only the missing entries will be generated. Note that a default/generated list specifies *no* inference for this parameter; i.e., it is fixed at its starting or default value, which may be appropriate for emulating deterministic computer code output. In such situations a value much smaller than the default may work even better (i.e., yield better out-of-sample predictive performance). The default was chosen conservatively

method — Specifies the method by which end-start candidates from X are chosen in order to predict at Xref. In brief, ALC ("alc", default) minimizes predictive variance; ALCRAY ("alcray")) executes a thrifty ALC-based search focused on rays emanating from the reference location [must have nrow(Xref) = 1]; ALCOPT ("alcopt") optimizes a continuous ALC analog via derivatives to and snaps the solution back to the candidate grid; MSPE ("mspe") augments ALC with extra derivative information to minimize mean-squared prediction error (requires extra computation); NN ("nn") uses nearest neighbor; and EFI ("fish") uses the expected Fisher information - essentially 1/G from Gramacy & Apley (2015) - which is global heuristic, i.e., not localized to Xref.

Xi.ret — A scalar logical indicating whether or not a vector of indices into X, specifying the chosen sub-design, should be returned on output

pall — a scalar logical (for laGP.R only) offering the ability to obtain predictions after every update (for progress indication and debugging), rather than after just the last update

close — a non-negative integer end < close <= nrow(X) specifying the number of NNs (to Xref) in X to consider when searching for elements of the sub-design; close = 0 specifies all. For method="alcray" and method="alcopt", this argument specifies the scope used to snap solutions obtained via analog continuous searches back to elements of X, otherwise there are no restrictions on those searches. Since these approximate searches are cheaper, they can afford a larger "snapping scope" hence the larger default

alc.gpu — a scalar logical indicating if a GPU should be used to parallelize the evaluation of the ALC criteria (method = "alc"). Requires the package be compiled with CUDA flags; see README/INSTALL in the package source for more details; currently only available for nrow(Xref) == 1 via laGP, not laGPsep or the .R variants, and only supports off-loading ALC calculation to the GPU

parallel — a switch indicating if any parallel calculation of the criteria is desired. Currently parallelization at this level is only provided for option method = "alc"). For parallel = "omp", the package must be compiled with OMP flags; for parallel = "gpu", the package must be compiled with CUDA flags see

README/INSTALL in the package source for more details; currently only available via laGP.R

numstart — a scalar integer indicating the number of rays for each greedy search when method="alcray" or the number of restarts when method="alcopt". More rays or restarts leads to a more thorough, but more computational intensive search. This argument is not involved in other methods

rect — an optional 2-by-ncol(X) matrix describing a bounding rectangle for X that is used by the "alcray" method. If not specified, the rectangle is calculated from range applied to the columns of X

lite — Similar to the predGP option of the same name, this argument specifies whether (TRUE, the default) or not (FALSE) to return a full covariance structure is returned, as opposed the diagonal only. A full covariance structure requires more computation and more storage. This option is only relevant when nrow(Xref) > 1

verb — a non-negative integer specifying the verbosity level; verb = 0 is quiet, and larger values cause more progress information to be printed to the screen

Details

A sub-design of X of size end is built-up according to the criteria prescribed by the method and then used to predict at Xref. The first start locations are NNs in order to initialize the first GP, via newGP or newGPsep, and thereby initialize the search. Each subsequent addition is found via the chosen criterion (method), and the GP fit is updated via updateGP or updateGPsep

The runtime is cubic in end, although the multiplicative "constant" out front depends on the method chosen, the size of the design X, and close. The "alcray" method has a smaller constant since it does not search over all candidates exhaustively.

After building the sub-design, local MLE/MAP lengthscale (and/or nugget) parameters are estimated, depending on the d and g arguments supplied. This is facilitated by calls to mleGP or jmleGP.

Finally predGP is called on the resulting local GP model, and the parameters of the resulting Student-t distribution(s) are returned. Unless Xi.ret = FALSE, the indices of the local design are also returned.

laGP.R and laGPsep.R are a prototype R-only version for debugging and transparency purposes. They are slower than laGP and laGPsep, which are primarily in C, and may not provide identical output in all cases due to differing library implementations used as subroutines; see note below for an example. laGP.R and other .R functions in the package may be useful for developing new programs that involve similar subroutines. The current version of laGP.R allows OpenMP and/or GPU parallelization of the criteria (method) if the package is compiled with the appropriate flags. See README/INSTALL in the package source for more information. For algorithmic details, see Gramacy, Niemi, & Weiss (2014)

Value

The output is a list with the following components.

mean — a vector of predictive means of length nrow(Xref)

s2 — a vector of Student-t scale parameters of length nrow(Xref)

df — a Student-t degrees of freedom scalar (applies to all Xref)

llik	a scalar indicating the maximized log likelihood or log posterior probability of the data/parameter(s) under the chosen sub-design; provided up to an additive constant
time	a scalar giving the passage of wall-clock time elapsed for (substantive parts of) the calculation
method	a copy of the method argument
d	a full-list version of the d argument, possibly completed by darg
g	a full-list version of the g argument, possibly completed by garg
mle	if d$mle and/or g$mle are TRUE, then mle is a data.frame containing the values found for these parameters, and the number of required iterations
Xi	when Xi.ret = TRUE, this field contains a vector of indices of length end into X indicating the sub-design chosen
close	a copy of the input argument

Note

laGPsep provides the same functionality as laGP but deploys a separable covariance function. However criteria (methods) EFI and MSPE are not supported. This is considered "beta" functionality at this time.

Note that using method="NN" gives the same result as specifying start=end, however at some extra computational expense.

Handling multiple reference locations (nrow(Xref) > 1) is "beta" functionality. In this case the initial start locations are chosen by applying NN to the average distances to all Xref locations. Using method="alcopt" causes exhaustive search to be approximated by a continuous analog via closed form derivatives. See alcoptGP for more details. Although the approximation provided has a spirit similar to method="alcray", in that both methods are intended to offer a thrifty alternative, method="alcray" is not applicable when nrow(Xref) > 1.

Differences between the C qsort function and R's order function may cause chosen designs returned from laGP and laGP.R (code and laGPsep and laGPsep.R) to differ when multiple X values are equidistant to Xref

Author(s)

Robert B. Gramacy <rbg@vt.edu> and Furong Sun <furongs@vt.edu>

References

F. Sun, R.B. Gramacy, B. Haaland, E. Lawrence, and A. Walker (2017) *Emulating satellite drag from large simulation experiments*; preprint on arXiv:1712.00182. http://arxiv.org/abs/1712.00182

R.B. Gramacy (2016). **laGP***: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in* R., Journal of Statistical Software, 72(1), 1-46; or see vignette("laGP")

R.B. Gramacy and B. Haaland (2016). *Speeding up neighborhood search in local Gaussian process prediction.* Technometrics, 58(3), pp. 294-303; preprint on arXiv:1409.0074 http://arxiv.org/abs/1409.0074

R.B. Gramacy and D.W. Apley (2015). *Local Gaussian process approximation for large computer experiments.* Journal of Computational and Graphical Statistics, 24(2), pp. 561-678; preprint on arXiv:1303.0383; http://arxiv.org/abs/1303.0383

R.B. Gramacy, J. Niemi, R.M. Weiss (2014). *Massively parallel approximate Gaussian process regression.* SIAM/ASA Journal on Uncertainty Quantification, 2(1), pp. 568-584; preprint on arXiv:1310.5182; http://arxiv.org/abs/1310.5182

See Also

vignette("laGP"), aGP, newGP, updateGP, predGP, mleGP, jmleGP, alcGP, mspeGP, alcrayGP, randLine ## path-based local prediction via laGP

Examples

```
## examining a particular laGP call from the larger analysis provided
## in the aGP documentation

## A simple 2-d test function used in Gramacy & Apley (2014);
## thanks to Lee, Gramacy, Taddy, and others who have used it before
f2d <- function(x, y=NULL)
  {
    if(is.null(y)) {
      if(!is.matrix(x) && !is.data.frame(x)) x <- matrix(x, ncol=2)
      y <- x[,2]; x <- x[,1]
    }
    g <- function(z)
      return(exp(-(z-1)^2) + exp(-0.8*(z+1)^2) - 0.05*sin(8*(z+0.1)))
    z <- -g(x)*g(y)
  }

## build up a design with N=~40K locations
x <- seq(-2, 2, by=0.02)
X <- as.matrix(expand.grid(x, x))
Z <- f2d(X)

## optional first pass of nearest neighbor
Xref <- matrix(c(-1.725, 1.725), nrow=TRUE)
out <- laGP(Xref, 6, 50, X, Z, method="nn")

## second pass via ALC, ALCOPT, MSPE, and ALC-ray respectively,
## conditioned on MLE d-values found above
out2 <- laGP(Xref, 6, 50, X, Z, d=out$mle$d)
out2.alcopt <- laGP(Xref, 6, 50, X, Z, d=out2$mle$d, method="alcopt")
out2.mspe <- laGP(Xref, 6, 50, X, Z, d=out2$mle$d, method="mspe")
out2.alcray <- laGP(Xref, 6, 50, X, Z, d=out2$mle$d, method="alcray")

## look at the different designs
plot(rbind(X[out2$Xi,], X[out2.mspe$Xi,]), type="n",
     xlab="x1", ylab="x2", main="comparing local designs")
points(Xref[1], Xref[2], col=2, cex=0.5)
text(X[out2$Xi,], labels=1:50, cex=0.6)
text(X[out2.alcopt$Xi,], labels=1:50, cex=0.6, col="forestgreen")
```

```
text(X[out2.mspe$Xi,], labels=1:50, cex=0.6, col="blue")
text(X[out2.alcray$Xi,], labels=1:50, cex=0.6, col="red")
legend("right", c("ALC", "ALCOPT", "MSPE", "ALCRAY"),
       text.col=c("black", "forestgreen", "blue", "red"), bty="n")

## compare computational time
c(nn=out$time, alc=out2$time, alcopt=out2.alcopt$time,
  mspe=out2.mspe$time, alcray=out2.alcray$time)

## Not run:
  ## A comparison between ALC-ex, ALC-opt and NN

  ## defining a predictive path
  wx <- seq(-0.85, 0.45, length=100)
  W <- cbind(wx-0.75, wx^3+0.51)

  ## three comparators from Sun, et al. (2017)
  ## larger-than-default "close" argument to capture locations nearby path
  p.alc <- laGPsep(W, 6, 100, X, Z, method="alc", close=10000, lite=FALSE)
  p.alcopt <- laGPsep(W, 6, 100, X, Z, method="alcopt", lite=FALSE)
  ## note that close=10*(1000+end) would be the default for method = "alcopt"
  p.nn <- laGPsep(W, 6, 100, X, Z, method="nn", close=10000, lite=FALSE)

  ## Mahalanobis distance to combine RMSE and covariance estimates
  mahdist <- function(Y, mu, Sigma) {
     Ymmu <- Y - mu
     Sigmai <- solve(Sigma)
     return(sqrt(t(Ymmu) %*% Sigmai %*% Ymmu))
  }

  ## comparison by Mahalanobis distance
  WY <- f2d(W)
  c(alc=mahdist(WY, p.alc$mean, p.alc$Sigma),
    alcopt=mahdist(WY, p.alcopt$mean, p.alcopt$Sigma),
    nn=mahdist(WY, p.nn$mean, p.nn$Sigma))

  ## comparison via time
  c(alc=p.alc$time, alcopt=p.alcopt$time, nn=p.nn$time)

  ## visualization
  ## first ALC-ex
  plot(W, type="l", bty="n", lwd=2, xlab="x1", ylab="x2",
    xlim=c(-2.25,0), ylim=c(-0.75,1.25), main="")
  points(W[length(wx)/2,1], W[length(wx)/2,2], col=2, pch=19)
  points(X[p.alc$Xi,], col="blue", cex=0.6)
  legend("bottomright", c("x0", "ALC-opt", "ALC-ex", "NN"),
    pch=c(19, 23, 21, 22), bty="n", cex=1.25,
    col=c("red", "purple", "blue", "forestgreen"))
  legend("topleft", "W(x0)", lty=1, col=1, lwd=2, bty="n",
    cex=1.25)
  ## then shifted NN
  lines(W[,1]+0.25, W[,2]-0.25)
  points(W[length(wx)/2,1]+0.25, W[length(wx)/2,2]-0.25, col=2, pch=19)
```

```
  points(X[p.nn$Xi,1]+0.25, X[p.nn$Xi,2]-0.25, pch=22,
    col="forestgreen", cex=0.6)
  ## finally shifted (in the other direction) ALC-opt
  lines(W[,1]-0.25, W[,2]+0.25)
  points(W[length(wx)/2,1]-0.25, W[length(wx)/2,2]+0.25, col=2, pch=19)
  points(X[p.alcopt$Xi,1]-0.25, X[p.alcopt$Xi,2]+0.25, pch=23,
    col="purple", cex=0.6)

## End(Not run)
```

`llikGP` *Calculate a GP log likelihood*

Description

Calculate a Gaussian process (GP) log likelihood or posterior probability with reference to a C-side GP object

Usage

```
llikGP(gpi, dab = c(0, 0), gab = c(0, 0))
llikGPsep(gpsepi, dab = c(0, 0), gab = c(0, 0))
```

Arguments

`gpi`	a C-side GP object identifier (positive integer); e.g., as returned by `newGP`
`gpsepi`	similar to `gpi` but indicating a separable GP object
`dab`	ab for the lengthscale parameter, see Details
`gab`	ab for the nugget parameter, see Details

Details

An "ab" parameter is a non-negative 2-vector describing shape and rate parameters to a Gamma prior; a zero-setting for either value results in no-prior being used in which case a log likelihood is returned. If both ab parameters are specified, then the value returned can be interpreted as a log posterior density. See `darg` for more information about ab

Value

A real-valued scalar is returned.

Author(s)

Robert B. Gramacy <rbg@vt.edu>

See Also

`mleGP`, `darg`

Examples

```
## partly following the example in mleGP
library(MASS)

## motorcycle data and predictive locations
X <- matrix(mcycle[,1], ncol=1)
Z <- mcycle[,2]

## get sensible ranges
d <- darg(NULL, X)
g <- garg(list(mle=TRUE), Z)

## initialize the model
gpi <- newGP(X, Z, d=d$start, g=g$start)

## calculate log likelihood
llikGP(gpi)
## calculate posterior probability
llikGP(gpi, d$ab, g$ab)

## clean up
deleteGP(gpi)
```

---

mleGP *Inference for GP correlation parameters*

Description

Maximum likelihood/a posteriori inference for (isotropic and separable) Gaussian lengthscale and nugget parameters, marginally or jointly, for Gaussian process regression

Usage

```
mleGP(gpi, param = c("d", "g"), tmin=sqrt(.Machine$double.eps),
  tmax = -1, ab = c(0, 0), verb = 0)
mleGPsep(gpsepi, param=c("d", "g", "both"), tmin=rep(sqrt(.Machine$double.eps), 2),
  tmax=c(-1,1), ab=rep(0,4), maxit=100, verb=0)
mleGPsep.R(gpsepi, param=c("d", "g"), tmin=sqrt(.Machine$double.eps),
  tmax=-1, ab=c(0,0), maxit=100, verb=0)
jmleGP(gpi, drange=c(sqrt(.Machine$double.eps),10),
  grange=c(sqrt(.Machine$double.eps), 1), dab=c(0,0), gab=c(0,0), verb=0)
jmleGP.R(gpi, N=100, drange=c(sqrt(.Machine$double.eps),10),
  grange=c(sqrt(.Machine$double.eps), 1), dab=c(0,0), gab=c(0,0), verb=0)
jmleGPsep(gpsepi, drange=c(sqrt(.Machine$double.eps),10),
  grange=c(sqrt(.Machine$double.eps), 1), dab=c(0,0), gab=c(0,0),
  maxit=100, verb=0)
jmleGPsep.R(gpsepi, N=100, drange=c(sqrt(.Machine$double.eps),10),
  grange=c(sqrt(.Machine$double.eps), 1), dab=c(0,0), gab=c(0,0),
  maxit=100, mleGPsep=mleGPsep.R, verb=0)
```

Arguments

gpi	a C-side GP object identifier (positive integer); e.g., as returned by newGP
gpsepi	similar to gpi but indicating a separable GP object, as returned by newGPsep
N	for jmleGP.R, the maximum number of times the pair of margins should be iterated over before determining failed convergence; note that (at this time) jmleGP uses a hard-coded N=100 in its C implementation
param	for mleGP, indicating whether to work on the lengthscale (d) or nugget (g) margin
tmin	for mleGP, smallest value considered for the parameter (param)
tmax	for mleGP, largest value considered for the parameter (param)
drange	for jmleGP, these are c(tmin, tmax) values for the lengthscale parameter; the default values are reasonable for 1-d inputs in the unit interval
grange	for jmleGP, these are c(tmin, tmax) values for the nugget parameter; the default values are reasonable for responses with a range of one
ab	for mleGP, a non-negative 2-vector describing shape and rate parameters to a Gamma prior for the parameter (param); a zero-setting for either value results in no-prior being used (MLE inference); otherwise MAP inference is performed
maxit	for mleGPsep this is passed as control=list(trace=maxit) to optim's L-BFGS-B method for optimizing the likelihood/posterior of a separable GP representation; this argument is not used for isotropic GP versions, nor for optimizing the nugget
dab	for jmleGP, this is ab for the lengthscale parameter
gab	for jmleGP, this is ab for the nugget parameter
mleGPsep	function for internal MLE calculation of the separable lengthscale parameter; one of either mleGPsep.R based on method="L-BFGS-B" using optim; or mleGPsep using the C entry point lbfgsb. Both options use a C backend for the nugget
verb	a verbosity argument indicating how much information about the optimization steps should be printed to the screen; verb <= 0 is quiet; for jmleGP, a verb - 1 value is passed to the mleGP or mleGPsep subroutine(s)

Details

mleGP and mleGPsep perform marginal (or profile) inference for the specified param, either the lengthscale or the nugget. mleGPsep can perform simultaneous lengthscale and nugget inference via a common gradient with param = "both". More details are provided below.

For the lengthscale, mleGP uses a Newton-like scheme with analytic first and second derivatives (more below) to find the scalar parameter for the isotropic Gaussian correlation function, with hard-coded 100-max iterations threshold and a sqrt(.Machine$double.eps) tolerance for determining convergence; mleGPsep.R uses L-BFGS-B via optim for the vectorized parameter of the separable Gaussian correlation, with a user-supplied maximum number of iterations (maxit) passed to optim. When maxit is reached the output conv = 1 is returned, subsequent identical calls to mleGPsep.R can be used to continue with further iterations. mleGPsep is similar, but uses the C entry point lbfgsb.

For the nugget, both mleGP and mleGPsep utilize a (nearly identical) Newton-like scheme leveraging first and second derivatives.

jmleGP and jmleGPsep provide joint inference by iterating over the marginals of lengthscale and nugget. The jmleGP.R function is an R-only wrapper around mleGP (which is primarily in C), whereas jmleGP is primarily in C but with reduced output and with hard-coded N=100. The same is true for jmleGPsep.

mleGPsep provides a param = "both" alternative to jmleGPsep leveraging a common gradient. It can be helpful to supply a larger maxit argument compared to jmleGPsep as the latter may do up to 100 outer iterations, cycling over lengthscale and nugget. mleGPsep usually requires many fewer total iterations, unless one of the lengthscale or nugget is already converged. In anticipation of param = "both" the mleGPsep function has longer default values for its bounds and prior arguments. These longer arguments are ignored when param != "both". At this time mleGP does not have a param = "both" option.

All methods are initialized at the value of the parameter(s) currently stored by the C-side object referenced by gpi or gpsepi. It is *highly recommended* that sensible range values (tmin, tmax or drange, grange) be provided. The defaults provided are too loose for most applications. As illustrated in the examples below, the darg and garg functions can be used to set appropriate ranges from the distributions of inputs and output data respectively.

The Newton-like method implemented for the isotropic lengthscale and for the nugget offers very fast convergence to local maxima, but sometimes it fails to converge (for any of the usual reasons). The implementation detects this, and in such cases it invokes a Brent_fmin call instead - this is the method behind the optimize function.

Note that the gpi or gpsepi object(s) must have been allocated with dK=TRUE; alternatively, one can call buildKGP or buildKGPsep - however, this is not in the NAMESPACE at this time

Value

A self-explanatory data.frame is returned containing the values inferred and the number of iterations used. The jmleGP.R and jmleGPsep.R functions will also show progress details (the values obtained after each iteration over the marginals).

However, the most important "output" is the modified GP object which retains the setting of the parameters reported on output as a side effect.

mleGPsep and jmleGPsep provide an output field/column called conv indicating convergence (when 0), or alternately a value agreeing with a non-convergence code provided on output by optim

Author(s)

Robert B. Gramacy <rbg@vt.edu>

References

For standard GP inference, refer to any graduate text, e.g., Rasmussen & Williams *Gaussian Processes for Machine Learning*;

See Also

vignette("laGP"), newGP, laGP, llikGP, optimize

Examples

```
## a simple example with estimated nugget
library(MASS)

## motorcycle data and predictive locations
X <- matrix(mcycle[,1], ncol=1)
Z <- mcycle[,2]

## get sensible ranges
d <- darg(NULL, X)
g <- garg(list(mle=TRUE), Z)

## initialize the model
gpi <- newGP(X, Z, d=d$start, g=g$start, dK=TRUE)

## separate marginal inference (not necessary - for illustration only)
print(mleGP(gpi, "d", d$min, d$max))
print(mleGP(gpi, "g", g$min, g$max))

## joint inference (could skip straight to here)
print(jmleGP(gpi, drange=c(d$min, d$max), grange=c(g$min, g$max)))

## plot the resulting predictive surface
N <- 100
XX <- matrix(seq(min(X), max(X), length=N), ncol=1)
p <- predGP(gpi, XX, lite=TRUE)
plot(X, Z, main="stationary GP fit to motorcycle data")
lines(XX, p$mean, lwd=2)
lines(XX, p$mean+1.96*sqrt(p$s2*p$df/(p$df-2)), col=2, lty=2)
lines(XX, p$mean-1.96*sqrt(p$s2*p$df/(p$df-2)), col=2, lty=2)

## clean up
deleteGP(gpi)

##
## with a separable correlation function
##

## 2D Example: GoldPrice Function, mimicking GP_fit package
f <- function(x)
{
  x1 <- 4*x[,1] - 2
  x2 <- 4*x[,2] - 2;
  t1 <- 1 + (x1 + x2 + 1)^2*(19 - 14*x1 + 3*x1^2 - 14*x2 + 6*x1*x2 + 3*x2^2);
  t2 <- 30 + (2*x1 -3*x2)^2*(18 - 32*x1 + 12*x1^2 + 48*x2 - 36*x1*x2 + 27*x2^2);
  y <- t1*t2;
  return(y)
}

## build design
library(tgp)
n <- 50 ## change to 100 or 1000 for more interesting demo
```

```
B <- rbind(c(0,1), c(0,1))
X <- dopt.gp(n, Xcand=lhs(10*n, B))$XX
## this differs from GP_fit in that we use the log response
Y <- log(f(X))

## get sensible ranges
d <- darg(NULL, X)
g <- garg(list(mle=TRUE), Y)

## build GP and jointly optimize via profile mehtods
gpisep <- newGPsep(X, Y, d=rep(d$start, 2), g=g$start, dK=TRUE)
jmleGPsep(gpisep, drange=c(d$min, d$max), grange=c(g$min, g$max))

## clean up
deleteGPsep(gpisep)

## alternatively, we can optimize via a combined gradient
gpisep <- newGPsep(X, Y, d=rep(d$start, 2), g=g$start, dK=TRUE)
mleGPsep(gpisep, param="both", tmin=c(d$min, g$min), tmax=c(d$max, d$max))
deleteGPsep(gpisep)
```

---

newGP *Create A New GP Object*

Description

Build a Gaussian process C-side object based on the X-Z data and parameters provided, and augment those objects with new data

Usage

```
newGP(X, Z, d, g, dK = FALSE)
newGPsep(X, Z, d, g, dK = FALSE)
updateGP(gpi, X, Z, verb = 0)
updateGPsep(gpsepi, X, Z, verb = 0)
```

Arguments

X	a matrix or data.frame containing the full (large) design matrix of input locations
Z	a vector of responses/dependent values with length(Z) = ncol(X)
d	a positive scalar lengthscale parameter for an isotropic Gaussian correlation function (newGP); or a vector for a separable version (newGPsep)
g	a positive scalar nugget parameter
dK	a scalar logical indicating whether or not derivative information (for the lengthscale) should be maintained by the GP object; this is required for calculating MLEs/MAPs of the lengthscale parameter(s) via mleGP and jmleGP

gpi	a C-side GP object identifier (positive integer); e.g., as returned by newGP
gpsepi	similar to gpi but indicating a separable GP object, as returned by newGPsep
verb	a non-negative integer indicating the verbosity level. A positive value causes progress statements to be printed to the screen for each update of i in 1:nrow(X)

Details

newGP allocates a new GP object on the C-side and returns its unique integer identifier (gpi), taking time which is cubic on nrow(X); allocated GP objects must (eventually) be destroyed with deleteGP or deleteGPs or memory will leak. The same applies for newGPsep, except deploying a separable correlation with limited feature set; see deleteGPsep and deleteGPseps

updateGP takes gpi identifier as input and augments that GP with new data. A sequence of updates is performed, for each i in 1:nrow(X), each taking time which is quadratic in the number of data points. updateGP also updates any statistics needed in order to quickly search for new local design candidates via laGP. The same applies to updateGPsep on gpsepi objects

Value

newGP and newGPsep create a unique GP indicator (gpi or gpsepi) referencing a C-side object; updateGP and updateGPsep do not return anything, but yields a modified C-side object as a side effect

Author(s)

Robert B. Gramacy <rbg@vt.edu>

References

For standard GP inference, refer to any graduate text, e.g., Rasmussen & Williams *Gaussian Processes for Machine Learning*.

For efficient updates of GPs, see:

R.B. Gramacy and D.W. Apley (2015). *Local Gaussian process approximation for large computer experiments.* Journal of Computational and Graphical Statistics, 24(2), pp. 561-678; preprint on arXiv:1303.0383; http://arxiv.org/abs/1303.0383

See Also

vignette("laGP"), deleteGP, mleGP, predGP, laGP

Examples

```
## for more examples, see predGP and mleGP docs

## simple sine data
X <- matrix(seq(0,2*pi,length=7), ncol=1)
Z <- sin(X)
```

```
## new GP fit
gpi <- newGP(X, Z, 2, 0.000001)

## make predictions
XX <- matrix(seq(-1,2*pi+1, length=499), ncol=ncol(X))
p <- predGP(gpi, XX)

## sample from the predictive distribution
library(mvtnorm)
N <- 100
ZZ <- rmvt(N, p$Sigma, p$df)
ZZ <- ZZ + t(matrix(rep(p$mean, N), ncol=N))
matplot(XX, t(ZZ), col="gray", lwd=0.5, lty=1, type="l",
       xlab="x", ylab="f-hat(x)", bty="n")
points(X, Z, pch=19, cex=2)

## update with four more points
X2 <- matrix(c(pi/2, 3*pi/2, -0.5, 2*pi+0.5), ncol=1)
Z2 <- sin(X2)
updateGP(gpi, X2, Z2)

## make a new set of predictions
p2 <- predGP(gpi, XX)
ZZ <- rmvt(N, p2$Sigma, p2$df)
ZZ <- ZZ + t(matrix(rep(p2$mean, N), ncol=N))
matplot(XX, t(ZZ), col="gray", lwd=0.5, lty=1, type="l",
       xlab="x", ylab="f-hat(x)", bty="n")
points(X, Z, pch=19, cex=2)
points(X2, Z2, pch=19, cex=2, col=2)

## clean up
deleteGP(gpi)
```

---

optim.auglag *Optimize an objective function under multiple blackbox constraints*

Description

Uses a surrogate modeled augmented Lagrangian (AL) system to optimize an objective function (blackbox or known and linear) under unknown multiple (blackbox) constraints via expected improvement (EI) and variations; a comparator based on EI with constraints is also provided

Usage

```
optim.auglag(fn, B, fhat = FALSE, equal = FALSE, ethresh = 1e-2,
  slack = FALSE, cknown = NULL, start = 10, end = 100,
  Xstart = NULL, sep = TRUE, ab = c(3/2, 8), lambda = 1, rho = NULL,
  urate = 10, ncandf = function(t) { t }, dg.start = c(0.1, 1e-06),
  dlim = sqrt(ncol(B)) * c(1/100, 10), Bscale = 1, ey.tol = 1e-2,
  N = 1000, plotprog = FALSE, verb = 2, ...)
```

```
optim.efi(fn, B, fhat = FALSE, cknown = NULL, start = 10, end = 100,
  Xstart = NULL, sep = TRUE, ab = c(3/2,8), urate = 10,
  ncandf = function(t) { t }, dg.start = c(0.1, 1e-6),
  dlim = sqrt(ncol(B))*c(1/100,10), Bscale = 1, plotprog = FALSE,
  verb = 2, ...)
```

Arguments

fn	function of an input (x), facilitating vectorization on a matrix X thereof, returning a list with elements $obj containing the (scalar) objective value and $c containing a vector of evaluations of the (multiple) constraint function at x. The fn function must take a known.only argument which is explained in the note below; it need not act on that argument
B	2-column matrix describing the bounding box. The number of rows of the matrix determines the input dimension (length(x) in fn(x)); the first column gives lower bounds and the second gives upper bounds
fhat	a scalar logical indicating if the objective function should be modeled with a GP surrogate. The default of FALSE assumes a known linear objective scaled by Bscale. Using TRUE is an "alpha" feature at this time
equal	an optional vector containing zeros and ones, whose length equals the number of constraints, specifying which should be treated as equality constraints (0) and which as inequality (1)
ethresh	a threshold used for equality constraints to determine validity for progress measures; ignored if there are no equality constraints
slack	A scalar logical indicating if slack variables, and thus exact EI calculations should be used. The default of slack = FALSE results in Monte Carlo EI approximation. One can optionally specify slack = 2 to get the slack = TRUE behavior, with a second-stage L-BFGS-B optimization of the EI acquisition applied at the end, starting from the best value found on the random search grid
cknown	A optional positive integer vector specifying which of the constraint values returned by fn should be treated as "known", i.e., not modeled with Gaussian processes
start	positive integer giving the number of random starting locations before sequential design (for optimization) is performed; start >= 6 is recommended unless Xstart is non-NULL; in the current version the starting locations come from a space-filling design via dopt.gp
end	positive integer giving the total number of evaluations/trials in the optimization; must have end > start
Xstart	optional matrix of starting design locations in lieu of, or in addition to, start random ones; we recommend nrow(Xstart) + start >= 6; also must have ncol(Xstart) = nrow(B)
sep	The default sep = TRUE uses separable GPs (i.e., via newGPsep, etc.) to model the constraints and objective; otherwise the isotropic GPs are used
ab	prior parameters; see darg describing the prior used on the lengthscale parameter during emulation(s) for the constraints

lambda	m-dimensional initial Lagrange multiplier parameter for m-constraints
rho	positive scalar initial quadratic penalty parameter in the augmented Lagrangian; the default setting of rho = NULL causes an automatic starting value to be chosen; see rejoinder to Gramacy, et al. (2016) or supplementary material to Picheny, et al. (2016)
urate	positive integer indicating how many optimization trials should pass before each MLE/MAP update is performed for GP correlation lengthscale parameter(s)
ncandf	function taking a single integer indicating the optimization trial number t, where start < t <= end, and returning the number of search candidates (e.g., for expected improvement calculations) at round t; the default setting allows the number of candidates to grow linearly with t
dg.start	2-vector giving starting values for the lengthscale and nugget parameters of the GP surrogate model(s) for constraints
dlim	2-vector giving bounds for the lengthscale parameter(s) under MLE/MAP inference, thereby augmenting the prior specification in ab
Bscale	scalar indicating the relationship between the sum of the inputs, sum(x), to fn and the output fn(x)$obj; note that at this time only linear objectives are fully supported by the code - more details below
ey.tol	a scalar proportion indicating how many of the EIs at ncandf(t) candidate locations must be non-zero to "trust" that metric to guide search, reducing to an EY-based search instead [choosing that proportion to be one forces EY-based search]
N	positive scalar integer indicating the number of Monte Carlo samples to be used for calculating EI and EY
plotprog	logical indicating if progress plots should be made after each inner iteration; the plots show three panels tracking the best valid objective, the EI or EY surface over the first two input variables (requires interp, and the parameters of the lengthscale(s) of the GP(s) respectively. When plotprog = TRUE the interp.loess function is used to aid in creating surface plots, however this does not work well with fewer than fifteen points. You may also provide a function as an argument, having similar arguments/formals as interp.loess. For example, we use interp below, which would have been the default if not for licensing incompatibilities
verb	a non-negative integer indicating the verbosity level; the larger the value the more that is printed to the screen
...	additional arguments passed to fn

Details

These subroutines support a suite of methods used to optimize challenging constrained problems from Gramacy, et al. (2016); and from Picheny, et al., (2016) see references below.

Those schemes hybridize Gaussian process based surrogate modeling and expected improvement (EI; Jones, et., al, 2008) with the additive penalty method (APM) implemented by the augmented Lagrangian (AL, e.g., Nocedal & Wright, 2006). The goal is to minimize a (possibly known) linear objective function f(x) under multiple, unknown (blackbox) constraint functions satisfying

`c(x) <= 0`, which is vector-valued. The solution here emulates the components of `c` with Gaussian process surrogates, and guides optimization by searching the posterior mean surface, or the EI of, the following composite objective

$$Y(x) = f(x) + \lambda^\top Y_c(x) + \frac{1}{2\rho}\sum_{i=1}^{m} \max(0, Y_{c_i}(x))^2,$$

where λ and ρ follow updating equations that guarantee convergence to a valid solution minimizing the objective. For more details, see Gramacy, et al. (2016).

A slack variable implementation that allows for exact EI calculations and can accommodate equality constraints, and mixed (equality and inequality) constraints, is also provided. For further details, see Picheny, et al. (2016).

The example below illustrates a variation on the toy problem considered in both papers, which bases sampling on EI. For examples making used of equality constraints, following the Picheny, et al (2016) papers; see the demos listed in the "See Also" section below.

Although it is off by default, these functions allow an unknown objective to be modeled (`fhat = TRUE`), rather than assuming a known linear one. For examples see `demo("ALfhat")` and `demo("GSBP")` which illustrate the AL and comparators in inequality and mixed constraints setups, respectively.

The `optim.efi` function is provided as a comparator. This method uses the same underlying GP models to with the hybrid EI and probability of satisfying the constraints heuristic from Schonlau, et al., (1998). See `demo("GSBP")` and `demo("LAH")` for `optim.efi` examples and comparisons between the original AL, the slack variable enhancement(s) on mixed constraint problems with known and blackbox objectives, respectively

Value

The output is a `list` summarizing the progress of the evaluations of the blackbox under optimization

`prog`	vector giving the best valid (`c(x) < 0`) value of the objective over the trials
`obj`	vector giving the value of the objective for the input under consideration at each trial
`X`	`matrix` giving the input values at which the blackbox function was evaluated
`C`	`matrix` giving the value of the constraint function for the input under consideration at each trial
`d`	`matrix` of lengthscale values obtained at the final update of the GP emulator for each constraint
`df`	if `fhat = TRUE` then this is a `matrix` of lengthscale values for the objective obtained at the final update of the GP emulator
`lambda`	a `matrix` containing `lambda` vectors used in each "outer loop" AL iteration
`rho`	a vector of `rho` values used in each "outer loop" AL iteration

Note

This function is under active development, especially the newest features including separable GP surrogate modeling, surrogate modeling of a blackbox objective, and the use of slack variables for

exact EI calculations and the support if equality constraints. Also note that, compared with earlier versions, it is now required to augment your blackbox function (`fn`) with an argument named `known.only`. This allows the user to specify if a potentially different object (with a subset of the outputs, those that are "known") gets returned in certain circumstances. For example, the objective is treated as known in many of our examples. When a non-null `cknown` object is used, the `known.only` flag can be used to return only the outputs which are known.

Older versions of this function provided an argument called `nomax`. The NoMax feature is no longer supported

Author(s)

Robert B. Gramacy <rbg@vt.edu>

References

Picheny, V., Gramacy, R.B., Wild, S.M., Le Digabel, S. (2016). "Bayesian optimization under mixed constraints with a slack-variable augmented Lagrangian". Preprint available on arXiv:1605.09466; http://arxiv.org/abs/1605.09466

Gramacy, R.B, Gray, G.A, Lee, H.K.H, Le Digabel, S., Ranjan P., Wells, G., Wild, S.M. (2016) "Modeling an Augmented Lagrangian for Improved Blackbox Constrained Optimization", *Technometrics* (with discussion), 58(1), 1-11. Preprint available on arXiv:1403.4890; http://arxiv.org/abs/1403.4890

Jones, D., Schonlau, M., and Welch, W. J. (1998). "Efficient Global Optimization of Expensive Black Box Functions." *Journal of Global Optimization*, 13, 455-492.

Schonlau, M., Jones, D.R., and Welch, W. J. (1998). "Global Versus Local Search in Constrained Optimization of Computer Models." In *New Developments and Applications in Experimental Design*, vol. 34, 11-25. Institute of Mathematical Statistics.

Nocedal, J. and Wright, S.J. (2006). *Numerical Optimization.* 2nd ed. Springer.

See Also

`vignette("laGP")`, `demo("ALfhat")` for blackbox objective, `demo("GSBP")` for a mixed constraints problem with blackbox objective, `demo("LAH")` for mix constraints with known objective, `optim.step.tgp` for unconstrained optimization; `optim` with `method="L-BFGS-B"` for box constraints, or `optim` with `method="SANN"` for simulated annealing

Examples

```
## this example assumes a known linear objective; further examples
## are in the optim.auglag demo

## a test function returning linear objective evaluations and
## non-linear constraints
aimprob <- function(X, known.only = FALSE)
{
  if(is.null(nrow(X))) X <- matrix(X, nrow=1)
  f <- rowSums(X)
  if(known.only) return(list(obj=f))
  c1 <- 1.5-X[,1]-2*X[,2]-0.5*sin(2*pi*(X[,1]^2-2*X[,2]))
```

```
  c2 <- rowSums(X^2)-1.5
  return(list(obj=f, c=cbind(c1,c2)))
}

## set bounding rectangle for adaptive sampling
B <- matrix(c(rep(0,2),rep(1,2)),ncol=2)

## optimization (primarily) by EI, change 25 to 100 for
## 99% chance of finding the global optimum with value 0.6
library(akima) ## for plotprog=interp
out <- optim.auglag(aimprob, B, end=25, plotprog=interp)

## using the slack variable implementation which is a little slower
## but more precise; slack=2 augments random search with L-BFGS-B

out2 <- optim.auglag(aimprob, B, end=25, slack=TRUE)
## Not run:
out3 <- optim.auglag(aimprob, B, end=25, slack=2)

## End(Not run)

## for more slack examples and comparison to optim.efi on problems
## involving equality and mixed (equality and inequality) constraints,
## see demo("ALfhat"), demo("GSBP") and demo("LAH")

## for comparison, here is a version that uses simulated annealing
## with the Additive Penalty Method (APM) for constraints
## Not run:
aimprob.apm <- function(x, B=matrix(c(rep(0,2),rep(1,2)),ncol=2))
{
  ## check bounding box
  for(i in 1:length(x)) {
    if(x[i] < B[i,1] || x[i] > B[i,2]) return(Inf)
  }

  ## evaluate objective and constraints
  f <- sum(x)
  c1 <- 1.5-x[1]-2*x[2]-0.5*sin(2*pi*(x[1]^2-2*x[2]))
  c2 <- x[1]^2+x[2]^2-1.5

  ## return APM composite
  return(f + abs(c1) + abs(c2))
}

## use SA; specify control=list(maxit=100), say, to control max
## number of iterations; does not easily facilitate plotting progress
out4 <- optim(runif(2), aimprob.apm, method="SANN")
## check the final value, which typically does not satisfy both
## constraints
aimprob(out4$par)

## End(Not run)
```

```
## for a version with a modeled objective see demo("ALfhat")
```

---

predGP *GP Prediction/Kriging*

Description

Perform Gaussian processes prediction (under isotropic or separable formulation) at new XX locations using a GP object stored on the C-side

Usage

```
predGP(gpi, XX, lite = FALSE, nonug = FALSE)
predGPsep(gpsepi, XX, lite = FALSE, nonug = FALSE)
```

Arguments

gpi	a C-side GP object identifier (positive integer); e.g., as returned by newGP
gpsepi	similar to gpi but indicating a separable GP object, as returned by newGPsep
XX	a matrix or data.frame containing a design of predictive locations
lite	a scalar logical indicating whether (lite = FALSE, default) or not (lite = TRUE) a full predictive covariance matrix should be returned, as would be required for plotting random sample paths, but substantially increasing computation time if only point-prediction is required
nonug	a scalar logical indicating if a (nonzero) nugget should be used in the predictive equations; this allows the user to toggle between visualizations of uncertainty due just to the mean, and a full quantification of predictive uncertainty. The latter (default nonug = FALSE) is the standard approach, but the former may work better in some sequential design contexts. See, e.g., ieciGP

Details

Returns the parameters of Student-t predictive equations. By default, these include a full predictive covariance matrix between all XX locations. However, this can be slow when nrow(XX) is large, so a lite options is provided, which only returns the diagonal of that matrix.

GP prediction is sometimes called "kriging", especially in the spatial statistics literature. So this function could also be described as returning evaluations of the "kriging equations"

Value

The output is a list with the following components.

mean	a vector of predictive means of length nrow(Xref)
Sigma	covariance matrix of for a multivariate Student-t distribution; alternately if lite = TRUE, then a field s2 contains the diagonal of this matrix
df	a Student-t degrees of freedom scalar (applies to all XX)

Author(s)

Robert B. Gramacy <rbg@vt.edu>

References

For standard GP prediction, refer to any graduate text, e.g., Rasmussen & Williams *Gaussian Processes for Machine Learning*

See Also

vignette("laGP"), newGP, mleGP, jmleGP,

Examples

```
## a "computer experiment" -- a much smaller version than the one shown
## in the aGP docs

## Simple 2-d test function used in Gramacy & Apley (2015);
## thanks to Lee, Gramacy, Taddy, and others who have used it before
f2d <- function(x, y=NULL)
  {
    if(is.null(y)) {
      if(!is.matrix(x) && !is.data.frame(x)) x <- matrix(x, ncol=2)
      y <- x[,2]; x <- x[,1]
    }
    g <- function(z)
      return(exp(-(z-1)^2) + exp(-0.8*(z+1)^2) - 0.05*sin(8*(z+0.1)))
    z <- -g(x)*g(y)
  }

## design with N=441
x <- seq(-2, 2, length=11)
X <- expand.grid(x, x)
Z <- f2d(X)

## fit a GP
gpi <- newGP(X, Z, d=0.35, g=1/1000)

## predictive grid with NN=400
xx <- seq(-1.9, 1.9, length=20)
XX <- expand.grid(xx, xx)
ZZ <- f2d(XX)

## predict
p <- predGP(gpi, XX, lite=TRUE)
## RMSE: compare to similar experiment in aGP docs
sqrt(mean((p$mean - ZZ)^2))

## visualize the result
par(mfrow=c(1,2))
image(xx, xx, matrix(p$mean, nrow=length(xx)), col=heat.colors(128),
      xlab="x1", ylab="x2", main="predictive mean")
```

```
image(xx, xx, matrix(p$mean-ZZ, nrow=length(xx)), col=heat.colors(128),
      xlab="x1", ylab="x2", main="bas")

## clean up
deleteGP(gpi)

## see the newGP and mleGP docs for examples using lite = FALSE for
## sampling from the joint predictive distribution
```

`randLine` *Generate two-dimensional random paths*

Description

Generate two-dimensional random paths (one-dimensional manifolds in 2d) comprising of different randomly chosen line types: linear, quadratic, cubic, exponential, and natural logarithm. If the input dimensionality is higher than 2, then a line in two randomly chosen input coordinates is generated

Usage

```
randLine(a, D, N, smin, res)
```

Arguments

`a`	a fixed two-element vector denoting the range of the bounding box (lower bound and upper bound) of all input coordinates
`D`	a scalar denoting the dimensionality of input space
`N`	a scalar denoting the desired total number of random lines
`smin`	a scalar denoting the minimum absolute scaling constant, i.e., the length of the shortest line that could be generated
`res`	a scalar denoting the number of data points, i.e., the resolution on the random path

Details

This two-dimensional random line generating function produces different types of `2d` random paths, including linear, quadratic, cubic, exponential, and natural logarithm.

First, one of these line types is chosen uniformly at random. The line is then drawn, via a collection of discrete points, from the origin according to the arguments, e.g., resolution and length, provided by the user. The discrete set of coordinates are then shifted and scaled, uniformly at random, into the specified 2d rectangle, e.g., $[-2, 2]^2$, with the restriction that at least half of the points comprising the line lie within the rectangle.

For a quick visualization, see Figure 15 in Sun, et al. (2017). Figure 7 in the same manuscript illustrates the application of this function in out-of-sample prediction using `laGP`, in `2d` and `4d`, respectively.

`randLine` returns different types of random paths and the indices of the randomly selected pair, i.e., subset, of input coordinates (when `D > 2`).

Value

randLine returns a list of lists. The outer list is of length six, representing each of the five possible line types (linear, quadratic, cubic, exponential, and natural logarithm), with the sixth entry providing the randomly chosen input dimensions.

The inner lists are comprised of $res \times 2$ data.frames, the number of which span N samples across all inner lists.

Note

Users should scale each coordinate of global input space to the same coded range, e.g., $[-2, 2]^D$, in order to avoid computational burden caused by passing global input space argument. Users may convert back to the natural units when necessary.

Author(s)

Furong Sun <furongs@vt.edu> and Robert B. Gramacy <rbg@vt.edu>

References

F. Sun, R.B. Gramacy, B. Haaland, E. Lawrence, and A. Walker (2017) *Emulating satellite drag from large simulation experiments*; preprint on arXiv:1712.00182. http://arxiv.org/abs/1712.00182

See Also

laGP, laGPsep

Examples

```
## 1. visualization of the randomly generated paths

## generate the paths
D <- 4
a <- c(-2, 2)
N <- 30
smin <- 0.1
res <- 100
line.set <- randLine(a=a, D=D, N=N, smin=smin, res=res)

## the indices of the randomly selected pair of input coordinates
d <- line.set$d

## visualization

## first create an empty plot
plot(0, xlim=a, ylim=a, type="l", xlab=paste("factor ", d[1], sep=""),
     ylab=paste("factor ", d[2], sep=""), main="2d random paths")
abline(h=(a[1]+a[2])/2, v=(a[1]+a[2])/2, lty=2)

## merge each path type together
```

```
W <- unlist(list(line.set$lin, line.set$qua, line.set$cub, line.set$ep, line.set$ln),
  recursive=FALSE)

## calculate colors to retain
n <- unlist(lapply(line.set, length)[-6])
cols <- rep(c("orange", "blue", "forestgreen", "magenta", "cornflowerblue"), n)

#* plot randomly generated paths with a centering dot in red at the midway point
for(i in 1:N){
  lines(W[[i]][,1], W[[i]][,2], col=cols[i])
  points(W[[i]][res/2,1], W[[i]][res/2,2], col=2, pch=20)
}

## 2. use the random paths for out-of-sample prediction via laGPsep

## test function (same 2d function as in other examples package)
## (ignoring 4d nature of path generation above)
f2d <- function(x, y=NULL){
  if(is.null(y)){
     if(!is.matrix(x) && !is.data.frame(x)) x <- matrix(x, ncol=2)
     y <- x[,2]; x <- x[,1]
  }
  g <- function(z)
  return(exp(-(z-1)^2) + exp(-0.8*(z+1)^2) - 0.05*sin(8*(z+0.1)))
  z <- -g(x)*g(y)
}

## generate training data using 2d input space
x <- seq(a[1], a[2], by=0.02)
X <- as.matrix(expand.grid(x, x))
Y <- f2d(X)

## example of joint path calculation folowed by RMSE calculation
## on the first random path
WW <- W[[sample(1:N, 1)]]
WY <- f2d(WW)

## exhaustive search via ``joint" ALC
j.exh <- laGPsep(WW, 6, 100, X, Y, method="alcopt", close=10000, lite=FALSE)
sqrt(mean((WY - j.exh$mean)^2)) ## RMSE

## repeat for all thirty path elements (way too slow for checking) and other
## local design choices and visualize RMSE distribution(s) side-by-side
## Not run:
  ## pre-allocate to save RMSE
  rmse.exh <- rmse.opt <- rmse.nn <- rmse.pw <- rmse.pwnn <- rep(NA, N)
  for(t in 1:N){

    WW <- W[[t]]
    WY <- f2d(WW)

    ## joint local design exhaustive search via ALC
    j.exh <- laGPsep(WW, 6, 100, X, Y, method="alc", close=10000, lite=FALSE)
```

```
    rmse.exh[t] <- sqrt(mean((WY - j.exh$mean)^2))

    ## joint local design gradient-based search via ALC
    j.opt <- laGPsep(WW, 6, 100, X, Y, method="alcopt", close=10000, lite=FALSE)
    rmse.opt[t] <- sqrt(mean((WY - j.opt$mean)^2))

    ## joint local design exhaustive search via NN
    j.nn <- laGPsep(WW, 6, 100, X, Y, method="nn", close=10000, lite=FALSE)
    rmse.nn[t] <- sqrt(mean((WY - j.nn$mean)^2))

    ## pointwise local design via ALC
    pw <- aGPsep(X, Y, WW, start=6, end=50, d=list(max=20), method="alc", verb=0)
    rmse.pw[t] <- sqrt(mean((WY - pw$mean)^2))

    ## pointwise local design via NN
    pw.nn <- aGPsep(X, Y, WW, start=6, end=50, d=list(max=20), method="nn", verb=0)
    rmse.pwnn[t] <- sqrt(mean((WY - pw.nn$mean)^2))

    ## progress meter
    print(t)
  }

  ## justify the y range
  ylim_RMSE <- log(range(rmse.exh, rmse.opt, rmse.nn, rmse.pw, rmse.pwnn))

  ## plot the distribution of RMSE output
  boxplot(log(rmse.exh), log(rmse.opt), log(rmse.nn), log(rmse.pw), log(rmse.pwnn),
          xaxt='n', xlab="", ylab="log(RMSE)", ylim=ylim_RMSE, main="")
  axis(1, at=1:5, labels=c("ALC-ex", "ALC-opt", "NN", "ALC-pw", "NN-pw"), las=1)

## End(Not run)
```

Index