Using the package `crmPack`: an introductory example.

Daniel Sabanés Bové

16th December 2014

This short vignette shall introduce into the usage of the package `crmPack`. Hopefully it makes it easy for you to set up your own CRM.

If you have any questions or feedback concerning the package, please write an email to me: `sabanesd@roche.com`. Thank you very much in advance!

1 Installation

Probably you have already installed the package. However, if not, here is a brief installation summary.

`crmPack` is relying on JAGS and WinBUGS (please click on the links for going to the webpages of the two projects) for the internal MCMC computations. While JAGS is required for the default MCMC runs, WinBUGS only needs to be installed currently if the dual-endpoint designs need to be run (because they don't run with JAGS at the moment). To increase the speed of the MCMC sampling, `BayesLogit` is currently used for the `LogisticNormal` model.

Furthermore, you first have to install some required packages, with the following command:

```
> install.packages(c("Rcpp", "RcppArmadillo", "rjags",
                     "ggplot2", "gridExtra", "GenSA", "BayesLogit", "mvtnorm"),
                   dependencies=TRUE)
```

Finally, you can obtain the newest version of the package. Download it by visiting the following URL: `https://stash.intranet.roche.com/stash/plugins/servlet/archive/projects/RSTAT/repos/crmpack?at=refs%2Fheads%2Fmaster` This will download a file called `crmpack-master.zip`. You then extract the zip file to a folder `crmpack-master`. Afterwards you can install the package from this folder, e.g. from within `R` using the command

```
> install.packages("path/to/the/directory/crmpack-master",
                   repos=NULL,
                   type="source")
```

Note that you have to specify the full directory to `crmpack-master`, unless this is a subdirectory of your current working directory.

2 Getting started

Before being able to run anything, you have to load the package with

```
> library(crmPack)
```

For browsing the help pages for the package, it is easiest to start the web browser interface with

```
> help.start()
```

then clicking on "Packages" and then searching for "crmPack" and clicking on the link. This gives you the list of all help pages available for the package.

First, we will set up a logistic normal model. Therefore, you can now click on the corresponding help page `LogisticLogNormal-class` as background information for the next steps.

3 Model setup

3.1 Logistic model with bivariate (log) normal prior

Let us start with setting up the logistic regression model, which will then be used in the following for the continual reassessment method. With the following command, we create a new model of class `LogisticLogNormal`, with certain mean and covariance prior parameters and reference dose:

```
> model <- new("LogisticLogNormal",
               mean=c(-0.85, 1),
               cov=
               matrix(c(1, -0.5, -0.5, 1),
                      nrow=2L),
               refDose=56)
```

This command may look unfamiliar to you, because it uses the `new` function. This is a special function in `R`, as it creates new objects of the class specified by its first argument (here: `"LogisticLogNormal"`). The `R`-package `crmPack` uses the S4 class system for implementation of the CRM designs. We can query the class an object belongs to with the `class` function:

```
> class(model)
```

```
[1] "LogisticLogNormal"
attr(,"package")
[1] "crmPack"
```

We can look in detail at the structure of `model` as follows:

```
> str(model)
```

```
Formal class 'LogisticLogNormal' [package "crmPack"] with 11 slots
  ..@ mean      : num [1:2] -0.85 1
  ..@ cov       : num [1:2, 1:2] 1 -0.5 -0.5 1
  ..@ refDose   : num 56
  ..@ datamodel :function ()
  ..@ priormodel:function ()
  ..@ datanames : chr [1:3] "nObs" "y" "x"
  ..@ modelspecs:function ()
  ..@ dose      :function (prob, alpha0, alpha1)
  ..@ prob      :function (dose, alpha0, alpha1)
  ..@ init      :function ()
  ..@ sample    : chr [1:2] "alpha0" "alpha1"
```

We see that the object has 11 slots, and their names. These can be accessed with the `@` operator (similarly as for lists the `$` operator), for example we can extract the `dose` slot:

```
> model@dose
```

```
function (prob, alpha0, alpha1)
{
    StandLogDose <- (logit(prob) - alpha0)/alpha1
    return(exp(StandLogDose) * refDose)
}
<environment: 0x000000000ee59f70>
```

This is the function that computes for a given probability `prob` and parameters `alpha0` and `alpha1` the dose that gives this probability. You can find out yourself about the other slots, by looking at the help page for `Model-class` in the help browser, because all models are just special cases of the general `Model` class. In the `Model-class` help page, you also find out that there are two additional specific model classes, namely `LogisticKadane` and `DualEndpoint`.

3.2 Advanced model specification

There are a few further advanced ways to specify a model in `crmPack`.

First, a minimal informative prior (Neuenschwander, Branson, and Gsponer, 2008) can be computed using the `MinimalInformative` function. The construction is based on the input of a minimal and a maximal dose, where certain ranges of DLT probabilities are deemed unlikely. A logistic function is then fitted through the corresponding points on the dose-toxicity plane in order to derive Beta distributions also for doses in-between. Finally these Beta distributions are approximated by a common `LogisticNormal` model. So the minimal informative construction avoids explicit specification of the parameters of the `LogisticNormal` model.

In our example, we could construct it as follows, assuming a minimal dose of 0.1 mg and a maximum dose of 100 mg:

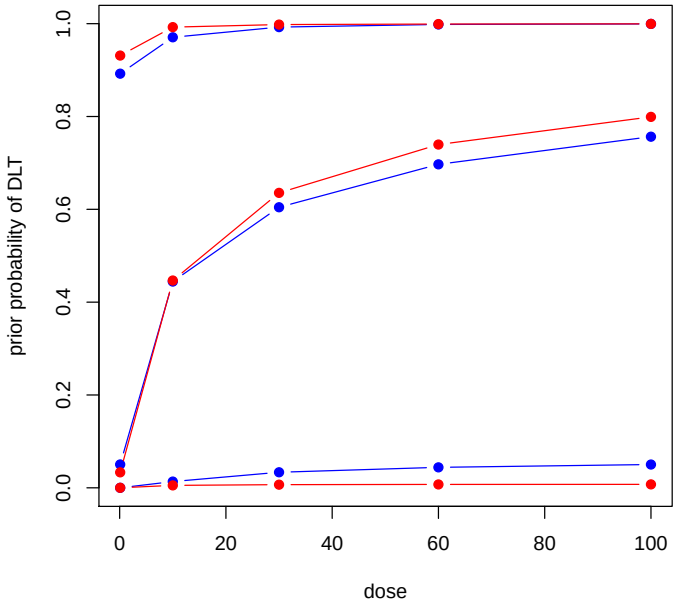
```
> set.seed(432)
> coarseGrid <- c(0.1, 10, 30, 60, 100)
> minInfModel <- MinimalInformative(dosegrid = coarseGrid,
                                    refDose=50,
                                    threshmin=0.2,
                                    threshmax=0.3,
                                    control=
                                    list(threshold.stop=0.03,
                                         maxit=200))
```

```
It: 1, obj value: 0.1872945506
It: 48, obj value: 0.1074607191
It: 75, obj value: 0.08777509852
```

We use a few grid points between the minimum and the maximum to guide the approximation routine, which is based on a stochastic optimization method (the `control` argument is for this optimization routine, please see the help page for `Quantiles2LogisticNormal` for details). Therefore we need to set a random number generator seed beforehand to be able to reproduce the results in the future. The `threshmin` and `threshmax` values specify the probability thresholds above and below, respectively, it is very unlikely (only 5% probability) of the probability of DLT at the minimum and maximum dose, respectively.

The result `minInfModel` is a list, and we can use its contents to illustrate the creation of the prior:

```
> matplot(x=coarseGrid,
          y=minInfModel$required,
          type="b", pch=19, col="blue", lty=1,
          xlab="dose",
          ylab="prior probability of DLT")
> matlines(x=coarseGrid,
           y=minInfModel$quantiles,
           type="b", pch=19, col="red", lty=1)
```



In this plot we see in blue the quantiles (2.5%, 50%, and 97.5%) of the Beta distributions that we approximate with the red quantiles of the logistic normal model. We see that the distance is quite small, and the maximum distance between any red and blue point is:

```
> minInfModel$distance
```

```
[1] 0.04674842
```

The final approximating model, which has produced the red points, is contained in the `model` element:

```
> str(minInfModel$model)
```

```
Formal class 'LogisticNormal' [package "crmPack"] with 12 slots
  ..@ mean      : num [1:2] 0.968 0.695
  ..@ cov       : num [1:2, 1:2] 8.939 0.938 0.938 0.308
  ..@ prec      : num [1:2, 1:2] 0.164 -0.501 -0.501 4.773
  ..@ refDose   : num 50
  ..@ datamodel :function ()
  ..@ priormodel:function ()
  ..@ datanames : chr [1:3] "nObs" "y" "x"
  ..@ modelspecs:function ()
  ..@ dose      :function (prob, alpha0, alpha1)
  ..@ prob      :function (dose, alpha0, alpha1)
  ..@ init      :function ()
  ..@ sample    : chr [1:2] "alpha0" "alpha1"
```

Here we see in the slots `mean`, `cov` the parameters that have been determined. At this point a slight warning: you cannot directly change these parameters in the slots of the existing model object, because the parameters have also been saved invisibly in other places in the model object. Therefore, always use the `new` command with the new parameters to create a new model object.

4 Data

If you are in the middle of a trial and you would like to recommend the next dose, then you have data from the previous patients for input into the model. This data needs to be captured in a `Data` object. For example:

```
> data <- new("Data",
              x=
              c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
              y=
              as.integer(c(0, 0, 0, 0, 0, 0, 1, 0)),
              cohort=
              as.integer(c(0, 1, 2, 3, 4, 5, 5, 5)),
              doseGrid=
              c(0.1, 0.5, 1.5, 3, 6,
                seq(from=10, to=80, by=2)))
```

Most important are `x` (the doses) and `y` (the DLTs, 0 for no DLT and 1 for DLT), as well as the dose grid `doseGrid`. All computations are using the dose grid specified in the `Data` object. So for example, except for patient number 7, all patients were free of DLTs.

Again, you can find out the details in the help page `Data-class`. Note that you have received a warning here, because you did not specify the patient IDs – however, automatic ones just indexing the patients have been created for you:

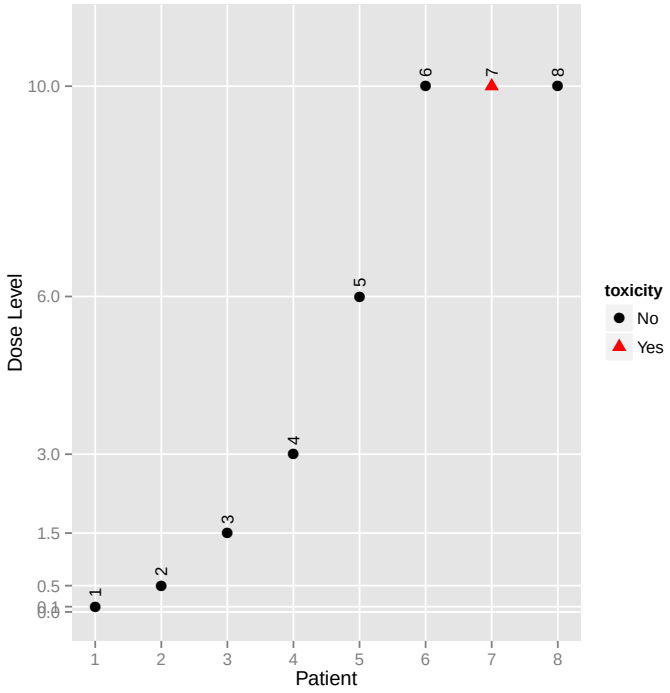
```
> data@ID
```

```
[1] 1 2 3 4 5 6 7 8
```

You can get a visual summary of the data by applying `plot` to the object:[1]

```
> print(plot(data))
```

[1] Note that for all `plot` calls in this vignette, you can leave away the wrapping `print` function call if you are working interactively with `R`. It is only because of the `Sweave` production of this vignette that the `print` statement is needed.



5 Obtaining the posterior

As said before, `crmPack` relies on MCMC sampling for obtaining the posterior distribution of the model parameters, given the data. The MCMC sampling can be controlled with an object of class `McmcOptions`, created for example as follows:

```
> options <- new("McmcOptions",
                 burnin=100,
                 step=2,
                 samples=2000)
```

Now the object `options` specifies that you would like to have 2000 parameter samples obtained from a Markov chain that starts with a "burn-in" phase of 100 iterations that are discarded, and then save a sample every 2 iterations. Note that these numbers are too low for actual implementation and only used for illustrating purposes here; normally you would specify at least the default parameters of the initialization method that is invoked by the `new` generic function: 10 000 burn-in iterations and 10 000 samples saved every 2nd iteration. You can look these up in help browser under the link "initialize-method" (there are multiple of those), described in the right column with "Initialization method for the "McmcOptions" class".

After having set up the options, you can proceed to MCMC sampling by calling the `mcmc` function:

```
> set.seed(94)
> samples <- mcmc(data, model, options)
```

```
|
|                                                    |   0%
|
|*                                                   |   2%
|
|**                                                  |   4%
|
|***                                                 |   6%
|
|****                                                |   8%
|
|*****                                               |  10%
|
|******                                              |  12%
|
|*******                                             |  14%
|
|********                                            |  16%
|
|*********                                           |  18%
|
|**********                                          |  20%
|
|***********                                         |  22%
|
|************                                        |  24%
|
|*************                                       |  26%
|
|**************                                      |  28%
|
|***************                                     |  30%
|
|****************                                    |  32%
|
|*****************                                   |  34%
|
|******************                                  |  36%
|
|*******************                                 |  38%
|
|********************                                |  40%
|
|*********************                               |  42%
|
|**********************                              |  44%
|
|***********************                             |  46%
|
|************************                            |  48%
|
|*************************                           |  50%
|
|**************************                          |  52%
```

```
  |
  |*************************                         |  54%
  |
  |**************************                        |  56%
  |
  |***************************                       |  58%
  |
  |****************************                      |  60%
  |
  |*****************************                     |  62%
  |
  |******************************                    |  64%
  |
  |*******************************                   |  66%
  |
  |********************************                  |  68%
  |
  |*********************************                 |  70%
  |
  |**********************************                |  72%
  |
  |***********************************               |  74%
  |
  |************************************              |  76%
  |
  |*************************************             |  78%
  |
  |**************************************            |  80%
  |
  |***************************************           |  82%
  |
  |****************************************          |  84%
  |
  |*****************************************         |  86%
  |
  |******************************************        |  88%
  |
  |*******************************************       |  90%
  |
  |********************************************      |  92%
  |
  |*********************************************     |  94%
  |
  |**********************************************    |  96%
  |
  |***********************************************   |  98%
  |
  |************************************************| 100%
```

The `mcmc` function takes the data object, the model and the MCMC options. By default, JAGS is used for obtaining the samples, and you could specify `program="WinBUGS"` to use WinBUGS instead. This will dynamically try to load the `R`-package `R2WinBUGS` to interface with WinBUGS. Therefore you must have installed both WinBUGS and the

R-package `R2WinBUGS` in order to use this option.

Finally, it is good practice to check graphically that the Markov chain has really converged to the posterior distribution. To this end, `crmPack` provides an interface to the convenient `R`-package `ggmcmc`. With the function `extract` you can extract the individual parameters from the object of class `Samples`. For example, we extract the α_0 samples:

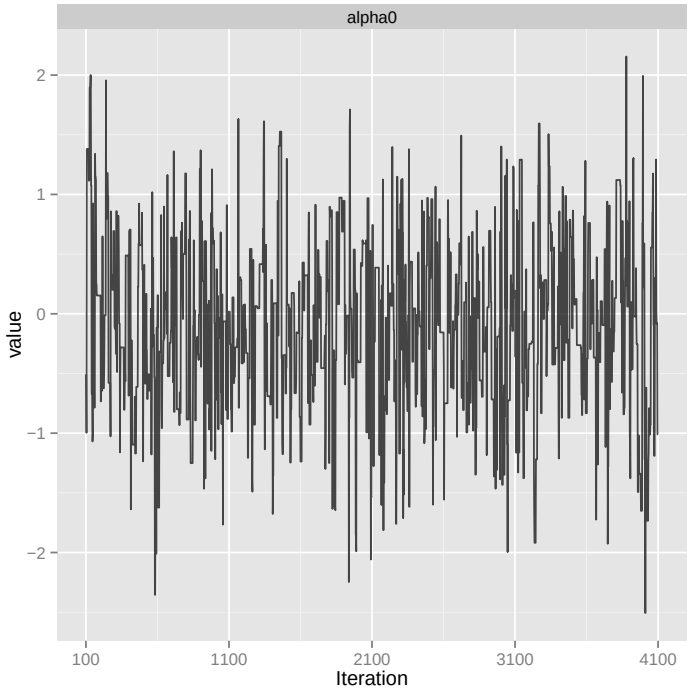
```
> ## look at the structure of the samples object:
> str(samples)
```

```
Formal class 'Samples' [package "crmPack"] with 2 slots
  ..@ data   :List of 2
  .. ..$ alpha0: num [1:2000] -0.508 -0.997 1.381 1.381 1.381 ...
  .. ..$ alpha1: num [1:2000] 0.73 0.529 1.338 1.338 1.338 ...
  ..@ options:Formal class 'McmcOptions' [package "crmPack"] with 3 slots
  .. .. ..@ iterations: int 4100
  .. .. ..@ burnin    : int 100
  .. .. ..@ step      : int 2
```

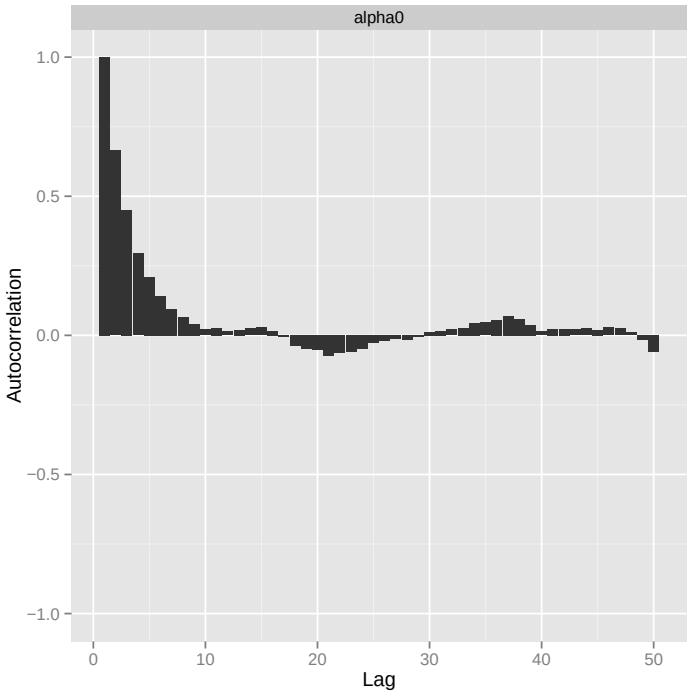
```
> ## now extract the alpha0 samples (intercept of the regression model)
> alpha0samples <- extract(samples, "alpha0")
```

`alpha0samples` now contains the α_0 samples in a format understood by `ggmcmc` and we can produce plots with it, e.g. a trace plot and an autocorrelation plot:

```
> library(ggmcmc)
> print(ggs_traceplot(alpha0samples))
```



```
> print(ggs_autocorrelation(alpha0samples))
```



So here we see that we have some autocorrelation in the samples, and might consider using a higher thinning parameter in order to decrease it.

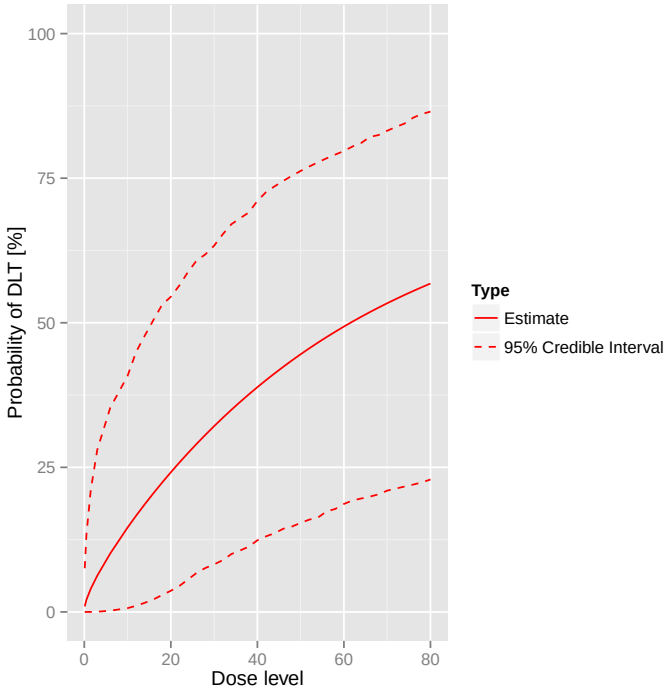
You can find other useful plotting functions in the package information:

```
> help(package="ggmcmc")
```

6 Plotting the model fit

After having obtained the parameter samples, we can plot the model fit, by supplying the samples, model and data to the generic plot function:

```
> print(plot(samples, model, data))
```



This plot shows the posterior mean curve and 95% equi-tailed credible intervals at each point of the dose grid from the `data` object.

Note that you can also produce a plot of the prior mean curve and credible intervals, i.e. from the model without any data. This works in principle the same way as with data, just that we use an empty data object:

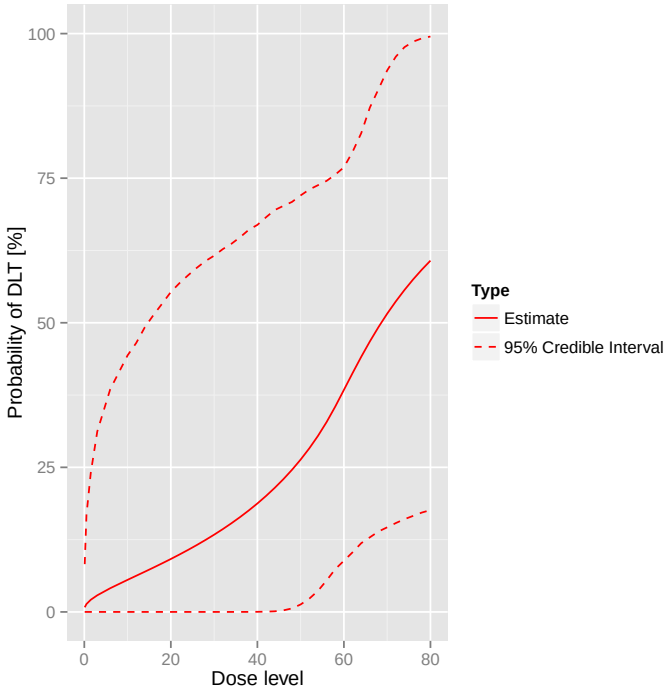
```
> ## provide empty dose and DLT vectors, and use same dose grid
> ## as before:
> emptydata <- new("Data",
                   x=numeric(),
                   y=integer(),
                   doseGrid=data@doseGrid)
> ## obtain prior samples with this Data object
> priorsamples <- mcmc(emptydata, model, options)
```

```
  |
  |                                                      |   0%
  |
  |*                                                     |   2%
  |
  |**                                                    |   4%
  |
  |***                                                   |   6%
  |
  |****                                                  |   8%
  |
  |*****                                                 |  10%
  |
```

```
|******                                  |  12%
|
|*******                                 |  14%
|
|********                                |  16%
|
|*********                               |  18%
|
|**********                              |  20%
|
|***********                             |  22%
|
|************                            |  24%
|
|*************                           |  26%
|
|**************                          |  28%
|
|***************                         |  30%
|
|****************                        |  32%
|
|*****************                       |  34%
|
|******************                      |  36%
|
|*******************                     |  38%
|
|********************                    |  40%
|
|*********************                   |  42%
|
|**********************                  |  44%
|
|***********************                 |  46%
|
|************************                |  48%
|
|*************************               |  50%
|
|**************************              |  52%
|
|***************************             |  54%
|
|****************************            |  56%
|
|*****************************           |  58%
|
|******************************          |  60%
|
|*******************************         |  62%
|
|********************************        |  64%
|
```

```
  |*********************************                  |  66%
  |
  |**********************************                 |  68%
  |
  |***********************************                |  70%
  |
  |************************************               |  72%
  |
  |*************************************              |  74%
  |
  |**************************************             |  76%
  |
  |***************************************            |  78%
  |
  |****************************************           |  80%
  |
  |*****************************************          |  82%
  |
  |******************************************         |  84%
  |
  |*******************************************        |  86%
  |
  |********************************************       |  88%
  |
  |*********************************************      |  90%
  |
  |**********************************************     |  92%
  |
  |***********************************************    |  94%
  |
  |************************************************   |  96%
  |
  |************************************************* |  98%
  |
  |**************************************************| 100%
```

```
> ## then produce the plot
> print(plot(priorsamples, model, emptydata))
```



7 Escalation Rules

For the dose escalation, there are four kinds of rules:

1. `Increments`: For specifying maximum allowable increments between doses
2. `NextBest`: How to derive the next best dose
3. `CohortSize`: For specifying the cohort size
4. `Stopping`: Stopping rules for finishing the dose escalation

We have listed here the classes of these rules, and there are multiple subclasses for each of them, which you can find as links in the help pages `Increments-class`, `NextBest-class`, `CohortSize-class` and `Stopping-class`.

7.1 Increments rules

Let us start with looking in detail at the increments rules. Currently two specific rules are implemented: Maximum relative increments based on the current dose (`IncrementsRelative`), and maximum relative increments based on the current cumulative number of DLTs that have happened (`IncrementsRelativeDLT`).

For example, in order to specify maximum increase of 100% for doses up to 20 mg, and 33% for doses above 20 mg, we can setup the following increments rule:

```
> myIncrements <- new("IncrementsRelative",
                      intervals=c(0, 20, Inf),
                      increments=c(1, 0.33))
```

Here the `intervals` slot specifies the intervals, in which the maximum relative `increments` (note: decimal values here, no percentages!) are valid.

The increments rule is used by the `maxDose` function to obtain the maximum allowable dose given the current data:

```
> nextMaxDose <- maxDose(myIncrements,
                         data=data)
> nextMaxDose
```

```
[1] 20
```

So in this case, the next dose could not be larger than 20 mg.

7.2 Rules for next best dose recommendation

There are two implemented rules for toxicity endpoint CRMs: `NextBestMTD` that uses the posterior distribution of the MTD estimate (given a target toxicity probability defining the MTD), and `NextBestNCRM` that implements the N-CRM, using posterior probabilities of target-dosing and overdosing at the dose grid points to recommend a next best dose.

For example, in order to use the N-CRM with target toxicity interval from 20% to 35%, and a maximum overdosing probability of 25%, we specify:

```
> myNextBest <- new("NextBestNCRM",
                    target=c(0.2, 0.35),
                    overdose=c(0.35, 1),
                    maxOverdoseProb=0.25)
```

Alternatively, we could use an MTD driven recommendation rule. For example, with a target toxicity rate of 33%, and recommending the 25% posterior quantile of the MTD, we specify

```
> mtdNextBest <- new("NextBestMTD",
                     target=0.33,
                     derive=
                     function(mtdSamples){
                         quantile(mtdSamples, probs=0.25)
                     })
```

Note that the `NextBestMTD` class is quite flexible, because you can specify a function `derive` that derives the next best dose from the posterior MTD samples.

During the study, in order to derive the next best dose, we supply the generic `nextBest` function with the rule, the maximum dose, the posterior samples, the model and the data:

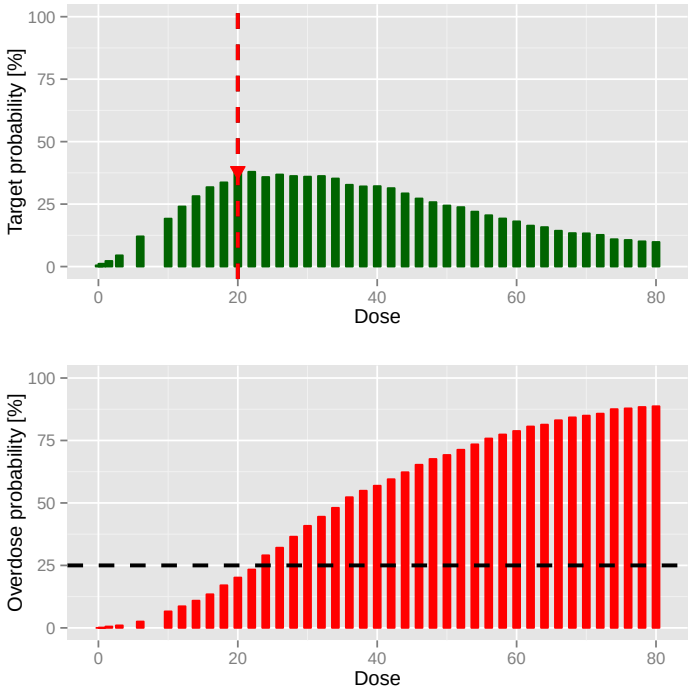
```
> doseRecommendation <- nextBest(myNextBest,
                                 doselimit=nextMaxDose,
                                 samples=samples, model=model, data=data)
```

The result is a list with two elements: `value` contains the numeric value of the recommended next best dose, and `plot` contains a plot that illustrates how the next best dose was computed. In this case we used the N-CRM rule, therefore the plot gives the target-dosing and overdosing probabilities together with the safety bar of 25%, the maximum dose and the final recommendation (the red triangle):

```
> doseRecommendation$value
```

```
[1] 20
```

```
> print(doseRecommendation$plot)
```



7.3 Cohort size rules

Similarly to the increments rules, you can define intervals in the dose space and/or the DLT space to define the size of the cohorts. For example, let's assume we want to have one patient only in the cohorts until we reach 30 mg or the first DLT is encountered, and then proceed with three patients per cohort.

We start by creating the two separate rules, first for the dose range:

```
> mySize1 <- new("CohortSizeRange",
                 intervals=c(0, 30, Inf),
                 cohortSize=as.integer(c(1, 3)))
```

Then for the DLT range:

```
> mySize2 <- new("CohortSizeDLT",
                 DLTintervals=c(0, 1, Inf),
                 cohortSize=as.integer(c(1, 3)))
```

Finally we combine the two rules by taking the maximum number of patients of both rules:

```
> mySize <- maxSize(mySize1, mySize2)
```

The `CohortSize` rule is used by the `size` function, together with the next dose and the current data, in order to determine the size of the next cohort:

```
> size(mySize,
       dose=doseRecommendation$value,
       data=data)
```

```
[1] 3
```

Because we have one DLT already, we would go for 3 patients for the next cohort.

Moreover, if you would like to have a constant cohort size, you can use the following `CohortSizeConst` class, which we will use (with three patients) for simplicity for the remainder of this vignette:

```
> mySize <- new("CohortSizeConst",
                size=3L)
```

7.4 Stopping rules

Stopping rules are often quite complex, and built from an and/or combination of multiple parts. Therefore the `crmPack` implementation mirrors this, and multiple atomic stopping rules can be combined easily. For example, let's assume we would like to stop the trial if there are at least 3 cohorts and at least 50% probability in the target toxicity interval $(20\%, 35\%)$, or the maximum sample size of 20 patients has been reached. Then we start by creating the three pieces the rule is composed of:

```
> myStopping1 <- new("StoppingMinCohorts",
                     nCohorts=3L)
> myStopping2 <- new("StoppingTargetProb",
                     target=c(0.2, 0.35),
                     prob=0.5)
> myStopping3 <- new("StoppingMaxPatients",
                     nPatients=20L)
```

Finally we combine these with the "and" operator `&` and the "or" operator `|`:

```
> myStopping <- (myStopping1 & myStopping2) | myStopping3
```

You can find a link to all implemented stopping rule parts in the help page `Stopping-class`.

During the study, any (atomic or combined) stopping rule can be used by the function `stopTrial` to determine if the rule has been fulfilled. For example in our case:

```
> stopTrial(stopping=myStopping, dose=doseRecommendation$value,
            samples=samples, model=model, data=data)
```

```
[1] FALSE
attr(,"message")
attr(,"message")[[1]]
attr(,"message")[[1]][[1]]
[1] "Number of cohorts is 6 and thus reached the prespecified minimum number 3"

attr(,"message")[[1]][[2]]
[1] "Probability for target toxicity is 38 % for dose 20 and thus below the required 50 %"


attr(,"message")[[2]]
[1] "Number of patients is 8 and thus below the prespecified maximum number 20"
```

We receive here `FALSE`, which means that the stopping rule criteria have not been met. The attribute `message` contains the textual results of the atomic parts of the stopping rule. Here we can read that the probability for target toxicity was just 31% for the recommended dose 20 mg and therefore too low, and also the maximum sample size has not been reached, therefore the trial shall continue.

8 Simulations

In order to run simulations, we first have to build a specific design, that comprises a model, the escalation rules, starting data, a cohort size (currently fixed during the trial) and a starting dose. It might seem strange at first sight that we have to supply starting data to the design, but we will show below that this makes sense. First, we use our `emptydata` object that only contains the dose grid, and a cohorts of 3 patients, starting from 0.1 mg:

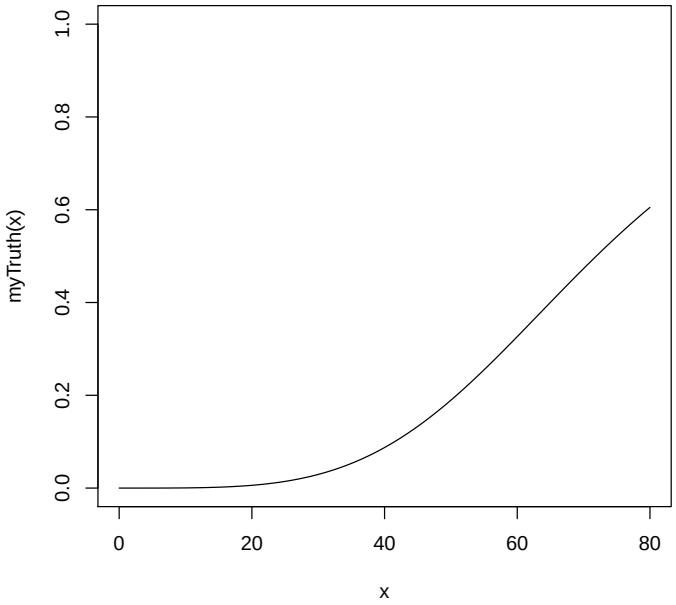
```
> design <- new("Design",
                model=model,
                nextBest=myNextBest,
                stopping=myStopping,
                increments=myIncrements,
                cohortSize=mySize,
                data=emptydata,
                startingDose=3)
```

8.1 Simulating from a true scenario

Next, we have to define a true scenario, from which the data should arise. In this case, this only requires a function that computes the probability of DLT given a dose. Here we use a specific case of the function contained in the model space:

```
> ## define the true function
> myTruth <- function(dose)
  {
```

```
      model@prob(dose, alpha0=-1, alpha1=4)
  }
> ## plot it in the range of the dose grid
> curve(myTruth(x), from=0, to=80, ylim=c(0, 1))
```



Now we can proceed to the simulations. We only generate 10 trial outcomes here for illustration, for the actual study this should be increased of course to at least 500:

```
> set.seed(819)
> time <- system.time(mySims <- simulate(design,
                                         truth=myTruth,
                                         args=NULL,
                                         firstSeparate=FALSE,
                                         nsim=10L,
                                         mcmcOptions=options,
                                         parallel=TRUE))[3]
> time
```

```
elapsed
  14.04
```

We have wrapped the call to `simulate` in a `system.time` to obtain the required time for the simulations (about 14 seconds in this case). The argument `args` could contain additional arguments for the `truth` function, which we did not require here and therefore set it to `NULL`. Note that we also pass again the MCMC options object, because during the trial simulations the MCMC routines are used. Finally, the argument `parallel` can

be used to enable the use of all processors of the computer for running the simulations in parallel. This can yield a meaningful speedup, especially for larger number of simulations.

As (almost) always, the result of this call is again an object with a class, in this case `Simulations`:

```
> class(mySims)
```

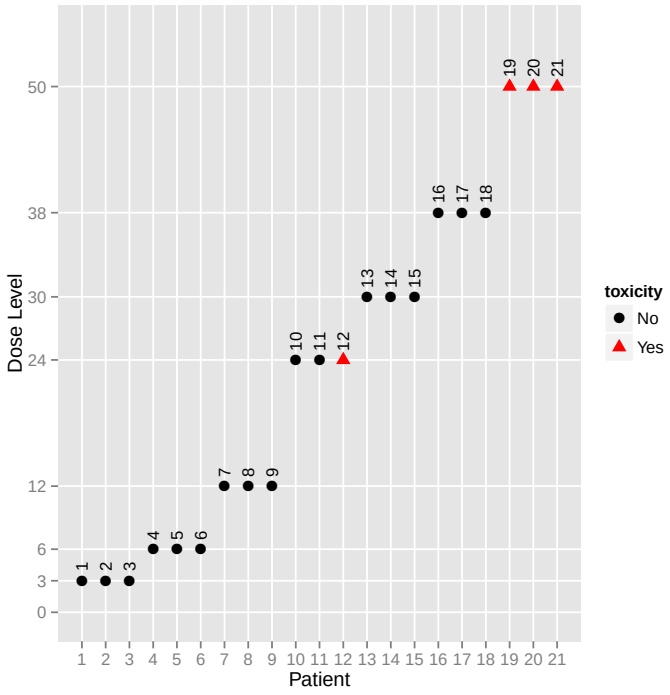
```
[1] "Simulations"
attr(,"package")
[1] "crmPack"
```

From the help page

```
> help("Simulations-class")
```

we find the description of the slots of `mySims`. In particular, the `data` slot contains the list of produced `Data` objects of the simulated trials. Therefore, we can plot the course of e.g. the third simulated trial as follows:

```
> print(plot(mySims@data[[3]]))
```



The final dose for this trial was

```
> mySims@doses[3]
```

```
[1] 36
```

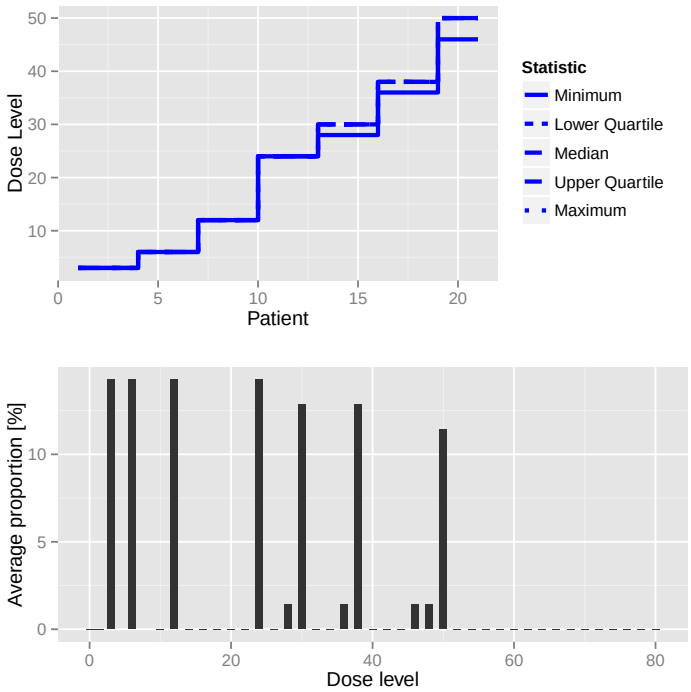
and the stopping reason was

```
> mySims@stopReasons[[3]]
```

```
[[1]]
[[1]][[1]]
[1] "Number of cohorts is 7 and thus reached the prespecified minimum number 3"

[[1]][[2]]
[1] "Probability for target toxicity is 49 % for dose 36 and thus below the required 50 %"


[[2]]
[1] "Number of patients is 21 and thus reached the prespecified maximum number 20"
```

Furthermore, with this object, we can apply two methods. First, we can plot it, i.e. we can apply the plot method:

```
> plot(mySims)
```



The resulting plot shows on the top panel a summary of the trial trajectories. Because of the low number of simulations here, we can essentially only see the median trajectory. On the bottom, the proportions of doses tried, averaged over the simulated trials, are shown. Note that you can select the plots by changing the `type` argument of `plot`, which by default is `type = c("trajectory", "dosesTried")`.

Second, we can summarize the simulation results. Here again we have to supply a true dose-toxicity function. We take the same (`myTruth`) as above:

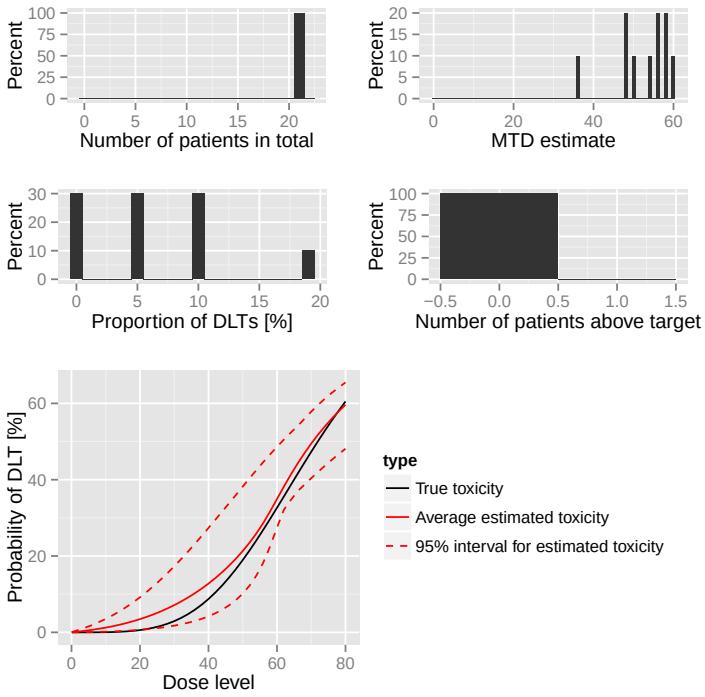
```
> summary(mySims,
          truth=myTruth)
```

```
Summary of 10 simulations

Target toxicity interval was 20, 35 %
Target dose interval corresponding to this was 50.8, 61.6
Intervals are corresponding to 10 and 90 % quantiles

Proportions of DLTs in the trials : mean 6 % (0 %, 10 %)
Mean toxicity risks for the patients : mean 4 % (4 %, 4 %)
Doses selected as MTD : mean 52.4 (46.8, 58.2)
True toxicity at doses selected : mean 23 % (16 %, 30 %)
Proportion of trials selecting target MTD: 60 %
Dose most often selected as MTD: 48
Observed toxicity rate at dose most often selected: 0 %
Fitted toxicity rate at dose most often selected : mean 19 % (9 %, 27 %)
Number of patients treated above target tox interval : mean 0 (0, 0)
Number of patients overall : mean 21 (21, 21)
```

Note that the observed toxicity rate at dose most often selected (52 mg) is not available, because no patients were actually treated at 52 mg during the simulations. This means the MTD was selected based on the evidence from the data at other dose levels.

Now we can also produce a plot of the summary results, which give a bit more detail than the textual summary we have just seen:

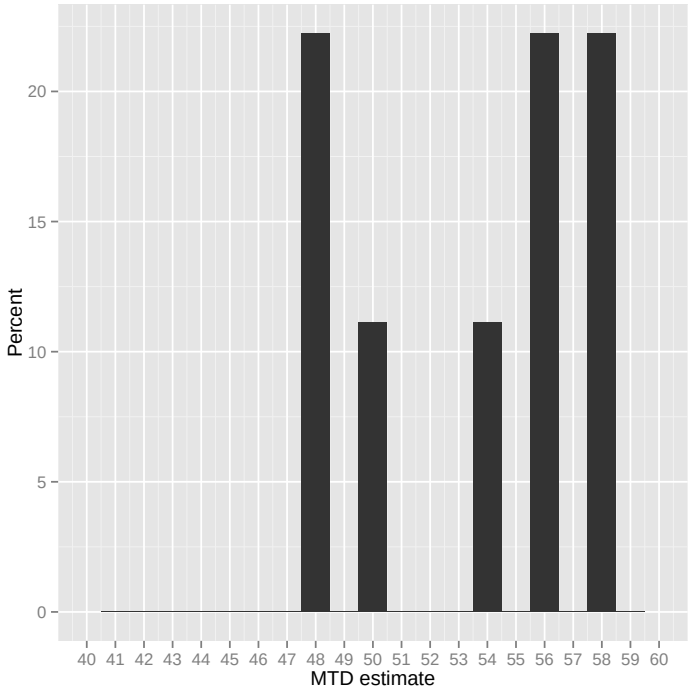
```
> simSum <- summary(mySims,
                    truth=myTruth)
> plot(simSum)
```



The top left panel shows the distribution of the sample size across the simulated trials. In this case all trials had 21 patients. The top right panel shows the distribution of the final MTD estimate / recommended dose across the simulated trials. The middle left panel shows the distribution across the simulations of the DLT proportions observed in the patients dosed. Here we see that not more than 14% of the patients had DLTs in the simulated trials. The middle right panel shows the distribution across simulations of the number of patients treated above the target toxicity window (here we used the default from 20% to 35%). We see here that in all trials, none of the patients were treated with too toxic doses. Finally, in the bottom panel we see a comparison of the true dose-toxicity curve (black) with the estimated dose-toxicity curves, averaged (continuous red line) across the trials and with 95% credible interval across the trials. Here we see a good correspondence of the truth and the estimation.

If we find that e.g. the top right plot with the distribution of the final selected doses is too small and shows not the right x-axis window, we can only plot this one and add x-axis customization on top: (see the `ggplot2` documentation for more on customizing)

```
> print(plot(simSum, type="doseSelected") +
        scale_x_continuous(breaks=40:60, limits=c(40, 60)))
```



8.2 Predicting the future course of the trial

By simulating parameters from their current posterior distribution instead of an assumed truth, it is possible to generate trial simulations from the posterior predictive distribution at any time point during the trial. This means that we can predict the future course of the trial, given the current data. In our illustrating example, this would work as follows.

The rationale of the `simulate` call is now that we specify as the `truth` argument the `prob` function from our assumed model, which has additional arguments (in our case `alpha0` and `alpha1`) on top of the first argument `dose`:

```
> model@prob
```

```
function (dose, alpha0, alpha1)
{
    StandLogDose <- log(dose/refDose)
    return(plogis(alpha0 + alpha1 * StandLogDose))
}
<environment: 0x000000000ee59f70>
```

For the simulations, these arguments are internally given by the values contained in the data frame given to `simulate` as the `args` argument. In our case, we want to supply the posterior samples of `alpha0` and `alpha1` in this data frame. We take only 50 out of the 2000 posterior samples in order to reduce the runtime for this example:

```
> postSamples <- as.data.frame(samples@data)[(1:20)*50, ]
> postSamples
```

```
          alpha0    alpha1
50    0.15334134 1.3864793
100  -0.11369470 1.0619764
150  -0.68533001 2.4217314
200  -1.23617044 0.7928227
250  -0.11645979 0.4328037
300   0.38468263 1.6950648
350   1.17594962 1.1756585
400   0.28184992 1.4310953
450  -0.07637122 2.1214294
500   0.01722731 2.7046601
550   0.07227262 0.7434383
600   0.04631815 1.4554316
650   0.28352908 0.7210675
700  -0.44900645 0.5663036
750  -1.23766733 1.4297970
800  -0.60103492 1.2925948
850  -1.25047260 0.3661834
900   0.94672574 1.3051034
950  -0.39043590 1.7878889
1000 -0.08722982 1.3003049
```

Therefore, each simulated trial will come from a posterior sample of our estimated model, given all data so far.

Furthermore we have to make a new `Design` object that contains the current data to start from, and the current recommended dose as the starting dose:

```
> nowDesign <- new("Design",
                   model=model,
                   nextBest=myNextBest,
                   stopping=myStopping,
                   increments=myIncrements,
                   cohortSize=mySize,
                   ## use the current data:
                   data=data,
                   ## and the recommended dose as the starting dose:
                   startingDose=doseRecommendation$value)
```

Finally we can execute the simulations:

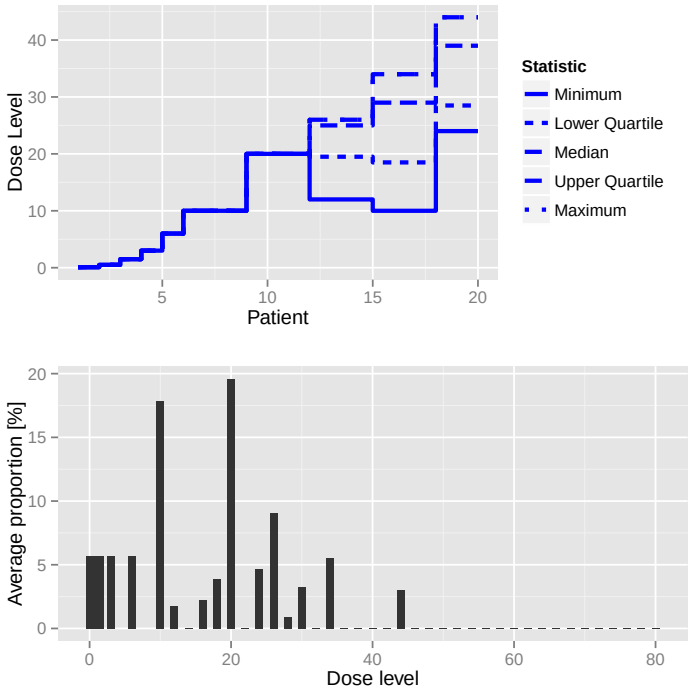
```
> set.seed(901)
> time <- system.time(futureSims <- simulate(## supply the new design here
                                             nowDesign,
                                             ## the truth is the assumed prob function
                                             truth=model@prob,
                                             ## further arguments are the
                                             ## posterior samples
                                             args=postSamples,
                                             ## no separate first patient
                                             firstSeparate=FALSE,
                                             ## do exactly so many simulations as
                                             ## we have samples
                                             nsim=nrow(postSamples),
```

```
                                        ## this remains the same:
                                        mcmcOptions=options,
                                        parallel=TRUE))[3]
> time
```

```
elapsed
  13.35
```

And now, exactly in the same way as above for the operating characteristics simulations, we can summarize the resulting predictive simulations, for example show the predicted trajectories of doses:

```
> plot(futureSims)
```



In the summary, we do not need to look at the characteristics involving the true dose-toxicity function, because in this case we are not intending to compare the performance of our CRM relative to a truth:

```
> summary(futureSims,
          truth=myTruth)
```

```
Summary of 20 simulations

Target toxicity interval was 20, 35 %
Target dose interval corresponding to this was 50.8, 61.6
Intervals are corresponding to 10 and 90 % quantiles
```

```
Proportions of DLTs in the trials : mean 17 % (5 %, 25 %)
Mean toxicity risks for the patients : mean 1 % (0 %, 3 %)
Doses selected as MTD : mean 30 (15.8, 56.2)
True toxicity at doses selected : mean 6 % (0 %, 27 %)
Proportion of trials selecting target MTD: 15 %
Dose most often selected as MTD: 18
Observed toxicity rate at dose most often selected: 33 %
Fitted toxicity rate at dose most often selected : mean 20 % (7 %, 30 %)
Number of patients treated above target tox interval : mean 0 (0, 0)
Number of patients overall : mean 18 (17, 20)
```

We see here e.g. that the estimated number of patients overall is 19, so 11 more than the current 8 patients are expected to be needed before finishing the trial.

References

Beat Neuenschwander, Michael Branson, and Thomas Gsponer. Critical aspects of the Bayesian approach to phase I cancer trials. *Statistics in medicine*, 27(13):2420–39, 2008. URL `http://onlinelibrary.wiley.com/doi/10.1002/sim.3230`.