Interacting with the Nature index database and using custom uncertainty distributions

Jens Åström, Bård Pedersen

2019-10-08

Contents

Implementing custom distributions

The purpose of specifying an uncertainty of any indicator is to be able to define a distribution, from which we can draw samples to use in the calculations of the Nature index. Before 2019, uncertainty of an indicator value could only be expressed in by its lower and upper quartiles. Using the point estimate and the lower and upper quartiles, we then find the distribution with the best fit, in a process we call "elicitation". This works well for many cases, and not so well for other cases. Preferably, we should be able to avoid the elicitation and specify the uncertainty distribution attached to an indicator value directly, if such information exists. There are for example cases where an indicator has a known distribution, or where we have a set of empirical values, perhaps from a model simulation or an MCMC run. For the indicators where the user directly specifies the uncertainty distribution, we can now sidestep the elicitation and instead sample directly from these given distributions.

To implement this in the `NIcalc` package, we make use of the `distr` package to generate various types of distributions. At this point, we allow for three types of distributions:

1. Named known distributions (Log-normal and Poisson currently supported.)
2. Empirical distributions (Sample draws, e.g. CODA output from Bayesian modelling)
3. Discrete distributions (Allows for a fixed number of possible values, with individual probabilities specified)

The first option already have sampling functions in R (rnorm, rpois). However, to streamline the process, we handle all cases similarly, by the `distr` package. Currently, we also allow for the traditional specification of uncertainty as well, so the user has the choice of specifying the uncertainty of an indicator value as either lower and upper quartiles, or using one of these custom distributions.

Example of using known distributions

Log-Normal distribution

Let's say we have an indicator with a log point estimate of 0.5. The uncertainty is specified (in this example) as a lognormal distribution with standard error of 0.1. A user case might be if the value of an indicator is simulated or estimated as a lognormally distributed variable with a standard error. We could make a sampling function of this distribution quite easily using the `distr` package.

```
nMean <- 0.5
nSd <- 0.1

N <- distr::Lnorm(mean = nMean, sd = nSd)
sample10Values <- N@r(10)
sample10Values
```

```
##  [1] 1.666882 1.563633 1.591371 1.652988 1.581858 1.808623 1.723583 1.596776
##  [9] 1.886419 1.482139
```

As you see, the distr package is not the most intuitive. For convenience, we have implemented the function `makeDistribution` in the `NIcalc` package. This function is used to create all types of custom uncertainties.

```
N <- makeDistribution(input = "logNormal", distParams = list(mean = 0.5, sd = 0.1))
```

To sample from the distribution, we use the function `distr::r` under the hood, as shown above. We wrap this in a function that we can use for any distribution.

```
sampleDistribution
```

```
## function(dist, nSamples = 10){
##   out <- distr::r(dist)(nSamples)
##   return(out)
## }
## <bytecode: 0xc425118>
## <environment: namespace:NIcalc>
```

```
sampleDistribution(N, 10)
```

```
##  [1] 1.414803 1.415585 1.848351 1.591335 1.562265 1.674117 1.866828 1.764209
##  [9] 1.824120 1.652853
```

Poisson distribution

Another potentially common user case is an indicator value that is expressed as a Poisson variable. Examples might be simulations of a population count. In this case, the distribution can be expressed using only the λ variable.

```
P <- makeDistribution("Poisson", distParams = list(lambda = 5))
```

```
sampleDistribution(P, 10)
```

```
##  [1] 7 7 7 4 9 2 5 3 9 6
```

The distribution object (`N` or `P`) can be stored in the Nature index database together with the indicator value and later be sampled directly from. (More on the database interaction below.)

Using empirical distributions

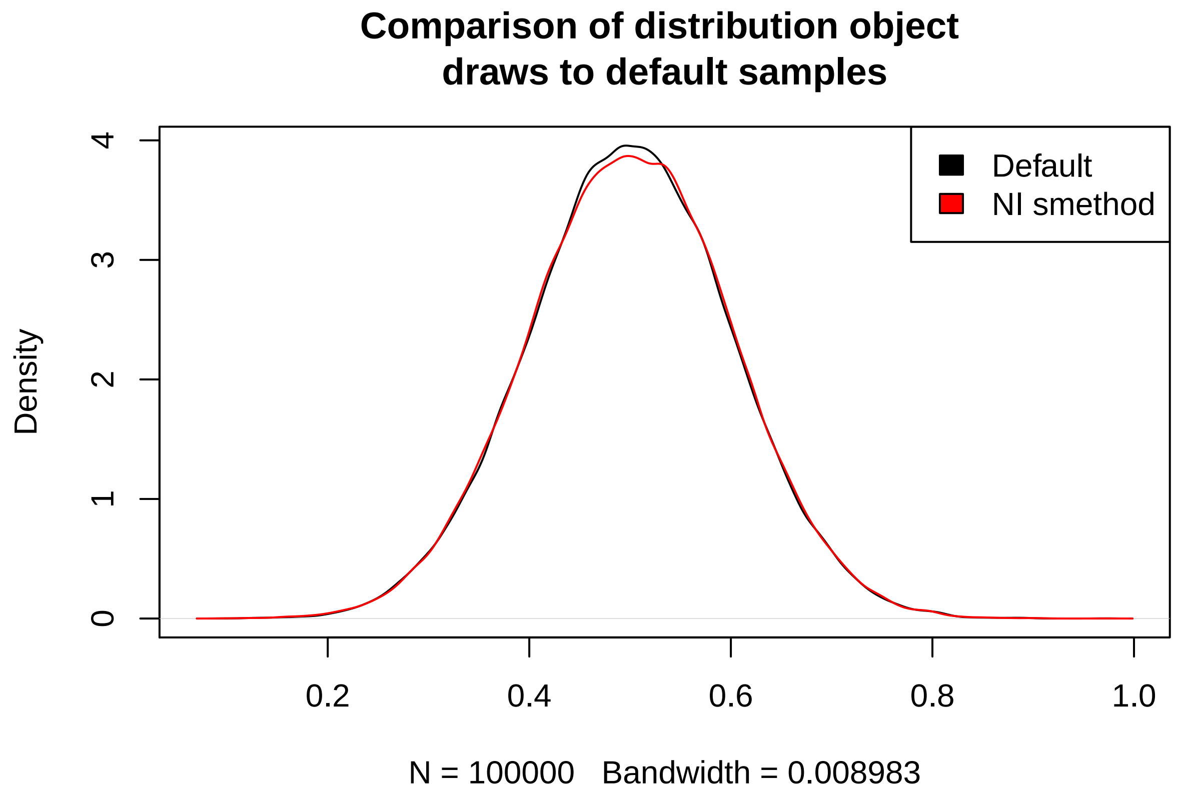
Empirical and discrete distributions can be handled similarly. Suppose we have output from a Bayesian MCMC model for a parameter, and we wish to use this to sample from. We could of course just do a `sample(codaObject)`, but here we use the `distr` package instead to keep us in a unified framework.

```
# Simulate an output from a MCMC run.
nSamples <- 1e+05
indicatorPosterior <- rnorm(nSamples, mean = 0.5, sd = 0.1)
E <- makeDistribution(indicatorPosterior)

sampleDistribution(E)
```

```
##  [1] 0.6048635 0.5314504 0.4733210 0.4804315 0.4463247 0.3893617 0.4404757
##  [8] 0.5212580 0.4366518 0.4968962
```

And just to confirm that sampling from this recently created object gives the (approximate) same answer as sampling from the original samples.

```
nSamples <- 1e+05
sampleMethod <- indicatorPosterior[x = sample(1:length(indicatorPosterior), size = nSamples,
    replace = T)]

NImethod <- sampleDistribution(E, nSamples)
plot(density(sampleMethod), col = 1, main = "Comparison of distribution object \ndraws to default sampl
points(density(NImethod), col = 2, type = "l")
legend("topright", legend = c("Default", "NI smethod"), fill = 1:2)
```



In this case, we simply store the `E` object and retrive the point estimate from this distribution as shown above. The size of such an empirical distribution of 10^5 samples, is 0.68 Mb. This is manageable for the database.

Discrete distributions

A discrete distribution is handled much the same way. This is just a set of possible values, with associated probabilities for each value (For the empirical distribution, each individual value have the same probability of being drawn.)

```r
allowedIndicatorValues <- c(0.1, 0.2, 0.3, 0.4)
indicatorValueProbabilities <- c(0.1, 0.4, 0.4, 0.1)

discretePointEstimate <- mean(allowedIndicatorValues)

D <- distr::DiscreteDistribution(allowedIndicatorValues, indicatorValueProbabilities)

sampleDistribution(D)
```

```
##  [1] 0.3 0.3 0.3 0.3 0.2 0.1 0.4 0.4 0.3 0.3
```

The same can be done via the `makeDistribution`.

```r
allowedIndicatorValues <- c(0.1, 0.2, 0.3, 0.4)
indicatorValueProbabilities <- c(0.1, 0.4, 0.4, 0.1)

myProbs <- cbind(allowedIndicatorValues, indicatorValueProbabilities)

D <- makeDistribution(myProbs)

sampleDistribution(D, 10)
```

```
##  [1] 0.3 0.1 0.2 0.4 0.4 0.2 0.2 0.1 0.3 0.2
```

Like before, we then store the `D` object together with the point estimate.

Storing the values and the `indicatorData`-class.

R objects cannot be stored directly in the database as they are. They can, however, be transformed into a raw data string and then stored. An R object can be stored as an element in an R data frame, if you add the column as a list, but it is rarely a good idea. This means we handle these custom distribution objects a little differently than when we use only point estimates and lower and upper quartiles.

We use the `indicatorData` class to store the information in R of the indicator values. This object type is used to retrieve and send indicator values to the NI database. It is simply a list in two parts. The first is a dataframe mimicking the information in a database table containing the individual indicator values. The last column in this dataframe contains a unique identifier, refering to a custom distribution, for the indicator values where this is used. The second part of the list is another list containing the custom distributions related to the data frame in the first element. A user should interact with these types of objects mainly through dedicated functions in the package. This is exemplified below.

```r
# display an example data set here Doesn't go well with Rmarkdown!
str(indicatorData)
```

Working with the Nature Index database

We communicate with the Nature index database through a web based API. R has nice functionality for this through the `httr` and `jsonlite`-packages. The `NIcalc` package contains functions for communicating with the database where the heavy lifting is done by these packages. It is a complex task however, to anticipate all the various error messages from the API, and display something understandable in R. If you get incomprehensible error messages when using the functions that communicate with the database, bear with us, and please let us know of your problems.

We currently support the main task of updating/setting indicator values via a series of steps:

1. Connect to the database (using `getToken`)
2. Identify which indicators the user can alter (using `getIndicators`)
3. Retrieve current indicator values for a given indicator (using `getIndicatorValues`)
4. Setting new indicator values (using `setIndicatorValues`), in the same step, the `makeDistribution` function is called.
5. Transforming non negative distribution parameters to log-normal distribution parameters, using the `normal2logNormal` function.
6. Writing new indicator values to the database (using `writeIndicatorValues`)

The procedure is designed to work with one indicator at a time. Although it is not recommended, it should also be possible to alter several indicators in the same function calls. See the very end for examples of how to set the values for many indicators at the same time.

Example of updating values

We connect to the database using a "token" which is issued individually and temporarily for a connection. This is retrieved by providing a username and password to the database. Use the same credentials as for the web-page http://naturindeks.nina.no. See further information on that page regarding the username and passwords. The `getToken` function retrieves a token which is used in further functions. Typically, the user only runs `getToken` in the beginning of a session and need not specify the token later on. For these examples, we use the indicators for butterflies and bumblebees.

```
## Token successfully retrieved from https://ninweb17.nina.no
```

```
getToken("your.username", "secretPassword")
```

The user can the retrive a list of the indicator values he/she is responsible for, or is allowed to alter. This is done by the `getIndicators` function. The `id` variable is later used to identify the indicator we wish to work with.

```
myIndicators <- getIndicators()
myIndicators
```

```
##    id                  name
## 1 351 Dagsommerfugler i skog
## 2 359          Humler i skog
```

The next typical step is to retrieve the current values for a given indicator. This can be done for all historical values in the database by simply:

```
indicatorData <- getIndicatorValues(indicatorID = 351)
```

Usually, the user is only interested in the latest values and do not need to change values for earlier years. We can then specify the current year for which we want to set values (at the time of writing, 2018). In case the indicator ID's change, we can also refer to the names of the indicators this way.

```
indicatorData <- getIndicatorValues(indicatorID = myIndicators$id[myIndicators$name ==
    "Dagsommerfugler i skog"], year = 2018)
indicatorData
```

```
## $indicatorValues
##    indicatorId          indicatorName areaId
## 10         351 Dagsommerfugler i skog   7040
## 20         351 Dagsommerfugler i skog   7041
## 30         351 Dagsommerfugler i skog   7042
##                                      areaName yearId yearName verdi nedre_Kvartil
## 10    Dagsommerfugler skog Østfold Vestfold      9     2018   0.9           0.7
## 20 Dagsommerfugler skog Vest-Agder Rogaland      9     2018   0.3            NA
## 30           Dagsommerfugler skog Trøndelag      9     2018   0.4            NA
##    ovre_Kvartil datatypeId    datatypeName unitOfMeasurement
## 10            1          2 Overvåkingsdata         Enhetsløs
## 20           NA         NA            <NA>         Enhetsløs
## 30           NA         NA            <NA>         Enhetsløs
##                  customDistributionUUID distributionName distributionId
## 10                                 <NA>             <NA>             NA
## 20 6ea2679c-b4bd-4f7a-8e5e-9f137261f257             <NA>             NA
## 30 4c37b474-d3f9-4309-baed-6a7009feb327             <NA>             NA
##    distParam1 distParam2
## 10         NA         NA
## 20         NA         NA
## 30         NA         NA
##
## $customDistributions
## $customDistributions$`6ea2679c-b4bd-4f7a-8e5e-9f137261f257`
## Distribution Object of Class: DiscreteDistribution
##
## $customDistributions$`4c37b474-d3f9-4309-baed-6a7009feb327`
## Distribution Object of Class: DiscreteDistribution
##
##
## attr(,"class")
## [1] "indicatorData" "list"
```

We can set an indicator value using the `setIndicatorValues` function. Here, we use the standard way with a point estimate and upper and lower quartiles.

```
updatedIndicatorData <- setIndicatorValues(indicatorData, areaId = 7040, years = 2018,
    est = 0.9, lower = 0.7, upper = 1)
updatedIndicatorData
```

```
## $indicatorValues
##    indicatorId          indicatorName areaId
## 10         351 Dagsommerfugler i skog   7040
## 20         351 Dagsommerfugler i skog   7041
## 30         351 Dagsommerfugler i skog   7042
##                                      areaName yearId yearName verdi nedre_Kvartil
## 10    Dagsommerfugler skog Østfold Vestfold      9     2018   0.9           0.7
## 20 Dagsommerfugler skog Vest-Agder Rogaland      9     2018   0.3            NA
## 30           Dagsommerfugler skog Trøndelag      9     2018   0.4            NA
##    ovre_Kvartil datatypeId    datatypeName unitOfMeasurement
## 10            1          1 Ekspervurdering         Enhetsløs
## 20          NA         NA            <NA>         Enhetsløs
## 30          NA         NA            <NA>         Enhetsløs
##                   customDistributionUUID distributionName distributionId
## 10                                  <NA>             <NA>             NA
## 20 6ea2679c-b4bd-4f7a-8e5e-9f137261f257             <NA>             NA
## 30 4c37b474-d3f9-4309-baed-6a7009feb327             <NA>             NA
##    distParam1 distParam2 distributionID
## 10         NA         NA             NA
## 20         NA         NA             NA
## 30         NA         NA             NA
##
## $customDistributions
## $customDistributions$`6ea2679c-b4bd-4f7a-8e5e-9f137261f257`
## Distribution Object of Class: DiscreteDistribution
##
## $customDistributions$`4c37b474-d3f9-4309-baed-6a7009feb327`
## Distribution Object of Class: DiscreteDistribution
##
##
## attr(,"class")
## [1] "indicatorData" "list"
```

We can also specify the values as a custom distribution made by the `makeDistribution` function. Here, the point estimate is automatically retrieved from the distribution. In this first example, we make a discrete distribution of three possible values. In the second example, we specify the parameters for a log-normal distribution

```r
updatedIndicatorData <- setIndicatorValues(updatedIndicatorData, areaId = 7041, year = 2018,
    distribution = makeDistribution(input = cbind(values = c(0.1, 0.3, 0.4), probs = c(0.33,
        0.33, 0.34))))

updatedIndicatorData <- setIndicatorValues(updatedIndicatorData, areaId = 7042, year = 2018,
    distribution = makeDistribution(input = "logNormal", distParams = list(mean = 40,
        sd = 2)))


updatedIndicatorData
```

```
## $indicatorValues
##    indicatorId          indicatorName areaId
## 10         351 Dagsommerfugler i skog   7040
## 20         351 Dagsommerfugler i skog   7041
## 30         351 Dagsommerfugler i skog   7042
##                                  areaName yearId yearName      verdi
## 10    Dagsommerfugler skog Østfold Vestfold      9     2018 9.0000e-01
## 20 Dagsommerfugler skog Vest-Agder Rogaland      9     2018 2.6811e-01
## 30           Dagsommerfugler skog Trøndelag      9     2018 1.7319e+18
##    nedre_Kvartil ovre_Kvartil datatypeId    datatypeName unitOfMeasurement
## 10           0.7            1          1 Ekspervurdering         Enhetsløs
## 20            NA           NA          1 Ekspervurdering         Enhetsløs
## 30            NA           NA          1 Ekspervurdering         Enhetsløs
##                   customDistributionUUID distributionName distributionId
## 10                                  <NA>             <NA>             NA
## 20 9bbd1fbb-2aa6-49d4-b915-64b62887c0eb             <NA>             NA
## 30 b98db62c-6abf-4ffa-9399-9404d893915e             <NA>             NA
##    distParam1 distParam2 distributionID
## 10         NA         NA             NA
## 20         NA         NA             NA
## 30         NA         NA             NA
##
## $customDistributions
## $customDistributions$`9bbd1fbb-2aa6-49d4-b915-64b62887c0eb`
## Distribution Object of Class: DiscreteDistribution
##
## $customDistributions$`b98db62c-6abf-4ffa-9399-9404d893915e`
## Distribution Object of Class: Lnorm
##  meanlog: 40
##  sdlog: 2
##
##
## attr(,"class")
## [1] "indicatorData" "list"
```

If we where to specify a Poisson distribution, the call would look something like this:

```
updatedIndicatorData <- setIndicatorValues(updatedIndicatorData, areaId = 7041, year = 2018,
    distribution = makeDistribution("Poisson", distParams = list(lambda = 23)))
updatedIndicatorData
```

```
## $indicatorValues
##    indicatorId          indicatorName areaId
## 10         351 Dagsommerfugler i skog   7040
## 20         351 Dagsommerfugler i skog   7041
## 30         351 Dagsommerfugler i skog   7042
##                                 areaName yearId yearName      verdi
## 10    Dagsommerfugler skog Østfold Vestfold      9     2018 9.0000e-01
## 20 Dagsommerfugler skog Vest-Agder Rogaland      9     2018 2.3000e+01
## 30           Dagsommerfugler skog Trøndelag      9     2018 1.7319e+18
##    nedre_Kvartil ovre_Kvartil datatypeId    datatypeName unitOfMeasurement
## 10           0.7            1          1 Ekspervurdering         Enhetsløs
## 20            NA           NA          1 Ekspervurdering         Enhetsløs
## 30            NA           NA          1 Ekspervurdering         Enhetsløs
##                   customDistributionUUID distributionName distributionId
## 10                                  <NA>             <NA>             NA
## 20 63164847-7aa7-49d3-8c8d-e46b31a54783             <NA>             NA
## 30 b98db62c-6abf-4ffa-9399-9404d893915e             <NA>             NA
##    distParam1 distParam2 distributionID
## 10         NA         NA             NA
## 20         NA         NA             NA
## 30         NA         NA             NA
##
## $customDistributions
## $customDistributions$`b98db62c-6abf-4ffa-9399-9404d893915e`
## Distribution Object of Class: Lnorm
##  meanlog: 40
##  sdlog: 2
##
## $customDistributions$`63164847-7aa7-49d3-8c8d-e46b31a54783`
## Distribution Object of Class: Pois
##  lambda: 23
```

```
##
##
## attr(,"class")
## [1] "indicatorData" "list"
```

If we would like to use an empirical distribution from for example a mcmc modelling run, we could do:

```
myCodasamples <- rnorm(n = 1000, mean = 15)  ## toy example coda results

updatedIndicatorData <- setIndicatorValues(updatedIndicatorData, areaId = 7041, year = 2018,
    distribution = makeDistribution(myCodasamples))
updatedIndicatorData
```

When you are satisfied with the new data, the next step is to import it to the database. For this we use the `writeIndicatorValues` function. This will only work when the database is open to receive new values, which might be allowed only for particular years.

```
writeIndicatorValues(updatedIndicatorData)
```

```
## Year 9 is closed for update
```

We can double check that the new values stuck by downloading them again. As we can se, we now get the updated values. Currently, potential old and now outdated distribution objects are not deleted. This may change in the future.

```
newValues <- getIndicatorValues(351, year = 2018)
newValues
```

```
## $indicatorValues
##    indicatorId          indicatorName areaId
## 10         351 Dagsommerfugler i skog   7040
## 20         351 Dagsommerfugler i skog   7041
## 30         351 Dagsommerfugler i skog   7042
##                                  areaName yearId yearName verdi nedre_Kvartil
## 10    Dagsommerfugler skog Østfold Vestfold      9     2018   0.9           0.7
## 20 Dagsommerfugler skog Vest-Agder Rogaland      9     2018   0.3            NA
## 30           Dagsommerfugler skog Trøndelag      9     2018   0.4            NA
##    ovre_Kvartil datatypeId    datatypeName unitOfMeasurement
## 10            1          2 Overvåkingsdata         Enhetsløs
## 20           NA         NA            <NA>         Enhetsløs
## 30           NA         NA            <NA>         Enhetsløs
##                  customDistributionUUID distributionName distributionId
## 10                                 <NA>             <NA>             NA
## 20 6ea2679c-b4bd-4f7a-8e5e-9f137261f257             <NA>             NA
## 30 4c37b474-d3f9-4309-baed-6a7009feb327             <NA>             NA
##    distParam1 distParam2
## 10         NA         NA
## 20         NA         NA
## 30         NA         NA
##
## $customDistributions
## $customDistributions$`6ea2679c-b4bd-4f7a-8e5e-9f137261f257`
## Distribution Object of Class: DiscreteDistribution
##
## $customDistributions$`4c37b474-d3f9-4309-baed-6a7009feb327`
## Distribution Object of Class: DiscreteDistribution
##
```

```
##
## attr(,"class")
## [1] "indicatorData" "list"
```

Example2: Updating in bulk, using the log-normal distribution

In this example we have indicators measured on a non-negative scale, i.e. the indicator can take values in the interval [0, +Inf]. Values like these should be represented as log-normal instead of normal to avoid sampling negative values when we account for uncertainty in the subsequent calculations.

Here, the vector estimatedStates contains estimated or predicted values. The vector standardErrors contain the corresponding standard errors of estimated or predicted states. Note that areaIDs and areaNames should correspond to IDs and names stored in the NI database.

```
estimatedStates <- c(1, 2)
standardErrors <- c(2, 0.5)
areaIDs <- c(7040, 7041)
areaNames <- c("Dagsommerfugler skog Østfold Vestfold", "Dagsommerfugler skog Vest-Agder Rogaland")
years <- c("Referanseverdi", "2018")
myData <- data.frame(areaIDs, areaNames, years, estimatedStates, standardErrors)
myData
```

```
##   areaIDs                                areaNames          years
## 1    7040    Dagsommerfugler skog Østfold Vestfold Referanseverdi
## 2    7041 Dagsommerfugler skog Vest-Agder Rogaland           2018
##   estimatedStates standardErrors
## 1               1            2.0
## 2               2            0.5
```

Assume each estimate/prediction is a stochastic variable with a lognormal distribution (i.e. normal distribution when measured on a log-scale). The parameters of each distribution can be calculated from the estimated values and their standard errors. The mean of the logarithmic distribution is then

$$meanLog = log\left(\frac{mean}{\sqrt{\frac{sd^2}{mean^2}+1}}\right)$$

, and the standard error of the logarithmic distribution is

$$\sqrt{log\left(1+\frac{sd^2}{mean^2}\right)}$$

. This can be specified in many ways, also as in the code below:

```
myData$muLogNormal <- log((myData$estimatedStates^2) * ((1/((myData$standardErrors^2) +
    (myData$estimatedStates^2)))^0.5))
myData$sigmaLogNormal <- (log(1 + (myData$standardErrors/myData$estimatedStates)^2))^0.5
```

To simplify, we have put these operations into the functions `normal2Lognormal`. To store the parameters of the corresponding log-normal distribution:

```
logNormalParams <- normal2Lognormal(mean = myData$estimatedStates, sd = myData$standardErrors)

myData$muLogNormal <- logNormalParams$mean
myData$sigmaLogNormal <- logNormalParams$sd
```

We can check that we get the correct values back (1 and 2 in this example) when sampling from this distribution, using the gamlss.dist package:

```
mean(gamlss.dist::rLOGNO(1e+06, mu = myData$muLogNormal[1], sigma = myData$sigmaLogNormal[1]))
```

```
## [1] 1.629381
```

```
sd(gamlss.dist::rLOGNO(1e+06, mu = myData$muLogNormal[1], sigma = myData$sigmaLogNormal[1]))
```

```
## [1] 5.272461
```

Instead of the LOGNO distribution functions in the `gamlss.dist` package, one may also use the rlnorm function in the `stats` package:

```
mean(rlnorm(1e+06, meanlog = myData$muLogNormal[1], sdlog = myData$sigmaLogNormal[1]))
```

```
## [1] 1.633812
```

If estimates and standard errors are calculated on a log-scale rather than the non-negative scale, they may be used directly as the parameters of the lognormal distribution:

```
myData$muLogNormal <- myData$estimatedStates
myData$sigmaLogNormal <- myData$standardErrors
```

Use makeDistribution to create distribution objects

Here we show a way to populate the new data in bulk for many indicators (here only for 2 rows, but the principle holds for many rows).

```
ddd <- NULL
for (i in 1:dim(myData)[[1]]) {
    ddd[i] <- list(makeDistribution(input = "logNormal", distParams = list(mean = myData$muLogNormal[i]
        sd = myData$sigmaLogNormal[i])))
}

ddd
```

```
## [[1]]
## Distribution Object of Class: Lnorm
##  meanlog: -0.80471895621705
##  sdlog: 1.6094379124341
##
## [[2]]
## Distribution Object of Class: Lnorm
##  meanlog: 0.662834869651728
##  sdlog: 0.0606246218164348
```

One may include the distribution objects in the dataframe myData. Note the trick with putting them in a list of the same length as the rows in the dataframe.

```
myData$distrObjects <- ddd
myData
```

```
##   areaIDs                                areaNames          years
## 1    7040    Dagsommerfugler skog Østfold Vestfold Referanseverdi
## 2    7041 Dagsommerfugler skog Vest-Agder Rogaland           2018
##   estimatedStates standardErrors muLogNormal sigmaLogNormal
## 1               1            2.0  -0.8047190     1.60943791
## 2               2            0.5   0.6628349     0.06062462
##                                          distrObjects
## 1 <S4 class 'Lnorm' [package "distr"] with 12 slots>
## 2 <S4 class 'Lnorm' [package "distr"] with 12 slots>
```

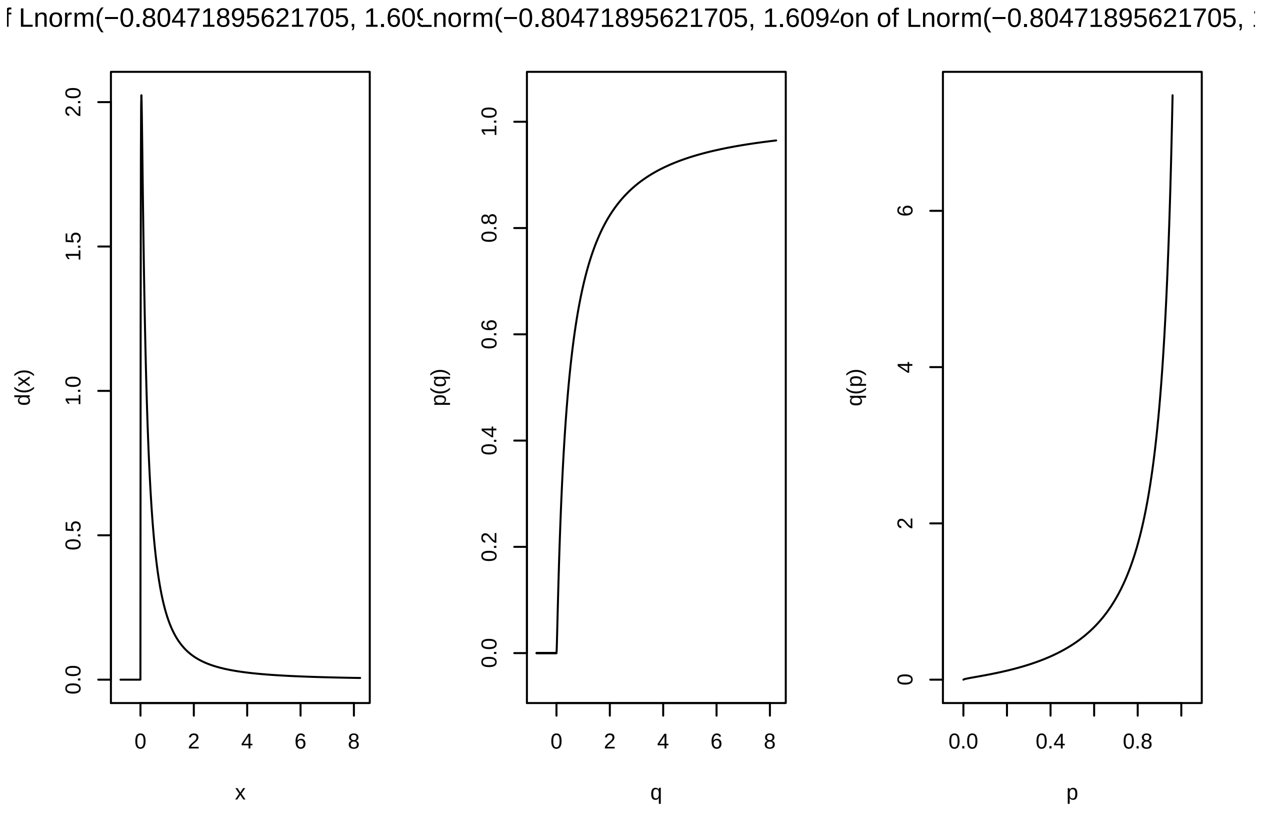
You can access individual distribution objects in the data.frame myData by treating it as an element in a list.

```
myData$distrObjects[[1]]
```

```
## Distribution Object of Class: Lnorm
##  meanlog: -0.80471895621705
##  sdlog: 1.6094379124341
```

This class comes with its own plotting function from the package `distr`. It can be sampled by a generic function in this package.

```
distr::plot(myData$distrObjects[[which(myData$areaNames == "Dagsommerfugler skog Østfold Vestfold" &
    myData$years == "Referanseverdi")]])
```



```
sampleDistribution(myData$distrObjects[[2]], 100)
```

```
##   [1] 1.902003 1.730056 1.843674 1.970402 1.808330 1.857039 1.864219 1.827363
##   [9] 1.909578 1.938335 2.110157 2.068979 1.792838 2.092669 1.907513 1.737817
##  [17] 1.921634 1.891100 1.986908 1.887410 2.114719 1.922369 1.911965 1.951497
##  [25] 1.994343 1.904062 1.829435 1.889865 1.926131 1.938280 2.172815 1.985989
##  [33] 2.090093 1.972085 1.896443 1.990042 1.899958 2.010013 1.905644 2.168841
##  [41] 1.925334 1.826863 1.887911 1.973579 2.123856 1.831671 2.079751 2.007434
##  [49] 1.981960 2.178201 2.014561 1.750044 1.626181 2.117003 1.875042 1.912858
##  [57] 2.045122 1.928157 2.060049 2.103488 2.123605 1.983969 2.010619 1.917178
##  [65] 1.975692 1.904322 2.099997 1.936950 1.942316 2.000435 2.050638 1.996003
##  [73] 1.882924 1.892216 1.877808 1.924079 2.030882 1.950720 1.904530 1.760588
##  [81] 1.856688 1.940769 1.853014 2.158860 1.920637 1.837866 1.611281 1.878639
##  [89] 1.848743 2.079129 2.133144 2.217964 1.934457 1.980863 2.165043 1.878335
##  [97] 2.050381 1.896569 1.928947 1.940800
```

As before, we first read a summary of the indicators you have access to from the NI database using getIndicators().

```r
getToken("your.username", "secretPassword")
myIndicators <- getIndicators()
myIndicators
```

Read indicatordata from the NI database for a selected indicator into an list-object. Although the function `getIndicatorValues` is made to operate with indicatorIDs, we can summon an indicator i cleartext like this. Leaving the parameter `years` empty (NULL), retrieves all years.

```r
indicatorData <- getIndicatorValues(indicatorID = myIndicators$id[myIndicators$name ==
    "Dagsommerfugler i skog"], years = c("Referanseverdi", "2018"))
```

Here we first copy indicatorData into new dataframe and keep indicatorData as backup if something goes wrong.

```r
updatedIndicatorData <- indicatorData
```

Use setIndicatorValues to update updatedIndicatorData. setIndicatorValues updates one indicator observation per call, so we here wrap it in a loop to set all values.

```r
for (i in 1:dim(myData)[[1]]) {
    updatedIndicatorData <- setIndicatorValues(updatedIndicatorData, areaId = myData$areaIDs[i],
        years = myData$years[i], distribution = myData$distrObjects[[i]])
}
```

Alternatively, you can set each indicator at a time, without using loops. Here we create the distribution at the same time (with transformed distribution parameters), and create the copy of the indicator data in one go.

```r
updatedIndicatorData <- setIndicatorValues(indicatorData, areaId = 7032, years = 2018,
    distribution = makeDistribution(input = "logNormal", distParams = normal2Lognormal(mean = 1,
        sd = 2)))
```

Write updated dataset into NI database after checking updatedIndicatorData

```r
updatedIndicatorData
writeIndicatorValues(updatedIndicatorData)
```