R packages rlibkriging and dolka

Yves Deville

2022, April

rlibkriging is an R package providing an interface to the *libKriging* library (Cpp Armadillo).

It allows

- to use kriging "à la *DiceKriging*", and also
- to write R code that is quite similar to the Python and Matlab/Octave code for the two other interfaces to the library.

These are two different goals!

For the first goal, an R interface quite similar to *DiceKriging* is provided.

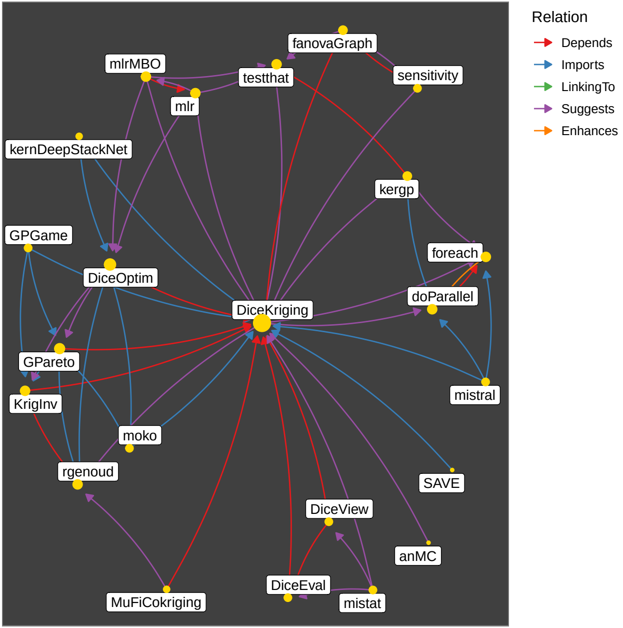
For the second goal there are/will be functionalities with no equivalent in *DiceKriging*, but some from other packages in R, Python or Matlab.

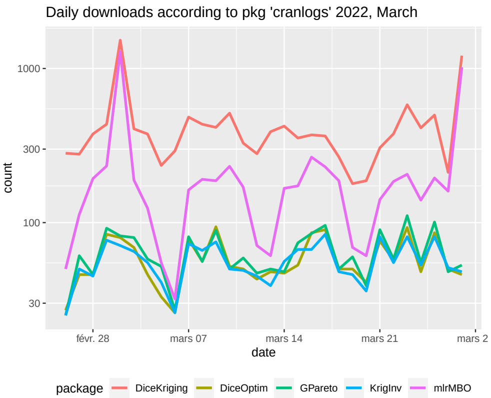
→ Use priors on correlation range as in *RobusGaSP*. Other references: *DACE*, *GPML*, *GPflow*, ...

Target use

Ideally, all R packages relying on *DiceKriging* for usual kriging steps could be made working with *libKriging* via its R interface *rlibkriging*. However, this requires a substantial amount of work and may take some time...

→ See package *dolka* later.


fanovaGraph
mlrMBO
testthat
sensitivity
mlr
kernDeepStackNet
kergp
GPGame
foreach
DiceOptim
doParallel
DiceKriging
GPareto
mistral
KrigInv
moko
rgenoud
SAVE
DiceView
anMC
DiceEval
MuFiCokriging
mistat
Relation
Depends
Imports
LinkingTo
Suggests
Enhances



Doing Bayes optimization is the main motivation to download *DiceKriging*!

R Interface Limitations

Due to the multi-language target of *libKriging*

- *rlibkriging* does not use formula or data frames.
- The designs **X** are numeric matrices.

  → Designs provided as data frames (as produced by `expand.grid`) are coerced into numeric matrices.
- The colnames of the designs are not used, so only the number of columns and the position are meaningful.

R pitfalls

Remind that R is not intended to be a matrix programming language, although it embeds matrix algebra.

If you are only an occasional R user, it may be better not to use data frames and stick to numeric vectors, matrices, arrays.

```
## bind the rows of two matrices
mat1 <- cbind(Before = c(1, 2), After = c(10, 11))
mat2 <- cbind(After = c(3, 4), Before = c(12, 13))
(mat <- rbind(mat1, mat2))

##      Before After
## [1,]      1    10
## [2,]      2    11
## [3,]      3    12
## [4,]      4    13

## bind the (coerced) data frames
df1 <- as.data.frame(mat1); df2 <- as.data.frame(mat2)
(df <- rbind(df1, df2))

##   Before After
## 1      1    10
## 2      2    11
## 3     12     3
## 4     13     4

c(mat = class(mat[2, ]), df = class(df[2, ]))

##          mat           df
##    "numeric" "data.frame"
```

data frame $\neq$ *matrix*

A data frame is a frame enclosing objects that are coped with by using their name. In most R packages, the columns of a data frame are not expected to be in a particular order. Moreover, a data frame contains usually more columns (variables) than we need.

$\rightarrow$ The same data frame can be used to compare models with different covariates

R pitfalls

Be careful

- When using `rbind`, `apply`, ...

  → Are you applying the function on a numeric vector or on a data frame? Will the elements be provided to the function in the right order?
- When preparing and using "new" data as required in `predict`, `simulate`, `update`.

This is especially true when kriging with $d = 4$ inputs or more with no natural order for them e.g. physical variables.

OO programming in R

R has several systems for Object-Oriented (OO) programming.
The S3 and S4 systems are used in *DiceKriging* and *rlibkriging*

In *DiceKriging* we find two main S4 classes.

- Class `"km"` for fitted "kriging" models. The main methods are `predict`, `simulate`, `update`.
- Class `"covStruct"` is for covariance kernels

  → Mainly for experimented users. Most users will only need to choose the name of the kernel: `"gauss"`, `"matern3_2"`, ...

Class "km" of DiceKriging

The methods for the class `"km"` are mainly classical methods as found for classes of "fitted models".

```
library(DiceKriging)
methods(class = "km")

## [1] coef      logLik    plot      predict   show      simulate
## [7] update
## see '?methods' for accessing help and source code
```

There are two creators for the class `"km"`: `km` and `kmData`.

→ The second one is closer to `lm`, because the response and the inputs are assumed to be in the same data frame passed through the `data` argument.

Classes in rlibkriging

The package *rlibkriging* exposes two classes for fitted "kriging models".

- S4 class `"KM"` which inherits from `"km"`.

  → Pretty much the same syntax as `"km"`, so quite R-specific.

- S3 class `"Kriging"`.

  → Intended to be only loosely related to R, in order to increase the inter-language consistency across R, Python and Matlab/Octave.

In both cases, we can use the three main methods `predict`, `simulate` and `update`. Yet the formal arguments and the content of the output differ.

S3 vs S4

- S4 classes are formally defined. Objects haves *slots* such as `myKM@X`.
- S3 objects are mainly list with a `"class"` attribute, they are coped with by “gentleman’s rules”.

  → The method `predict` for the class `"Kriging"` is implemented via the function `predict.Kriging` which is called “internally”.

When a function is ("is registered as") a method, a *dispatch mechanism* is invoked at the call, based on the class of the first argument e.g. `object` for `predict` methods.

→ The name of the formal argument is the same across methods. All `predict` method have (or should have) their first formal argument name `"object"`.

The S3 and S4 systems are widely compatible so you may use methods without even knowing if they are S3 or S4.

→ There are quite technical functions such as `findMethod` to better understand what happens.

```r
library(rlibkriging)
methods(class = "KM")

## [1] predict  show     simulate update
## see '?methods' for accessing help and source code

methods(class = "Kriging")

##  [1] as.km         as.list       coerce        initialize
##  [5] leaveOneOut   logLikelihood logMargPost   predict
##  [9] print         show          simulate      slotsFromS3
## [13] update
## see '?methods' for accessing help and source code
```

In case of doubt, use `getS3method` for a an S3 method or `getMethod` for an S4 method

```
p3 <- getS3method(f = "predict", class = "lm")
p4 <- getMethod(f = predict, signature = "km")
```

We get the functions and we can see their code and know from what package they come.

→ It may happen that, for a given generic function and a given class, the method exist both as S3 and S4. This is the case for `predict` and `"km"`.

Inheriting from `"km"`

Since `"KM"` inherits from `"km"` it inherits all the methods implemented for `"km"` when the package *DiceKriging* is attached.

→ So the class `"KM"` has a `coef` method.

As a general rule do not call S3 methods directly, even when this is possible

good ☺ `predict(object = myObject)`

bad ☹ `predict.km(object = myObject)`

With the second syntax, we "hard code" that `myObject` must have class `"km"`. We also disable the use of possibly inherited method.

As far as possible, use methods to extract a slot or element in an object.

→ E.g. use `logLik(myObj)` method rather than `myObj$logLik` or `myObj@logLik`.

This makes robustness w.r.t. internal changes.

Mind that two different systems of formal arguments and of output names are used for `"KM"` and `"Kriging"`, ...

→ See the help of the two related `predict` methods.

Writing wrappers between the two naming systems is useless since the methods of `"KM"` already are wrappers for methods of `"Kriging"`.

dolka

The *DiceOptim* package is devoted to kriging-based Bayesian Optimization relying on *DiceKriging*. It could be made working with the library *libKriging* via *rlibkriging*.

In a first stage, a few functions of *DiceOptim* have been re-factored and included into a new package *dolka*.

→ Maybe "DiceOptim with LibKriging Alternative".

dolka

The package *dolka*

- Should work with *DiceKriging* and *rlibkriging* thanks to the use of methods.
- Refactors code from *DiceOptim* in order to make it less dependent on historical artefacts.

  → The Bayes criteria such as `EI` optionally provide the gradient with a straightforward syntax and output structure.
- Is a transitional/temporary package since the job should be done by/with experts in Bayesian Optimization.

Bayesian Optimization

A costly-to-evaluate function $f(\mathbf{x})$ is to be minimised over a domain in $\mathbb{R}^d$, say an hypercube.

Given a number of n evaluations $f(\mathbf{x}_i)$ at n design points $\mathbf{x}_i$ for $i = 1$ to n, we can use kriging to guess one or several "new" inputs $\mathbf{x}^{\text{new}}$ where the function is likely to take a small value. Then the new input(s) can be included in the design is a sequential fashion.

Considering $\{f(\mathbf{x})\}_{\mathbf{x}}$ as a path of a GP, kriging provides us with the distribution of $f(\mathbf{x}^{\text{new}})$ conditional on $\mathbf{f}(\mathbf{x}_{1:n}) := [f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)]^\top$.

The improvement is then the non-negative random variable

$$I(\mathbf{x}) := \left[\min_{1 \leqslant i \leqslant n} f(\mathbf{x}_i) - f(\mathbf{x}) \right]_+$$

where $z_+ := \max\{z, 0\}$ is a positive part of z. Its distribution is the mixture of truncated Gaussian and of a probability mass at zero.

We want to get the largest possible distribution for the improvement $I(\mathbf{x})$ according to some (partial) order on distributions.

→ The distribution being conditional on the known points.

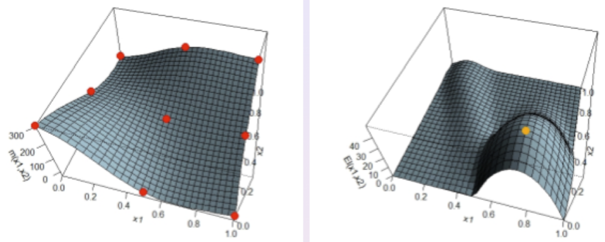
A good choice for $\mathbf{x}^{\text{new}}$ is the/a value maximizing the Expected Improvement

$$\mathrm{EI}(\mathbf{x}) := \mathbb{E}\left\{ I(\mathbf{x}) \,\middle|\, \mathbf{f}(\mathbf{x}_{1:n}) \right\}$$

We could use as well a quantile of the improvement, or some other criterion of improvement to be maximized.

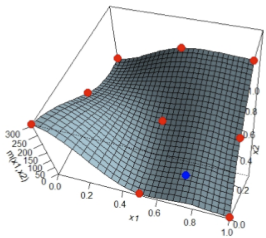
A 2D example (Branin function)

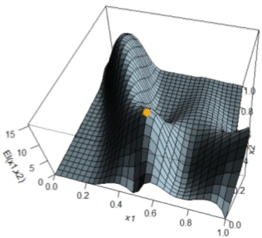
Left: Kriging model surface. Right: EI surface.



A 2D example (Branin function)

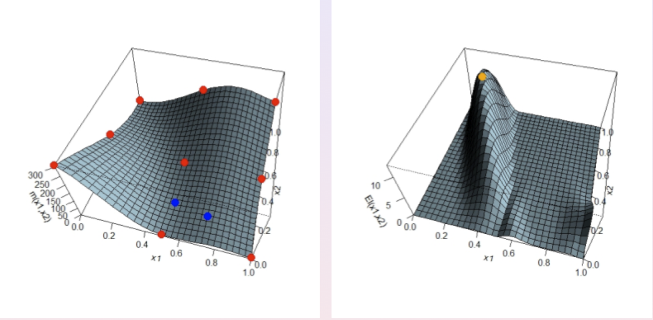
Left: Kriging model surface. Right: EI surface.

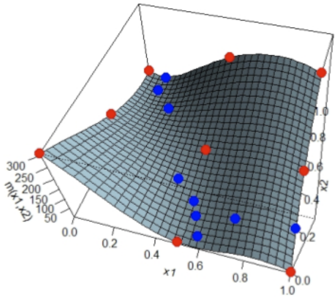




A 2D example (Branin function)

Left: Kriging model surface. Right: EI surface.





Initial designs (red points) and "new" designs added across iterations (blue points).

This illustration was provided by the authors and contributors of *DiceOptim*.

EGO (Efficient Global Optimization) algorithm

Given an initial design $\mathbf{X}$ and the corresponding vector of responses $\mathbf{f}$, we fit a kriging model .

Then repeat the following steps.

1. Find a design $\mathbf{x}^\star$ that maximizes the criterion.

   → We have to optimize a function of $\mathbf{x}$ which depends on the current model.

2. Evaluate the function $f(\mathbf{x})$ for $\mathbf{x} = \mathbf{x}^\star$.

3. Update the kriging model with the new point.

   → Add the new row $\mathbf{x}^\star$ to $\mathbf{X}$ and the new response $f(\mathbf{x}^\star)$ to the vector $\mathbf{f}$.

Stop if the improvement is small or if our evaluation budget is consumed.

Making things work

Using methods for the specific class of Kriging model that we want to use.

- *In step 1* the criterion function has formal arguments `x` and `model`. It calls the `predict` method.
- *In step 3*, we use the `update` method for the class of kriging models.

We can use `"km"` of *DiceKriging* or `"KM"` from *rlibkriging*. And more...

Optimization of the Criterion of Improvement

The criterion of improvement $C(\mathbf{x}^{\text{new}})$ to be maximized (say) can be computed using the values at $\mathbf{x}^{\text{new}}$ of the kriging mean $m(\mathbf{x})$, and the kriging variance $\sigma^2(\mathbf{x})$.

→ These are the *conditional* expectation and variance.

The criterion to be maximized has often many local maxima.

→ Roughly speaking there are as many local maxima as they are "holes" between the design points.

We want to perform a *global* optimization rather than a *local* one.

→ Genetic algorithms (as in packages *rgenoud*), CMAES as in package *cmaes*, ... or custom methods.

Using derivatives

It can help to use the derivatives of the criterion (gradient w.r.t. $\mathbf{x}$).

Since $m(\mathbf{x})$ and $\sigma^2(\mathbf{x})$ are standard output from the `predict` method, we can simply enhance this method so that it optionally computes the gradients as well.

→ When computed with the other kriging results, the gradients come at a small cost.

Using derivatives in `predict`

For now

- *DiceKriging* and *DiceOptim* do not rely on `predict` to pass the derivatives.

  → This makes *DiceOptim* strongly dependent on *DiceKriging*.

- *dolka* overloads (or "masks") the `predict` method from *DiceKriging* to add the derivatives.

  → This possibly confusing behaviour will no longer be needed it the authors of *DiceKriging* accept a small change in `predict`.

- *rlibkriging* will soon provide suitably enhanced `predict` methods for the classes `"KM"` and `"Kriging"`.

So we can easily switch from one `predict` to another, possibly relying on *libKriging*.

Update

The second key ingredient in kriging-based Bayes optimization is the `update method`.

Once the objective function $f(\mathbf{x})$ is evaluated at $\mathbf{x}^{\text{new}}$ we have to incorporate the new input point in the kriging design, possibly re-estimating the parameters (correlation ranges and variance). The fitted model object then embeds $n+1$ points $\mathbf{x}_i$ with $\mathbf{x}_{n+1} := \mathbf{x}^{\text{new}}$.

Roadmap

- Allow the use of different classes of kriging objects, including `"km"` and `"KM"`.

  → Using methods: `predict` and `update`.

- Allow the use of different optimization functions to maximize the chosen criterion.

  → `stats::optim`, `rgenoud::genoud`. But why not use *nloptr* or other packages?

	DiceOptim	*dolka*
DiceKriging "km"	max_EI max_qEI and much more.	max_EI_genoud, max_EI_cmaes* max_qEI_genoud
rlibkriging "KM"	not available yet requires changes in *DiceOptim*	max_EI_genoud, max_EI_cmaes* max_qEI_genoud

Table: Optimizers for criteria of improvement, available or soon available. Names of optimizers not requiring derivatives are starred $\star$.

Your turn

We will perform a few iterations of the EGO algorithm with EI criterion the `branin` function.

→ Available as `DiceKriging::branin`.

The goal is to a flexible implementation where we can use either *DiceKriging* or *rlibkriging* and we can easily switch from *DiceOptim* to *dolka*.

Your turn

Depending on your experience in Bayesian Optimization, you can find `EGO0.R` (minimal help), `EGO1.R`, ...

The suggested schedule is to write your program from scratch before using any help.

References I