Local Approximate SVD-based GP Models

January 29, 2018

`lasvdgp-package` *Local Approximate SVD-baed GP Models*

Description

Local approximate SVD-based GP model (lasvdGP) for large-scale dynamic computer experiments. This package intend to address the issue that the input set of the dynamic computer experiments is too large (as large as more than tens of thousands) for the full GP models to emulate. As is well known, the time complexity of fitting a GP model is $O(N^3)$ where N is the number of design points. To resolve the big N issue, we fit local SVD-based GP models on a small neighborhood set of every test inputs. The neighborhood sets are selected by the algorithm proposed in Zhang et al. (2017).

The neighborhood selection and SVD-based GP model fitting algorithm is convenient for parallelization. In this package, we provide two ways to parallelize the algorithm,

- parallelization via the R package "parallel". It is a process-level parallelism.
- parallelization via openMP. It is a thread-level parallelism.

The parallelization can achieve nearly linear speed since the procedure on each test point is independent and identical.

Author(s)

Ru Zhang <`14rz14@queensu.ca`>

References

Zhang, R., Lin, C.D. and Ranjan, P. (2017) *Local Gaussian Process Model for Large-scale Dynamic Computer Experiments*, arXiv preprint arXiv:1611.09488.

lasvdgpWorker *Local Approximate SVD-baed GP Models*

Description

Fit a local approximate SVD-based GP model with test set `X0`, design set `X` and response matrix `resp`. The local design consists of `nn` out of which `n0` points are selected by the Euclidean distance. If `n0` = `nn`, it performs the naive approach knnsvdGP. The functions `lasvdgpWorker` and `lasvdgpms` are executed in sequential way. The functions `lasvdgpParallel` and `lasvdgpmsParal` parallelize the execution via the R package "parallel". The functions `lasvdgpOmp` and `lasvdgpmsOmp` parallelize the execution by openMP.

Usage

```
lasvdgpWorker(X0, design, resp, n0, nn,
              nfea = min(1000,nrow(design)),
              nsvd = nn, nadd = 1,
              frac = .9, gstart = 0.001,
              resvdThres = min(5, nn-n0),
              every = min(5,nn-n0),centralize=FALSE,
              maxit=100, verb=0)
lasvdgpms(X0, design, resp, n0, nn,
          nfea = min(1000,nrow(design)),
          nsvd = nn, nadd = 1,
          frac = .9, gstart = 0.001,
          resvdThres = min(5, nn-n0),
          every = min(5,nn-n0),
          nstarts = 5,centralize=FALSE,
          maxit=100, verb=0)
lasvdgpParallel(X0, design, resp, n0, nn,
                nfea = min(1000,nrow(design)),
                nsvd = nn, nadd = 1,
                frac = .9, gstart = 0.001,
                resvdThres = min(5, nn-n0),
                every = min(5,nn-n0),centralize=FALSE,
                maxit=100, verb=0, nthread = 4, clutype="FORK")
lasvdgpmsParal(X0, design, resp, n0, nn,
               nfea = min(1000,nrow(design)),
               nsvd = nn, nadd = 1,
               frac = .9, gstart = 0.001,
               resvdThres = min(2, nn-n0),
               every = min(5,nn-n0),
               nstarts = 5,centralize=FALSE,
               maxit=100, verb=0,
               nthread = 4, clutype="FORK")
lasvdgpOmp(X0, design, resp, n0, nn,
           nfea = min(1000,nrow(design)),
           nsvd = nn, nadd = 1,
           frac = .9, gstart = 0.001,
           resvdThres = min(5, nn-n0),
           every = min(5,nn-n0),centralize=FALSE,
           maxit=100, verb=0,nthread=4)
lasvdgpmsOmp(X0, design, resp, n0, nn,
             nfea = min(1000,nrow(design)),
             nsvd = nn, nadd = 1,
             frac = .9, gstart = 0.001,
             resvdThres = min(5, nn-n0),
             every = min(5,nn-n0),nstarts=5,
             centralize=FALSE, maxit=100, verb=0,
             nthread=4)
```

Arguments

`X0`	An M by d matrix of test inputs, where M is the number of test points and d is the dimension of input. The neighborhood will be search and the SVD-based GP models will be fitted on every point (row) of `X0`.
`design`	An N by d matrix of design inputs, where N is the number of design points. The neighborhood points will be selected out of the points (rows) in `design`.
`resp`	An L by N response matrix of `design`, where L is the length of the time series outputs, N is the number of design points.
`n0`	The number of points in the initial neighborhood set. The initial neighborhood set is selected by the Euclidean distance.
`nn`	The total number of neighborhood points. The `nn-n0` points are selected sequentially by the proposed algorithm.
`nfea`	The number of feasible points within which to select the neighborhood points. This function will only consider the `nfea` design points closest to the test point in terms of Euclidean distance when selecting neighborhood points.
`nsvd`	The number of design points closet to the test points on whose response matrix to perform the initial singular value decomposition.
`nadd`	The number of neighborhood points selected at one iteration.
`frac`	The threshold in the cumulative percentage criterion to select the number of SVD bases.
`gstart`	The starting number and upper bound of for estimating the nugget parameter. If `gstart = sqrt(.Machine$double.eps)`, the nugget will be fixed at `sqrt(.Machine$double.eps)`.
`resvdThres`	The threshold to re-perform SVD. After every `resvdThres` points have been included into the neighborhood set, the SVD of the response matrix will be re-performed and the SVD-based GP model will be refitted.
`every`	The threshold to refit GP models without re-perform SVD. After every `every` points have been included into the neighborhood set, the GP models will be refitted. But the SVD will not be re-performed. It is suggested `every <= resvdThres`.
`nstart`	The number of starting points used in the numerical maximization of the posterior density function. The larger `nstart` will typically lead to more accurate prediction but longer computational time.
`centralize`	If `centralize=TRUE` the response matrix will be centralized (subtract the mean) before the start of the algorithm. The mean will be added to the predictive mean at the finish of the algorithm.
`maxit`	Maximum number of iterations in the numerical optimization algorithm for maximizing the posterior density function.

verb	A nonnegative integer indicates the level of printing on the screen. If `verb=0` the function is executed in silence.
nthread	The number of threads (process) used in parallel execution of this function.
clutype	The type of cluster in the R package "parallel" to perform parallelization.

Value

pmean	An L by M matrix of posterior predictive mean for the response at the test set `X0`
ps2	An L by M matrix of posterior predictive variance for the response at the test set `X0`

Author(s)

Ru Zhang <14rz14@queensu.ca>

Examples

```
## load the R package of simulation functions
library("simfuncs")
library("lhs")

timepoints <- seq(0,1,len=200)
design <- lhs::randomLHS(10000,3)
test <- lhs::randomLHS(2000,3)
## evaluate the response matrix on the design matrix
resp <- apply(design,forretal,1,timepoints)

n0 <- 15
nn <- 30
gs <- sqrt(.Machine$double.eps)
## knnsvdGP approach with neighborhood size nn, openMP parallelization
retnn <- lasvdgpOmp(test,design,resp,nn,nn,frac=.95,gstart=gs,
                    centralize=TRUE,nthread=4)
## lasvdGP approach with neighborhood size nn and initial neighborhood size n0,
## openMP parallelization
retgg <- lasvdgpOmp(test,design,resp,n0,nn,frac=.95,gstart=gs,
                    centralize=TRUE,nthread=4)
## knnsvdGP using R package "parallel" for parallelization
retnnp <- lasvdgpParallel(test,design,resp,nn,nn,frac=.95,gstart=gs,
                          centralize=TRUE,nthread=4,clutype="PSOCK")
## lasvdGP using R package "parallel" for parallelization
retggp <- lasvdgpParallel(test,design,resp,n0,nn,frac=.95,gstart=gs,
                          centralize=TRUE,nthread=4,clutype="PSOCK")
```