Using the package crmPack: introductory examples

Daniel Sabanés Bové

27th March 2015
Package version 0.0.23

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` or even better post it on the Hive page: `https://roche.jiveon.com/projects/crmpack` Thank you very much in advance!

1 Installation

Probably you have already installed the package. If not, please refer to the separate installation document available from the Hive page: `https://roche.jiveon.com/servlet/JiveServlet/download/38-91276/installation.docx`

`crmPack` is relying on JAGS (please click on the link for going to the webpage of the project) for the internal MCMC computations. WinBUGS is no longer required. To increase the speed of the MCMC sampling, `BayesLogit` is currently used for the `LogisticNormal` model.

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

```
> crmPackHelp()
```

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 <- LogisticLogNormal(mean=c(-0.85, 1),
                             cov=
                                 matrix(c(1, -0.5, -0.5, 1),
                                        nrow=2),
                             refDose=56)
```

The R-package `crmPack` uses the S4 class system for implementation of the dose-escalation designs. There is the convention that class initialization functions have the same name as the class, and all class names are capitalized. We can query the class that 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: 0x000000000aa44358>
```

This is the function that computes for given parameters `alpha0` and `alpha1` the dose that gives the probability `prob` for a dose-limiting toxicity (DLT). 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 four additional specific model classes that are subclasses of the `Model` class, namely `LogisticLogNormalSub`, `LogisticNormal`, `LogisticKadane` and `DualEndpoint`.

3.2 Advanced model specification

There are a few further, advanced ways to specify a model object 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` (or `LogisticLogNormal`) model. So the minimal informative construction avoids explicit specification of the prior parameters of the logistic regression 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:

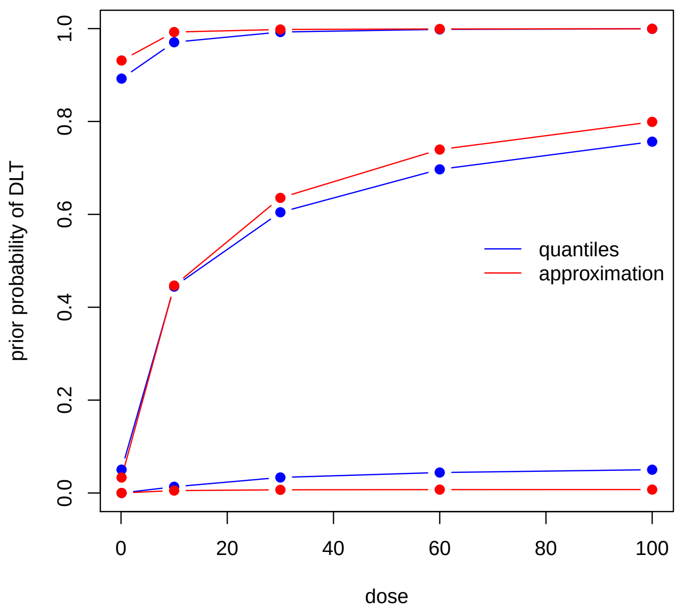
```
> set.seed(432)
> coarseGrid <- c(0.1, 10, 30, 60, 100)
> minInfModel <- MinimalInformative(dosegrid = coarseGrid,
                                    refDose=50,
                                    threshmin=0.2,
                                    threshmax=0.3,
                                    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)
> legend("right",
         legend=c("quantiles", "approximation"),
         col=c("blue", "red"),
         lty=1,
         bty="n")
```



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` list 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 class initialization function to create a new model object, if new parameters are required. But if we want to further use the approximation model, we can save it under a shorter name, e.g.:

```
> myModel <- minInfModel$model
```

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 <- Data(x=c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10),
               y=c(0, 0, 0, 0, 0, 0, 1, 0),
               cohort=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.

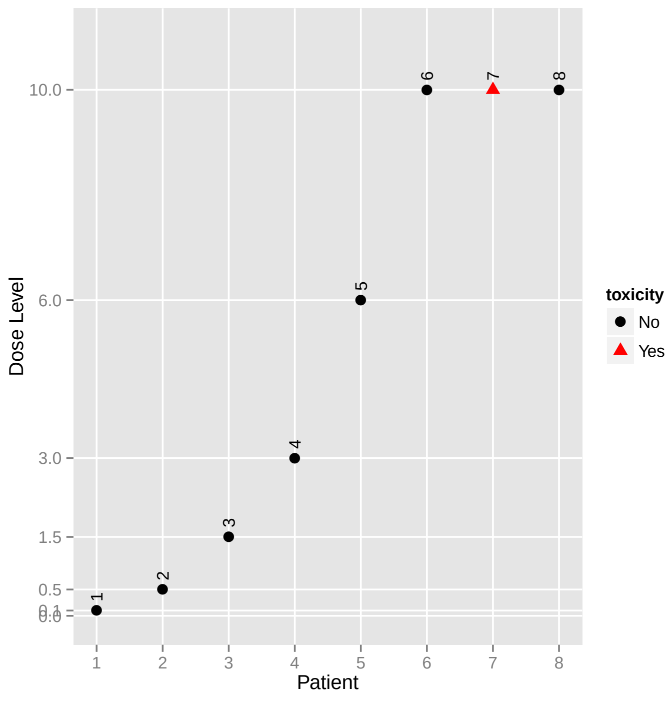
Again, you can find out the details in the help page `Data-class`. Note that you have received a warning here, because you did not specify the patient IDs – however, automatic ones just indexing the patients have been created for you:

```
> 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))
```



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 <- 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

[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.

low for actual implementation and only used for illustrating purposes here; normally you would specify at least the default parameters of the initialization function `McmcOptions`: 10 000 burn-in iterations and 10 000 samples saved every 2nd iteration. You can look these up in help browser under the link "McmcOptions".

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)
```

The `mcmc` function takes the data object, the model and the MCMC options. By default, JAGS is used for obtaining the samples. Use the option `verbose=TRUE` to show a progress bar and detailed JAGS messages.

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 `get` you can extract the individual parameters from the object of class `Samples`. For example, we extract the α_0 samples:

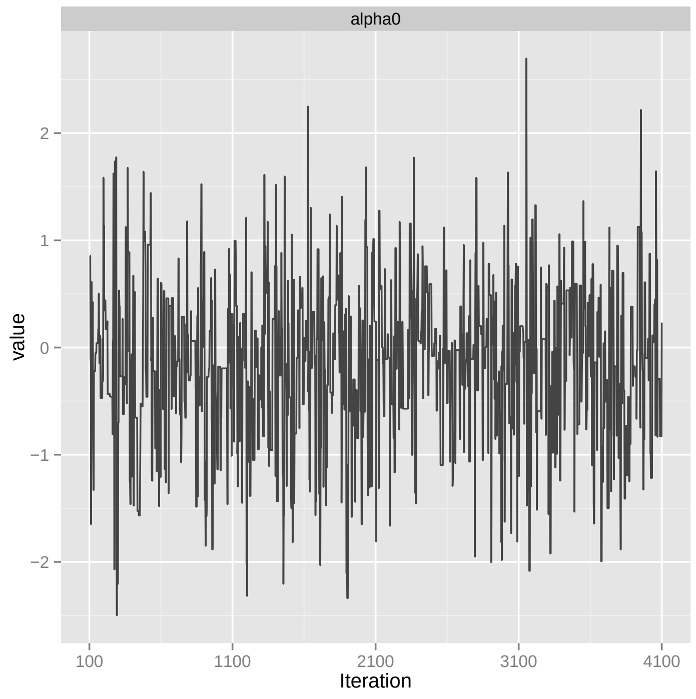
```
> ## look at the structure of the samples object:
> str(samples)
```

```
Formal class 'Samples' [package "crmPack"] with 2 slots
  ..@ data   :List of 2
  .. ..$ alpha0: num [1:2000] 0.855 0.855 0.855 -0.116 -0.116 ...
  .. ..$ alpha1: num [1:2000] 0.777 0.777 0.777 0.753 0.753 ...
  ..@ options:Formal class 'McmcOptions' [package "crmPack"] with 3 slots
  .. .. ..@ iterations: int 4100
  .. .. ..@ burnin    : int 100
  .. .. ..@ step      : int 2
```

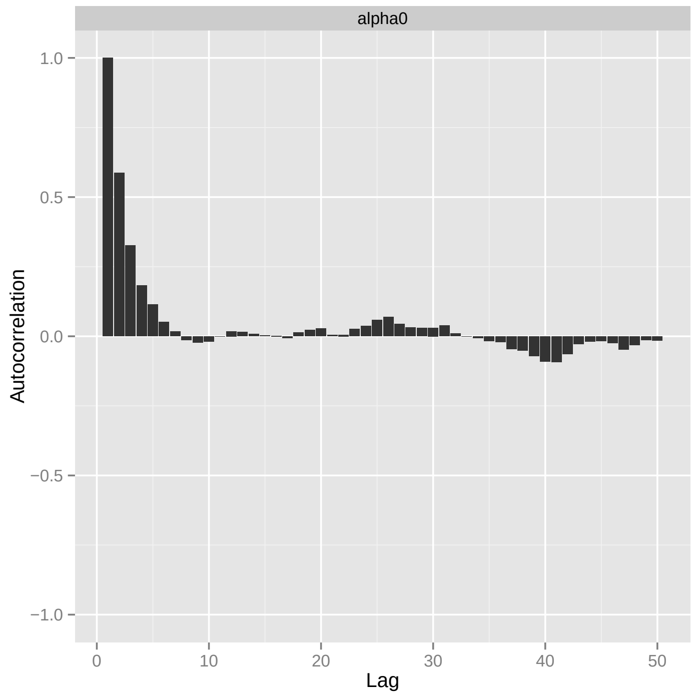
```
> ## now extract the alpha0 samples (intercept of the regression model)
> alpha0samples <- get(samples, "alpha0")
```

`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.

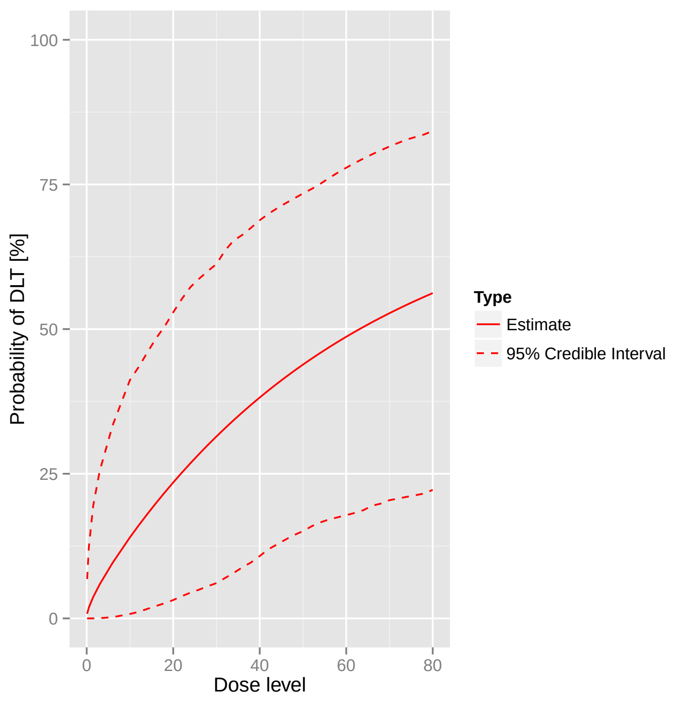
You can find other useful plotting functions in the package information:

```
> help(package="ggmcmc", help_type="html")
```

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:

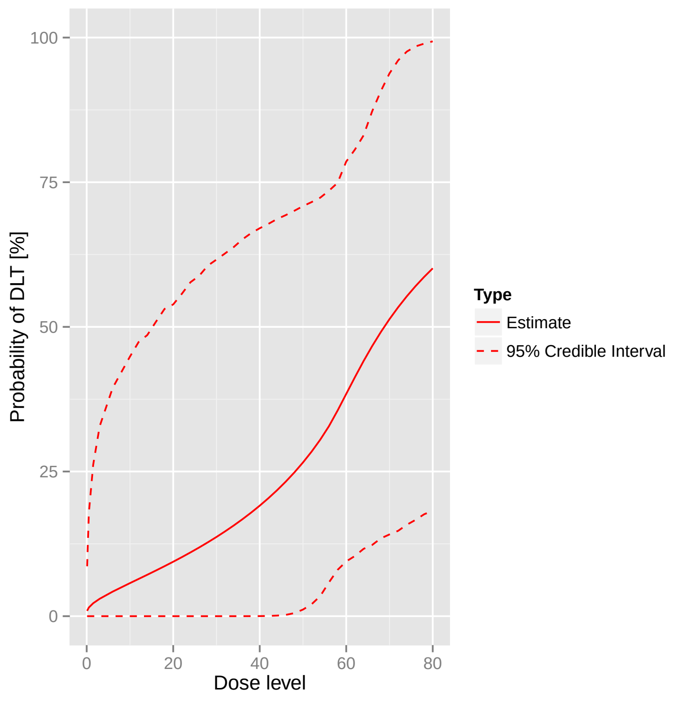
```
> print(plot(samples, model, data))
```



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 only the dose grid:
> emptydata <- Data(doseGrid=data@doseGrid)
> ## obtain prior samples with this Data object
> priorsamples <- mcmc(emptydata, model, options)
> ## 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 <- IncrementsRelative(intervals=c(0, 20),
                                     increments=c(1, 0.33))
```

Here the `intervals` slot specifies the left bounds of 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 <- 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 <- 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:

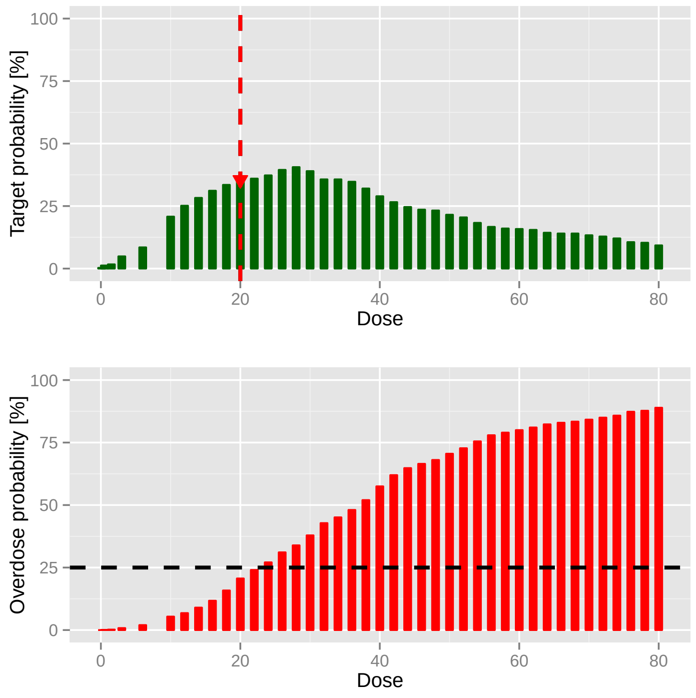
```
> doseRecommendation <- nextBest(myNextBest,
                                 doselimit=nextMaxDose,
                                 samples=samples, model=model, data=data)
```

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 <- CohortSizeRange(intervals=c(0, 30),
                             cohortSize=c(1, 3))
```

Then for the DLT range:

```
> mySize2 <- CohortSizeDLT(DLTintervals=c(0, 1),
                           cohortSize=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 <- CohortSizeConst(size=3)
```

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 <- StoppingMinCohorts(nCohorts=3)
> myStopping2 <- StoppingTargetProb(target=c(0.2, 0.35),
                                    prob=0.5)
> myStopping3 <- StoppingMinPatients(nPatients=20)
```

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 35 % for dose 20 and thus below the required 50 %"


attr(,"message")[[2]]
[1] "Number of patients is 8 and thus below the prespecified minimum 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 30% 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

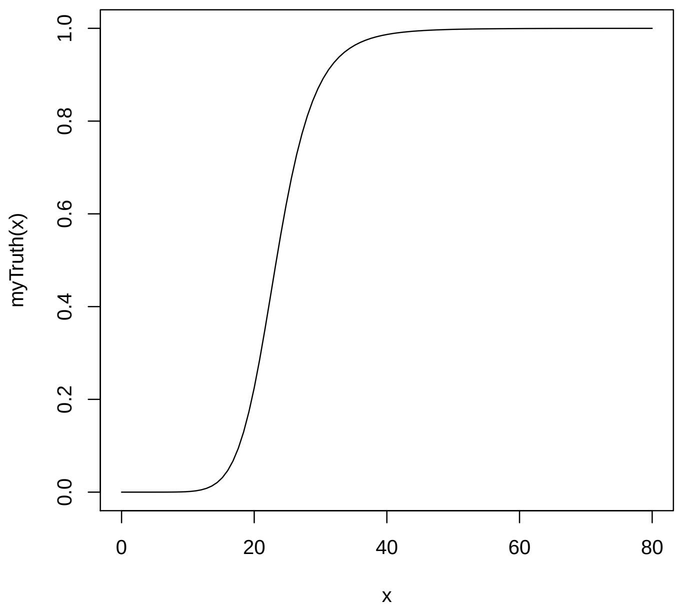
In order to run simulations, we first have to build a specific design, that comprises a model, the escalation rules, starting data, a cohort size (currently fixed during the trial) and a starting dose. It might seem strange at first sight that we have to supply starting data to the design, but we will 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 <- 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=7, alpha1=8)
  }
> ## 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 100 trial outcomes here for illustration, for the actual study this should be increased of course to at least 500:

```
> time <- system.time(mySims <- simulate(design,
                                         args=NULL,
                                         truth=myTruth,
                                         nsim=100,
                                         seed=819,
                                         mcmcOptions=options,
                                         parallel=TRUE))[3]
> time
```

```
elapsed
  81.63
```

We have wrapped the call to `simulate` in a `system.time` to obtain the required time for the simulations (about 82 seconds in this case). The argument `args` could contain additional arguments for the `truth` function, which we did not require here and therefore let it at the default `NULL`. We specify the number of simulations with `nsim` and the random number generator seed with `seed`. 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)
```

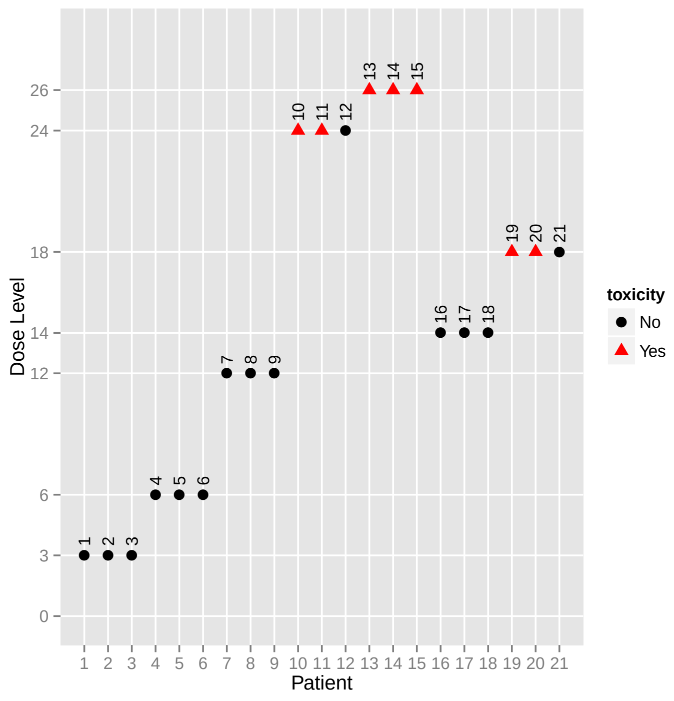
```
[1] "Simulations"
attr(,"package")
[1] "crmPack"
```

From the help page

```
> help("Simulations-class", help="html")
```

we see that this class is a subclass of the “GeneralSimulations” class. By looking at the help pages for “Simulations” and the parent class “GeneralSimulations”, we can find the description of all 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] 16
```

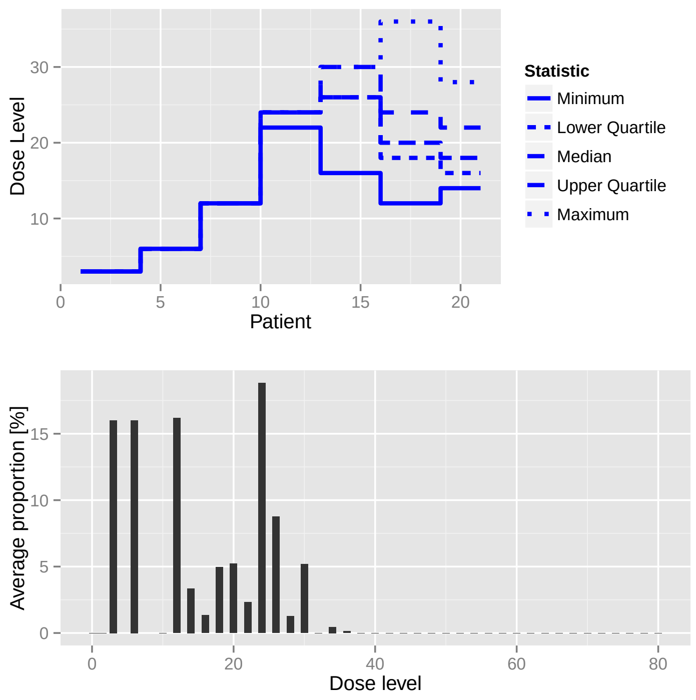
and the stopping reason was

```
> 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 56 % for dose 16 and thus above the required 50 %"


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

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

```
> print(plot(mySims))
```



The resulting plot shows on the top panel a summary of the trial trajectories. 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:

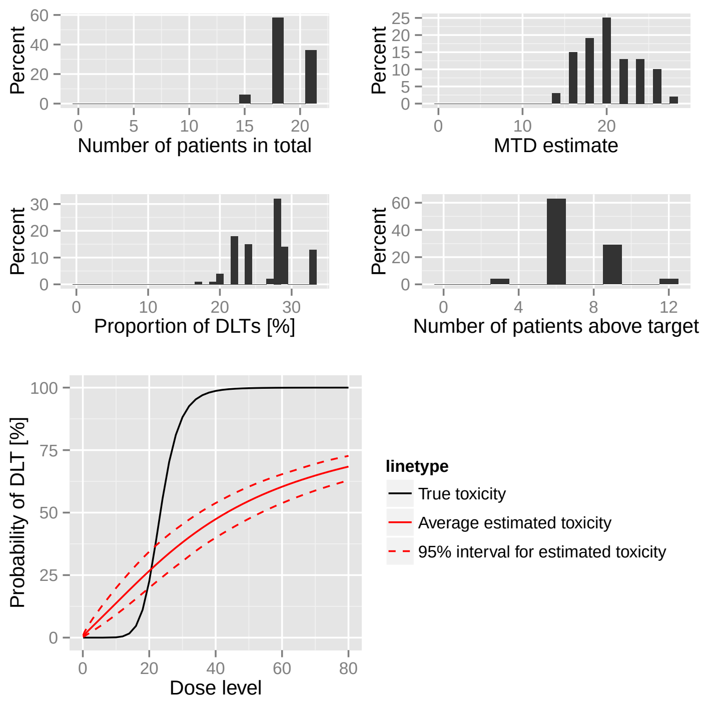
```
> summary(mySims,
          truth=myTruth)
```

```
Summary of 100 simulations

Target toxicity interval was 20, 35 %
Target dose interval corresponding to this was 19.6, 21.6
Intervals are corresponding to 10 and 90 % quantiles

Number of patients overall : mean 19 (18, 21)
Number of patients treated above target tox interval : mean 7 (6, 9)
Proportions of DLTs in the trials : mean 26 % (22 %, 33 %)
Mean toxicity risks for the patients : mean 26 % (18 %, 34 %)
Doses selected as MTD : mean 20.4 (16, 26)
True toxicity at doses selected : mean 29 % (5 %, 70 %)
Proportion of trials selecting target MTD: 25 %
Dose most often selected as MTD: 20
Observed toxicity rate at dose most often selected: 23 %
Fitted toxicity rate at dose most often selected : mean 27 % (21 %, 32 %)
```

Note that sometimes the observed toxicity rate at the dose most often selected (here 20 mg) is not available, because it can happen that no patients were actually treated that dose during the simulations. (Here it is available.) This illustrates that the MTD can be selected based on the evidence from the data at other dose levels – which is an advantage of model-based dose-escalation designs.

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

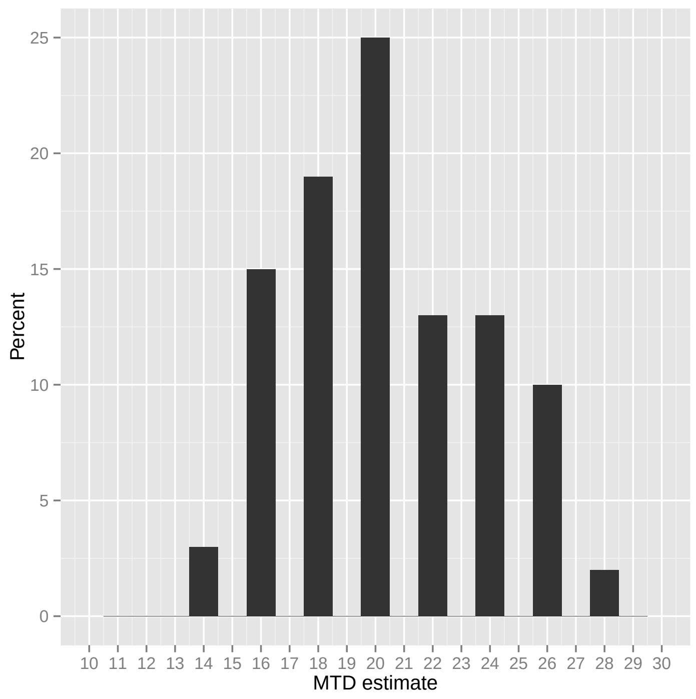
```
> simSum <- summary(mySims,
                    truth=myTruth)
> print(plot(simSum))
```



The top left panel shows the distribution of the sample size across the simulated trials. In this case the trials had between 15 and 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 in most trials between 20 and 30% of the patients had DLTs. 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%). 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 that the steep true dose-toxicity curve is not recovered by the model fit.

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 information on customizing the plots)

```
> dosePlot <- plot(simSum, type="doseSelected") +
        scale_x_continuous(breaks=10:30, limits=c(10, 30))
> print(dosePlot)
```



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: 0x000000000aa44358>
```

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    0.702251325 0.4536462
100  -2.204545161 1.5423571
150  -1.205057702 2.2682090
200  -0.459867721 1.2273199
250   0.605039226 1.5172805
300   0.106094712 1.1018841
350  -0.307693286 0.9927107
400   0.006182696 0.4175478
450  -0.178415278 0.6002686
500   0.316565610 1.0027199
550  -1.169523805 1.8327901
600  -0.485561605 0.8332442
650  -1.198065520 0.5144423
700  -0.466094586 0.6709419
750   0.092395469 1.7562952
800   0.917129172 0.7363936
850  -0.445657146 0.5083458
900  -2.011125006 3.0098420
950  -0.593644102 0.4814157
1000  0.242875509 0.7727379
```

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 <- 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)
```

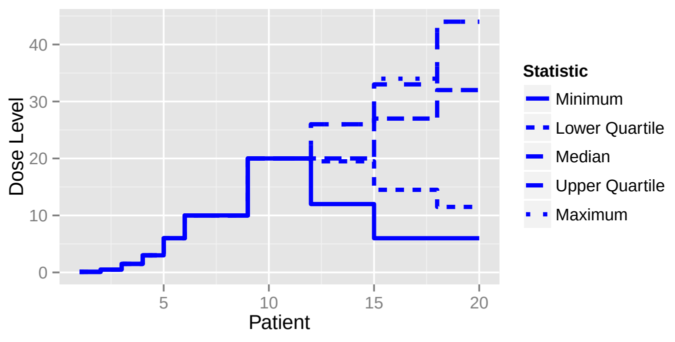
Finally we can execute the simulations:

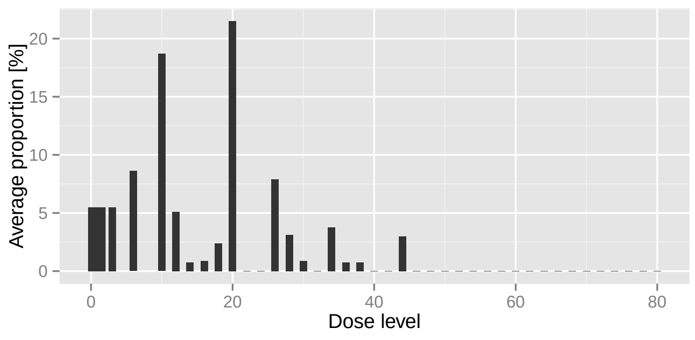
```
> time <- system.time(futureSims <- simulate(
      ## supply the new design here
      nowDesign,
      ## the truth is the assumed prob function
      truth=model@prob,
      ## further arguments are the
      ## posterior samples
      args=postSamples,
      ## do exactly so many simulations as
      ## we have samples
      nsim=nrow(postSamples),
      seed=918,
      ## this remains the same:
      mcmcOptions=options,
        parallel=TRUE))[3]
> time
```

```
elapsed
  14.83
```

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:

```
> print(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 19.6, 21.6
Intervals are corresponding to 10 and 90 % quantiles

Number of patients overall : mean 18 (17, 20)
Number of patients treated above target tox interval : mean 4 (0, 9)
```

```
Proportions of DLTs in the trials : mean 20 % (10 %, 30 %)
Mean toxicity risks for the patients : mean 22 % (5 %, 43 %)
Doses selected as MTD : mean 25.8 (10, 47)
True toxicity at doses selected : mean 49 % (0 %, 100 %)
Proportion of trials selecting target MTD: 5 %
Dose most often selected as MTD: 10
Observed toxicity rate at dose most often selected: 32 %
Fitted toxicity rate at dose most often selected : mean 16 % (6 %, 26 %)
```

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.

9 Simulating 3+3 design outcomes

While `crmPack` focuses on model-based dose-escalation designs, it now includes the 3+3 design in order to allow for convenient comparisons. Note that actually no simulations would be required for the 3+3 design, because all possible outcomes can be enumerated, however we still rely here on simulations for consistency with the overall `crmPack` design.

The easiest way to setup a 3+3 design is the function `ThreePlusThreeDesign`:

```
> threeDesign <- ThreePlusThreeDesign(doseGrid=c(5, 10, 15, 25, 35, 50, 80))
> class(threeDesign)
```

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

We have used here a much coarser dose grid than for the model-based design before, because the 3+3 design cannot jump over doses. The starting dose is automatically chosen as the first dose from the grid. The outcome is a `RuleDesign` object, and you have more setup options if you directly use the `RuleDesign()` initialization function. We can then simulate trials, again assuming that the `myTruth` function gives the true dose-toxicity relationship:

```
> threeSims <- simulate(threeDesign,
                        nsim=1000,
                        seed=35,
                        truth=myTruth,
                        parallel=TRUE)
```

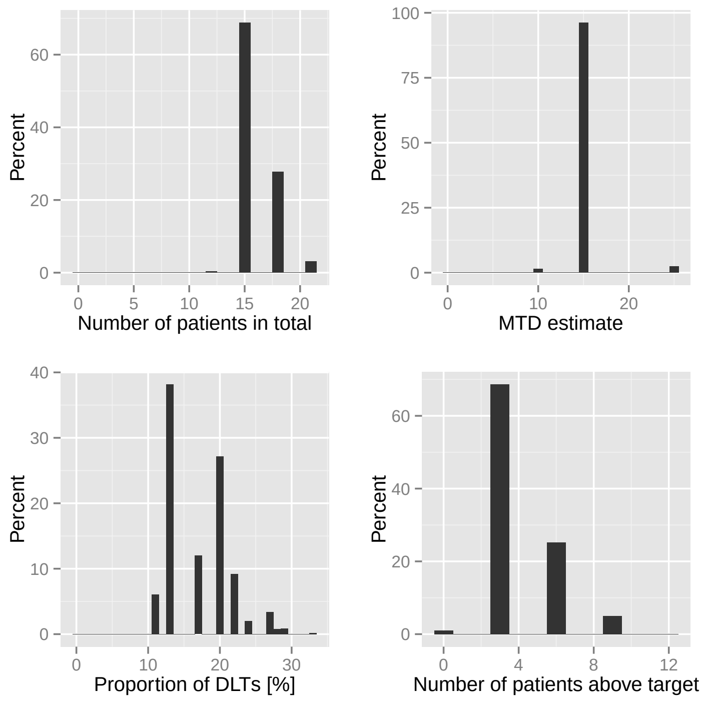
As before for the model-based design, we can summarize the simulations:

```
> threeSimsSum <- summary(threeSims,
                          truth=myTruth)
> threeSimsSum
```

```
Summary of 1000 simulations

Target toxicity interval was 20, 35 %
Target dose interval corresponding to this was 19.6, 21.6
Intervals are corresponding to 10 and 90 % quantiles

Number of patients overall : mean 16 (15, 18)
Number of patients treated above target tox interval : mean 4 (3, 6)
Proportions of DLTs in the trials : mean 17 % (13 %, 22 %)
Mean toxicity risks for the patients : mean 17 % (14 %, 22 %)
Doses selected as MTD : mean 15.2 (15, 15)
True toxicity at doses selected : mean 4 % (3 %, 3 %)
Proportion of trials selecting target MTD: 0 %
Dose most often selected as MTD: 15
Observed toxicity rate at dose most often selected: 3 %
```

Here we see that 15 mg was the dose most often selected as MTD, and this is actually too low when comparing with the narrow target dose interval going from 19.6 to 21.6 mg. This is an inherent problem of dose-escalation designs where the dose grid has to be coarse: you might not know before starting the trial which is the range where you need a more refined dose grid. In this case we obtain doses that are too low, as one can see from the average true toxicity of 4 % at doses selected. Graphical summaries are again obtained by calling "plot" on the summary object:

```
> print(plot(threeSimsSum))
```



10 Dual-endpoint designs

The latest version of `crmPack` includes dual-endpoint designs. These are still under development, and so far we have not yet published a peer-reviewed paper on these methods, therefore please consider them as experimental.

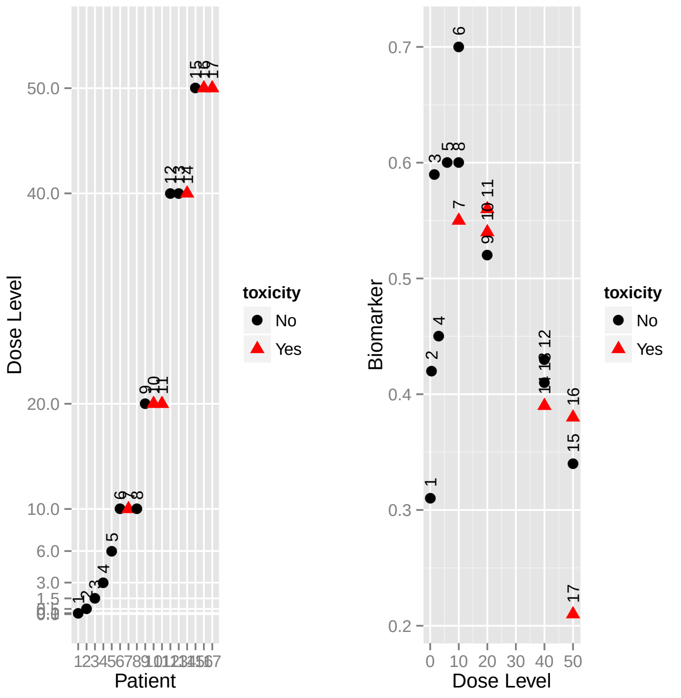
In the help page "DualEndpoint-class" the general model structure is described, and on the Hive page more information can be found. Basically the idea is that a (single) biomarker variable is the second endpoint of the dose-escalation design, with the aim to maximize the biomarker response while controlling toxicity in a safe range. This is useful when it can not be assumed that just increasing the dose will always lead to a better clinical response.

Let's look at the data structure. Here is an example:

```
> data <- DataDual(
      x=
          c(0.1, 0.5, 1.5, 3, 6, 10, 10, 10,
            20, 20, 20, 40, 40, 40, 50, 50, 50),
      y=
          c(0, 0, 0, 0, 0, 0, 1, 0,
            0, 1, 1, 0, 0, 1, 0, 1, 1),
      w=
          c(0.31, 0.42, 0.59, 0.45, 0.6, 0.7, 0.55, 0.6,
            0.52, 0.54, 0.56, 0.43, 0.41, 0.39, 0.34, 0.38, 0.21),
      doseGrid=
          c(0.1, 0.5, 1.5, 3, 6,
            seq(from=10, to=80, by=2)))
```

The corresponding plot can again be obtained with:

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



Here we see that there seems to be a maximum biomarker response at around 10 mg already. In order to model this data, we consider a dual-endpoint model with RW1 structure for the dose-biomarker relationship:

```
> model <- DualEndpointRW(mu=c(0, 1),
                          Sigma=matrix(c(1, 0, 0, 1), nrow=2),
                          sigma2betaW=
                          0.01,
                          sigma2W=
                          c(a=0.1, b=0.1),
                          rho=
                          c(a=1, b=1),
                          smooth="RW1")
```

We use a smoothing parameter $\sigma^2_{\beta_W} = 0.01$, an inverse-gamma prior $\mathrm{IG}(0.1, 0.1)$ on the biomarker variance σ^2_W and a uniform prior (or $\mathrm{Beta}(1, 1)$ prior) on the correlation ρ between the latent DLT and the biomarker variable.

As the dual-endpoint models are more complex, it is advisable to use a sufficiently long Markov chain for fitting them. Here we just use for illustration purposes quite a small Markov chain – again, for the real application, this would need to be at least 100 times longer!

```
> options <- McmcOptions(burnin=100,
                         step=2,
                         samples=500)
```

Then we can obtain the MCMC samples:

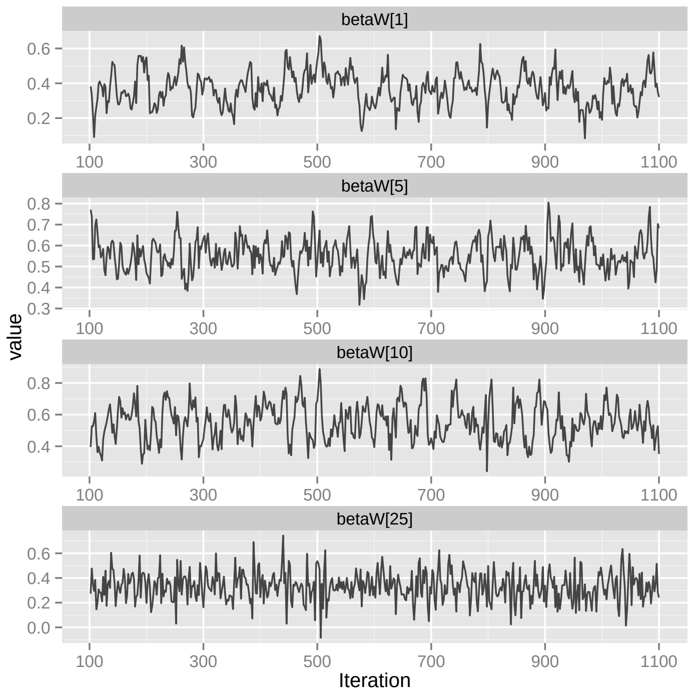
```
> samples <- mcmc(data, model, options)
```

And we check the convergence by picking a few of the fitted biomarker means and plotting their traceplots:

```
> data@nGrid
```

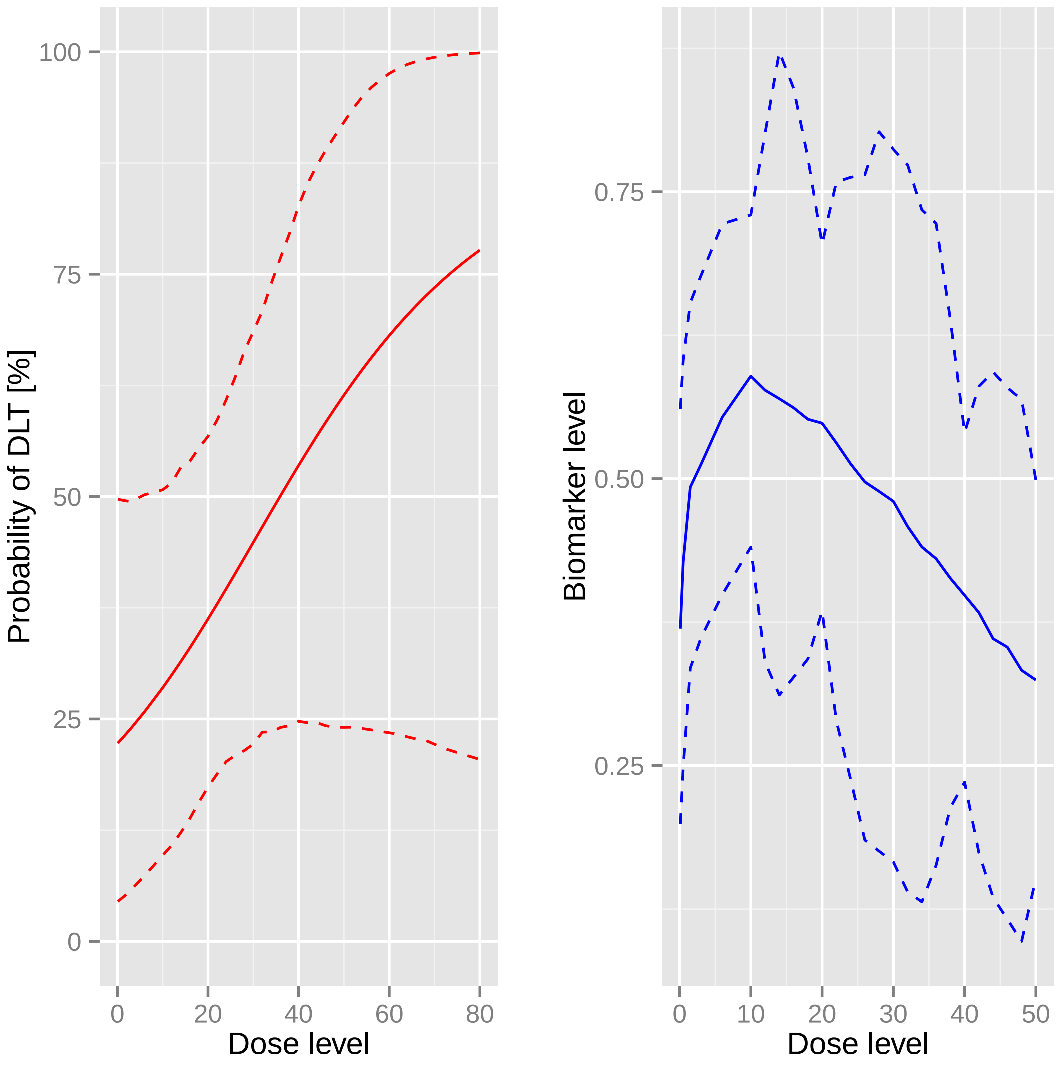
```
[1] 41
```

```
> betaWpicks <- get(samples, "betaW", c(1, 5, 10, 25))
> ggs_traceplot(betaWpicks)
```



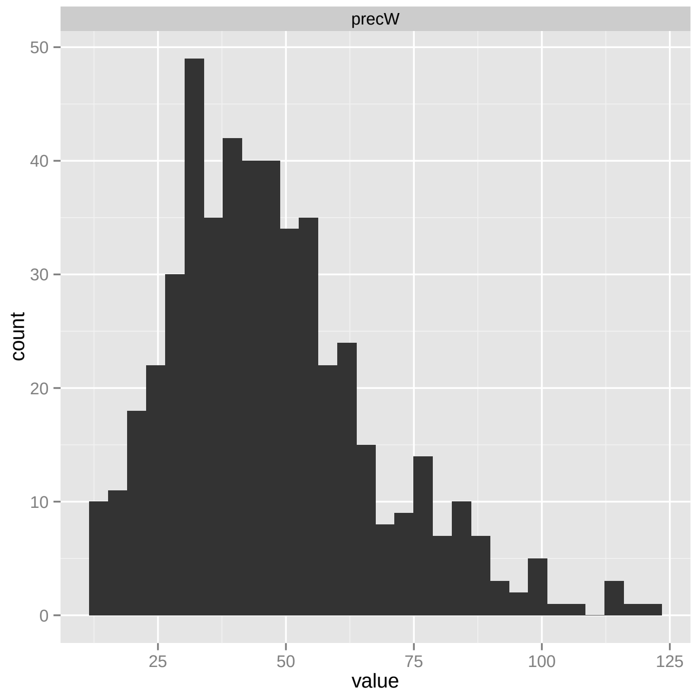
Here all 4 $\beta_{W,j}$ $(j = 1, 5, 10, 25)$ means, which are the biomarker means at the first, 5th, 10th and 25th gridpoint, respectively, seem to have converged, as the traceplots show. (Remember that `data@nGrid` gives the number of gridpoints.) So we can plot the model fit:

```
> print(plot(samples, model, data, extrapolate=FALSE))
```



We specify `extrapolate = FALSE` to focus the biomarker plot in the right panel on the observed dose range, so we don't want to extrapolate the biomarker fit to higher dose levels. We can also look at the estimated biomarker precision $1/\sigma_W^2$. For that we extract the precision "precW" and then use another `ggmcmc` function to create the histogram:

```
> ggs_histogram(get(samples, "precW"))
```



For the selection of the next best dose, a special class "NextBestDualEndpoint" has been implemented. It tries to maximize the biomarker response, under an NCRM-type safety constraint. If we want to have at least 90% of the maximum biomarker response, and a 25% maximum overdose probability for the next dose, we specify:

```
> myNextBest <- NextBestDualEndpoint(target=0.9,
                                     overdose=c(0.35, 1),
                                     maxOverdoseProb=0.25)
```

In our example, and assuming a dose limit of 50 mg given by the maximum allowable increments, the next dose can then be found as follows:

```
> nextDose <- nextBest(myNextBest,
                       doselimit=50,
                       samples=samples,
                       model=model,
                       data=data)
> nextDose$value
```

```
[1] 6
```

A corresponding plot can be produced by printing the "plot" element of the returned list:

```
> print(nextDose$plot)
```

Here the bottom panel shows (as for the NCRM) the overdose probability, and we see that doses above 6 mg are too toxic. In the top panel, we see the probability for each dose to reach at least 90% of the maximum biomarker response in the dose grid — this is here our target probability. While the numbers are low, we clearly see that there is a local maximum at 10 mg of the target probability, confirming what we have seen in the previous data and model fit plots.

A corresponding stopping rule exists. When we have a certain probability to be above a relative biomarker target, then the "StoppingTargetBiomarker" rule gives back `TRUE` when queried if it has been fulfilled by the `stopTrial` function. For example, if we require at least 50% probability to be above 90% biomarker response, we specify:

```
> myStopping4 <- StoppingTargetBiomarker(target=0.9,
                                         prob=0.5)
```

In this case, the rule has not been fulfilled yet, as we see here:

```
> stopTrial(myStopping4, dose=nextDose$value,
            samples, model, data)
```

```
[1] FALSE
attr(,"message")
[1] "Probability for target biomarker is 10 % for dose 6 and thus below the required 50 %"
```

Again, this dual-endpoint specific rule can be combined as required with any other stopping rule. For example, we could combine it with a maximum sample size of 40 patients:

```
> myStopping <- myStopping4 | StoppingMinPatients(40)
```

If one or both of the stopping rules are fulfilled, then the trial is stopped.

Let's try to build a corresponding dual-endpoint design. We start with an empty data set, and use the relative increments rule defined in a previous section and use a constant cohort size of 3 patients:

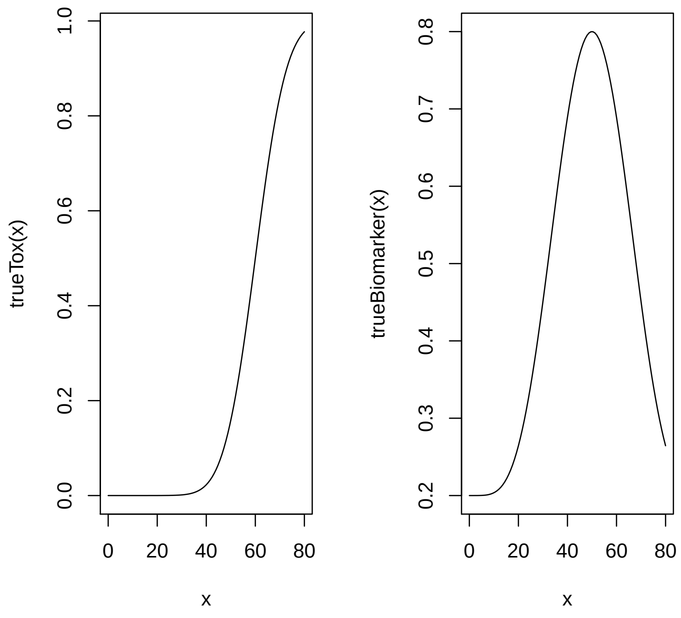
```
> emptydata <- DataDual(doseGrid=data@doseGrid)
> design <- DualDesign(model=model,
                       data=emptydata,
                       nextBest=myNextBest,
                       stopping=myStopping,
                       increments=myIncrements,
                       cohortSize=CohortSizeConst(3),
                       startingDose=6)
```

In order to study operating characteristics, we need to determine true biomarker and DLT probability functions. Here we are going to use a biomarker function from the beta family. Note that there is a corresponding "DualEndpointBeta" model class, that allows to have dual-endpoint designs with the beta biomarker response function. Have a look at the corresponding help page for more information on that. But let's come back to our scenario definition:

```
> betaMod <- function (dose, e0, eMax, delta1, delta2, scal)
  {
      maxDens <- (delta1^delta1) * (delta2^delta2)/((delta1 + delta2)^(delta1 + delta2))
      dose <- dose/scal
      e0 + eMax/maxDens * (dose^delta1) * (1 - dose)^delta2
  }
> trueBiomarker <- function(dose)
  {
      betaMod(dose, e0=0.2, eMax=0.6, delta1=5, delta2=5 * 0.5 / 0.5, scal=100)
  }
> trueTox <- function(dose)
  {
      pnorm((dose-60)/10)
  }
```

We can draw the corresponding curves:

```
> par(mfrow=c(1, 2))
> curve(trueTox(x), from=0, to=80)
> curve(trueBiomarker(x), from=0, to=80)
```



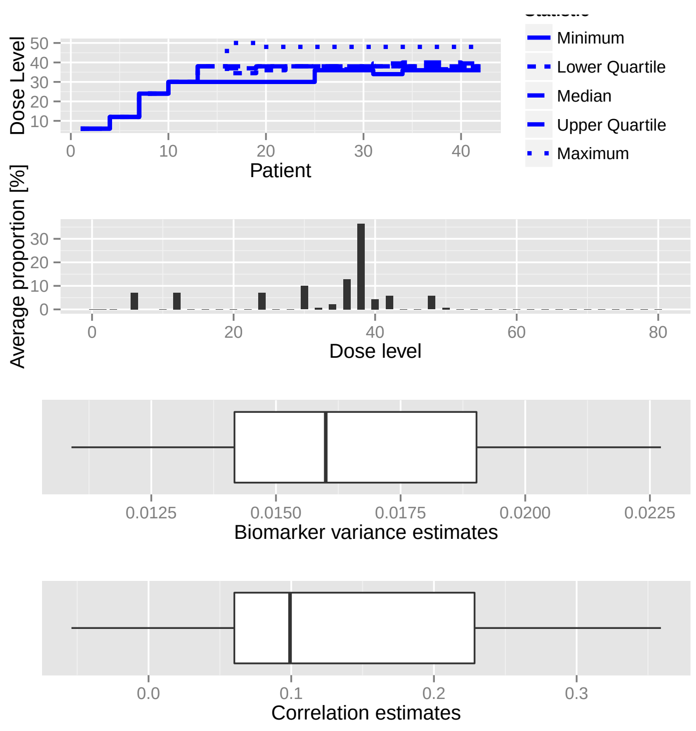
So the biomarker response peaks at 50 mg, where the toxicity is still low. After deciding for a true correlation of $\rho = 0$ and a true biomarker variance of $\sigma_W^2 = 0.01$ (giving a high signal-to-noise ratio), we can start simulating trials (starting each with 6 mg):

```
> mySims <- simulate(design,
                     trueTox=trueTox,
                     trueBiomarker=trueBiomarker,
                     sigma2W=0.01,
                     rho=0,
                     nsim=10,
                     parallel=TRUE,
                     seed=3,
                     startingDose=6,
                     mcmcOptions =
                         McmcOptions(burnin=1000,
                                     step=1,
                                     samples=3000))
```

Note that we are having a "small" MCMC option set here, in order to reduce simulation time — for the real application, this should be "larger".

Plotting the result gives not only an overview of the final dose recommendations and trial trajectories, but also a summary of the biomarker variance and correlation estimates in the simulations:

```
> print(plot(mySims))
```



Finally, a summary of the simulations can be obtained with the corresponding function:

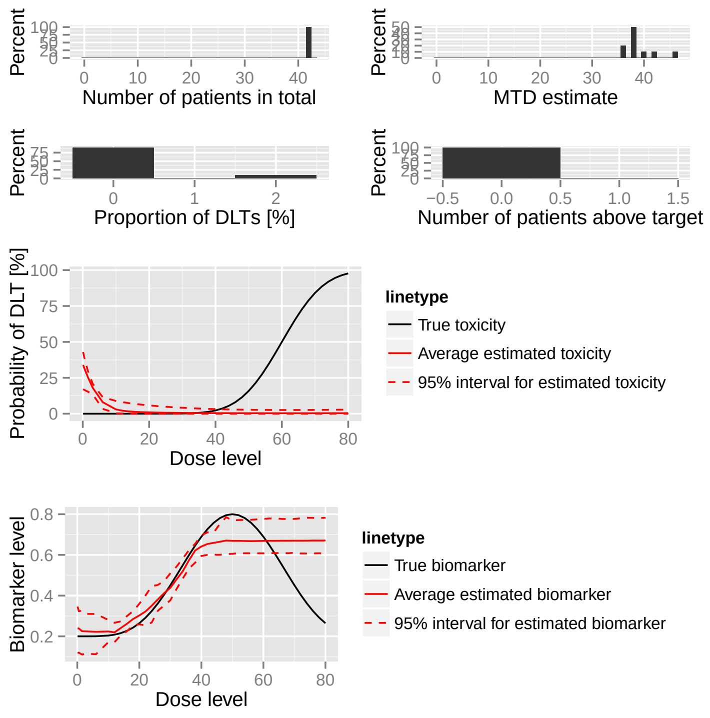
```
> sumOut <- summary(mySims,
                     trueTox=trueTox,
                     trueBiomarker=trueBiomarker)
> sumOut
```

```
Summary of 10 simulations

Target toxicity interval was 20, 35 %
Target dose interval corresponding to this was 51.6, 56.1
Intervals are corresponding to 10 and 90 % quantiles

Number of patients overall : mean 42 (42, 42)
Number of patients treated above target tox interval : mean 0 (0, 0)
Proportions of DLTs in the trials : mean 0 % (0 %, 0 %)
Mean toxicity risks for the patients : mean 2 % (1 %, 3 %)
Doses selected as MTD : mean 39 (36, 42.4)
True toxicity at doses selected : mean 2 % (1 %, 4 %)
Proportion of trials selecting target MTD: 0 %
Dose most often selected as MTD: 38
Observed toxicity rate at dose most often selected: 0 %
Fitted toxicity rate at dose most often selected : mean 0 % (0 %, 0 %)
Fitted biomarker level at dose most often selected : mean 0.6 (0.6, 0.7)
```

We see here that all trials proceeded until the maximum sample size of 40 patients (reaching 42 because of the cohort size 3). The doses selected are lower than the toxicity target range, because here we were aiming for a biomarker target instead, and the true biomarker response peaked at 50 mg.

The corresponding plot looks as follows:

```
> print(plot(sumOut))
```



We see that the average biomarker fit is not too bad in the range up to 50 mg, but the toxicity curve fit is bad — probably a result of the very low frequency of DLTs. Again the warning here: the dual-endpoint designs are still experimental!

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`.