MCPModGeneral Vignette

Ian Laga

22 August, 2019

1 Introduction

`MCPModGeneral` is an extension of the `DoseFinding` package, streamlining the analysis of non-normal endpoints. Almost all of the functions in the `DoseFinding` package rely on the user supplying μ, the estimated dose-response coefficients, and S, the variance-covariance matrix of μ. However, S is difficult to know beforehand, and for functions like `powMCT`, differ for each alternative model. The `MCPModGeneral` package does not require the user to supply a matrix for S and instead calculates the theoretical variance-covariance matrix for the negative binomial, binomial, and Poisson distributions. Alternatively, the emperical covariance matrices can be estimated via simulation.

Users can also use the `MCPModGeneral` package to fit the full MCPMod procedure on negative binomial, Poisson, and binomial data, as well as basic survival data. The relevant functions for fitting the models are `prepareGen` and `MCPModGen`. The full capabilities of these functions will be explored later in the vignette.

As with the `DoseFinding` package, the `MCPModGeneral` package still requires the user to create and specify the potential dose-response curves. The `DoseFinding` package provides the `guesst` and `Mods` functions to create these models, but ultimately, the dose-response curves come from prior knowledge or discussion with clinicians. For the entirety of this vignette, it is assumed that the user is able to construct the potential models. Recall that dose-response curves must be constructed on the same scale as the ANOVA output, meaning if negative binomial data is to be analyzed via a GLM with a log-link, the dose-response curve should represent the log of the means at each dose. The `MCPModGeneral` package allows usage with the most commonly used links. See the `family` page from the `stats` package for a list of the common links.

2 Exploratory Stage

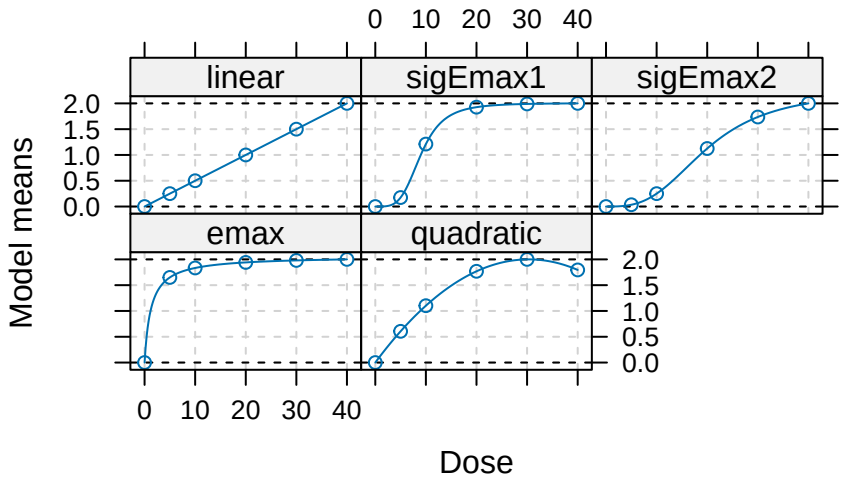
Before collecting data, it is useful to understand the properties of the potential data set. The two relevant functions in the `MCPModGeneral` package are `powMCTGen` and `sampSizeMCTGen`. `powMCTGen` will calculate the power of the multiple contrast test for the patient allocation and dose-response curves provided. These are extensions of the `powMCT` and `sampSizeMCT` packages from the `DoseFinding` package. While the `DoseFinding` functions can handle non-normal data, they require a user-provided covariance matrix. We find this unneccesarily prohibative in common use since the covariance function is unlikely to be known in advance.

Furthermore, only a single covariance matrix may be supplied, while the covariance matrix for non-normal data typically depends on the expected value at the doses. The `MCPModGeneral` functions work by calculating the theoretical covariance matrices from the means of the supplied models and then calling the `DoseFinding` functions. All theoretical covariance matrices were derived by hand for the most common links.

2.1 Power Calculation

The original `powMCT` function calculates the power for a multiple contrast test under each alternative model. As discussed above, the `powMCTGen` function works by calculating the theoretical covariance matrix at each of the alternative models and then calling `powMCTGen`. We added two new capabilities to the `powMCTGen` function. First, the user can change the doses in the function instead of having to redefine a new `Mods` object with the new doses. Second, the user can supply theoretical responses. Note that only one power calculation will be provided in this case, since the responses are no longer assumed to come from each of the models. Now, let us create the set of candidate models we will use for the following examples. We include 5 candidate models in our study, shown below.

```
dose.vec = c(0, 5, 10, 20, 30, 40)
models.full = Mods(doses = dose.vec, linear = NULL,
                   sigEmax = rbind(c(9, 4), c(20, 3)),
                   emax = 1.25, quadratic = -0.044/2.667,
                   placEff = 0, maxEff = 2)
plot(models.full)
```



The above plot shows the potential dose-response curves on the link-scale (e.g. on the log-scale for negative binomial data). The doses considered are $(0, 5, 10, 20, 30, 40)$. Now consider a trial where 30 patients receive each dose. We wish to calculate the probability of accepting at least one alternative model given the true underlying dose-response curve. We first assume that the endpoints are negative binomial counts and the dispersion parameter is 0.1. The following chunks of code demonstrate three ways to calculate these powers. The "verbose" parameter specifies whether the assumed patient allocation should be printed. This is useful to ensure that the patient allocation is what was intended.

We demonstrate the use of `powMCTGen` below.

Using the "arm" Ntype

```
## Look at the power for each possible DR-curve
powMCTGen(30, "negative binomial", "log", modelPar = 0.1,
          Ntype = "arm", alpha = 0.05, altModels = models.full, verbose = T)
#> [1] "the patient allocation is given by:"
#>   Dose nPatients
#> 1    0        30
#> 2    5        30
#> 3   10        30
#> 4   20        30
#> 5   30        30
#> 6   40        30
#>    linear  sigEmax1  sigEmax2      emax quadratic
#> 0.8649376 0.9524492 0.9348414 0.8475181 0.8860520
```

Using the "total" Ntype

```
powMCTGen(180, "negative binomial", "log", modelPar = 0.1,
          Ntype = "total", alpha = 0.05, altModels = models.full, verbose = T)
```

Using the "actual" Ntype

```
powMCTGen(c(30,30,30,30,30,30), "negative binomial", "log", modelPar = 0.1,
          Ntype = "actual", alpha = 0.05, altModels = models.full, verbose = T)
```

Now let's assume our dose-response models define the probability of occurance on the probit scale. We need only to change the `family` and `link` arguments, and remove the `modelPar` argument. Note that the `modelPar` is simply ignored for `family = "binomial"`.

```
powMCTGen(30, "binomial", "probit",
          Ntype = "arm", alpha = 0.05, altModels = models.full)
```

By default, the doses are assumed to be the same doses used to construct the alternative models. However, the user now can also supply their own doses to see what effect the choice of doses has on the power. Consider the next example, where the doses are either very small or very large, which no doses in the middle of the dose-range. We see that the power decreases for some models, especially "sigEmax2". This is because certain choices of doses do a better job of defining the underlying shape of the curves.

```
powMCTGen(30, "binomial", "probit",
          Ntype = "arm", alpha = 0.05, doses = c(0, 1, 2, 36, 38, 40),
          altModels = models.full, verbose = TRUE)
#> [1] "the patient allocation is given by:"
#>   Dose nPatients
#> 1    0        30
#> 2    1        30
#> 3    2        30
#> 4   36        30
```

```
#> 5   38        30
#> 6   40        30
#>    linear  sigEmax1  sigEmax2      emax quadratic
#> 0.9999939 0.9999951 0.9999961 0.9999171 0.9999864
```

Another nice extension to the `DoseFinding` package is the power function can now handle theoretical response values. Instead of supplying a dose-response curve, the user can instead supply theoretical mean values (on the link scale) at respective doses. These doses do not have to be the same as the doses used to construct the alternative models.

```
## Now consider power at some theoretical DR-values
powMCTGen(30, "negative binomial", "log", modelPar = 0.1,
          theoResp = c(0, 0.2, 1.8), doses = c(0, 20, 40),
          alpha = 0.05, altModels = models.full)
#> [1] 0.6434992
```

This can also be used to test the type-1 error, if needed, although this should always be α.

```
## Can also check type-1 error
powMCTGen(30, "negative binomial", "log", modelPar = 0.01, theoResp = rep(0, 5),
          doses = c(0, 50, 10, 20, 30),
          alpha = 0.05, altModels = models.full)
#> [1] 0.05015167
```

2.2 Sample Size Calculations

One could keep adjusting the patient allocation in `powMCTGen` in order to reach some target power, but `sampSizeMCTGen` offers an automated process. In addition to the family, link, parameters, and models, the user also needs to supply a "guess" of the largest sample size needed to reach a certain power, and the desired power. The following line of code calcualted the optimal sample size for binomial data with more patients allocated in the ratios $3:1:2:2:2:2$ for the respective doses, in order to reach a minimum power of 0.8.

```
sampSizeMCTGen("binomial", "logit", upperN = 50, Ntype = "arm",
               altModels = models.full, alpha = 0.05, alRatio = c(3/2, 1/2, 1, 1, 1, 1),
               sumFct = "min", power = 0.8)
#> Sample size calculation
#>
#> alRatio: 3 1 2 2 2 2
#> Total sample size: 84
#> Sample size per arm: 21 7 14 14 14 14
#> targFunc: 0.8126
```

If `verbose = TRUE`, current iteration, N, and power is printed at each step. Let's pay more attention to the `sumFct` argument. Recall that `powMCTGen` returns a vector of powers, one for each alternative model. `sumFct` converts this vector into a single value. The default is to consider the minimum of the powers, so that in the worst case (the true model is the model which has the lowest power), the power is still above the target. Like `sampSizeMCT`, default functions are "min", "mean", and "max". However, `sumFct` can also handle user-supplied functions, as long as the function accepts a patient allocation argument of the `Ntype` specified, and returns a numeric value.

```
sampSizeMCTGen("negative binomial", "log", modelPar = 0.1, upperN = 50, Ntype = "arm",
               altModels = models.full, alpha = 0.05,
               sumFct = "max", power = 0.8, verbose = T)
#> Iter: 1, N = 31, current value = 0.957
#> Iter: 2, N = 22, current value = 0.8808
#> Iter: 3, N = 17, current value = 0.7964
#> Iter: 4, N = 20, current value = 0.8518
#> Iter: 5, N = 18, current value = 0.8178
#> Sample size calculation
#>
#> alRatio: 1 1 1 1 1 1
#> Total sample size: 108
#> Sample size per arm: 18 18 18 18 18 18
#> targFunc: 0.8178
```

Just like `powMCTGen`, `sampSizeMCTGen` can also handle theoretical responses at supplied doses.

```
sampSizeMCTGen("negative binomial", "log", modelPar = 0.1, upperN = 100, Ntype = "total",
               alRatio = c(3/2, 1/2, 1),
               theoResp = c(0, 0.2, 1.8), doses = c(0, 20, 40),
               altModels = models.full, alpha = 0.05)
#> Sample size calculation
#>
#> alRatio: 3 1 2
#> Total sample size: 114
#> Sample size per arm: 57 19 38
#> targFunc: 0.8017
```

3 Data Analysis Stage

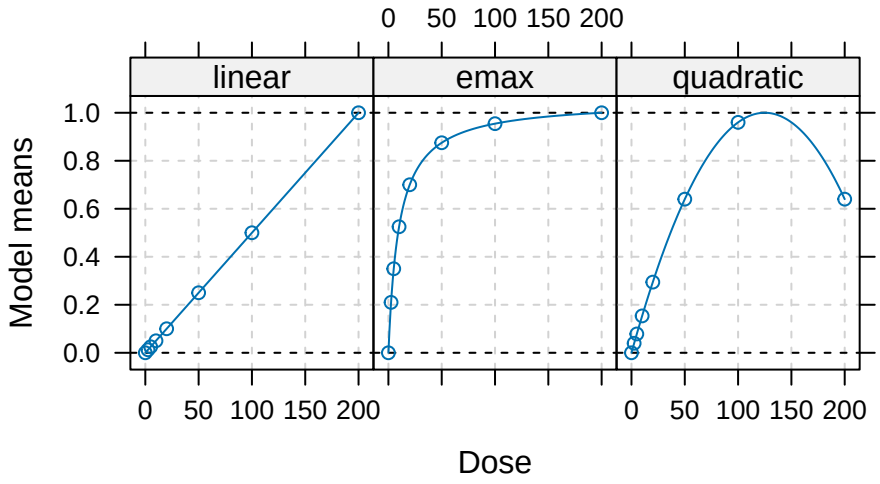
In this section, we will explore the two methods for analyzing non-normal data using the `MCPModGeneral` package. The first method uses the `prepareGen` function to retrieve the $\hat{\mu}$ vector and $\hat{S}$ matrix, and then passes these objects into `DoseFinding` functions. The second method combines the two steps into one via the `MCPModGen` function.

3.1 Binomial Data

First, consider the data. This data set is provided in the `DoseFinding` package, and the primary endpoint was the binary value "pain freedom at 2 hours postdose." We also construct three candidate models for demonstration. These models were not based off any prior knowledge, and serve only as demonstration of the approach.

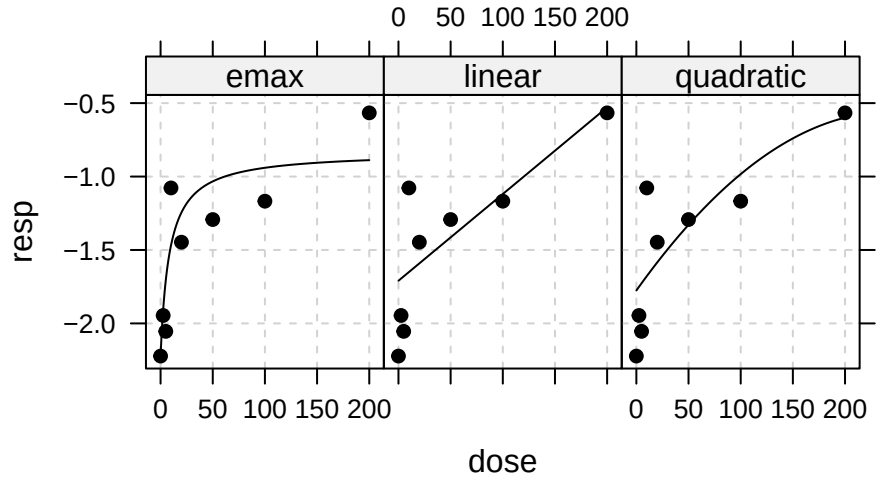
```
data(migraine)
migraine$pfrat = migraine$painfree / migraine$ntrt
migraine
#>    dose painfree ntrt      pfrat
#> 1   0.0       13  133 0.09774436
#> 2   2.5        4   32 0.12500000
#> 3   5.0        5   44 0.11363636
#> 4  10.0       16   63 0.25396825
```

```
#> 5  20.0       12   63 0.19047619
#> 6  50.0       14   65 0.21538462
#> 7 100.0       14   59 0.23728814
#> 8 200.0       21   58 0.36206897

models = Mods(linear = NULL, emax = 10, quadratic = c(-0.004), doses = migraine$dose)
plot(models)
```



We can prepare the data manually using the `prepareGen` function and then forward the output to the `MCPMod` function from the `DoseFinding` package. This two step process is shown below.

```
mu.S = prepareGen(family = "binomial", link = "logit", w = "ntrt", dose = "dose",
                  resp = "pfrat", data = migraine)
mcp.hand = MCPMod(dose = mu.S$data$dose, resp = mu.S$data$resp, models = models,
                  S = mu.S$S, Delta = 0.2, type = "general")
plot(mcp.hand)
```



```
mcp.hand
#> MCPMod
#>
#> Multiple Contrast Test:
#>           t-Stat   adj-p
#> emax       4.061 < 0.001
#> linear     3.703 < 0.001
#> quadratic  3.079 0.00209
#>
#> Estimated Dose Response Models:
#> linear model
#>     e0  delta
#> -1.710  0.006
#>
#> emax model
#>     e0   eMax   ed50
#> -2.219  1.387  8.473
#>
#> quadratic model
#>     e0     b1     b2
#> -1.776  0.010  0.000
#>
#> Selected model (AIC): emax
#>
#> Estimated TD, Delta=0.2
#>    linear      emax quadratic
#>   33.8758    1.4274   20.9810
```

Alternatively, we can perform the same calculations, but in one step via the `MCPModGen` function.

```
mcp.gen = MCPModGen("binomial", "logit", returnS = F, w = "ntrt", dose = "dose",
            resp = "pfrat", data = migraine, models = models, Delta = 0.2)
mcp.gen
#> MCPMod
#>
#> Multiple Contrast Test:
#>           t-Stat   adj-p
#> emax       4.061 < 0.001
#> linear     3.703 < 0.001
#> quadratic  3.079 0.00216
#>
#> Estimated Dose Response Models:
#> linear model
#>     e0  delta
#> -1.710  0.006
#>
#> emax model
#>     e0   eMax   ed50
#> -2.219  1.387  8.473
#>
#> quadratic model
#>     e0     b1     b2
#> -1.776  0.010  0.000
```

```
#>
#> Selected model (AIC): emax
#>
#> Estimated TD, Delta=0.2
#>    linear      emax quadratic
#>   33.8758    1.4274   20.9810
```

Notice that the results for the two methods are almost identical. Slightly different values are given (e.g. the adj-p values), but this is due to the computation method.

3.2 Negative Binomial Data

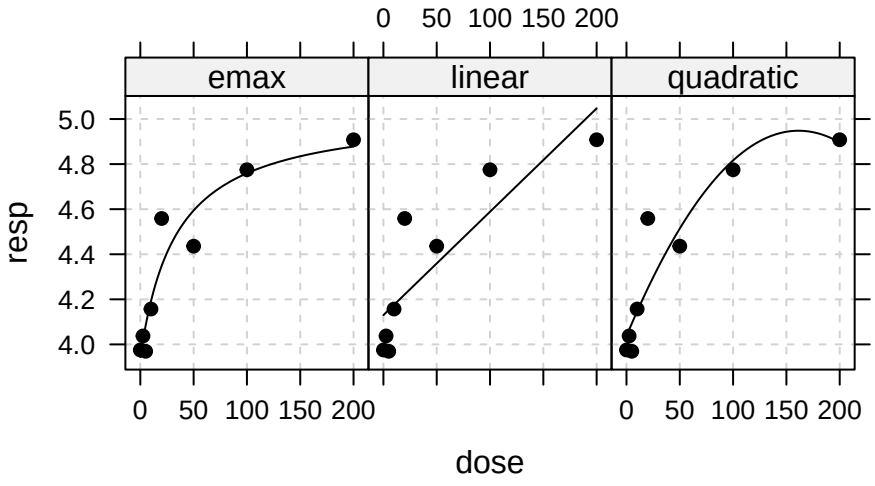
The `DoseFinding` package does not have any negative binomial data, so we will simulate data according to the same model curves we used for the binomial data. Five random observations are shown from the data set to demonstrate the structure of the data. To demonstrate the affect that additional covariates has on the model, we will use extremely large sample sizes for each arm (300 patients per arm) and a very large gender effect.

```
## Simulate some negative binomial data according to one of the models
set.seed(188)
mean.vec = getResp(models)[,2]
dose.dat = c()
resp.dat = c()
gender.dat = c()
for(i in 1:length(migraine$dose)){
  gender.tmp = rbinom(300, 1, prob = 0.3)
  gender.dat = c(gender.dat, gender.tmp)
  dose.dat = c(dose.dat, rep(migraine$dose[i], 300))
  resp.dat = c(resp.dat, rnbinom(300, mu = exp(mean.vec[i] + 5*gender.tmp), size = 1))
}
nb.dat = data.frame(dose = dose.dat, resp = resp.dat, gender = gender.dat)
nb.dat[sample(1:nrow(nb.dat), 5),]
#>       dose resp gender
#> 876    5.0   24      1
#> 398    2.5    1      0
#> 2281 200.0    3      0
#> 1780  50.0    9      0
#> 2126 200.0  453      1
```

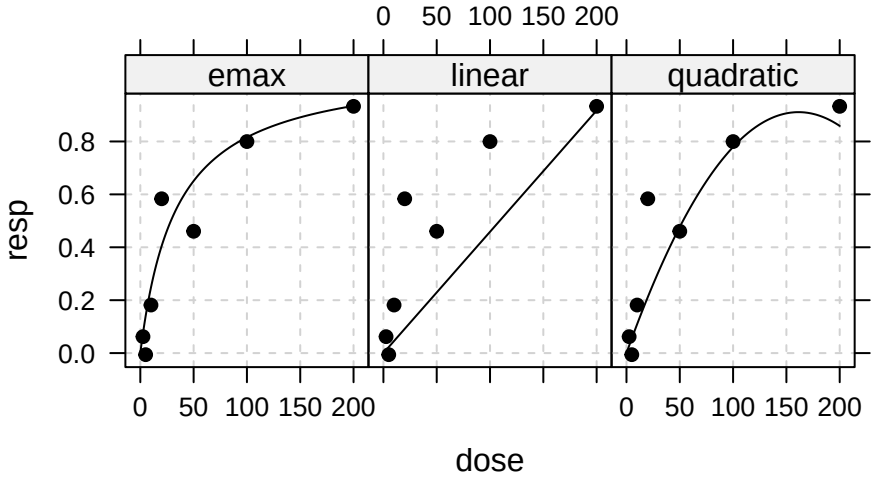
Now we will perform the MCP-Mod procedure on the negative binomial data, both using the default approach, and also on the placebo adjusted data. Note that we also show the different ways to supply the `dose` and `resp` arguments, which differ slightly from the `DoseFinding` package which does not use character vectors. We show only the results for two of the objects (one default, one placebo adjusted). Note that if `returnS` is TRUE, the returned object will have three arguments. One with the MCPMod object, one with $\hat{\mu}$ and the doses, and one with $\hat{S}$.

```
mcp.nb1 = MCPModGen("negative binomial", link = "log", returnS = T,
          dose = "dose", resp = "resp", data = nb.dat, models = models, Delta = 0.6)

mcp.nb2 = MCPModGen("negative binomial", link = "log", returnS = T,
          dose = dose.dat, resp = resp.dat, models = models, Delta = 0.6)
```

```r
mcp.nb3 = MCPModGen("negative binomial", link = "log", returnS = T, placAdj = T,
          dose = "dose", resp = "resp", data = nb.dat, models = models, Delta = 0.6)

mcp.nb4 = MCPModGen("negative binomial", link = "log", returnS = T, placAdj = T,
          dose = dose.dat, resp = resp.dat, models = models, Delta = 0.6)

names(mcp.nb1)
#> [1] "MCPMod" "data"   "S"
mcp.nb1$data
#>    dose     resp
#> 1   0.0 3.975311
#> 2   2.5 4.037421
#> 3   5.0 3.969348
#> 4  10.0 4.156954
#> 5  20.0 4.558707
#> 6  50.0 4.435765
#> 7 100.0 4.774885
#> 8 200.0 4.908036


mcp.nb1$MCPMod$doseEst
#>    linear      emax quadratic
#> 130.84452  41.37549  67.10633
#> attr(,"addPar")
#> [1] 0.6
mcp.nb4$MCPMod$doseEst
#>    linear      emax quadratic
#> 130.84452  41.37549  67.10633
#> attr(,"addPar")
#> [1] 0.6

plot(mcp.nb1$MCPMod)
```



```r
plot(mcp.nb4$MCPMod)
```

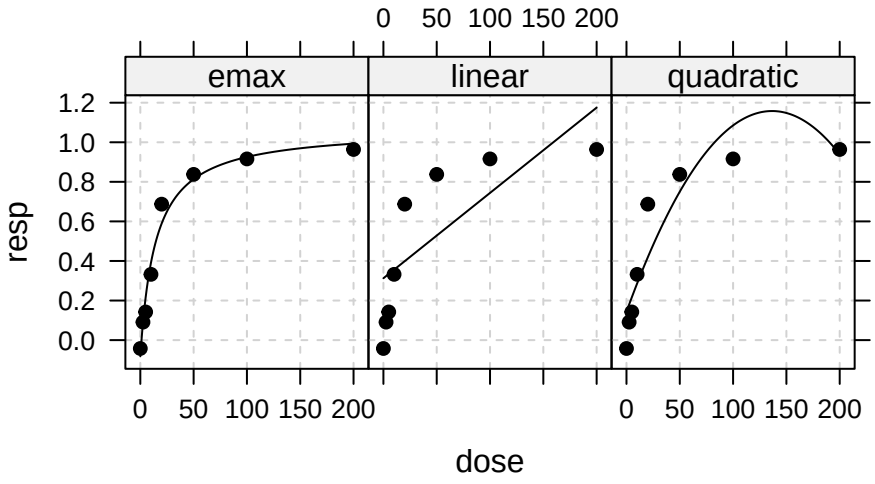


Additional Covariates

The `MCPModGeneral` function can also incorporate additional covariates. Consider the same constructed negative binomial data from above. Notice how we removed the effect of `gender` on the results. We will now easily consider the `gender` of the patients using the `addCovars` parameter. Notice how `gender` is a categorical variable, so when we provide the formula in the `addCovars` argument, we need to specify that we want to treat it as a factor. If `gender` is already a factor in the data frame, then the model will still treat it as a factor.

```r
mcp.covars = MCPModGen("negative binomial", link = "log", returnS = F, addCovars = ~ factor(gender),
                       dose = "dose", resp = "resp", data = nb.dat, models = models, Delta = 0.6)

mcp.covars
#> MCPMod
#>
#> Multiple Contrast Test:
#>            t-Stat  adj-p
#> emax       15.509 <0.001
#> quadratic 14.384 <0.001
#> linear     12.122 <0.001
#>
#> Estimated Dose Response Models:
#> linear model
#>    e0 delta
#> 0.313 0.004
#>
#> emax model
#>     e0   eMax   ed50
#> -0.082  1.152 14.330
#>
#> quadratic model
```

```
#>    e0    b1    b2
#> 0.143 0.015 0.000
#>
#> Selected model (AIC): emax
#>
#> Estimated TD, Delta=0.6
#>    linear      emax quadratic
#>  139.1470   15.5723   49.2665
plot(mcp.covars)
```



Let's now see the target dose estimates for our models against the true target dose.

```
TD(models, Delta = 0.6)[2]
#>     emax
#> 13.33333

mcp.nb1$MCPMod$doseEst[mcp.nb1$MCPMod$selMod]
#>     emax
#> 41.37549
mcp.covars$doseEst[mcp.covars$selMod]
#>     emax
#> 15.57235
```

We can clearly see that including the covariates greatly improves the accuracy of the TD estimate.

Relative Risk

The relative risk measures either the risk ratio for binomial data or the rate ratio for count data. The main advantage of using the relative risk is that interpretation is incredibly easy and consistent under different data distributions. While the canonical link for negative binomial data is the log-link, it is often difficult to propose candidate models in terms of log-mean. Other links are even more confusing.

For binary data, the relative risk is better known as the risk ratio, where the relative risk for dose a against the placebo p is given by

$$RR_a = \frac{\left(\sum_{i=1}^{n_a} I(outcome)_{a,i}\right)/n_a}{\left(\sum_{i=1}^{n_p} I(outcome)_{p,i}\right)/n_p} \tag{1}$$

where n_a denotes the number of patients who received dose a.

For count data (e.g. negative binomial and Poisson), the relative risk is better known as the rate ratio, where the relative risk for dose a against the placebo p is given by

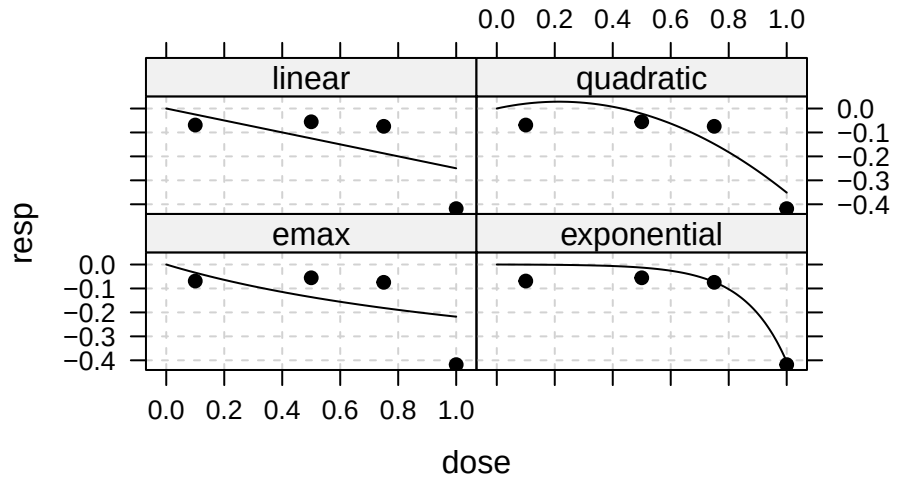
$$RR_a = \frac{\left(\sum_{i=1}^{n_a} \#events_{a,i}\right) / \left(\sum_{i=1}^{n_a} time_{a,i}\right)}{\left(\sum_{i=1}^{n_p} \#events_{p,i}\right) / \left(\sum_{i=1}^{n_p} time_{p,i}\right)} \tag{2}$$

where $\#events_{a,i}$ denotes the number events that occurred for patient i of dose a in the recording time $time_{a,i}$.

Users can specify candidate models on the relative risk scale and apply the traditional MCP-Mod procedure via the `MCPModGeneral` package. For power analysis, the user must also supply the mean at the placebo, denoted `placEff`. An example of data generated from a candidate relative risk dose-response curve and analysis of the data is shown in the following chunk of code.

```
set.seed(1786)
doses = c(0, 0.1, 0.5, 0.75, 1)
n.vec = c(30, 20, 23, 19, 32)
n.doses = length(doses)
models = Mods(doses = doses, linear = NULL, emax = 0.1, exponential = 0.2,
              quadratic = -0.75, placEff = 1, maxEff = -0.3)

## Perform power-analysis
powMCTGen(n.vec, "binomial", "risk ratio", altModels = models, placEff = 0.9,
          Ntype = "actual")
#> using 'placAdj' specification from contMat object
#> using 'placAdj' specification from contMat object
#> using 'placAdj' specification from contMat object
#> using 'placAdj' specification from contMat object
#>      linear        emax exponential   quadratic
#>   0.8987091   0.9597508   0.8011131   0.9106507




## Simulate the data according to the exponential curve
means = getResp(models)[,3]*0.9
cbind(Dose = doses, Means = means)
#>      Dose     Means
#> 0    0.00 0.9000000
#> 0.1  0.10 0.8988118
#> 0.5  0.50 0.8795183
#> 0.75 0.75 0.8239505
#> 1    1.00 0.6300000

resp.dat = c()
for(i in 1:n.doses){
  resp.dat = c(resp.dat, rbinom(1, size = n.vec[i], prob = means[i]))
}
```

```r
bin.dat = data.frame(dose = doses, resp = resp.dat/n.vec, w = n.vec)



## Fit using the package

mod.pack = MCPModGen("binomial", "risk ratio", returnS = F, w = "w", dose = "dose", resp = "resp",
                     data = bin.dat, models = models, Delta = 0.1)
#> Forcing 'placAdj = TRUE' for risk ratio links

plot(mod.pack)
```




```r
## Look at the MED estimate
mod.pack$doseEst[mod.pack$selMod]
#> exponential
#>   0.7955145
TD(models, Delta = 0.1, direction = "decreasing")[3]
#> exponential
#>   0.7829547
```

Conclusion

The `MCPModGeneral` package allows users to use the methods from the `DoseFinding` package without having to code additional data analysis. While fitting the actual MCP-Mod procedure for non-normal data is often very straightforward, full power-analysis requires extensive coding, which the `MCPModGeneral` package does for the user. The other significant contribution to the `DoseFinding` package is the inclusion of the relative risk link. Together, the `MCPModGeneral` package and `DoseFinding` package can apply the MCP-Mod procedure to the most common data distributions.