DejaVu User Guide

Nikolas S. Burkoff[**] and David Ruau[*]

[*]AstraZeneca, B&I, Advanced Analytics Centre, UK
[**]Tessella, 26 The Quadrant, Abingdon Science Park, Abingdon, OX14 3YS, UK

November 8, 2024

1 Introduction

The `dejaVu` package performs multiple imputation on recurrent event data sets. The package allows users to import or simulate a "complete" data set (one without any subjects dropping out). User-specified dropout mechanisms can then be applied to this data to generate a "dropout" data set (or a dropout set can be imported directly). A multiple imputation procedure is then applied to the dropout data set and the package allows user specified imputation mechanisms to be created. Finally, the imputed data sets can be analysed and their results combined using Rubin's rules.

2 Running a Single MI Example

In this section we walk through a single MI example:

- Simulating Complete Data
- Simulating Subject Dropout
- Generating MI data sets
- Analysing results using Rubin's formula

See later in the guide for using covariates with the package and for details of running a large number of examples and combining the results to estimate summary statistics such as power and Type I error.

In order to use the functionality of the package, it must be loaded:

```
library(dejaVu)
```

Note: due to namespace conflicts between libraries `MASS` and `trafficlight`, `dejaVu` does will not work if `trafficlight` is loaded.

For reproducibility we set the random seed:

```
set.seed(1298711)
```

2.1 Simulating Complete Data

The `SimulateComplete` function is used to generate a complete data set of subject outcomes for a recurrent event study with follow up time T and the number of events for each subject, n_i is given by a negative binomial process.

Under the negative binomial assumption the time interval between adjacent event for a given subject are exponentially distributed with rate λ_i/T where $\lambda_i \sim \text{Gamma}(1/k, k\mu)$ and k is the dispersion parameter and μ the mean number of events (k and μ can be different for each treatment group).

Specifically,

$$\mathbb{P}(n_i = r) = NB(p, \gamma) = \frac{\Gamma(\gamma + r)}{\Gamma(r+1)\Gamma(\gamma)} p^r (1-p)^\gamma \tag{1}$$

where $\gamma = 1/k$ and $p = \frac{k\mu}{1+k\mu}$.

The `SimulateComplete` function takes the following arguments:

- `study.time` $= T$
- `number.subjects`: the number of subjects in each arm to simulate - if it is a single number then this will be used for both arms, otherwise a vector `c(number.control,number.active)`
- `event.rates` $= \mu/T$: the event rates for each arm
- `dispersions` $= k$, again either a single number if k is the same for both arms otherwise a vector.
- `dejaData`, this advanced option is used only when subject covariates are included; see later in the guide for further information

An example of the data generation procedure:

```
complete <- SimulateComplete(study.time=365,
                      number.subjects=50,
                      event.rates=c(0.01,0.005),
                      dispersions=0.25)
print(complete)

## complete dataset
## Study follow up period: 365
## Negative binomial event rates: 0.01 0.005
## Negative binomial dispersion: 0.25
## Data:
## 'data.frame': 100 obs. of  5 variables:
##  $ Id             : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ arm            : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ censored.time  : num  365 365 365 365 365 365 365 365 365 365 ...
##  $ observed.events: num  4 2 2 3 10 6 15 5 3 5 ...
##  $ actual.events  : num  4 2 2 3 10 6 15 5 3 5 ...

summary(complete)

## complete dataset
## Study follow up period: 365
## Subjects (per arm): 50 50
## Subject dropouts (per arm): 0 0
## Number of events (per arm): 212 69
## Total Time at risk (per arm): 18250 18250
## Empirical event rates (per arm): 0.01161644 0.003780822
```

We can also access the data directly:

```
head(complete$data)

##   Id arm censored.time observed.events actual.events
## 1  1   0           365               4             4
## 2  2   0           365               2             2
## 3  3   0           365               2             2
## 4  4   0           365               3             3
## 5  5   0           365              10            10
## 6  6   0           365               6             6

#The event times for subject with Id 1
complete$event.times[[1]]

## [1] 104.3637 173.7090 195.1324 200.4362
```

The `SimulateComplete` function returns a `SingleSim` object. See `help(SingleSim)` for further details.

2.2 Subject Dropout

Given a `SingleSim` object, we can use the `SimulateDropout` function to apply a dropout mechanism to create a data set which includes subject dropout. In this example we use a simple MCAR example.

The `ConstantRateDrop` mechanism will give subjects exponentially distributed drop out times with subject specific rate R where $R = (\texttt{rate})\exp(X_i)$ where $X_i \sim \mathcal{N}(0,\texttt{var})$. If the simulated drop out time is $< T$ then the subject drops out and any events which occur after dropout are not observed.

```
ConstantRateDrop(rate=0.0025,var=1)

## Dropout Type: MCAR
## Dropout Mechanism: Constant rate per subject
## Parameters:
##    rate :  0.0025
##    between.subject.var :  1
```

This dropout mechanism can be used to create a data set with subject dropouts.

```
with.MCAR.dropout <- SimulateDropout(complete,
                    drop.mechanism=ConstantRateDrop(rate=0.0025,
                                                    var=1)) #var by default=0
```

Note the `censored.time` and `observed.events` have been updated:

```
summary(with.MCAR.dropout)

## dropout dataset
## Study follow up period: 365
## Subjects (per arm): 50 50
## Subject dropouts (per arm): 31 29
## Number of events (per arm): 126 35
## Total Time at risk (per arm): 10233.38 12147.64
## Empirical event rates (per arm): 0.01231265 0.002881218
```

```
head(with.MCAR.dropout$data)

##   Id arm censored.time observed.events actual.events
## 1  1   0     365.00000               4             4
## 2  2   0      19.36295               0             2
## 3  3   0     365.00000               2             2
## 4  4   0     365.00000               3             3
## 5  5   0      16.19086               2            10
## 6  6   0     365.00000               6             6
```

See later in the tutorial for other available dropout mechanisms and a description of how to implement your own.

2.3 Generating MI data sets

Given a `SingleSim` object we can fit a negative binomial model using the `Simfit` function. **Note a warning will be displayed if the model fit does not converge this is possible, especially if there are a small number of subjects**.

```
my.fit <- Simfit(with.MCAR.dropout,
                 equal.dispersion=TRUE)
```

The function `glm.nb` from the package `MASS` is used with the following models:

- `observed.events` ∼ `arm + offset(log(censored.time))` if `equal.dispersion` is `TRUE`
- `observed.events` ∼ `offset(log(censored.time))` if `equal.dispersion` is `FALSE` and separate models are fitted for each arm

It is possible to include covariates in the model formula using the `covar` argument; see Section 5 for further details. Additional arguments to `Simfit` are passed to the model fitting function. For example `Simfit(with.MCAR.dropout,maxit=50)` will increase the number of iterations when fitting the model (see `help(glm.control)` for further details).

This creates a `SingleSimFit` object:

```
class(my.fit)

## [1] "SingleSimFit"

summary(my.fit)

## Summary for model fit for dropout data set
## Treatment Effect: 0.2248795
## SE (log) treatment effect: 0.247891
## 95% CI: [0.138339, 0.365557]
## Rate Estimates (per arm): 0.01265197 0.002845169
## Negative binomial dispersion model parameter: 0.4803851
## p-value: 1.749095e-09
## Note p-value is associated with this individual data set

#Can access the individual elements of the summary object
x <- summary(my.fit)
x$pval

## [1] 1.749095e-09
```

We can output γ and μ which would be used for the imputation (see Equation (1)) – γ, a vector containing the 1/dispersion for first the control arm and then the active arm and μ a matrix containing the predicted mean number of events per subject, one row per subject, one column for if the subject were to have been in the control arm, the second for if the subject had been in the active arm.

The function `genCoeff.function` inside the `SingleSimFit` object takes one logical argument `use.uncertainty`. If `FALSE` then no uncertainty is incorporated into the parameter estimates output. By default, it is set to `TRUE`, whereby each call to the function generates its own sample of γ and μ which takes into account uncertainty using the `mvrnorm` function from `MASS`.

Note that if there are covariates in the model then μ is a function of these covariates.

```
#First get values direct from model fit
gamma_and_mu <- my.fit$genCoeff.function(use.uncertainty=FALSE)

## Warning in my.fit$genCoeff.function(use.uncertainty = FALSE): Not using uncertainty in parameters
therefore imputation will not be proper

gamma_and_mu$gamma

## [1] 2.081663 2.081663

head(gamma_and_mu$mu, 5)

##             [,1]        [,2]
## [1,] 0.01265197 0.002845169
## [2,] 0.01265197 0.002845169
## [3,] 0.01265197 0.002845169
## [4,] 0.01265197 0.002845169
## [5,] 0.01265197 0.002845169

#Now sample uncertainty in coefficients
#each imputation will call this function
#itself and so generate its own coefficients
#for the imputation
head(my.fit$genCoeff.function(use.uncertainty=TRUE)$mu,5)

##             [,1]        [,2]
## [1,] 0.01402533 0.003738611
## [2,] 0.01402533 0.003738611
## [3,] 0.01402533 0.003738611
## [4,] 0.01402533 0.003738611
## [5,] 0.01402533 0.003738611
```

Given a `SingleSimFit` object we can generate a set of imputed data sets:

```
imputed.data.sets <- Impute(fit = my.fit,
                            impute.mechanism = weighted_j2r(trt.weight=0),
                            N=10)

#output the number of subject dropouts in each arm
imputed.data.sets$dropout

## [1] 31 29
```

The `Impute` function requires three arguments, the fit, an impute mechanism and N the number of data sets to impute. We are using the weighted j2r impute mechanism with treatment weight = 0 which implies missing counts for subjects in both arms will be imputed according to the mean of the control arm conditioned on the number of subject's observed events. See later in the tutorial for further details, other available impute mechanisms and a description of how to implement your own.

2.4 Fitting MI datasets

It is possible to access the individual imputed data sets (note these data sets assume the imputed data is 'truth' and hence as far as this data set is concerned the number of subject dropouts is 0):

```
sixth.data.set <- GetImputedDataSet(imputed.data.sets,index=6)

summary(sixth.data.set)

## imputed dataset
## Study follow up period: 365
## Subjects (per arm): 50 50
## Subject dropouts (per arm): 0 0
## Number of events (per arm): 232 93
## Total Time at risk (per arm): 18250 18250
## Empirical event rates (per arm): 0.01271233 0.00509589

head(sixth.data.set$data)

##   Id arm censored.time observed.events actual.events actual.censored.time
## 1  1   0           365               4             4            365.00000
## 2  2   0           365               4             2             19.36295
## 3  3   0           365               2             2            365.00000
## 4  4   0           365               3             3            365.00000
## 5  5   0           365               6            10             16.19086
## 6  6   0           365               6             6            365.00000
```

Note the `actual.censored.time` column is the time the subject was actually censored at whereas the `censored.time` column is the time the subject was censored after applying the imputation. Similarly the `actual.events` is the actual number of events which occurred and `observed.events` is the number of events which were 'observed' including imputed events.

We can fit a model to imputed data sets. By default the `Simfit` function fits a negative binomial model, however, by using the family argument a Poission or quasi-Poisson model can be used instead (the same model formula is used as in the negative binomial case however `glm` is used rather than `glm.nb`)

```
sixth.fit <- Simfit(sixth.data.set,
                    family="poisson")

summary(sixth.fit)

## Summary for model fit for imputed data set
## Treatment Effect: 0.4008621
## SE (log) treatment effect: 0.1227306
## 95% CI: [0.3151567, 0.5098747]
## Rate Estimates (per arm): 0.01271233 0.00509589
## Negative binomial dispersion model parameter: NA
## p-value: 9.453043e-14
## Note p-value is associated with this individual data set
```

It is possible to fit the entire set of imputed data sets in one go, again using the `Simfit` function (again the `covar` argument can be used to include covariates in the model fit):

```
fitted <- Simfit(imputed.data.sets,
                 family="negbin") #negbin is the default
```

We can create a data frame collating the fitted values:

```
head(as.data.frame(fitted))

##   imputeID control.rate active.rate treatment.effect        se         pval
## 1        1   0.01260274 0.004054795        0.3217391 0.1928930 4.127982e-09
## 2        2   0.01243836 0.006520548        0.5242291 0.1819008 3.845993e-04
## 3        3   0.01276712 0.006136986        0.4806867 0.1963729 1.912148e-04
## 4        4   0.01282192 0.005972603        0.4658120 0.1808050 2.385183e-05
## 5        5   0.01123288 0.005972603        0.5317073 0.1901999 8.968152e-04
## 6        6   0.01271233 0.005095890        0.4008621 0.1856446 8.473658e-07
##   dispersion
## 1  0.4836589
## 2  0.5069815
## 3  0.6335479
## 4  0.4810659
## 5  0.5530908
## 6  0.4850217
```

We can also summarise the results – the values shown here are calculated using Rubin's formula [1].

```
summary(fitted)

## Summary for imputed data sets
## Treatment Effect: 0.4757741
## SE (log) treatment effect: 0.2790041
## Degrees of freedom: 33.13335
## p-value: 0.01187991
## Adjusted d.o.f: 19.25946
## Adjusted p-value: 0.01527075
## Average dispersion: 0.5844744
## Number of subjects dropped out per arm: 31 29
```

3 Running Complete Scenarios

In order to calculate the summary statistics such as power it is necessary to repeat the procedure multiple times. In this example we show how to easily replicate and combine the results.

First we create a function which outputs a list of the summaries of the fits we are interested in:

```r
example.scenario <- function(){

  #simulate a complete data set
  sim <- SimulateComplete(study.time=365,number.subjects=125,
                      event.rates=c(0.01,0.005),dispersions=0.25)

  #take the simulated data set and apply an MCAR dropout mechanism...
  sim.with.MCAR.dropout <- SimulateDropout(sim,
                     drop.mechanism=ConstantRateDrop(rate=0.0025))

  #fit a Negative Binomial model
  with.MCAR.fit <- Simfit(sim.with.MCAR.dropout,equal.dispersion=TRUE)

  #we can impute a set of 10 sets following the j2r mechanism using the fit
  impute.data.sets <- Impute(with.MCAR.fit,impute.mechanism = weighted_j2r(trt.weight=0),N=10)

  #we can then fit models to the entire imputed data set
  fit.imputed.set <- Simfit(impute.data.sets)

  #output the summary values
  return(list(MI=summary(fit.imputed.set), #for MI
              dropout=summary(with.MCAR.fit), #for dropout
              complete=summary(Simfit(sim)))) #for complete data set
}
```

Next we run the simulation a large number of times (In this example, two of the 6000 model fits did not fully converge, hence the warning messages):

```r
answer <- replicate(500,example.scenario(),simplify = FALSE)

## Warning in theta.ml(Y, mu, sum(w), w, limit = control$maxit, trace = control$trace > :  iteration
limit reached
## Warning in theta.ml(Y, mu, sum(w), w, limit = control$maxit, trace = control$trace > :  iteration
limit reached
```

and process the results using the `extract_results` function (note the `extract_results` is a simple wrapper around the `CreateScenario` function):

```r
#answer contains a list of lists each containing 3 SimFit objects
names(answer[[1]])

## [1] "MI"       "dropout"  "complete"

answer[[1]]$MI

## Summary for imputed data sets
## Treatment Effect: 0.7025427
## SE (log) treatment effect: 0.1235926
## Degrees of freedom: 56.02235
## p-value: 0.00599837
## Adjusted d.o.f: 40.59495
## Adjusted p-value: 0.006721929
## Average dispersion: 0.2129433
## Number of subjects dropped out per arm: 79 74

names(answer[[2]])

## [1] "MI"       "dropout"  "complete"

length(answer)

## [1] 500

#we can create a list with only the MI results
MI.fits <- lapply(answer,"[[","MI")

#the 2nd MI fit
MI.fits[[2]]

## Summary for imputed data sets
## Treatment Effect: 0.6568822
## SE (log) treatment effect: 0.1108937
## Degrees of freedom: 88.89575
## p-value: 0.000274139
## Adjusted d.o.f: 58.10357
## Adjusted p-value: 0.00036062
## Average dispersion: 0.1698235
## Number of subjects dropped out per arm: 75 84

#and create the scenario
MI.answer <- CreateScenario(MI.fits,description="the description of the scenario")


#The extract_results function can be used to both extract the list and
#create the scenario in one go
MI.answer <- extract_results(answer,name="MI",
                             description="Using j2r multiple imputation")
dropout.answer <- extract_results(answer,name="dropout",
                             description="Using no imputation")
complete.answer <- extract_results(answer,name="complete",
                             description="Using complete data sets")


class(MI.answer)

## [1] "Scenario"
```

We can output a summary of the simulations as a data frame and summarize the results:

```
head(as.data.frame(MI.answer))

##   replica treatment.effect        se         pval       df dispersion
## 1       1        0.7025427 0.1235926 5.998370e-03  56.02235  0.2129433
## 2       2        0.6568822 0.1108937 2.741390e-04  88.89575  0.1698235
## 3       3        0.6946910 0.1221099 4.187133e-03  57.20043  0.2384786
## 4       4        0.6906176 0.1235480 3.820314e-03  67.58339  0.2628547
## 5       5        0.6533744 0.1054918 6.730548e-05 348.02909  0.2547627
## 6       6        0.6043190 0.1088671 8.455864e-06 138.59122  0.2382916
##   dropout.control.rate dropout.active.rate
## 1                0.632               0.592
## 2                0.600               0.672
## 3                0.688               0.520
## 4                0.608               0.568
## 5                0.584               0.584
## 6                0.592               0.568
```

```
summary(dropout.answer)
```

```
## Scenario summary
## Description: Using no imputation
## Estimated Treatment Effect: 0.4967566
## Event Rate Reduction: 50.32434%
## Estimated SE (log) treatment effect: 0.122086
## Using alpha=0.05
##   power: 1
## Dropout rate summary statistics
## Control arm:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.440   0.568   0.600   0.597   0.632   0.728
## Active arm:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.464   0.568   0.600   0.598   0.632   0.736
```

```
summary(complete.answer)
```

```
## Scenario summary
## Description: Using complete data sets
## Estimated Treatment Effect: 0.5001534
## Event Rate Reduction: 49.98466%
## Estimated SE (log) treatment effect: 0.1025977
## Using alpha=0.05
##   power: 1
## Dropout rate summary statistics
## Control arm:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0       0       0       0       0       0
## Active arm:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0       0       0       0       0       0
```

```
#and can access individual elements
x <- summary(MI.answer,use.adjusted.pval=TRUE,alpha=0.025)

#power is calculated as the proportion of replicas which have
#pvalue < alpha
x$power
```

```
## [1] 0.958
```

4 Additional Dropout and Imputing Information

Additional Dropout mechanisms Alongside the `ConstantRateDrop` function. Additional dropout mechanisms have been implemented:

The `LinearRateChangeDrop` function allows a piecewise exponential drop out function where after j events subjects have drop out rate R_j where $R_j = C_j \exp(X_j)$ where $X_j \sim \mathcal{N}(0, \sigma^2)$ and $C_j = C + jD$ for constants C and D.

```
drop.mec <- LinearRateChangeDrop(starting.rate=0.0025, #C in text above
                                 rate.change=0.0005, #D in text above
                                 var=1) #sigma^2 in text above by default var=0

drop.mec

## Dropout Type: MAR
## Dropout Mechanism: Linear change in rate after each event
## Parameters:
##    starting.rate :  0.0025
##    rate.change.after.event :  5e-04
##    between.subject.var :  1

with.MAR.dropout <- SimulateDropout(complete,
                    drop.mechanism=drop.mec)
```

See later in the guide for creating custom dropout mechanisms.

Additional Imputing Mechanisms By altering the `trt.weight` argument, the `weighted_j2r` imputing mechanism can be used to generate different imputing mechanisms:

If `trt.weight = 0` then imputation using this mechanism will follow the jump to reference (j2r) model whereby missing counts for subjects in both arms will be imputed according to the mean of the placebo arm conditioned on the subject's observed number of events

If `trt.weight = 1` then imputation using this mechanism will follow the MAR model whereby missing counts for subjects in each arm will be imputed according to the event rate of subjects in its treatment group conditioned on the subject's observed number of events

Explicitly, when using the `weighted_j2r` function, with `trt.weight`$= \omega$, $\mu = (\mu_c, \mu_a)$, the (possibly subject-specific) expected means, and $\gamma = (\gamma_c, \gamma_a)$ are 1/dispersions, which are parameter estimates sampled with uncertainty from a model fit[1], for a subject with O_i observed events and 'imputation time' $t_i = T-$censor.time, the number of imputed events is negative binomial given by

- $NB(p = \frac{\mu_c t_i}{\gamma_c + \mu_c T}, \gamma = \gamma_c + O_i)$ for subjects in the control arm.
- $NB(p = (1-\omega)q_c + \omega q_a, \gamma = \gamma_a + O_i)$ for subjects in the active arm where $q_c = \frac{\mu_c t_i}{\gamma_a + \mu_a(T - t_i) + \mu_c t_i}$ and $q_a = \frac{\mu_a t_i}{\gamma_a + \mu_a T}$.

Finally if `trt.weight = 1` an additional argument `delta` can be included. It should be a vector of length 2, `c(control.delta,treatment.delta)` and in this case the mean number of expected events for the imputed missing data is multipled by the appropriate delta (and so p is adjusted appropriately):

```
  weighted_j2r(trt.weight=1,delta=c(1,1.4))
```

[1] using a normal approximation to sample the coefficeints and log(1/dispersion) which are asymptotically uncorrelated

```
## Imputation Method: weighted_j2r
## Parameters:
##    trt.weight :  1
##    delta :  1 1.4
```

5 Using covariates & creating new dropout/imputing mechanisms

In this Section we show how to perform multiple imputation using a data set with covariates. Here we use a set of covariates to simulate a “complete data set” of events. **Note** that it is possible to import a complete/dropout data set on which imputation can be perfomed, see `help(ImportSim)` for examples.

In order to use the `SimulateComplete` function with covariates, we require a data frame containing the covariates together with three additional columns: subject Id, treatment arm (0 for control or 1 for active) and a rate column (typically a function of the covariates/treatment arm). The rate column will replace the `event.rates` argument, so it is a subject specific μ/T.

Users have complete freedom over the number and type of covariates. In this example we use two covariates: `pre.exa`, the number of events in the previous year, and `steroid` a variable for whether the subject is using steroids (1 yes, 0 no).

We first generate some covariates for a 100 subject trial with equal randomization proportions:

```
covar.df <- data.frame(Id=1:100,
                       arm=c(rep(0,50),rep(1,50)),
                       pre.exa=rbinom(n=100,size=15,prob=0.4),
                       steroid=rbinom(n=100,size=1,prob=0.2))
```

We then define the subject specific event rate to be given by the following formula:

$$0.001 + 0.002\text{pre.exa} + 0.005(1 - \text{steroid}) + 0.008(1 - \text{arm})$$

```
covar.df$rate <- 0.001 + 0.002*covar.df$pre.exa +
                 0.005*(1-covar.df$steroid) + 0.008*(1-covar.df$arm)

head(covar.df)

##   Id arm pre.exa steroid  rate
## 1  1   0      10       0 0.034
## 2  2   0       3       0 0.020
## 3  3   0       5       0 0.024
## 4  4   0       7       0 0.028
## 5  5   0       8       0 0.030
## 6  6   0       9       0 0.032
```

Finally we can simulate a complete data set using the `dejaData` argument to the `SimulateComplete` function:

```
complete.covar <- SimulateComplete(study.time=365,
                      dispersion=0.25,
                      dejaData = MakeDejaData(covar.df,arm="arm",
                                              Id="Id",rate="rate"))

head(complete.covar$data)
```

```
##   Id arm censored.time observed.events actual.events pre.exa steroid
## 1  1   0           365              26            26      10       0
## 2  2   0           365               4             4       3       0
## 3  3   0           365              22            22       5       0
## 4  4   0           365               6             6       7       0
## 5  5   0           365               4             4       8       0
## 6  6   0           365              10            10       9       0
```

The `MakeDejaData` function requires 4 arguments: the data frame and the column names used for the treatment arm, the subject Id and the event rate. The value in the event rate column for each subject is used as the negative binomial event rate (μ/T) for this subject when simulating event times.

Implementing new drop out mechanisms (advanced!) - the idea is for a developer to add new mechanisms, though advanced R users can follow these instructions. It is possible to implement your own drop out mechanisms using the `CreateNewDropoutMechanism` function. A `DropoutMechanism` object should be created which contains 5 elements. We show a very simple example here and see `help(DropoutMechanism.object)` and `help(CreateNewDropoutMechanism)` for further details.

In our example we say subject dropouts are exponentially distributed with subject specific rate R where $R = (\texttt{rate})\exp(X_i)$ where $X_i \sim \mathcal{N}(0, \texttt{var})$ and there are different rates dependent on whether the subject is on steroids.

```
#we create a function which returns the new dropout mechanism
steroidMCAR <- function(steroid.rate, non.steroid.rate,var=0){

  #First we create a function which must take in two arguments,
  #event.times - a list of a single subject's event times
  #data - a row of the data frame containing the subject details
  #and outputs the time of subject dropout
  GetDropTime <- function(event.times,data){
    rate <- if(data$steroid==1) steroid.rate else non.steroid.rate
    rate <- rate*exp(rnorm(1,mean = 0,sd = sqrt(var)))
    dropout.time <- rexp(1,rate)
    return(min(dropout.time, data$censored.time))
  }

  #we create a vector of the columns from the data frame that
  #are used in the GetDropTime function
  cols.needed <- c("censored.time","steroid")

  #we call the CreateNewDropoutMechanism function
  #with the following arguments
  CreateNewDropoutMechanism(type="MNAR",
                            text="Rate dependent on steroid use", #the text to be output
                            cols.needed=cols.needed, #see above
                            GetDropTime=GetDropTime, #see above
                            parameters=list(steroid.rate=steroid.rate,
                                            non.steroid.rate=non.steroid.rate,
                                            var=var) #The parameters to be output
                            )

}
```

We can then use the `steroidMCAR` function:

```
#we can view the dropout mechanism
steroidMCAR(steroid.rate = 0.005, non.steroid.rate = 0.025)

## Dropout Type: MNAR
## Dropout Mechanism: Rate dependent on steroid use
## Parameters:
##    steroid.rate :  0.005
##    non.steroid.rate :  0.025
##    var :  0

#we can use it
dropout.covar <- SimulateDropout(complete.covar,
                    drop.mechanism=steroidMCAR(steroid.rate = 0.0025,
                                   non.steroid.rate = 0.001))

summary(dropout.covar)

## dropout dataset
## Study follow up period: 365
## Subjects (per arm): 50 50
## Subject dropouts (per arm): 20 18
## Number of events (per arm): 361 283
## Total Time at risk (per arm): 13985.88 14190.8
## Empirical event rates (per arm): 0.02581175 0.01994249
```

By using the `covar` argument with the `SimFit` function, covariates can be included in the model fit. In the code below we include the covariate `pre.exa` in the model fit (so the model becomes `observed.events` ∼ `arm + offset(log(censored.time))+ pre.exa`).

```
dropout.fit <- Simfit(dropout.covar,
                 equal.dispersion=TRUE,
                 covar=~pre.exa)

#The values of mu now depend on subject's pre.exa value
gamma_mu <- dropout.fit$genCoeff.function(use.uncertainty=TRUE)
head(gamma_mu$mu)

##             [,1]       [,2]
## [1,] 0.05599407 0.03905091
## [2,] 0.01562708 0.01089850
## [3,] 0.02250293 0.01569380
## [4,] 0.03240412 0.02259900
## [5,] 0.03888488 0.02711877
## [6,] 0.04666179 0.03254247
```

Implementing new imputing mechanisms (advanced!) - the idea is for a developer to add new mechanisms, though advanced R users could follow these instructions It is possible to implement your own imputing mechanisms using the `CreateNewImputeMechanism` function. A `ImputeMechanism` object should be created which contains 4 elements. We show a toy example here and see `help(ImputeMechanism.object)` and `help(CreateNewImputeMechanism)` for further details.

In our toy[2] example we would like subjects in the treatment group to have no imputed events and subjects in the control group to have either 0 or 1 events with a given probability which depends on their `steroid` covariate value. If subjects have an event it is midway between dropping out and the end of the follow up period.

```r
#we create a function which returns the new dropout mechanism
#arguments are parameters for probability control arm having event
my.example.impute <- function(steroid.prob,non.steroid.prob){


  #We need a function which takes in a SingleSimFit object
  #and returns a list with 2 elements:
  #newevent.times which contains vectors of the imputed event times for each subject
  #new.censored.times, a vector of the times the imputed data subjects dropout
  #(the code in this function has been designed for clarity as does NOT follow
  #R best practice)
  impute <- function(fit){

    #how many subjects are there in the data frame?
    number.of.subjects <- numberSubjects(fit)

    #subject follow up time
    study.time <- fit$singleSim$study.time

    #After imputing data, all subjects are followed up
    #for study.time
    new.censored.times <- rep(study.time,number.of.subjects)

    #The subject data
    data <- fit$singleSim$data

    #the imputed event times for each subject
    newevent.times <- list()

    #Note could access the mu, gamma taking into
    #account uncertainty in parameter estimates for the imputation
    gamma_mu <-  fit$genCoeff.function(use.uncertainty=TRUE)
    #so gamma_mu£mu[i,] is (mu_c,mu_a) for subject i

    #for each subject create a vector of imputed event times
    #if no events are imputed then use numeric(0)
    for(id in 1:number.of.subjects){

      #assume by default no events are imputed
      newevent.times[[id]] <- numeric(0)

      #time left on study
      time.left <- study.time - data[id,]$censored.time

      #if ti = 0 then subject didn't drop out so
```

[2]this is clearly not a sensible imputing method but shows how to implement imputing mechanisms

```
      #no imputed events and if in treatment group then no new events
      if(data[id,]$arm==1 || time.left==0) next;

      #get the probability of an event
      prob <- if(data[id,]$steroid==1)steroid.prob else non.steroid.prob

      #did subject have an event?
      if(rbinom(n=1,size = 1,prob=prob)!= 0){
        newevent.times[[id]] <- data[id,]$censored.time + 0.5*time.left
      }
    }

    #return the appropriate list
    return(list(new.censored.times=new.censored.times,
                newevent.times=newevent.times))

  }

  #we create a vector of the columns from the data frame that
  #are used in the impute function
  cols.needed <- c("censored.time","arm","steroid")

  CreateNewImputeMechanism(name="my new impute mechanism", #name for outputting
                  cols.needed=cols.needed, #see above
                  impute=impute, #see above
                  parameters=list(steroid.prob=steroid.prob,
                                  non.steroid.prob=non.steroid.prob)) #extra parameters
}

#we can view the impute mechanism
my.example.impute(steroid.prob=0.5,non.steroid.prob = 0.9)

## Imputation Method: my new impute mechanism
## Parameters:
##    steroid.prob :  0.5
##    non.steroid.prob :  0.9
```

For a more complex example (aimed for developers) see the functions: `weighted_j2r` and `dejaVu:::.internal.impute`. Using the imputation mechanism, MI can be performed and models (with covariates) fitted to the data as before:

```
imputed.covar <- Impute(fit = dropout.fit,
                  impute.mechanism = my.example.impute(steroid.prob=0.5,
                                       non.steroid.prob = 0.9),
                  N=10)

fitted.covar <- Simfit(imputed.covar,
                 family="negbin",
                 covar=~pre.exa)

summary(fitted.covar)
```

```
## Summary for imputed data sets
## Treatment Effect: 0.7058284
## SE (log) treatment effect: 0.1657061
## Degrees of freedom: 13612974
## p-value: 0.03551686
## Adjusted d.o.f: 94.98204
## Adjusted p-value: 0.03816193
## Average dispersion: 0.5036714
## Number of subjects dropped out per arm: 20 18
```

References

[1] Donald B Rubin. Multiple imputation for nonresponse in surveys (wiley series in probability and statistics). 1987.