Using the `Infusion` package

François Rousset

August 21, 2019

This is a gentle introduction to the R package `Infusion`, which name stands for `Inf`erence `us`ing simula`tion`. `Infusion` implements a method called summary likelihood, suitable for statistical inference based on simulation of processes for which the likelihood function is not available. Summary-likelihood inference is basically the same idea as discussed by Diggle & Gratton (1984) in the first dedicated discussion of generic approaches for simulation-based inference. Summary likelihood is not full-data likelihood but is still a form of likelihood, which one can evaluate if the full data have been thrown away and only some summary statistics have been retained. The following simple examples show both the logic of the method and how to perform inference with the package functions. After illustrating the basic workflow, it shows how different "projection methods" can be used to reduce the number of summary statistics.

This document actually describes two variants of the basic idea, differing in the way they estimate the likelihood surface and in the simulation design they use for that purpose. One variant, described in Rousset *et al.* (2017) (and closest to Diggle & Gratton, 1984), first estimates the distribution of summary statistics for a relatively small number of parameter points, using a (moderately) large number of simulation of the data-generating process for each parameter point. The other variant uses a table of simulations containing a single simulation of the data-generating process for each of large number of parameter points. This simulation design is the current focus of development, as yet-unpublished simulations seem to confirm that it can be more efficiently exploited than the previous one. Therefore, it is the main focus of exposition here, and the original method is described afterwards (Section 2).

Contents

1 The workflow from simulation to inference

The examples were run with R Under development (unstable) (2019-06-11 r76694) and `Infusion` 1.4.3.

1.1 Toy example with most informative statistics

Our first example should be fast, easily visualized, and easily comparable to alternative methods. We will therefore estimate the mean and variance of a Gaussian distribution from the sample mean and sample variance of 40 Gaussian deviates. Since the statistics considered each contain all information about each of the parameters of the Gaussian distribution, we expect the results to be practically equivalent to standard likelihood-based inference.

We first simulate these two summary statistics in a format appropriate for interaction with the package functions. We define the function that returns the summary statistics

```
## the summary-statistic function
myrnorm <- function(mu,s2,sample.size) {
  s <- rnorm(n=sample.size,mean=mu,sd=sqrt(s2))
  return(c(mean=mean(s),var=var(s)))
}
```

and we produce a realization that will stand for the actual data to be analyzed:

```
set.seed(123)
ssize <- 40 # sample size through all examples
Sobs <- myrnorm(mu=4,s2=1,sample.size=ssize) ## stands for the ac-
tual data to be analyzed
Sobs
```

```
##  mean   var
## 4.045 0.806
```

from which we can immediately say that the ML estimate of the mean is 4.05, and the ML estimate of the variance is `Sobs["var"]*(ssize-1)/ssize`=0.786.[1]

The Student t-based exact 95% confidence interval for the mean μ is [3.76, 4.33]. It uses the exact distribution of t given the sample variance. Alternatively, the more generally available χ^2 approximation for the distribution of the (profile) log-likelihood ratio (LR) may be used to construct approximate LR confidence intervals. For the mean, this interval is [3.77, 4.32]. Likewise, confidence intervals for the variance can be constructed using the same profile LR approximation, or the exact distribution of the sample variance. The exact interval for σ^2 is [0.541, 1.33] and the approximate one is [0.522, 1.26]. The approximate CIs appear similar to the exact intervals, except for the upper bound of the variance.

We will practically recover the likelihood ratio CIs by simulation. For that purpose, we will explore the parameter space to evaluate the probability density of the statistics for different parameter values. As shown in the next section, this exploration is iterative, a first step providing first estimates from which new parameters points are chosen, for which new simulations are performed, and refined estimates are produced. In approximate Bayesian computation, (vector-valued) parameters are sampled from a prior distribution, and one data sample is drawn for each (vector-valued) parameter. The same type of input is handled here. We first load the package and set some options:

```
options(digits=3)
library(Infusion)
# to use parallelisation when running infer_logLs():
Infusion.options(nb_cores=7) # Adapt this to your computer.
```

We use the `add_reftable` function from the package to build the simulation table,

```
set.seed(123)
# Uniform sampling in parameter space:
npoints_j <- 600
ssize_j <- 40
```

[1] The likelihood of the data can be expressed in terms of the sample mean and variance as the normal density of the mean, times a χ^2_{n-1} density for a rescaled variance, times a correction for the rescaling (Davison, 2003, pp. 66,75). The log-likelihood is

```
logLnorm <- function(mu,s2,Sobs) { dnorm(Sobs["mean"],mean=mu,sd=sqrt(s2/ssize),log=TRUE)+
dchisq((ssize-1)*Sobs["var"]/s2,df = (ssize-1),log=TRUE)+log((ssize-1)/s2) }
```

But the methods described here ignore such specific results.

```
parsp_j <- data.frame(mu=runif(npoints_j,min=2.8,max=5.2),
                      s2=runif(npoints_j,min=0.4,max=2.4),sample.size=ssize_j)
# Build simulation table:
simuls_j <- add_reftable(Simulate="myrnorm",par.grid=parsp_j)
```

Here the simulation function `myrnorm` is available in the `R` session to produce samples from the distribution of the summary statistics. In more involved applications the simulation code may not be callable from `R`. This case is also handled by `add_reftable`, using its `newsimuls` argument. We then use the function `infer_SLik_joint` to build the first inference of the likelihood surface

```
# Infer surface:
slik_j <- infer_SLik_joint(simuls_j,stat.obs=Sobs)

## Densities modeling: 600 points;
##  joint density modeling: 7 clusters;
```

`slik_j` is an object of class `SLik_j` (`slik`, pronounced "sleek", refers to `s`ummary `lik`elihood, and "`j`" refers to the `j`oint density of parameters and statistics, used together with the prior density to construct the likelihood surface). Parameter inference can then be performed as if the summary likelihood surface was a full-data likelihood surface. In particular, we can obtain from it the "maximum summary likelihood" (MSL) estimate as well as confidence intervals, using "profile summary-likelihood ratios":

```
slik_j <- MSL(slik_j) ## find the maximum of the log-likelihood sur-
face

## 600 simulated samples;
## * Summary ML: *
##    mu    s2  logL
## 3.992 0.823 1.993
## Computing RMSEs... (may be slow)
## *** Interval estimates and RMSEs ***
##          low.mu  up.mu low.s2 up.s2
## par       3.750 4.3335 0.4892 1.432
## par_RMSE  0.024 0.0226 0.0232 0.132
## LR_RMSE   0.404 0.3817 0.2801 0.573
```

Information about the MSL estimates (and intervals) has been added to `slik_j`, which remains of class `SLik_j`. The `confint` function can also be used to compute intervals for given parameters (as in `confint(slik_j,"mu")`). We can visualize the result by `plot(slik_j,filled=TRUE)` (Figure 1).

The estimates have some random error since the probability densities on which they are based are themselves estimated with some error. We can refine these estimates iteratively, by

```
plot(slik_j,filled=FALSE)
```

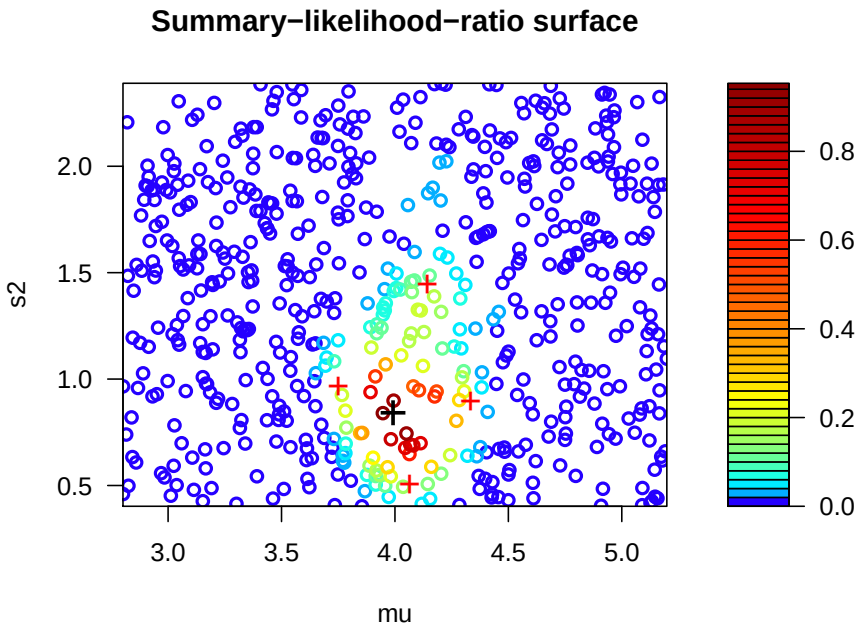


Figure 1: First inference of the likelihood surface
The black and red crosses mark respectively the estimated ML point and the confidence intervals points, that is, the outmost points on the contour defined by the profile likelihood threshold for the profile confidence intervals. There is a pair of CI points for each interval. The contour line delimits the estimated 95% confidence region for the two parameters.

```
maxit <- 5 ## maximum number of 'refine' iterations for all examples
slik_j <- refine(slik_j,filled=TRUE,verbose=FALSE,maxit=maxit)
slik_j

## *** Summary ML (6 iterations, 3600 points): ***
##        mu        s2      logL RMSE_logL
##     4.049     0.802     1.907     0.052
## *** Interval estimates and RMSEs ***
##          low.mu  up.mu low.s2  up.s2
## par      3.7596 4.2879 0.4697 1.2295
## par_RMSE 0.0177 0.0139 0.0301 0.0182
## LR_RMSE  0.2678 0.1396 0.2499 0.1344
```

The results are plotted in Figure 2 (generated by the command `plot(slik_j,filled=TRUE)`). These iterations have led us closer to the expected LR confidence intervals bounds. The `maxit` and `precision` arguments of `refine` can be used to control the number of iterations to perform. In particular, a given `precision` is compared to a root mean square error (RMSE) of estimation of the maximum log summary-likelihood and of profile log-likelihood ratios, and iterations will stop if the requested precision is reached for all points.

1.2 Toy example reconsidered: projecting summary statistics

The previous example was atypical in several ways. In particular it used the minimal number of statistics required to estimate the parameters. In this section we consider how to address the more relevant case where more statistics have been considered. in this case, it may be useful to reduce the number of statistics to the number of parameters. Such a reduction is useful because it will substantially reduce the computation time of the smoothing of empirical distribution of summary statistics, which might otherwise become prohibitive as the number of statistics increases. The need to reduce the number of summary statistics is also discussed in the ABC literature, where neural networks (Blum & François, 2010), boosting procedures based on weighting of different predictors (Aeschbacher *et al.*, 2012), random forests (Pudlo *et al.*, 2016), and simple linear regression (Fearnhead & Prangle, 2012) have been used. Kriging can also be used. We can refer to all relevant methods as "projection methods", and will illustrate some of them on the following example. The package provides a `project` function that acts as an interface for various methods that can be used for projection.

1.2.1 The workflow with projection

Random forests are **currently** the default method called by `project`, which is called as follows: It calls `ranger` from the `ranger` package.

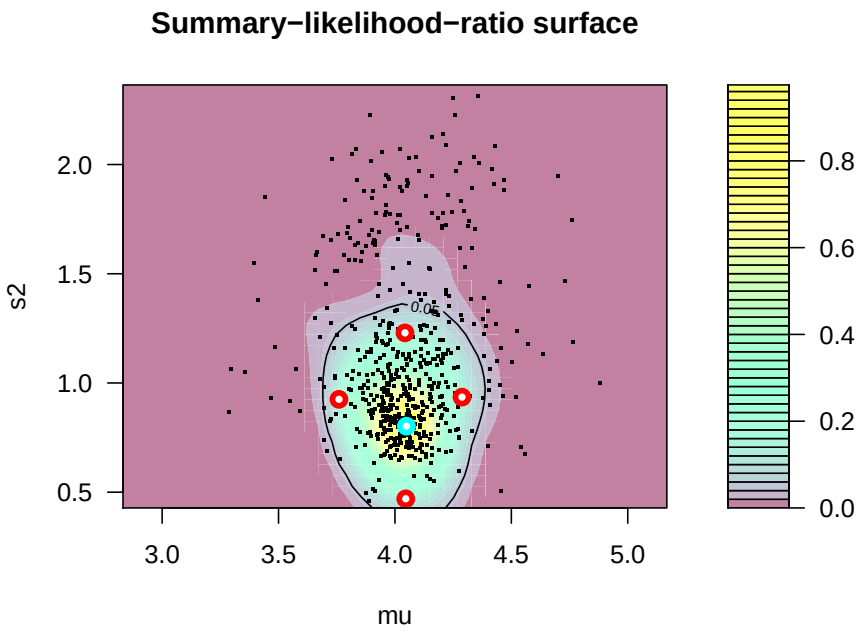
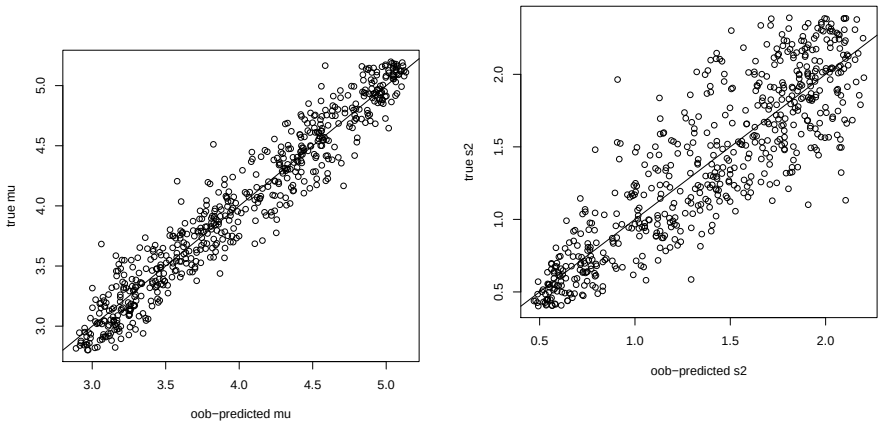


Figure 2: Refined inference of the likelihood surface
See legend of Fig. 1. The black dots mark points added in the latest iteration.

```
mufit <- project("mu",stats=c("mean","var"),data=simuls_j)
s2fit <- project("s2",stats=c("mean","var"),data=simuls_j)
```

```
## Selecting 'ranger' as default
method
```



The predictor for each parameter defines a new statistic that is to be used in further inferences. Ideally, the new statistic should be as close as possible to a bijection with the parameter. In other words, its expectation should be a monotonous function of the parameter, and it should have minimal variance (conditional on the parameter). The parameter will be poorly inferred in regions where the regression of the projected statistic to the parameter is flat, which sometimes occurs at the edge of the simulated parameter space (the simulated parameter range should therefore be extended if the inferences suggest this part of parameter space is relevant).

We then use a similarly named function, `project`, to create the observed value and the simulated distributions of the "projected" statistics from the original statistics.[2]:

```
projSobs <- project(Sobs,projectors=list(MEAN=mufit,VAR=s2fit))
projSimuls <- project(simuls_j,projectors=list(MEAN=mufit,VAR=s2fit))
```

This gives names `MEAN` and `VAR` to the projected statistics. These names can be chosen *ad libitum* by the user provided they are different from the names of the un-projected statistics, and should be used consistently over the different `project` calls.

[2]in R jargon, `project` is a "generic function" with different "methods" to construct the projections and to apply them, the appropriate method being selected on the basis of the type of first argument, here either a character string or simulated distributions.

Next we proceed as in the first toy example, but using the projected statistics. `refine` will be aware that projected statistics are used, because the information is kept in the input and output objects through all steps. In particular, it will perform simulation of new distributions of the un-projected statistics, then compute the projected statistics from them. The workflow from this point is therefore exactly the same as for un-projected statistics:

```
slik <- infer_SLik_joint(projSimuls,stat.obs=projSobs,verbose=FALSE)
slik <- MSL(slik, verbose=FALSE)
slik <- refine(slik,maxit=maxit,verbose=FALSE)
slik

## *** Summary ML (6 iterations, 3596 points): ***
##        mu        s2      logL RMSE_logL
##    4.1033    0.8038    1.0716    0.0711
## *** Interval estimates and RMSEs ***
##          low.mu  up.mu  up.s2
## par      3.8188 4.3554 1.2949
## par_RMSE 0.0308 0.0295 0.0512
## LR_RMSE  0.4029 0.2692 0.2962
```

See Fig. 3. The results are roughly similar to the first analysis. In general we could expect the confidence intervals to be wider than when the statistically most informative statistics are used, because the use of projection methods cannot in general recover such statistics. A further difference is that the MSE diagnostics of the likelihood surface may be higher than in the original analysis with true mean and variance, in which case more iterations would be required to obtain precise estimates of the surface.

It is possible to update the projections in each iteration, by calling `refine(., update_projectors=TRUE)`.

1.2.2 Handling possible overfitting of machine-learning methods

Our toy example uses the most informative (sufficient) statistics, the sample mean and variance. One can dream of a perfect projection method that recovers these statistics, but a "sufficient" projection method only needs to find a predictor of the parametric mean that is a bijection of the sample mean, and a predictor of the parametric variance that is a bijection of the sample variance. This result is here more closely achieved by building the projectors using Kriging (i.e., fitting a linear mixed model with a random effect autocorrelated in parameter space; here fitted using restricted likelihood methods).

```
REmufit <- project("mu",stats=c("mean","var"),data=simuls_j,method="REML",verbose=FALSE)
REs2fit <- project("s2",stats=c("mean","var"),data=simuls_j,method="REML",verbose=FALSE)
```

with the following relationships between original and projected statistics: It follows that the results of the analyses using such projections are are hardly

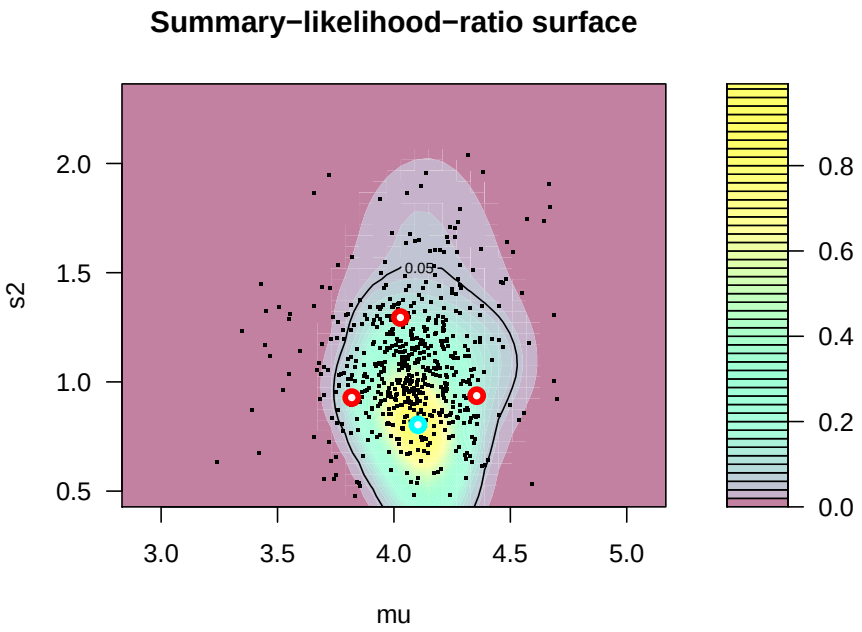
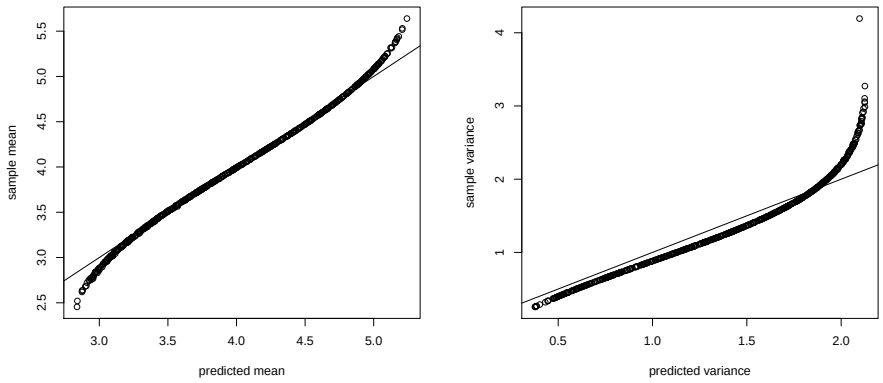


Figure 3: Refined inference of the likelihood surface, using initial projectors

```
plot(predict(REs2fit,newdata=simuls_j),simuls_j$var,xlab="predicted variance",ylab="sample v
abline(0,1)
```



different from the results without projections. No overfitting can happen in this case.

A variant of Kriging which performs similarly but may be faster is available by `project(...,method="GCV")` (which uses generalized cross-validation instead of REML).

However, Kriging is not the most appropriate or convenient projection method for all applications. So we reconsider our example using random forests. In that case, the predictions for the parameters correlate less well to the sufficient statistics than when using the REML method: This means that these predictions are not the sufficient statistics and are therefore less efficient. Confidence intervals based on them should be wider than those based on using sufficient statistics.

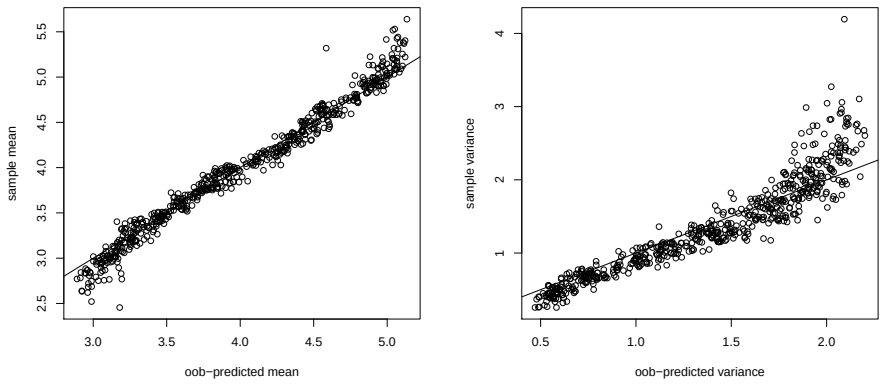
The above plots show out-of-bag predictions for the training samples. `Infusion` uses out-of-bag predictions whenever relevant on `ranger` (or `randomForest`) objects. By contrast, `predict(s2fit,data=simuls_j)` would provide predictions that overfit the training sample, and confidence intervals based on them would be too narrow. This problem must be taken care of when using alternative projection methods.

Out-of-bag predictions are definitely better, but may not be distributed identically to predictions on data not used for training, in which case updating projectors in each iteration is not a good idea.

1.2.3 A slightly more realistic example using poorer summary statistics

We revise our example to show that the method provides reasonable results when we do not provide the best possible summary statistics. In particular, in

```
plot(mufit$predictions,simuls_j$mean,xlab="oob-predicted mean",ylab="sample mean")
abline(0,1)
plot(s2fit$predictions,simuls_j$var,xlab="oob-predicted variance",ylab="sample variance")
abline(0,1)
```



practical applications there are typically many summary statistics, and it worth trying to construct a new efficient statistic for each parameter by combining the non-efficient ones by a projection method. Here, however, for expository purposes, we retain the other simple features of the two-parameter Gaussian model, but we compute "blurred" means and variances, for example the mean and variance of an exponential transformation of the normal random deviates:

```
blurred <- function(mu,s2,sample.size) {
  s <- rnorm(n=sample.size,mean=mu,sd=sqrt(s2))
  s <- exp(s/4)
  return(c(mean=mean(s),var=var(s)))
}
```

We accordingly consider a blurred version of our original data:

```
set.seed(123)
dSobs <- blurred(mu=4,s2=1,sample.size=40) ## stands for the actual data to be analyzed
```

and new simulated distributions

```
dsimuls <- add_reftable(,Simulate="blurred",par.grid=parsp_j,verbose=FALSE)
```

We now apply the workflow that seems the best according to our previous experiments:

```
mufit <- project("mu",stats=c("mean","var"),data=dsimuls,verbose=FALSE)
s2fit <- project("s2",stats=c("mean","var"),data=dsimuls,verbose=FALSE)
dprojectors <- list(MEAN=mufit,VAR=s2fit)
dprojSobs <- project(dSobs,projectors=dprojectors)
dprojSimuls <- project(dsimuls,projectors=dprojectors,verbose=FALSE)
dslik <- infer_SLik_joint(dprojSimuls,stat.obs=dprojSobs,verbose=FALSE)
dslik <- MSL(dslik, verbose=FALSE)
dslik <- refine(dslik,maxit=maxit,verbose=FALSE)
#
trainsample <- sample(nrow(dslik$raw_data),1000)
mufit <- project("mu",stats=c("mean","var"),data=dslik$raw_data[trainsample,],verbose=FALSE)
s2fit <- project("s2",stats=c("mean","var"),data=dslik$raw_data[trainsample,],verbose=FALSE)
tdprojectors <- list(MEAN=mufit,VAR=s2fit)
tdprojSobs <- project(dSobs,projectors=tdprojectors)
tdprojSimuls <- project(dslik$raw_data[-trainsample,],projectors=tdprojectors)
tdslik <- infer_SLik_joint(tdprojSimuls,stat.obs=tdprojSobs,verbose=FALSE)
tdslik <- MSL(tdslik)

## 2602 simulated samples;
## * Summary ML: *
##    mu    s2  logL
## 4.095 0.679 3.047
## Computing RMSEs... (may be slow)
## *** Interval estimates and RMSEs ***
##          low.mu  up.mu up.s2
## par      3.8220 4.3438 1.254
## par_RMSE 0.0226 0.0175 0.159
## LR_RMSE  0.3107 0.2872 0.347
```

2 The more primitive inference method

In this method, one tries to estimate the distribution of the summary statitics independently for a more limited number of parameter points. The function `init_grid` can be used to generate a first set of parameter points:

```
parsp <- init_grid(lower=c(mu=2.8,s2=0.2,sample.size=40),
upper=c(mu=5.2,s2=3,sample.size=40))
```

It does this in a way suitable for the inference of a likelihood surface, by generating an irregular grid of parameter values, with duplicate values for some parameters. We use the `add_simulation` function from the package to build a list of simulated distributions from this set of parameter values:

```
osimuls <- add_simulation(NULL,Simulate="myrnorm",par.grid=parsp,verbose=FALSE)
```

For each parameter point, `add_simulation` has simulated an empirical distribution (by default, of 1000 realizations). Here the simulation function `myrnorm` is available in the `R` session to produce samples from the distribution of the summary statistics. In more involved applications the simulation code may not be callable from `R`. This case is also handled by `add_simulation`, using its `newsimuls` argument.

We estimate the probability density ("likelihood") of the observed summary statistics for each simulated distribution:

```
densv <- infer_logLs(osimuls,stat.obs=Sobs,verbose=FALSE)
```

`infer_logLs` infers a probability density by smoothing the empirical distribution of the simulated statistics for each parameter value. From all estimated likelihoods, we estimate a summary likelihood surface by smoothing the density estimates:

```
slik <- infer_surface(densv) ## infer a log-likelihood surface

##
## Using REML to infer the S-likelihood surface...
```

`slik` is an object of class `SLik`. The workflow for such objects is identical to the one previously explained for `SLik_j` objects:

```
slik <- MSL(slik, verbose=FALSE) ## find the maximum of the log-likelihood sur-
face
slik <- refine(slik,filled=TRUE,verbose=FALSE,maxit=5)
slik

## *** Summary ML (6 iterations, 125 points): ***
##        mu        s2      logL RMSE_logL
##     4.032     0.747     1.822     0.045
## *** Interval estimates and RMSEs ***
##          low.mu  up.mu  low.s2   up.s2
## par      3.7615 4.3281 0.52145 1.25794
## par_RMSE 0.0034 0.0038 0.00211 0.00747
## LR_RMSE  0.0486 0.0495 0.04424 0.04832
```

The results are plotted in Figure 4 (generated by the command `plot(slik,filled=TRUE)`). The same functions previously applied to `SLik_j` objects, such as `plot` (see

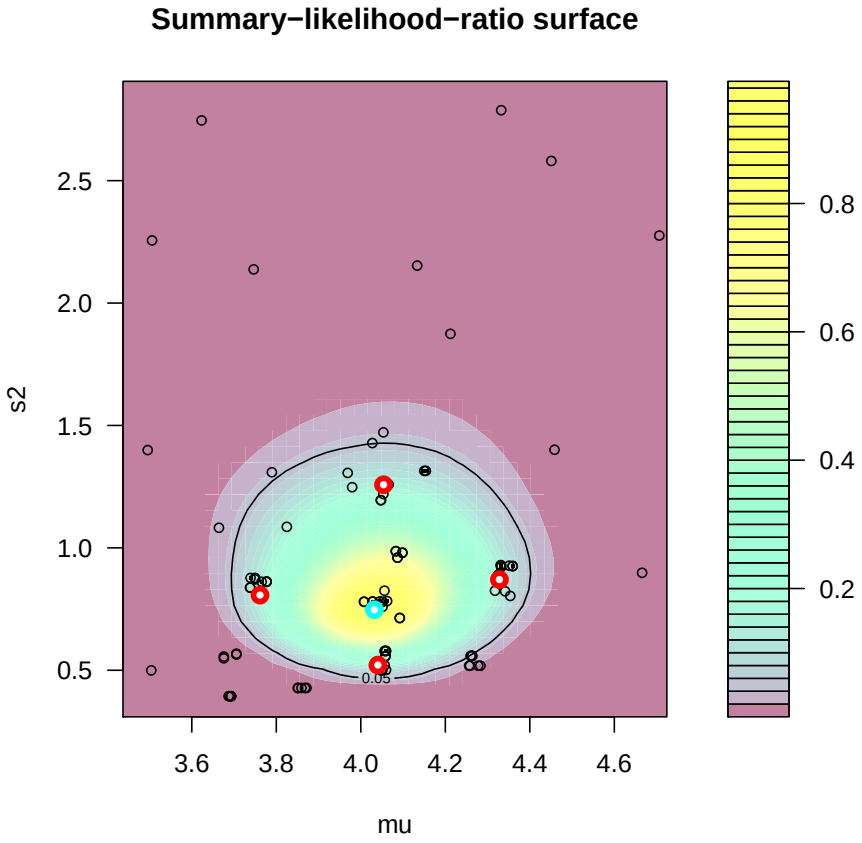


Figure 4: Refined inference of the likelihood surface
See legend of Fig. 3. The smaller black dots mark points added in the latest iteration.

Fig. 1), `refine`, etc., can indeed all be applied to `SLik` objects.

These iterations have led us much closer to the expected LR confidence intervals bounds. The `maxit` and `precision` arguments of `refine` can be used to control the number of iterations to perform until a certain precision of estimation of the maximum log summary-likelihood and of profile log-likelihood ratios is reached. In the present case, all the RMSEs shown in the last line of output are above the default precision sought, so if we increase `maxit`, further iterations will be run (but this is not shown in this first example).

3 How to...

3.1 How to infer composite parameters?

Suppose you have run simulations for a model with two parameters (say population size N and mutation rate μ) and wish to construct a confidence interval for their product (or the classical $\theta = 4N\mu$ parameter of population genetic models). You can write a function to compute the necessary likelihood profiles from a summary likelihood object, but if you are not into that, you can also modify a reference table of simulations previously used to make inferences about (N, μ), as follows. An `SLik_j` object holds a cumulative reference table as its `$raw_data` element. This table has columns for parameters and columns for summary statistics. Then you should *replace* one of the parameter columns by a column for the composite parameter, and use it as `newsimuls` input. Further, attributes of the table may also need to be modified.

3.2 How to update projections as new simulations are accumulated?

For this purpose, it is recommended to use the `refine(., update_projectors=TRUE)` (for the reference table method only), as it is otherwise easy to get this wrong.

3.3 How to fix some parameters instead of estimating them?

There is no need to define new functions in this case. Just include the fixed values of these parameters in the grid of parameters simulated. The package function will detect and handle constant parameters. Actually, the first example already worked in such a way. `sample_size` was fixed. It is not a statistical parameter one wish to estimate, but rather an argument of the simulation function. The package takes arguments of the simulation function from the parameter grid, and it assumes that the variable arguments are the parameters to be estimated.

3.4 How to test a parameter value?

If you want a good accuracy is the evaluation of the likelihood ratio for a target parameter value `parH0`, it may be useful to simulate several empirical distribu-

tions for this parameter. Note that the sensitivity of the p-value to uncertainty in the likelihood ratio is different for low and high p-values, so if you want a good accuracy is the evaluation of the p-value, more simulations may be necessary for large p-values. A crude but useful rule for determining the number of simulations to control uncertainty of p-value, using an object `slik`, is to estimate the current standard error of the p-value as

```
sd_p <- sqrt(slik$fit$phi) *
          dchisq( 2*( slik$MSL$maxlogL - predict(slik, newdata=parH0)[1] ), df=2)
```

where `slik$fit$phi` is a crude measure of uncertainty of log-likelihood estimates from each simulated distribution, and the `dchisq` call quantifies the sensitivity of the p-value to uncertainty in the likelihood ratio. Then to reduce the standard error of the p-value to (say) 0.005 you need to perform `ceiling(sd_p/0.005)` simulations:

```
nrep <- ceiling(sd_p/0.005)
gr <- as.data.frame(t(replicate(nrep, c(mu=4, s2=1, sample.size=ssize))))
simulsH0 <- add_simulation(NULL, Simulate="myfunction", par.grid=gr)
```

and update the object:

```
slik <- refine(slik, newsimuls=simulsH0)
predict(slik, newdata=parH0) ## new estimate of target likelihood
```

4 A closer look at the methods

4.1 Using alternative projection methods

4.1.1 Using neural networks as projectors

Deep learling methods can be used (`method="keras"`), though the implemententation is limited and yet little tested. One can also call neural network methods from the `caret` by the argument `method="neuralNet"` of `project` (this method is predefined by `Infusion`, and uses the `train` function from the `caret` package for cross-validation):

```
mufit <- project("mu",stats=c("mean","var"),data=osimuls,method="neuralNet",verbose=FALSE)

## projection data reduced according to 'knotnbr' argument.
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info
= trainInfo, :  There were missing values in resampled performance
measures.

s2fit <- project("s2",stats=c("mean","var"),data=osimuls,method="neuralNet",verbose=FALSE)

## projection data reduced according to 'knotnbr' argument.
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info
= trainInfo, :  There were missing values in resampled performance
measures.
```

The work flow is otherwise (exactly) unchanged.

```
projSobs <- project(Sobs,projectors=list("MEAN"=mufit,"VAR"=s2fit))
projSimuls <- project(osimuls,projectors=list("MEAN"=mufit,"VAR"=s2fit))
cdensv <- infer_logLs(projSimuls,stat.obs=projSobs,verbose=FALSE)
nnslik <- infer_surface(cdensv)

##
## Using REML to infer the S-likelihood surface...

nnslik <- MSL(nnslik,verbose=FALSE)
nnslik <- refine(nnslik,maxit=maxit,verbose=FALSE)

## Inaccurate MSE computation, presumably from nearly-singular co-
variance matrices.

nnslik

## *** Summary ML (6 iterations, 125 points): ***
##        mu        s2      logL RMSE_logL
##     4.039     0.816     1.646        NA
## *** Interval estimates and RMSEs ***
##           low.mu   up.mu low.s2  up.s2
## par      3.77066 4.32475 0.5246 1.2417
## par_RMSE 0.00448 0.00528 0.0043 0.0110
## LR_RMSE  0.06188 0.06463 0.0599 0.0676
```

See Fig. 5. The results are again close to those using the most informative statistics.

4.2 Density estimation

4.2.1 Using alternative density estimators

Many methods have been defined for density estimation from an empirical sample. For any of them, once the density is estimated, the log density of the observed summary statistics can be inferred. `infer_logLs` by default uses methods from the `Rmixmod` package to fit a mixture of gaussian distributions to the empirical distributions, and we do not encourage users to tinker with this (by default, `mclust` will be used if `Rmixmod` is not available). But it is possible to try alternative smoothing methods, using the `method` argument of the `infer_logLs` function.

In general, a wrapper function has to be defined for each density estimation method as different methods have different formal arguments, and as `infer_logLs` must output more than the estimated density. Some such wrappers are predefined in the package (see the `infer_logLs` documentation for more information about them).

```
plot(nnslik,filled=TRUE)
```

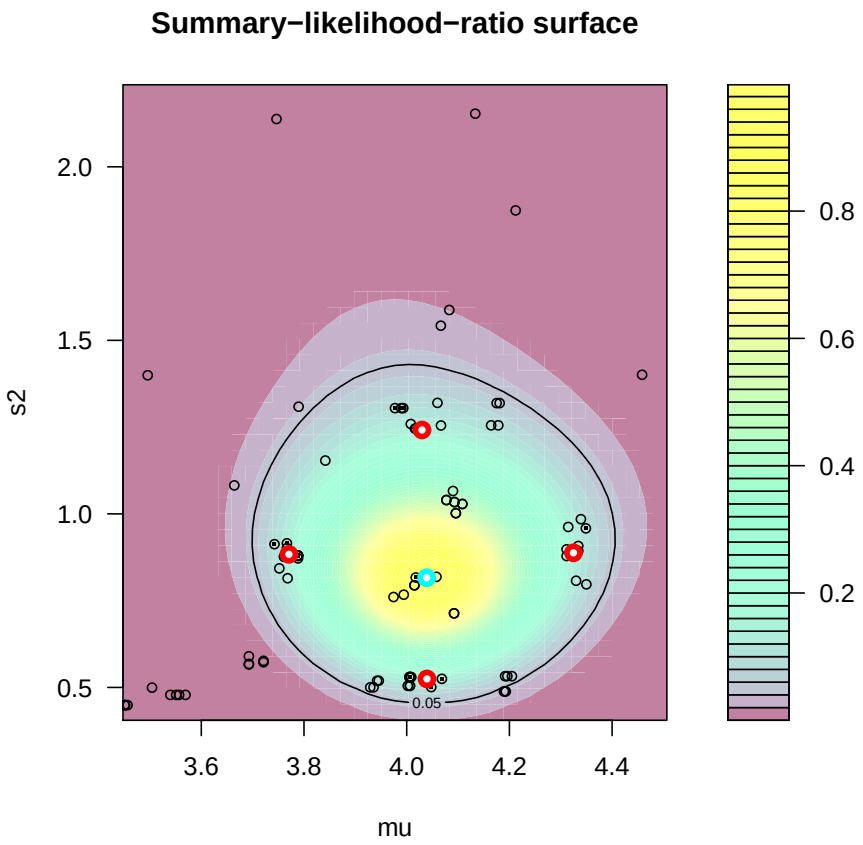


Figure 5: Refined inference of the likelihood surface, using neural networks as projectors

An example of such a wrapper, based on the `ks` (kernel smoothing) package is presented below. Note that this package which has many limitations that make it poorly suitable for simulation-based inference.

```
library(ks)
useks <- function(EDF,stat.obs,logLname,verbose) {
  chull <- resetCHull(EDF,formats=c("vertices","constraints"))
  if ( ! isPointInCHull(stat.obs,constraints=chull[c("a","b")])) {
    ## quick gaussian approximation
    obscov <- cov(EDF)
    obsmean <- colMeans(EDF)
    logL <- (-determinant(obscov)$modulus -log(2*pi)*length(obsmean)
             - (stat.obs-obsmean)%*%solve(obscov,(stat.obs-obsmean)) )
    logL <- logL/2
    isValid <- FALSE
  } else {
    #Hmat <- Hlscv.diag(EDF) ## bandwidth estimation
    Hmat <- Hscv.diag(EDF) ## bandwidth estimation
    fit <- kde(EDF,H=Hmat) ## smoothing
    logL <- log(predict(fit,x=stat.obs))
    isValid <- TRUE
  }
  names(logL) <- logLname
  unlist(c(attr(EDF,"par"),logL,isValid=isValid))
}
```

As shown here, the function must return the parameter values under which the empirical distribution was simulated, the inferred log density, and a boolean `isValid` which typically (as is the case here) indicates whether the observed summary statistics are within the convex envelope of the simulated empirical distribution (which is tested by the `spaMM::isPointInCHull` function). `isValid` can be set always to `TRUE` if you wish to skip this test. However, the test is useful as it is generally not useful to estimate the density when the observed summary statistics are out of the convex hull. In that case, a crude approximation of the likelihood (such as given by a multivariate normal density, as shown in the example) is sufficient. Further methods can readily be implemented by changing only the `bandwidth estimation` and `smoothing` lines in the above code, provided the density estimator has a `predict` method. A `predict` method is often provided by the developers of the packages implementing the methods, although sometimes one as to provide one (e.g., for `Rmixmod`).

We then proceed by using `infer_logLs(....,method="useks")` and the usual work flow. Note the additional `packages` argument naming additional packages to be loaded on the cores for a parallel computation:

```
densks <- infer_logLs(osimuls,stat.obs=Sobs,method="useks",verbose=c(FALSE,TRUE),
                      packages="ks")

## 35 distributions tagged as 'outlier'(s))

slikks <- infer_surface(densks)

##
## Using REML to infer the S-likelihood surface...

slikks <- MSL(slikks,verbose=FALSE) ## find the maximum of the log-
likelihood surface
slikks <- refine(slikks,maxit=maxit,verbose=FALSE)
slikks

## *** Summary ML (6 iterations, 126 points): ***
##        mu        s2      logL RMSE_logL
##     4.061     0.790     1.749     0.071
## *** Interval estimates and RMSEs ***
##           low.mu   up.mu  low.s2 up.s2
## par      3.75639 4.34025 0.53830 1.277
## par_RMSE 0.00998 0.00793 0.00521 0.026
## LR_RMSE  0.11538 0.10645 0.08807 0.112
```

The results (Fig. 6) are similar to those given by the previous methods, except that the intervals are slightly broader, suggesting (if anything) that the projected statistics are slightly less efficient.

Acknowledgements

The development of `Infusion` has benefitted from discussions with Jean-Michel Marin. Alex Gouy, Camille Martinez-Almoyna and Alex Courtiol have contributed to testing the first versions, and Raphael Leblois the more recent ones.

References

Aeschbacher, S., Beaumont, M. A., Futschik, A., 2012. A novel approach for choosing summary statistics in approximate Bayesian computation. *Genetics* **192**: 1027–1047.

Blum, M. G. B., François, O., 2010. Non-linear regression models for approximate Bayesian computation. *Stat. Computing* **20**: 63–73.

Davison, A. C., 2003. *Statistical models*. Cambridge Univ. Press.

Diggle, P. J., Gratton, R. J., 1984. Monte Carlo methods of inference for implicit statistical models. *J. R. Stat. Soc. B* **46**: 193–227.

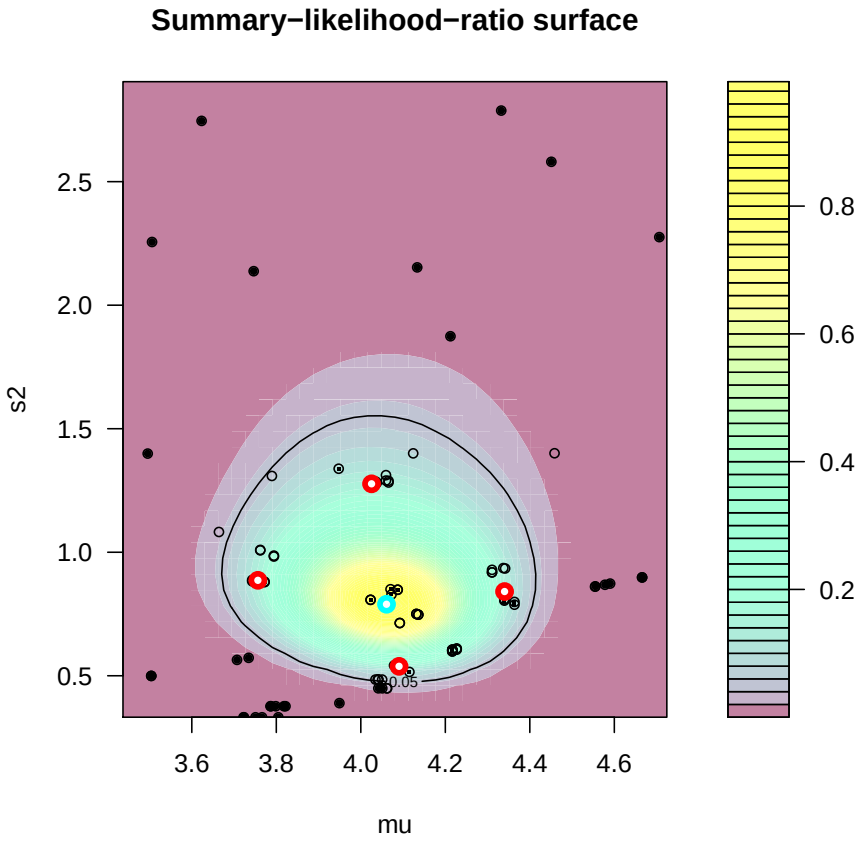


Figure 6: Inference of the likelihood surface after density estimation by kernel smoothing

Fearnhead, P., Prangle, D., 2012. Constructing summary statistics for approximate Bayesian computation: semi-automatic approximate Bayesian computation (with discussion). *J. R. Stat. Soc. B* **74**: 419–474.

Pudlo, P., Marin, J.-M., Estoup, A., Cornuet, J.-M., Gautier, M., Robert, C. P., 2016. Reliable ABC model choice via random forests. *Bioinformatics* **32**: 859–866.

Rousset, F., Gouy, A., Martinez-Almoyna, C., Courtiol, A., 2017. The summary-likelihood method and its implementation in the Infusion package. *Mol. Ecol. Res.* **17**: 110–119.