METRUM RESEARCH GROUP

Simulating Clinical Trials in Alzheimer's Disease

Daniel G. Polhamus, Ph.D., James A. Rogers, Ph.D.

Contents

Chapter 0

Getting Started

0.1 System Requirements

The R version and the versions of all packages used in this manual are described in Table 1. If you are unable to produce the same output using more recent versions of any of these components, please contact me at danp@metrumrg.com and I will see what I can do.

If you install the following three packages: **contrast**, **reshape**, and **adsim**, all of the other required packages will get installed automatically. The packages **contrast** and **reshape** are maintained on CRAN and may therefore be installed from R with the menu selections "Packages" → "Install package(s)". The package "adsim" has been provided as a zip file with the training material. On R for Windows, you can install this with the menu selections "Packages" → "Install package(s) from local zip file".

Once the packages have been installed, you can load all of them in an R session with the single command:

Listing 1:

```
library(adsim)
```

Although the operating system is unlikely to matter, it doesn't hurt to record this as well. The code in this manual was executed under the followng OS:

Listing 2:

```
Sys.info()['version']
```

```
"Darwin Kernel Version 10.8.0: Tue Jun  7 16:33:36 PDT 2011; root:
   xnu-1504.15.3~1/RELEASE_I386"
```

Table 1: Versions of base R and supporting packages used to generate this manual

pkgVersions	Package	Version
1	adsim	2.1
2	Hmisc	3.8-3
3	survival	2.36-5
4	splines	2.12.2
5	stats	2.12.2
6	graphics	2.12.2
7	grDevices	2.12.2
8	utils	2.12.2
9	datasets	2.12.2
10	methods	2.12.2
11	base	2.12.2

0.2 Other Supporting Materials

Most of the R code in this book has been extracted into pure R scripts. Depending on your preference, you can either copy directly from this book and paste to the R prompt, or you can work from the scripts. In any case, it is the same R code in both places.

Chapter 1

Basic Concepts

1.1 Random Number Generators and Reproducibility

By default, R sets the initial states of its random number generators (RNGs) based on the clock time when an RNG is first invoked in a session. For the sake of reproducibility, we do not rely on this default behavior and instead explicity set the initial state of our random number generators (RNGs) in our scripts.

Listing 1.1:

```
set.seed(1) # or set.seed(314159), or whatever ...
```

The situation is more complicated if the simulations involve parallelization. In this course we are assuming that all iterations of a given simulation are run serially (presumably on a single computer), so that the random number generators advance in predictable ways.

1.2 Monte Carlo Error

Keeping track of the RNG state is just fine for reproducibility, but we would also like to be able to run the same simulation with two different initial RNG states and get answers that are "pretty close", i.e. we would like the Monte-Carlo error to be small. If the thing we are estimating by simulation is a probability (e.g. the power of a given trial design under a particular alternative), then each iteration of the simulation may be viewed as an independent binomial trial, so the standard error of the resulting estimate is approximately $\sqrt{\hat{p} \times (1-\hat{p})/B}$ where $\hat{p}$ is our simulation-based estimate of the probability and B is the number of simulations.

Exercise: Suppose you are running a simulation to estimate the power of a given trial design under a particular alternative. You think the true power is probably close to 80%, and

you want your simulation-based estimate to be accurate to within 1%. How many iterations should your simulation include?

1.3 Conditioning and Marginalizing

Conventional power calculations computations can be expressed as conditional ("if-then") probability statements. For example, consider the power of a simple parallel group design with 64 patients per group, assuming the primary analysis is just a two sample T-test, under the alternative hypothesis that the effect size is 0.5. We could compute this analytically, but the example is more instructive if we pretend we don't know about the analytical solution and instead use simulation to get the answer:

Listing 1.2:

```
oneTrial <- function(nPerGroup = 64) {
  true.means <- c(0, 0.5)
  true.sd <- 1
  dat <- data.frame(Treatment = rep(c("Pbo", "CP"), each =
     nPerGroup),
                    Response = rnorm(2 * nPerGroup,
                      mean = rep(true.means, each = nPerGroup),
                      sd = true.sd)
                    )
  pval <- t.test(Response ~ Treatment, data = dat)$p.value
  return(pval)
  }
nSim <- 10^3
pvals <- replicate(nSim, oneTrial())
sum(pvals < 0.05) / nSim
```

```
[1] 0.803
```

So, the power under that alternative is about 80%. To get this estimate, we averaged over the likely values of future data, but we kept our true parameter values fixed. We can express this as a conditional probability statement: "The probability of rejecting the null hypothesis, *conditional* on the true difference in means being 0.5 and the (common) standard deviation being 1, is 80%".

Because there is always uncertainty with respect to the true difference in means and with respect to the standard deviation, traditional power calculations are almost always presented as *multiple* "if-then" scenarios:

- **if** the true difference in means is 0.5 and the standard deviation is 1, **then** the probability of rejecting the null hypothesis is 80%.

- **if** the true difference in means is 0.4 and the standard deviation is 1, **then** the probability of rejecting the null hypothesis is 61%.
- **if** the true difference in means is 0.5 and the standard deviation is 1.1, **then** the probability of rejecting the null hypothesis is 45%.
- etc.

Excercise: Verify the last two bullet points by simulation.

Another approach is to *marginalize* (take the average) over all the likely true values of the unknowns. To do this we need to express our beliefs about the likely true values as probability distributions:

Listing 1.3:

```
oneTrial2 <- function(nPerGroup = 64) {
  true.means <- c(0, rnorm(1, 0.5, 0.05) )
  true.sd <- rgamma(1, 1, 1)
  dat <- data.frame(Treatment = rep(c("Pbo", "CP"), each =
     nPerGroup),
                    Response = rnorm(2 * nPerGroup,
                      mean = rep(true.means, each = nPerGroup),
                      sd = true.sd)
                    )
  pval <- t.test(Response ~ Treatment, data = dat)$p.value
  return(pval)
  }
nSim <- 10^4
pvals <- replicate(nSim, oneTrial2())
sum(pvals < 0.05) / nSim
```

```
[1] 0.7671
```

This marginal version of power is sometimes called "predictive power". It represents an average over both the future data *and* our beliefs about the true values of the underlying population parameters.

Either approach, if used with care, is amenable to providing an honest assessment of the probability of rejecting the null hypothesis, so it is reasonable to let the choice be dictated by the preferences of the eventual audience. An advantage of the "if-then" approach is that it exposes more of the "black box", making it easier for the consumers of the analysis to understand the sensitivity of the conclusions to parameters that they may have strong opinions about. On the other hand, the predictive approach has the advantage of providing a concise and comprehensive summary, and avoids swamping the audience with more "if-then" scenarios than they can digest.

An appealing hybrid approach is to marginalize with respect to the parameters for which there is likely to be substantial agreement (e.g. variance parameters and other ancillary parameters that can be estimated with historical data), and present "if-then" scenarios for the more contentious parameters (e.g. the mean difference between treatment and placebo, for which there is sometimes no historical data).

In any event, as we run clinical trial simulations and interpret the results, it will be important to keep straight what we have conditioned on and what we have averaged over.

Exercise: What happens to the predictive power in the last example if we quadruple our uncertainty about the true difference in means?

1.4 Probability Distributions

Simulation from our model will involve sampling from Normal, Gamma, and Beta distributions. We briefly review the latter two distributions for those who may be less familiar with them.

1.4.1 The Beta Distribution

The Beta distribution is a two-parameter distribution with support on the interval $[0,1]$. It is usually parameterized (and is parameterized in the R function `rbeta()`) in terms of α (`shape1` in R) and β (`shape2` in R) in such a way that the expectation and variance are:

$$\mathrm{E}[X] = \frac{\alpha}{\alpha+\beta} \ ; \ \mathrm{Var}[X] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}.$$

To make the distribution easier to work with for modeling purposes, we reparameterize as follows:

$$(\alpha,\beta) \rightarrow \left(\frac{\alpha}{\alpha+\beta}, \alpha+\beta\right) \equiv (\theta,\tau).$$

Our parameterization has the advantage that θ is the expected value of the distribution:

$$\mathrm{E}[X] = \theta \ ; \ \mathrm{Var}[X] = \frac{\theta(1-\theta)}{\tau+1}.$$

Note we can express the original α and β in terms of our new parameters as:

$$\alpha = \theta\tau \ ; \ \beta = (1-\theta)\tau.$$

Therefore, if we want to sample from a Beta distribution with mean 0.2 and standard deviation 0.02 we would do:

Listing 1.4:

```
my.mean <- 0.2
my.sd <- 0.02
theta <- my.mean
tau <- theta * (1-theta) / my.sd^2
rbeta(10, theta * tau, (1-theta) * tau)
```

```
[1] 0.2279981 0.2331807 0.2062680 0.1819488 0.2345863
[6] 0.2121873 0.1999433 0.2113867 0.2152330 0.1896086
```

Exercise: Generate a sample of size 1000 from a Beta distribution with mean = 0.35 and standard deviation = 0.05. Verify that the sample has the desired properties and make a histogram of it. Try the same thing again but changing the mean to 0.9 and the standard deviation to 0.1.

1.4.2 The Gamma Distribution

The Gamma distribution is a two parameter generalization of the Chi-squared distribution, with support on the positive Real line. It is often parameterized (and is parameterized in the R function `rgamma()`) in terms of α (`shape` in R) and β (`rate` in R) such that the expectation and variance are:

$$\mathrm{E}[X] = \frac{\alpha}{\beta} \ ; \ \mathrm{Var}[X] = \frac{\alpha}{\beta^2}.$$

We again reparameterize in a way that will make the distribution easier to work with in the context of our model:

$$(\alpha, \beta) \rightarrow (\alpha, \beta/\alpha) \equiv (\kappa, \phi^2).$$

With this parameterization, the mean of the distribution is $1/\phi^2$ (it's not obvious here, but this turns out to be a convenient property), and the coefficient of variation for the distribution is $1/\sqrt{\kappa}$.

Note we can express the original α and β in terms of our new parameters as:

$$\alpha = \kappa \; ; \; \beta = \kappa\phi^2.$$

Therefore, if we want to sample from a Gamma distribution with a mean of 10 and a CV of 10%, we would do:

Listing 1.5:

```
my.mean <- 10
my.cv <- 0.1
kappa <- 1 / my.cv^2
phiSquared <- 1 / my.mean
rgamma(10, kappa, kappa * phiSquared)
```

```
[1]  8.458743  8.678287  9.196287 10.050364  9.971803
[6]  9.843299 10.473241 10.677312 11.404397  9.381377
```

Exercise: Generate a sample of size 1000 from a Gamma distribution with mean = 0.25 and with coefficient of variation = 50 %. Verify that the sample has the desired properties and make a histogram of it.

Chapter 2

Simulating Natural Progression

In this chapter we are going to simulate new observations for new subjects in a new study. To keep things relatively simple on the first pass, we won't add in any placebo or drug effects, corresponding to the simulation of "naturally progressing" patients. Let's do this for 100 new subjects in a single new (non-interventional) study, with entry criteria restricing baseline MMSE scores to the range 16–26:

Listing 2.1:

```
nSubj <- 100
```

The posterior distribution from our model fit has been saved as a matrix (each row is a separate sample from the posterior, and each column is a different parameter). We can load this matrix as follows:

Listing 2.2:

```
library(adsim)
data(posterior)
```

To further simplify matters on this first pass, we are just going to work with point estimates of our population parameters, pretending for now that these estimates have no uncertainty associated with them. By "population" parameters, we mean only those parameters that are necessary to generate predictions of new observations in new subjects in new studies.

We will take the posterior medians as our point estimates:

Listing 2.3:

```
popPars <- apply(posterior, 2, median)
```

2.1 Sampling from a population of studies

Our model assumes that each study has its own "true" intercept and slope, all of which are sampled from some larger population of study-specific intercepts and slopes. In the notation used in the manuscript, this is expressed as:

$$\mu_{\eta,k} \sim \mathrm{N}\left(\nu_\eta, \psi_\eta^2\right) \tag{2.1}$$
$$\mu_{\alpha,k} \sim \mathrm{N}\left(\nu_\alpha, \psi_\alpha^2\right) \tag{2.2}$$

We do this in R with:

Listing 2.4:

```
set.seed(314159)
muEta <- rnorm(1, popPars['nuEta'], popPars['psiEta'] )
muAlpha <- rnorm(1, popPars['nuAlpha'], popPars['psiAlpha']  )
```

Note these parameter values need to be transformed in order to be interpreted on the original scale - see Table A.1 in the Appendix.

Exercise:

- What is the approximate cross-study population average rate of natural progression, expressed in points per year?
- Make a histrogram to show the posterior uncertainty in that value
- What is the approximate rate of natural progression for your study (i.e. the one you sampled with the R code above), expressed in points per year?
- Draw an entire sample of study-specific slopes, express them in points-per-year, and make a histogram.

At this point, the slope and intercept values correspond to a study accepting only individuals who have:

- Baseline MMSE=21
- 75 year old
- Male
- Non-carrier status for the ApoE-ε4 allele

(these were chosen as the reference values at the modeling stage). Note that, *conditional on the reference values*, the study-specific values for slope and intercept are assumed to be independent, however their common dependence on the covariates (which will be introduced

momentarily) implies that slope and intercept values are in fact stochastically dependent when looking across studies.

Our model also assumes that each study has its own "true" inter-subject variance in intercepts, its own "true" inter-subject variance in slopes, and its own "true" residual variance:

$$\begin{aligned} 1/\sigma^2_{\eta,k} &\sim \text{Gamma}(\kappa_\eta, \kappa_\eta \phi^2_\eta) \\ 1/\sigma^2_{\alpha,k} &\sim \text{Gamma}(\kappa_\alpha, \kappa_\alpha \phi^2_\alpha) \\ \tau_k &\sim \text{Gamma}(\kappa_\varepsilon, \kappa_\varepsilon \phi^2_\varepsilon). \end{aligned}$$

This aspect of the model reflects the fact that studies vary in the stringency of their eligibility criteria (and a study with more restrictive eligibility criteria will presumably have less inter-subject variability, for example), as well as the fact that studies vary in terms of the experience of the investigators and the training of the investigators (which would plausibly lead to inter-study differences in residual variance).

To sample the study-specific variance components in R we do:

Listing 2.5:

```
tauEta <- rgamma(1, popPars['kappaEta'], popPars['kappaEta'] *
   popPars['phiEta']^2)
sigmaEta <- 1 / sqrt(tauEta)
tauAlpha <- rgamma(1, popPars['kappaAlpha'], popPars['kappaAlpha']
    * popPars['phiAlpha']^2)
sigmaAlpha <- 1 / sqrt(tauAlpha)
tauResid <- rgamma(1, popPars['kappaEpsilon'], popPars['
   kappaEpsilon'] * popPars['phiEpsilon']^2)
```

We can again refer to Table A.1 to convert some of these standard deviations to the natural scale:

Listing 2.6:

```
## Cross-study average of inter-individual SD:
popPars['phiEta'] * 70 / 4
```

```
phiEta
7.2415
```

Listing 2.7:

```
## Inter-individual SD for "our study":
sigmaEta * 70 / 4
```

```
[1] 6.595239
```

Listing 2.8:

```
## Cross-study average of residual SD:
70 * sqrt( (25/70) * (1 - 25/70) * popPars['phiEpsilon']^2 / (1 +
   popPars['phiEpsilon']^2) )
```

```
phiEpsilon
  3.509151
```

Listing 2.9:

```
## Residual SD for "our study":
70 * sqrt( (25/70) * (1 - 25/70) * (1/tauResid) / (1 + 1/tauResid)
    )
```

```
[1] 3.602373
```

Exercise: The efficiency of a cross-over design depends on the size of the residual variance relative to the inter-subject variance. At baseline, what proportion of the total within-study variance is attributable to residual variation? Answer this at both the population level and in terms of the study that you have sampled.

2.2 Sampling subjects within the study

As mentioned previously, the covariates in our core model are baseline MMSE, age, gender, and ApoE4 status. Let's assume we are simulating a study with eligibility restricted to baseline MMSE in the range 16–26. In the absence of any other information, it probably makes sense to assume a Uniform distribution of MMSE scores within this range:

Listing 2.10:

```
bmmse <- runif(nSubj, 16, 26)
```

The other covariates can be simulated uniformly as well, but this will not reflect correlations present in realistic trial data. Instead, we can use the covariate model (tuned on the CAMD dataset) to generate We simulate ApoE4 status first, and then simulate realistic values of age conditional on the simulated value of ApoE4 status. The simulation functions are hidden, so we'll expose them using the ":::" function:

Listing 2.11:

```
apo <- adsim:::.simApoE(nSubj, popPars)
age <- adsim:::.simAge(nSubj, popPars, apo)
gender <- adsim:::.simGen(nSubj, popPars)
```

According to our model, the subject-level random effects are distributed as:

$$\eta_{pk}|\text{ study } k \sim \text{N}\left(\mu_{\eta,k}, \sigma^2_{\eta,k}\right) \tag{2.3}$$
$$\alpha_{pk}|\text{ study } k \sim \text{N}\left(\mu_{\alpha,k}, \sigma^2_{\alpha,k}\right), \tag{2.4}$$

where

$$\begin{aligned}
\mu_{\eta,k} &\leftarrow \mu_{\eta,k} + \lambda_{\eta,bMMSE}(bMMSE_{pk} - 21) && (2.5)\\
\mu_{\alpha,k} &\leftarrow \mu_{\alpha,k} + \lambda_{\alpha,bMMSE}(bMMSE_{pk} - 21) && (2.6)\\
&\quad + \lambda_{\alpha,bAge}(bAge_{pk} - 75) && (2.7)\\
&\quad + \lambda_{\alpha,Apo1} I(ApoE = 1) + \lambda_{\alpha,Apo2} I(ApoE = 2) && (2.8)\\
&\quad + \lambda_{\alpha,Gender} I(Gender = Female) && (2.9)
\end{aligned}$$

where $bMMSE$ denotes baseline MMSE score, $bAge$ is baseline age, and $ApoE = 1$ is a heterozygous carrier and $ApoE = 2$ is a homozygous carrier for the ApoE-ε4 allele.

We can first compute the means of these distributions, which are the *expected* intercepts and slopes for subjects with these baseline MMSE scores:

Listing 2.12:

```
muEtaAdj <- muEta + popPars['lambdaEtaBMMSE'] * (bmmse - 21)
muAlphaAdj <- muAlpha + popPars['lambdaAlphaBMMSE'] * (bmmse - 21)
    + popPars['lambdaAlphaAge'] * (age -75) + popPars['
   lambdaAlphaApo1'] * (apo==1) + popPars['lambdaAlphaApo2'] * (
   apo==2) + popPars['lambdaAlphaGen'] * gender
```

And we then sample the subject-level random effects as:

Listing 2.13:

```
eta <- rnorm(nSubj, muEtaAdj, sigmaEta)
alpha <- rnorm(nSubj, muAlphaAdj, sigmaAlpha)
```

Let's start putting these values in a data frame and adding the time dimension, assuming an 80 week study with assessments every 4 weeks:

Listing 2.14:

```
times <- seq(0, 80, 4)
subjId <- 1:nSubj
dat <- expand.grid(times, subjId)
names(dat) <- c("Week", "SubjId")
dat$BMMSE <- bmmse[dat$SubjId]
```

```
dat$Eta <- eta[dat$SubjId]  # (not available in a real data set)
dat$Alpha <- alpha[dat$SubjId] # (not available in a real data set
   )
```

Given the intercepts and slopes, we can compute the logit of the conditional expectation (a conditional expectation in this case being a simulated observation minus any residual variability), which the model defines as:

$$\log\left[\frac{\theta_{ipk}}{1-\theta_{ipk}}\right] \quad = \quad \eta_{pk} + \alpha_{pk} t_{ipk}, \tag{2.10}$$

We do this in R with:

Listing 2.15:

```
dat$LogitCondExp <- with(dat, Eta + Alpha * Week)
```

We can then apply the inverse logit transformations to convert these to conditional expectations on the natural scale (these are the θs in the notation of the manuscript, multiplied by 70):

$$\theta_{ipk} \quad = \quad 1/(1+\exp(-(\eta_{pk} + \alpha_{pk} t_{ipk})) \tag{2.11}$$

Listing 2.16:

```
dat$Theta <- with(dat, 1 / (1 + exp(-LogitCondExp)))
```

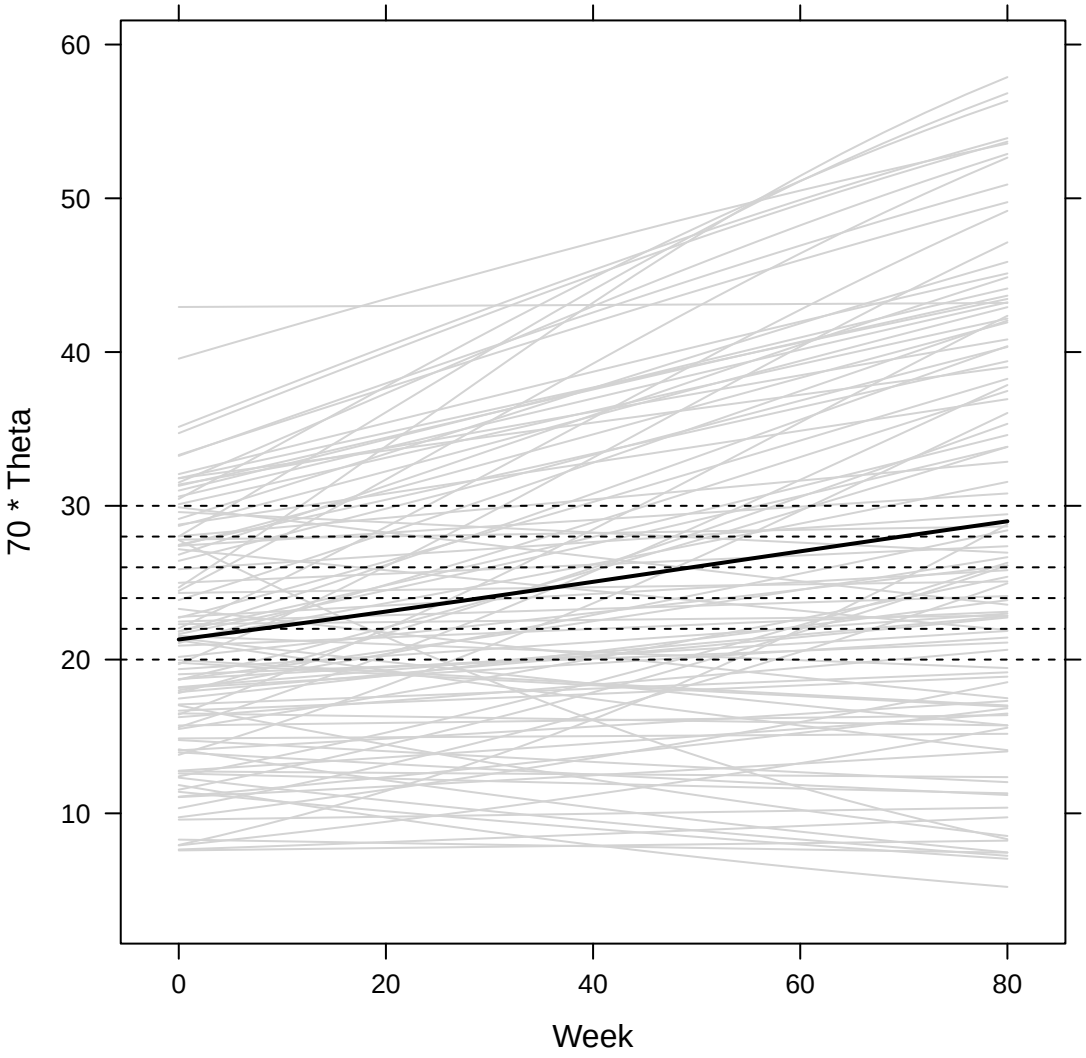
Let's take a look at these conditionally expected time-courses:

Listing 2.17:

```
library(lattice)
```

Listing 2.18:

```
print(
      xyplot(70 * Theta  ~  Week, groups = SubjId, data = dat,
             type = "l",
             col = "lightgrey",
             panel = function(x, y, ...) {
               panel.superpose(x, y, ...)
               panel.average(x, y, col = 'black', type = 'l',
                             lwd = 2, horizontal = FALSE)
               panel.abline(h = seq(20, 30, 2), lty = 2)
             }
             )
      )
```



2.3 Adding residual variability

It is now an easy matter to add residual variation around our conditional expectations:

$$ADAS_{ipk}/70 \,|\, \text{patient } p \;\sim\; \text{Beta}\left(\theta_{ipk}\tau_k,\, \left(1-\theta_{ipk}\right)\tau_k\right). \tag{2.12}$$

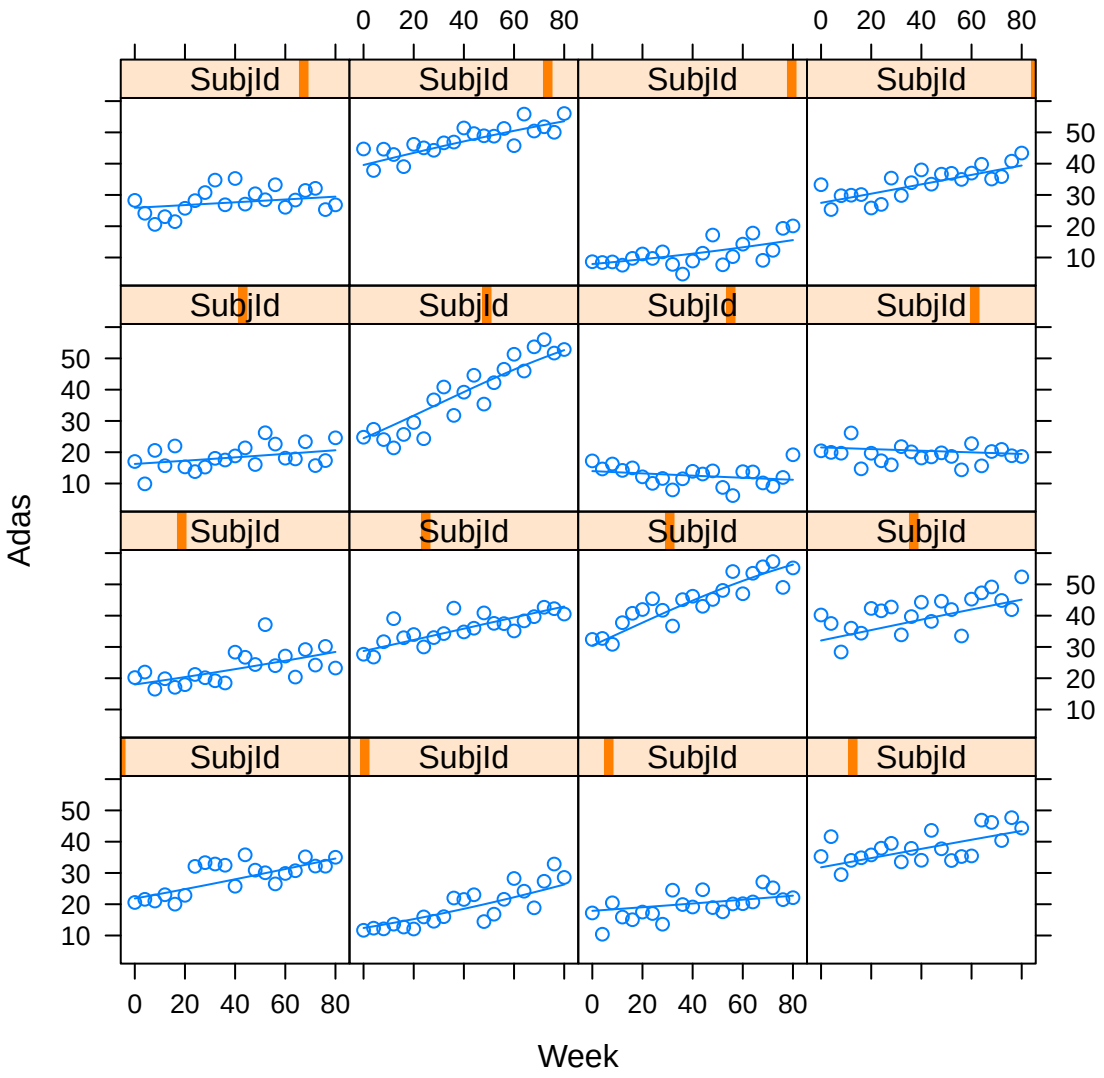
Listing 2.19:

```
dat$Adas <- with(dat, 70 * rbeta(length(Theta), Theta * tauResid,
   (1 - Theta) * tauResid))
```

Let's see what the simulated observed time-courses look like for the first sixteen subjects:

Listing 2.20:

```
print(
      xyplot(Adas  ~  Week | SubjId,
             data = dat,
             subset = SubjId <= 16,
             panel = function(x, y, subscripts, ...) {
               panel.xyplot(x, y, ...)
               panel.xyplot(x, 70 * dat$Theta[subscripts], type =
                  'l')
               }
             )
      )
```


SubjId
Adas
Week
0 20 40 60 80
10 20 30 40 50

Chapter 3

Adding the Effects of Intervention

In this chapter we will create a simulated data set corresponding to an interventional trial.

This time we will need to take a few new parameters along for the ride:

Listing 3.1:

```
library(adsim)
data(posterior)
popPars <- apply(posterior, 2, median)
```

According to our model, the conditional expectation on the logit scale is computed as the sum of terms for the natural disease state, the incremental placebo effect, and the incremental drug effects. We therefore retrace our steps from last chapter, up to the point of computing the conditional expectations on the logit scale.

This time let's simulate a study with 150 subjects with baseline MMSE 16–26, and assessments every 4 weeks for 80 weeks:

Listing 3.2:

```
set.seed(314159)
nSubj <- 150
times <- seq(0, 80, 4)
bmmse <- runif(nSubj, 16, 26)
apo <- adsim:::.simApoE(nSubj, popPars)
age <- adsim:::.simAge(nSubj, popPars, apo)
gender <- adsim:::.simGen(nSubj, popPars)
muEta <- rnorm(1, popPars['nuEta'], popPars['psiEta'] )
muAlpha <- rnorm(1, popPars['nuAlpha'], popPars['psiAlpha']  )
tauEta <- rgamma(1, popPars['kappaEta'], popPars['kappaEta'] *
   popPars['phiEta']^2)
sigmaEta <- 1 / sqrt(tauEta)
```

```
tauAlpha <- rgamma(1, popPars['kappaAlpha'], popPars['kappaAlpha']
    * popPars['phiAlpha']^2)
sigmaAlpha <- 1 / sqrt(tauAlpha)
tauResid <- rgamma(1, popPars['kappaEpsilon'], popPars['
   kappaEpsilon'] * popPars['phiEpsilon']^2)
muEtaAdj <- muEta + popPars['lambdaEtaBMMSE'] * (bmmse - 21)
muAlphaAdj <- muAlpha + popPars['lambdaAlphaBMMSE'] * (bmmse - 21)
    + popPars['lambdaAlphaApo1'] * (apo==1) + popPars['
   lambdaAlphaApo2'] * (apo==2) + popPars['lambdaAlphaAge'] * (age
   -75) + popPars['lambdaAlphaGen'] * gender
eta <- rnorm(nSubj, muEtaAdj, sigmaEta)
alpha <- rnorm(nSubj, muAlphaAdj, sigmaAlpha)
subjId <- 1:nSubj
dat <- expand.grid(times, subjId)
names(dat) <- c("Week", "SubjId")
dat$BMMSE <- bmmse[dat$SubjId]
dat$Apo <- apo[dat$SubjId]
dat$Age <- age[dat$SubjId]
dat$Gender <- gender[dat$SubjId]
dat$Eta <- eta[dat$SubjId]  # (not available in a real data set)
dat$Alpha <- alpha[dat$SubjId] # (not available in a real data set
   )
dat$LogitCondExp <- with(dat, Eta + Alpha * Week)
```

3.1 Adding the Placebo Effect

We now add the incremental effect of placebo. The model describes the placebo effect using a Bateman function:

$$E_{\text{PBO},ipk} = \beta \left(e^{-k_{el} t_{ipk}} - e^{-k_{eq} t_{ipk}} \right) \tag{3.1}$$

For technical reasons, a reparameterization of the placebo parameters was used at the model fitting stage. We first need to back-transform to get to the original (more useable) parameterization:

Listing 3.3:

```
kel <- popPars['kel']
keq <- kel + popPars['keqMinusKel']
beta <- - popPars['aucPlacebo'] / (1 / kel - 1 / keq)
```

Now we can compute the placebo effect on the logit scale at each assessment time:

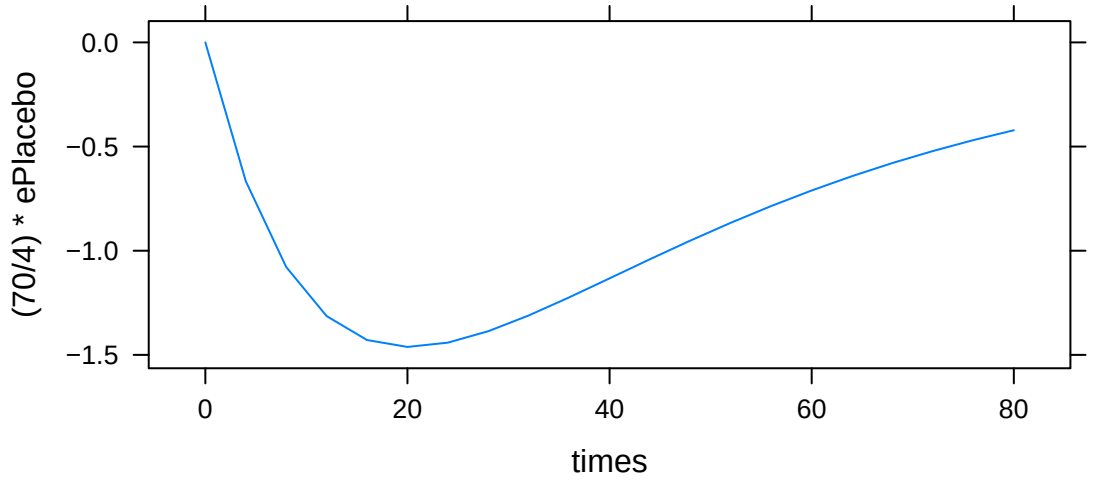
Listing 3.4:

```
ePlacebo <- beta * ( exp( -kel * times) - exp( -keq * times ) )
```

Before incorporating this in our data set, let's get a feel for what this incremental effect looks like. Because `ePlacebo` represents an effect on the logit scale, we need to convert it back to the original scale to make it interpretable. We can do this approximately with the linear approximation to the logit.

Listing 3.5:

```
library(lattice)
print(
      xyplot( (70/4) * ePlacebo  ~  times, type = 'l')
      )
```



We add this placebo effect as follows:

Listing 3.6:

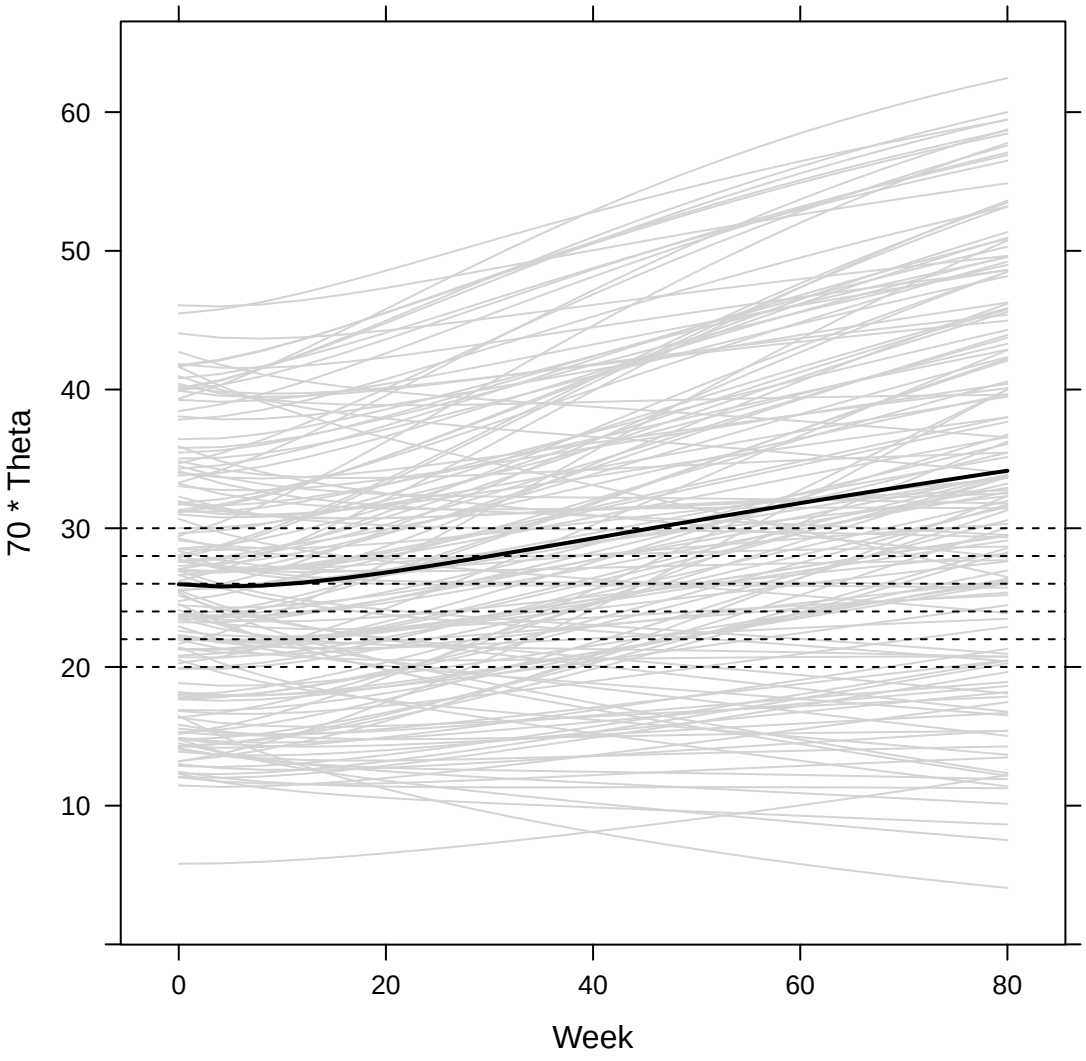
```
dat$LogitCondExp <- dat$LogitCondExp + rep(ePlacebo, nSubj)
```

(I replicated `ePlacebo` to make the logic a little easier to follow, but we could have also just taken advantage of R's recycling rules and computed this as `dat$LogitCondExp + ePlacebo`).

Now we'll do the same thing as in the previous chapter, first plotting the conditionally expected time-courses:

Listing 3.7:

```
dat$Theta <- with(dat, exp(LogitCondExp) / (1 + exp(LogitCondExp))
   )
print(
      xyplot(70 * Theta  ~  Week, groups = SubjId, data = dat,
             type = "l",
             col = "lightgrey",
             panel = function(x, y, ...) {
               panel.superpose(x, y, ...)
               panel.average(x, y, col = 'black', type = 'l',
                             lwd = 2, horizontal = FALSE)
               panel.abline(h = seq(20, 30, 2), lty = 2)
             }
             )
      )
```

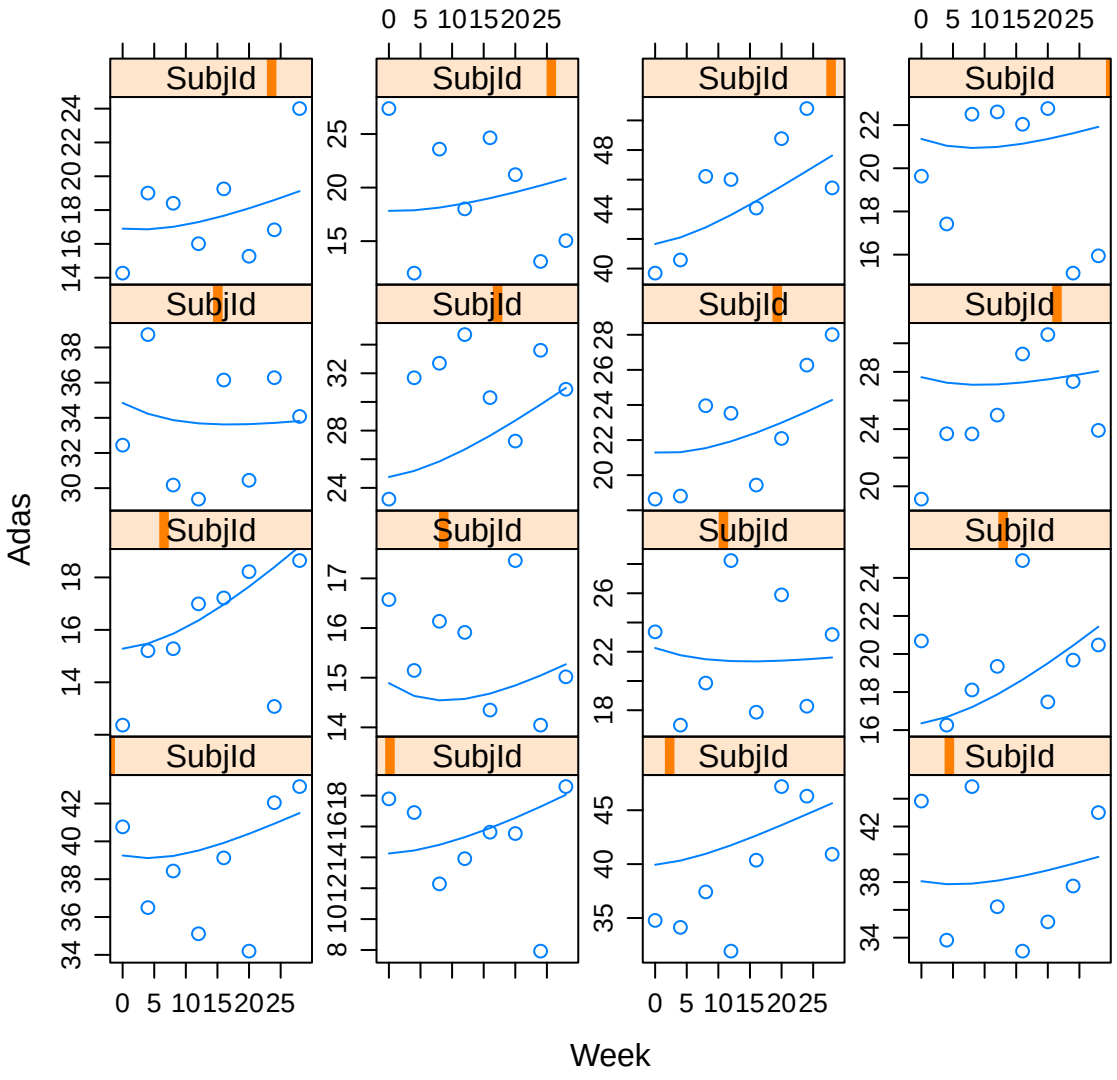


Now we can add the residual variability and plot some of the individual time-courses:

Listing 3.8:

```
dat$Adas <- with(dat, 70 * rbeta(length(Theta), Theta * tauResid,
   (1 - Theta) * tauResid))
print(
      xyplot(Adas  ~  Week | SubjId,
             data = dat,
             subset = SubjId <= 16 & Week <= 28,
             panel = function(x, y, subscripts, ...) {
               panel.xyplot(x, y, ...)
               panel.xyplot(x, 70 * dat$Theta[subscripts], type =
```

```
                    'l')
                },
              scales = list(y = list(relation = "free"))
              )
      )
```



3.2 Adding "Symptomatic" Drug Effects

As an illustration, we will simulate a trial randomizing 50 patients to placebo, 50 to donepezil 5 mg QD, and 50 to donepezil 10 mg QD. The parameters describing drug properties are the `eStar`, `et50`, and `gamma` parameters. The drugs are indexed as 1 = donepezil, 2 = galantamine, 3 = rivastigmine, and were assigned reference doses (for modeling purposes) of 5 mg QD, 24 mg QD, and 6 mg QD respectively.

The model describes the drug effects and their washout as:

$$E_{\mathrm{DRG},ipk} = (D_{ipk})^{\gamma_{d(p)}} \frac{E_{\Delta,d(p)} t_{ipk}}{ET_{50,d(p)} + t_{ipk}} \tag{3.2}$$

$$W_{\mathrm{DRG}}(t) = \exp\left\{ -\frac{\ln(2)\, t}{ET_{50,\mathrm{Wash(DRG)}}} \right\}. \tag{3.3}$$

We will put off the usage of $W_{\mathrm{DRG}}(t)$ until the trial simulation chapter noting that specification of the drug washout is done by simply setting $ET_{50,\mathrm{Wash(DRG)}}$ to the desired washout half-life.

A reparameterization for drug parameters was used at the modeling stage. Focusing on donepezil, we revert to the original parameterization with:

Listing 3.9:

```
eStar <- popPars['eStar[1]']
et50 <- popPars['et50[1]']
gamma <- popPars['gamma[1]']
b <- 12 / et50
eDelta <- - (1 + b) * eStar / b
```

The model estimated incremental effect of donepezil 5 mg QD at each of the assessment times is then:

Listing 3.10:

```
eDon5 <- eDelta * times / (et50 + times)
```

The model effect of 10 mg QD is computed by adjusting the effect at the reference dose as follows:

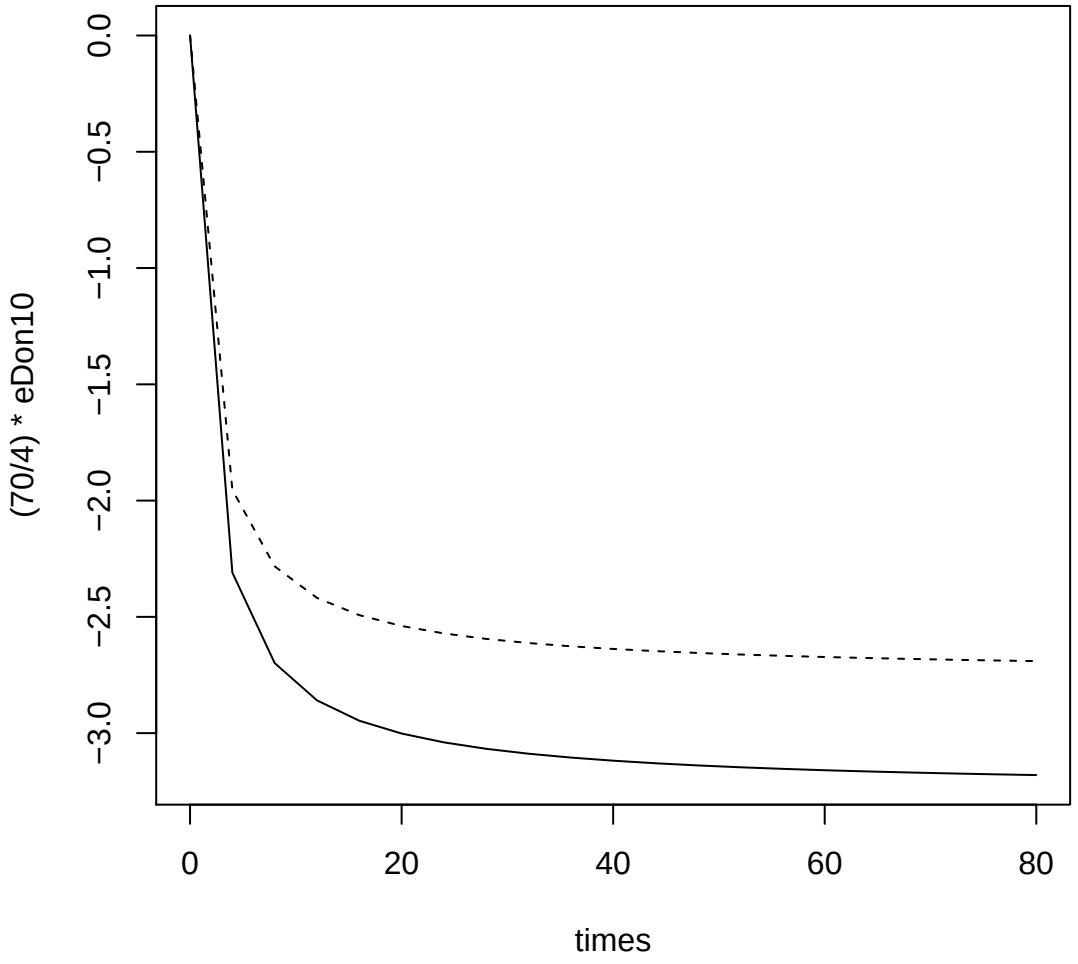
Listing 3.11:

```
refDose <- 5
eDon10 <-  eDon5 * (10/refDose)^gamma
```

As with the placebo effect, we will convert this approximately to the original scale in order to visualize it in isolation.

Listing 3.12:

```
plot(times, (70 / 4) * eDon10, type = 'l')
lines(times, (70 / 4) * eDon5, type = 'l', lty = 2)
```



Listing 3.13:

```
trts <- c("Placebo", "Donepezil 5 mg", "Donepezil 10 mg")
dat$Treatment <- rep(trts, each = 50)[dat$SubjId]
dat$Treatment <- factor(dat$Treatment, level = trts)
```

```
dat <- within(dat, LogitCondExp[Treatment == "Donepezil 5 mg"] <-
   LogitCondExp[Treatment == "Donepezil 5 mg"] + eDon5)
dat <- within(dat, LogitCondExp[Treatment == "Donepezil 10 mg"] <-
    LogitCondExp[Treatment == "Donepezil 10 mg"] + eDon10)
```

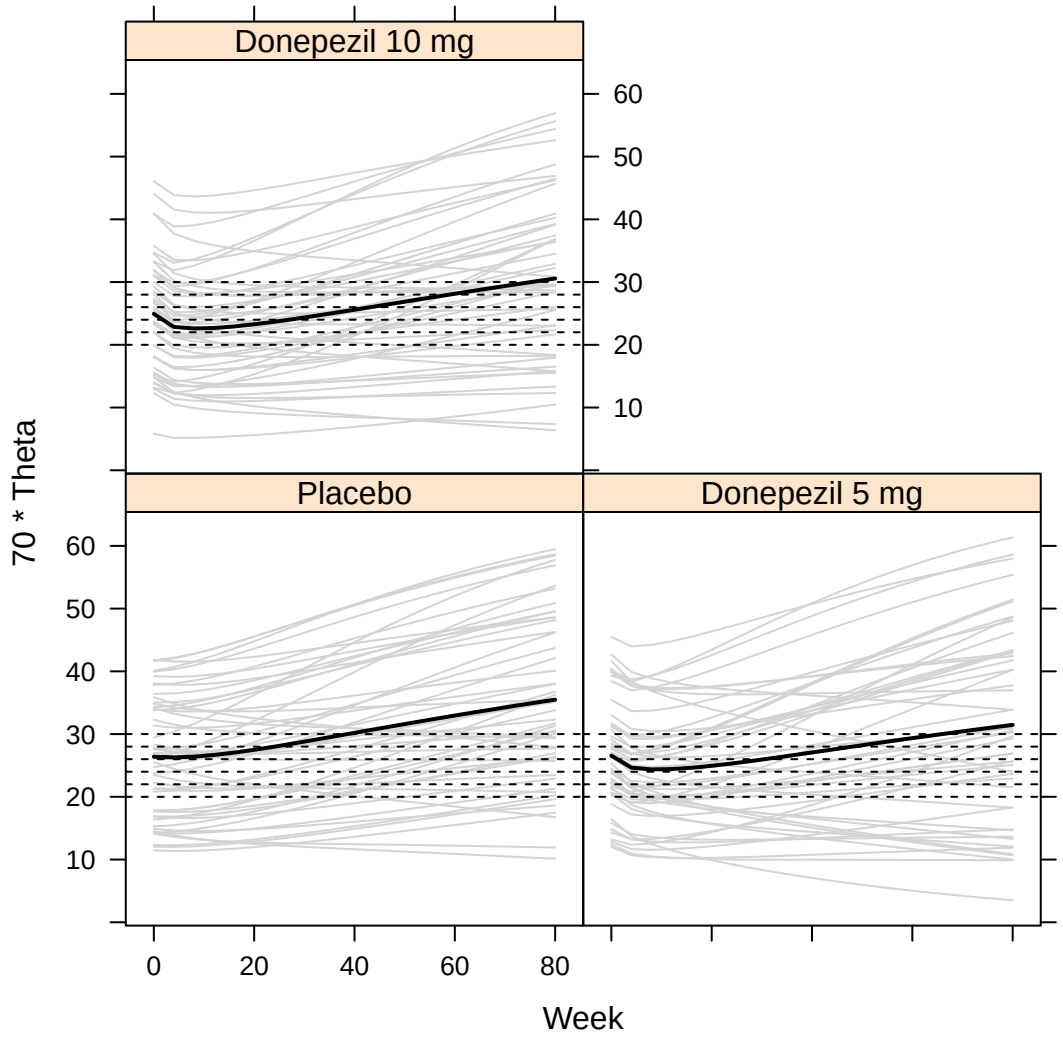
Based on these new conditional expectations on the logit scale, we need to recompute our conditional expectations on the original scale, and then regenerate the residual variability on top of that:

Listing 3.14:

```
dat$Theta <- with(dat, exp(LogitCondExp) / (1 + exp(LogitCondExp))
   )
dat$Adas <- with(dat, 70 * rbeta(length(Theta), Theta * tauResid,
   (1 - Theta) * tauResid))
```

Listing 3.15:

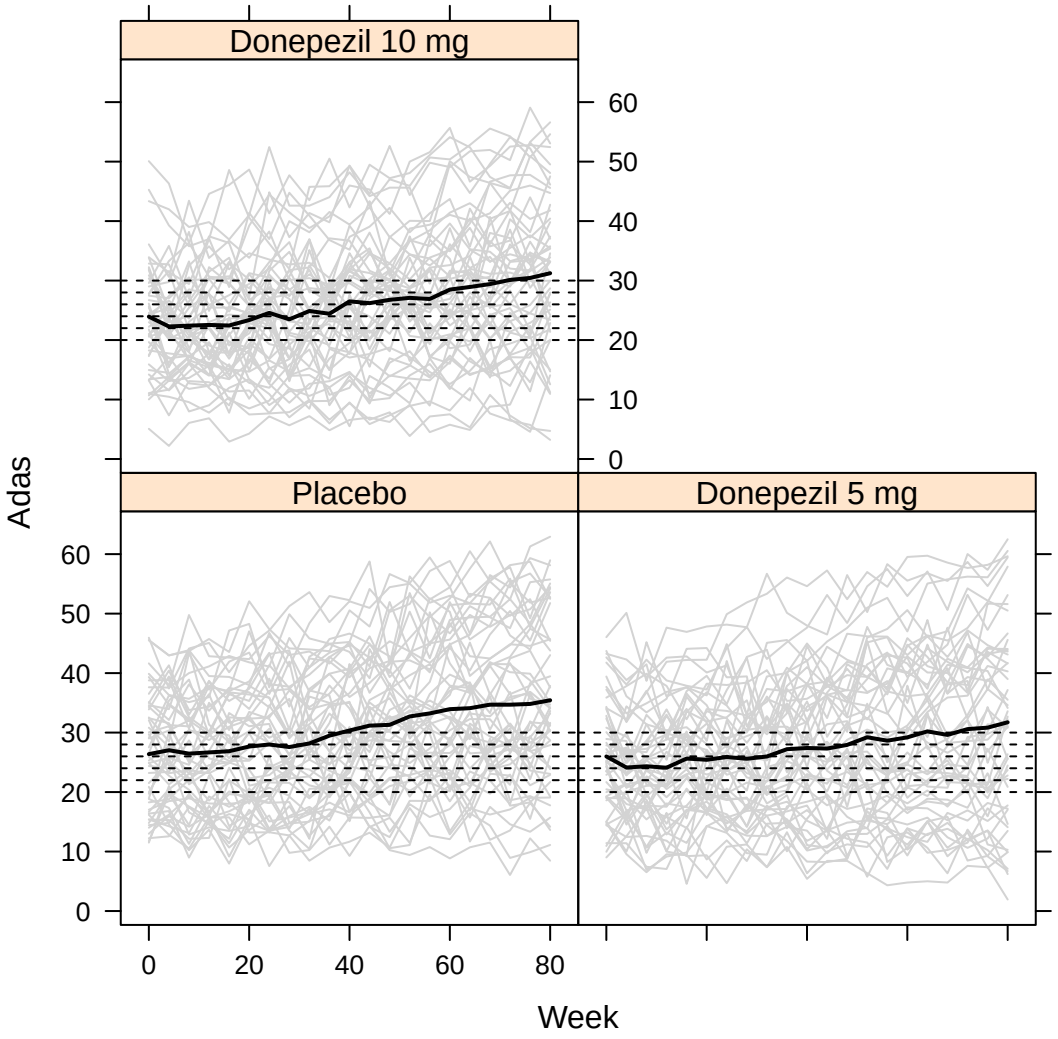
```
print(
      xyplot(70 * Theta  ~  Week | Treatment, groups = SubjId, data
          = dat,
             type = "l",
             col = "lightgrey",
             panel = function(x, y, ...) {
               panel.superpose(x, y, ...)
               panel.average(x, y, col = 'black', type = 'l',
                             lwd = 2, horizontal = FALSE)
               panel.abline(h = seq(20, 30, 2), lty = 2)
             }
             )
      )
```



Listing 3.16:

```
print(
      xyplot(Adas  ~  Week | Treatment, groups = SubjId, data = dat
         ,
             type = "l",
             col = "lightgrey",
             panel = function(x, y, ...) {
               panel.superpose(x, y, ...)
               panel.average(x, y, col = 'black', type = 'l',
                             lwd = 2, horizontal = FALSE)
               panel.abline(h = seq(20, 30, 2), lty = 2)
             }
```

```
                    )
          )
```



3.3 Adding "Disease Modifying" Drug Effects

Disease modifying agents are often conceptualized as affecting the "slope" parameters in our model. (There is no logical or physiological reason why a disease modifying effect would have to manifest itself in this way, but this is perhaps the most common way of thinking about what a disease modifying agent would do.) We will simulate a drug that reduces the "rate of

natural progression" by 50%.

Listing 3.17:

```
dmEffect <- 0.5
```

Our model involves several levels of "slope" parameters: the overall population slope (ν_α), the true study-specific slopes for baseline MMSE = 21 ($\mu_{\alpha,k}$), the true study-specific slopes adjusted for baseline MMSE, ApoE, age, and gender, and the true subject-specific slopes (α_{pk}). A proportional adjustment of subject-specific slopes probably does not conform to our intuitions about disease modification, since it would imply an inhibition of both cognitive decline (for those with postive slopes) *and* an inhibition of cognitive improvement (for those relatively few subjects with negative slopes). Any of the higher-level slope parameters are reasonable candidates for modification; in our example we will modify the true study-specific slopes adjusted for baseline MMSE ($\mu_{\alpha,k} + \lambda_{\alpha,bMMSE}(BMMSE_{pk} - 21) + \lambda_{\alpha,Age}...$), which are almost guaranteed to be positive for all baseline MMSE values.

Rather than trying to modify our existing data set, let's build a new one from scratch. Some of our steps can be retraced exactly:

Listing 3.18:

```
set.seed(142536)
nSubj <- 600
times <- seq(0, 80, 4)
bmmse <- runif(nSubj, 16, 26)
apo <- adsim:::.simApoE(nSubj, popPars)
age <- adsim:::.simAge(nSubj, popPars, apo)
gender <- adsim:::.simGen(nSubj, popPars)
muEta <- rnorm(1, popPars['nuEta'], popPars['psiEta'] )
muAlpha <- rnorm(1, popPars['nuAlpha'], popPars['psiAlpha']  )
tauEta <- rgamma(1, popPars['kappaEta'], popPars['kappaEta'] *
   popPars['phiEta']^2)
sigmaEta <- 1 / sqrt(tauEta)
tauAlpha <- rgamma(1, popPars['kappaAlpha'], popPars['kappaAlpha']
    * popPars['phiAlpha']^2)
sigmaAlpha <- 1 / sqrt(tauAlpha)
tauResid <- rgamma(1, popPars['kappaEpsilon'], popPars['
   kappaEpsilon'] * popPars['phiEpsilon']^2)
muEtaAdj <- muEta + popPars['lambdaEtaBMMSE'] * (bmmse - 21)
muAlphaAdj <- muAlpha + popPars['lambdaAlphaBMMSE'] * (bmmse - 21)
    + popPars['lambdaAlphaApo1'] * (apo==1) + popPars['
   lambdaAlphaApo2'] * (apo==2) + popPars['lambdaAlphaAge'] * (age
   -75) + popPars['lambdaAlphaGen'] * gender
```

Let's hold off on creating `Eta` and `Alpha` (we need to make the treatment assignments first before computing the `Alpha` values), but otherwise we can continue with the creation of the

simulated data set:

Listing 3.19:

```
subjId <- 1:nSubj
dat <- expand.grid(times, subjId)
names(dat) <- c("Week", "SubjId")
dat$BMMSE <- bmmse[dat$SubjId]
dat$ApoE <- apo[dat$SubjId]
dat$Gender <- gender[dat$SubjId]
dat$Age <- age[dat$SubjId]
trts <- c("Placebo", "DM Agent")
assignments <- rep(trts, each = nSubj / 2)
dat$Treatment <- assignments[dat$SubjId]
dat$Treatment <- factor(dat$Treatment, level = trts)
```

Now that we have treatment assignments, we can modify the appropriate slope values and continue on our merry way.

Listing 3.20:

```
muAlphaAdj[assignments == "DM Agent"] <- dmEffect * muAlphaAdj[
   assignments == "DM Agent"]
eta <- rnorm(nSubj, muEtaAdj, sigmaEta)
alpha <- rnorm(nSubj, muAlphaAdj, sigmaAlpha)
dat$Eta <- eta[dat$SubjId]  # (not available in a real data set)
dat$Alpha <- alpha[dat$SubjId] # (not available in a real data set
   )
dat$LogitCondExp <- with(dat, Eta + Alpha * Week)
```

Don't forget to add in the placebo effect:

Listing 3.21:

```
kel <- popPars['kel']
keq <- kel + popPars['keqMinusKel']
beta <- - popPars['aucPlacebo'] / (1 / kel - 1 / keq)
ePlacebo <- beta * ( exp( -kel * times) - exp( -keq * times ) )
dat$LogitCondExp <- dat$LogitCondExp + ePlacebo
```

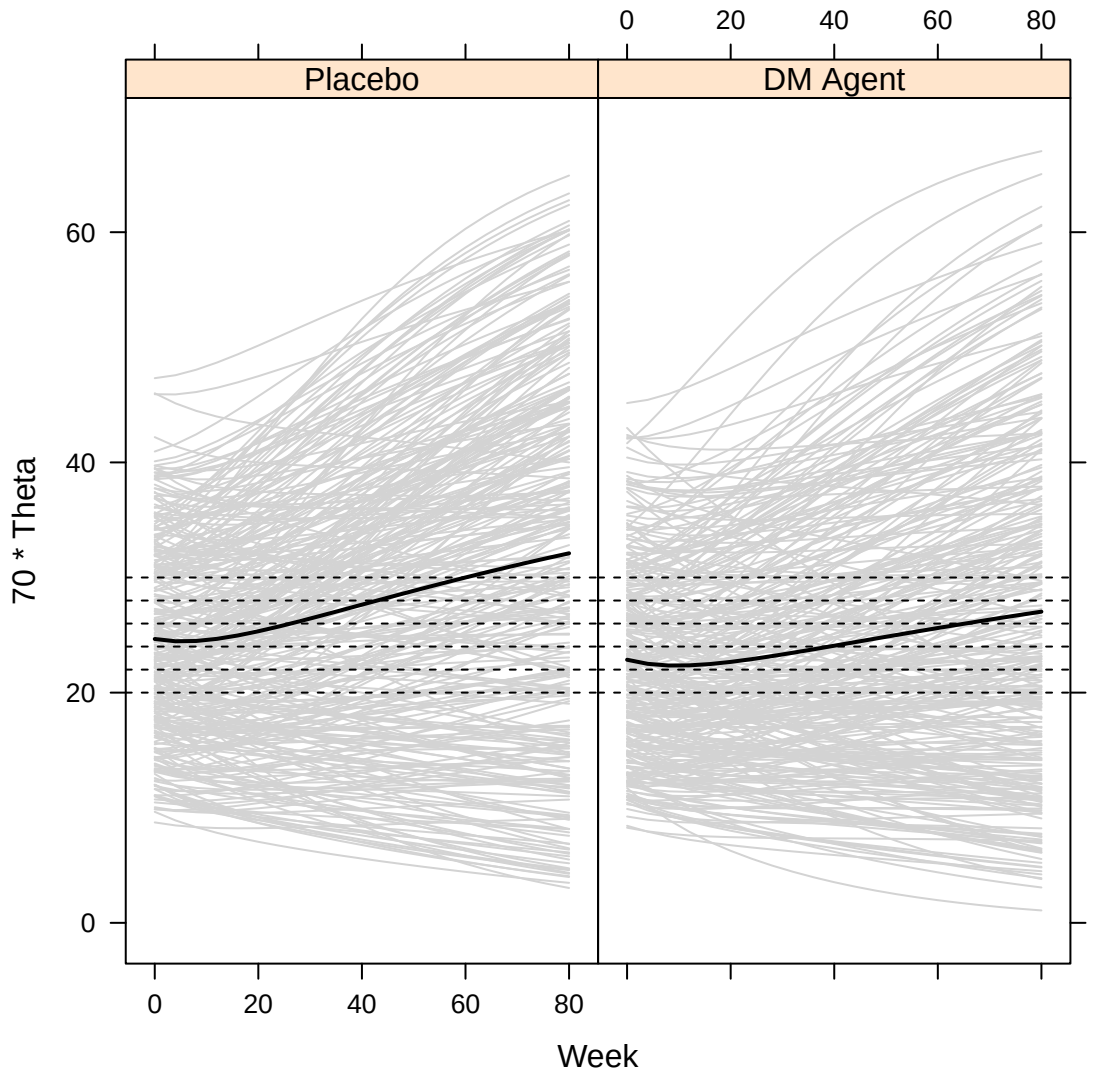
At this point we could add in some symptomatic effects as well, just as we did in the previous section. We will make such an example in a later chapter. For now, we finish the job with just the DM effects:

Listing 3.22:

```
dat$Theta <- with(dat, exp(LogitCondExp) / (1 + exp(LogitCondExp))
   )
dat$Adas <- with(dat, 70 * rbeta(length(Theta), Theta * tauResid,
   (1 - Theta) * tauResid))
```

Listing 3.23:

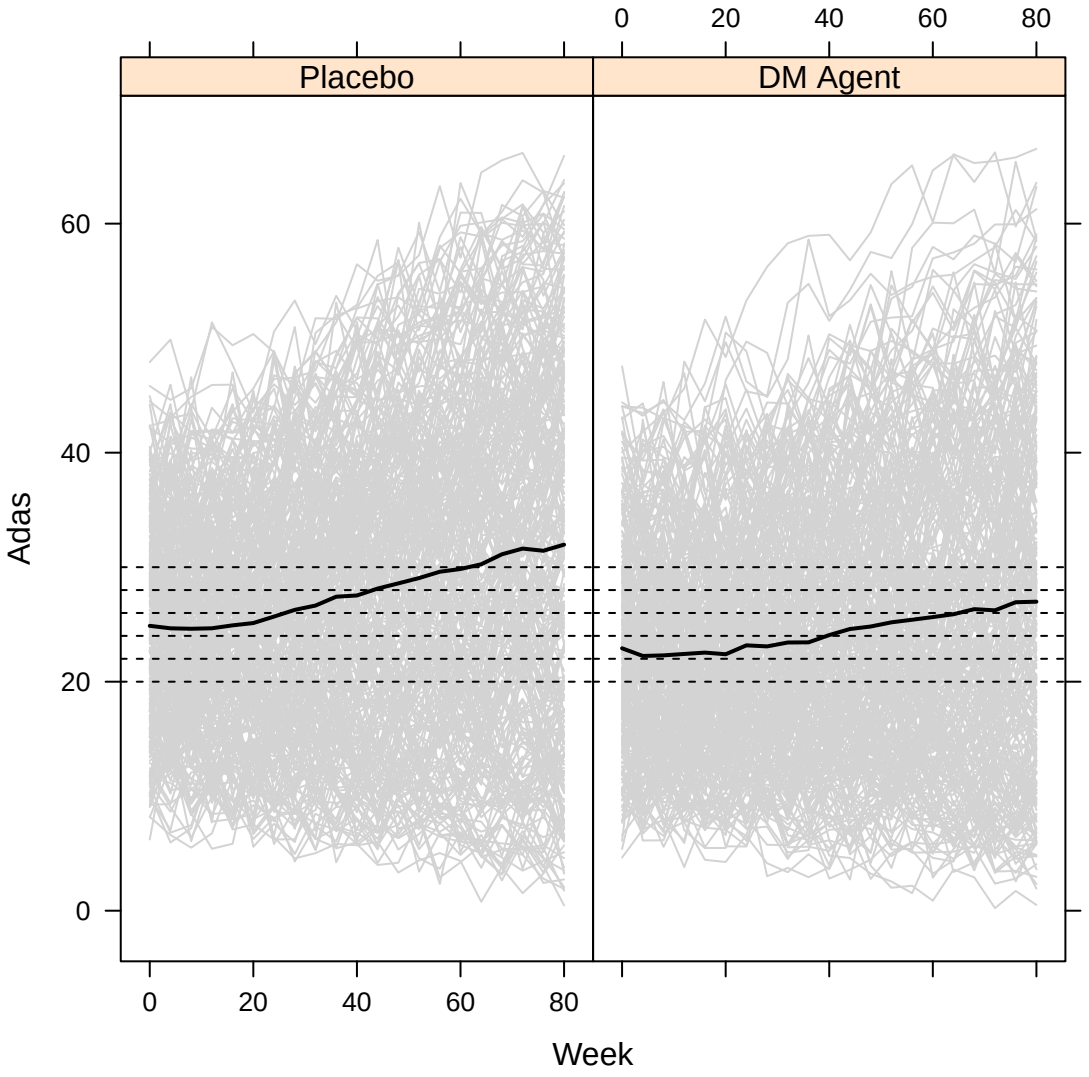
```
print(
      xyplot(70 * Theta  ~  Week | Treatment, groups = SubjId, data
          = dat,
             type = "l",
             col = "lightgrey",
             panel = function(x, y, ...) {
               panel.superpose(x, y, ...)
               panel.average(x, y, col = 'black', type = 'l',
                             lwd = 2, horizontal = FALSE)
               panel.abline(h = seq(20, 30, 2), lty = 2)
             }
             )
      )
```



Listing 3.24:

```
print(
      xyplot(Adas  ~  Week | Treatment, groups = SubjId, data = dat
         ,
             type = "l",
             col = "lightgrey",
             panel = function(x, y, ...) {
               panel.superpose(x, y, ...)
               panel.average(x, y, col = 'black', type = 'l',
                             lwd = 2, horizontal = FALSE)
               panel.abline(h = seq(20, 30, 2), lty = 2)
             }
```

```
                    )
          )
```


Placebo
DM Agent
Adas
Week
0
20
40
60
80

Chapter 4

Simulation to Evaluate Trial Designs

In the previous two chapters we have simulated single replicate of a clinical trial data set. While the simulation of a single trial may be helpful in order to generate a toy data set, we ultimately want to make statements about trial *designs*, and to do that we need to generate many trial replicates for each design of interest.

We will work with the same posterior samples as in previous chapters.

Listing 4.1:

```
library(adsim)
data(posterior)
```

And we can again start with point estimates:

Listing 4.2:

```
postMed <- apply(posterior, 2, median)
```

4.1 Simulation of a Cross-over Design

Cross-over designs have been considered in a number of AD therapeutic development programs at the proof-of-concept stage. It is well known that cross-over designs have greater efficiency than parallel group designs, though this advantage may be offset by the introduction of biases when treatment-by-period interactions are present. Perhaps the most common concern with respect to treatment-by-period interactions is the possibility of incomplete washout of a symptomatic effect, a concern that can be mitigated with the incorporation of a washout period between the two treatment periods. A washout period is of no use however, when a drug's effects are (at least partially) disease modifying: such effects, by definition, persist even after complete pharmacokinetic washout.

Therefore, when a cross-over design is considered to evaluate a drug whose effects may be at least partly disease modifying, it is prudent to assess the magnitude of the bias that might plausibly arise. We now provide an example where such an assessment is made by simulation.

We begin by introducing the functions `acRecruit()`, `acRandomize()`, and `acRun()` from the **adsim** package. This function bundles together many of the operations that were done in scripted fashion in the preceding chapters.

Suppose we want to consider the possibility that our drug has both a symptomatic effect and a disease modifying effect. We will assume the symptomatic effect follows a longitudinal Emax model with an ET_{50} of one week (similar to donepezil), and that the net symptomatic effect at 12 weeks is 1.4 points. With regard to the disease modifying component, we will assume (somewhat ambitiously) a 50% reduction in the rate of increase in ADAS-cog.

We can approximately translate the symptomatic effect at 12 weeks (on the original scale) to an Emax value on the logit scale as follows:

Listing 4.3:

```
eDelta.use <-  gLogitDelta(deltaAtRefTime = -1.4,
                           refTime = 12, # (weeks)
                           refAdas = 22.5, # this is the
                              approximate baseline ADAS
                           et50 = 1
                           )
eDelta.use$logitEmax
```

```
        12
-0.1010641
```

The Emax value we just computed is an approximation, and before proceeding we should check on the validity of this approximation. We can do this by simulating either a) the delta resulting from a single, very large trial with a parallel design and an assessment at 12 weeks or b) the mean delta of thousands of smaller trials with a parallel design and an assessment at 12 weeks. The latter option fits in to the overarching theme of this document well and has the additional benefit of being easily parallelized. In simulating these trials, we can demonstrate the three functions mentioned above. First, we begin by specifying the treatment sequence for each arm of a parallel design:

(NOTE: While we won't parallelize the problem here, note that the assumption is that these repliacted trials are independent and can be simulated easily in a distributed fashion. Interested readers with access to unix are referred to the `mclapply` function of the `multicore` package.)

Listing 4.4:

```
treatments <-  c("Placebo", "Test")
```

```
TrtSeqTable <- data.frame(Arm = LETTERS[1:2],
                          DrugDose = treatments,
                          DoseBegin = c(0, 0),
                          DoseEnd = c(12, 12)
                          )
TrtSeqTable
```

```
  Arm DrugDose DoseBegin DoseEnd
1   A  Placebo         0      12
2   B     Test         0      12
```

Now we proceed to simulation of the patient demographics and randomization into one of the two arms outlined in the treatment sequence table. After parameterizing the treatment, we'll use `acRecruit` to simulate patient demographics, `acRandomize` to assign them to an arm, and `acRun` to generate the ADAS-cog values for each patient, at each time point (just 0 and 12 weeks here). The function `acRandomize` allows for specification of randomization function (the argument `randfunc`). By default, `randfunc` is `blockRand` which can be quite inefficient for large simulations. For tuning purposes, use the function `simpleRand` included as part of the `adsim` package, which quite simply splits the patients into `n.levels` groups.

Listing 4.5:

```
simpleRand
```

```
function (n, num.levels)
{
    rep(LETTERS[1:num.levels], each = n/num.levels)
}
<environment: namespace:adsim>
```

To simulate the "true delta" and tune the drug parameterization to match our specified `delta`, we proceed as follows:

Listing 4.6:

```
n.per.arm <- 30
n.enrolled <- n.per.arm*2
treatments <- c("Placebo", "Test")
lbMmse.use <- 14
ubMmse.use <- 26
TrtParTable <- data.frame(
                          DrugDose = treatments,
                          emaxSx = c(0, eDelta.use$logitEmax),
                          et50Sx = 1,
                          et50SxWash = 1, ## Washout half-life
```

```
                            eDm = c(0, 0)  ## Just tuning Emax now
                            )
TrtParTable
```

```
   DrugDose     emaxSx et50Sx et50SxWash eDm
    Placebo  0.0000000      1          1   0
12     Test -0.1010641      1          1   0
```

Listing 4.7:

```
## Without making the code parallel, it takes roughly one minute
## to run 10^4 trials on a 2.4Ghz machine.
## Using 10^5 trials corresponds to a 0.01 SE, so scale according
   to your
## desired level of precision.

minutes <- 10
nSims <- minutes*10^4
trueDelta <- lapply(1:nSims, function(i){
  patientTable <- acRecruit(n.enrolled, postMed,
                            lbMmse=lbMmse.use, ubMmse=ubMmse.use
                            )
  baseTable <- acRandomize(patientTable, TrtSeqTable, TrtParTable,
                           randfunc=simpleRand)
  bigDat <- acRun(p=postMed, baseTable, timevec=c(0,12))
  mns <- with(bigDat, tapply(AdasCog, list(DrugDose, Time), mean)
     )
  apply(mns, 2, diff)["12"]
})
trueDelta <- unlist(trueDelta)
mean(trueDelta)                          ## Estimated effect
```

```
[1] -1.351082
```

Listing 4.8:

```
sd(trueDelta)/sqrt(length(trueDelta))    ## Standard error
```

```
[1] 0.008112507
```

The approximation does not correspond exactly to an effect of -1.4, we can adjust this and simulate the exact scenario that we are advertising by using a technical device sometimes referred to as the "fudge factor":

Listing 4.9:

```
fudgeFactor <- -1.4 / mean(trueDelta)
```

```
eDelta.use <-  gLogitDelta(deltaAtRefTime = -1.4 * fudgeFactor,
                           refTime = 12, # (weeks)
                           refAdas = 22.5, # this is the
                              approximate baseline ADAS
                           et50 = 1
                           )
TrtParTable <- data.frame(
                           DrugDose = treatments,
                           emaxSx = c(0, eDelta.use$logitEmax),
                           et50Sx = 1,
                           et50SxWash = 1,
                           eDm = c(0, 0)  ## Just tuning Emax now
                           )
trueDelta <- lapply(1:nSims, function(i){
  patientTable <- acRecruit(n.enrolled, postMed, lbMmse=lbMmse.use
     , ubMmse=ubMmse.use)
  baseTable <- acRandomize(patientTable, TrtSeqTable, TrtParTable)
  bigDat <- acRun(p=postMed, baseTable, timevec=c(0,12))
  mns <- with(bigDat, tapply(AdasCog, list(DrugDose, Time), mean)
     )
  apply(mns, 2, diff)["12"]
})
trueDelta <- unlist(trueDelta)
mean(trueDelta)  # Should get you close enough to the advertised
   value Emaxf, if not you need larger nSims
```

```
[1] -1.397416
```

Listing 4.10:

```
sd(trueDelta)/sqrt(length(trueDelta))
```

```
[1] 0.008067079
```

If this does not get you close enough to eDelta, then you can increase `nSims` to reach the desired precision, adjusting the fudge factor. In the interest of running this quickly, we will proceed with the given precision. Since we are also envisaging a 50% DM effect, let's determine what the 12 week delta will be with that factored in:

Listing 4.11:

```
TrtParTable <- data.frame(
                           DrugDose = treatments,
                           emaxSx = c(0, eDelta.use$logitEmax),
                           et50Sx = 1,
                           et50SxWash = 1,
```

```
                          eDm = c(0, .5)
                          )
trueDelta <- lapply(1:nSims, function(i){
  patientTable <- acRecruit(n.enrolled, postMed, lbMmse=lbMmse.use
     , ubMmse=ubMmse.use)
  baseTable <- acRandomize(patientTable, TrtSeqTable, TrtParTable)
  bigDat <- acRun(p=postMed, baseTable, timevec=c(0,12))
  mns <- with(bigDat, tapply(AdasCog, list(DrugDose, Time), mean)
     )
  apply(mns, 2, diff)["12"]
})
trueDelta <- unlist(trueDelta)
trueDiff <- mean(trueDelta)
sd(trueDelta)/sqrt(length(trueDelta))
```

```
[1] 0.007999302
```

Now that we have our drug parameters determined, let's use the functions to generate a single replicate of a realistically sized cross-over design. The candidate design that we will evaluate is a cross-over design with 30 patients per arm, two 12 week treatment periods and a 3 week washout in between. Additionally, we will add in our dropout model. So far we have simulated under the pretense that there is no dropout, but the posterior matrix provided in the `posterior` object actually contains parameters that pertain to a dropout model fit to the CAMD data set. Specifically, a Weibull hazard model was used such that:

$$\begin{aligned} \ln(\lambda_{pjk}) &= \beta_0 + \beta_{Age} * (age - 75) + \beta_{bMMSE} * (bMMSE - 21) + \omega_k \\ t_{pjk}|\omega_k &\sim \text{Weibull}(\lambda_{pjk}, v) \\ \omega_k &\sim \text{N}(0, \sigma_\omega^2). \end{aligned}$$

Using the function `dropTime`, we can simulate the dropout times given the simulated covariates, as illustrated below.

Listing 4.12:

```
n.per.arm <- 30
n.enrolled <- n.per.arm * 2
treatments <- c("Test")  ## Both Arms will receive the treatment
TrtSeqTable <- data.frame(Arm = LETTERS[1:2],
                          DrugDose = treatments,
                          DoseBegin = c(0, 15),
                          DoseEnd = c(12, 27)
                          )
TrtParTable <- data.frame(
```

```
                            DrugDose = treatments,
                            emaxSx = eDelta.use$logitEmax,
                            et50Sx = 1,
                            et50SxWash = 1,
                            eDm = 0.5
                            )
patientTable <- acRecruit(n.enrolled, postMed, lbMmse = lbMmse.use
   , ubMmse = ubMmse.use)
baseTable <- acRandomize(patientTable, TrtSeqTable, TrtParTable)
baseTable$dropT <- dropTime(baseTable, postMed)
t.global <- seq(0, 27, 3)
xOverDat <- acRun(p=postMed, xFrame=baseTable, timevec=t.global)
xOverDat <- xOverDat[ xOverDat$Time < xOverDat$dropT, ] ##
   Observable values
```

We can perform a typical primary analysis on this data set using the function `pAnalyzeXover` (this function is really just an example, you may want to write your own function and tailor it more exactly to the primary analysis that is envisaged for your trial). The function requires a few other variables to be passed in as part of the patient set,

- Period : The period number (1 or 2)
- Stratum : Levels of strata, for example on BMMSE
- TimeWithin : Time since beginning of current period
- Site : Study site for the patient.

Defaults for these may be incorporated into `pAnalyzeXover` in future versions of the `adsim` package, but for now we demonstrate reasonable choices:

Listing 4.13:

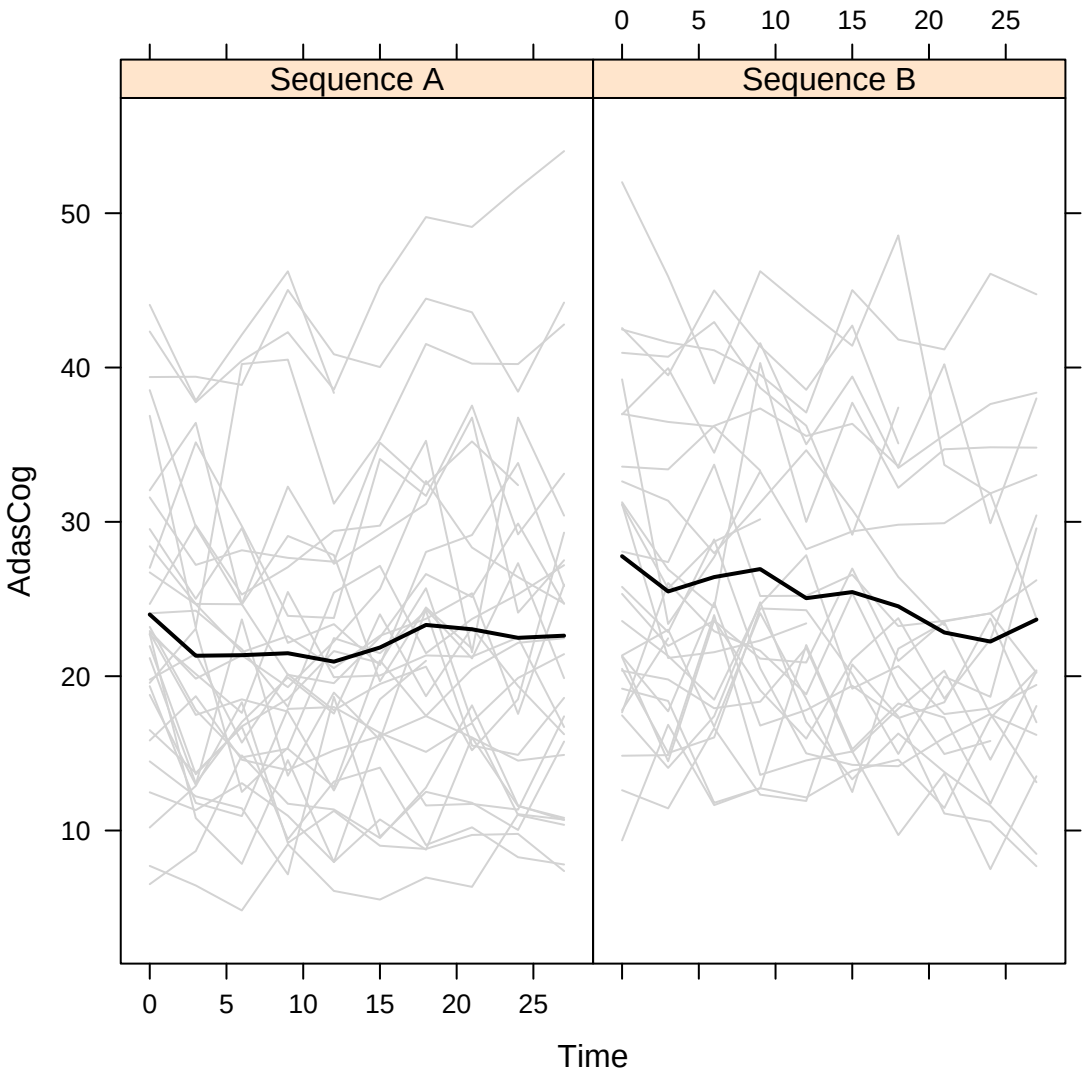
```
xOverDat$Period <- 1 + with(xOverDat, Time >= 15)
xOverDat$Stratum <- factor(ifelse(xOverDat$Bmmse < 20, 'mild', '
   moderate'))
xOverDat$TimeWithin <- with(xOverDat, Time %% 15)
nSites <- 6  ## Just assign these randomly, we test for a site
   effect to assure there isn't one
site <- sample(1:nSites, length(unique(xOverDat$Patient)), replace
   =TRUE)
xOverDat$Site <- factor(site[xOverDat$Patient])
pAnalyzeXover(xOverDat, ep.time = 12)
```

```
   Contrast           SE        Lower        Upper
-1.62471083   0.74587921  -3.08660723  -0.16281444
   testStat           df       Pvalue
-2.17824924 526.00000000   0.02983086
```

If we want we can also visualize the simulated data set with something like:

Listing 4.14:

```
library(lattice)
print(
      xyplot(AdasCog ~ Time | paste("Sequence", Arm), data =
         xOverDat,
             groups = Patient,
             type = 'l',
             panel = function(x, y, ...) {
               panel.xyplot(x, y, ...)
               panel.average(x, y, horizontal = FALSE, col = '
                  black', lwd = 2)
             },
             col = "lightgrey"
             )
      )
```

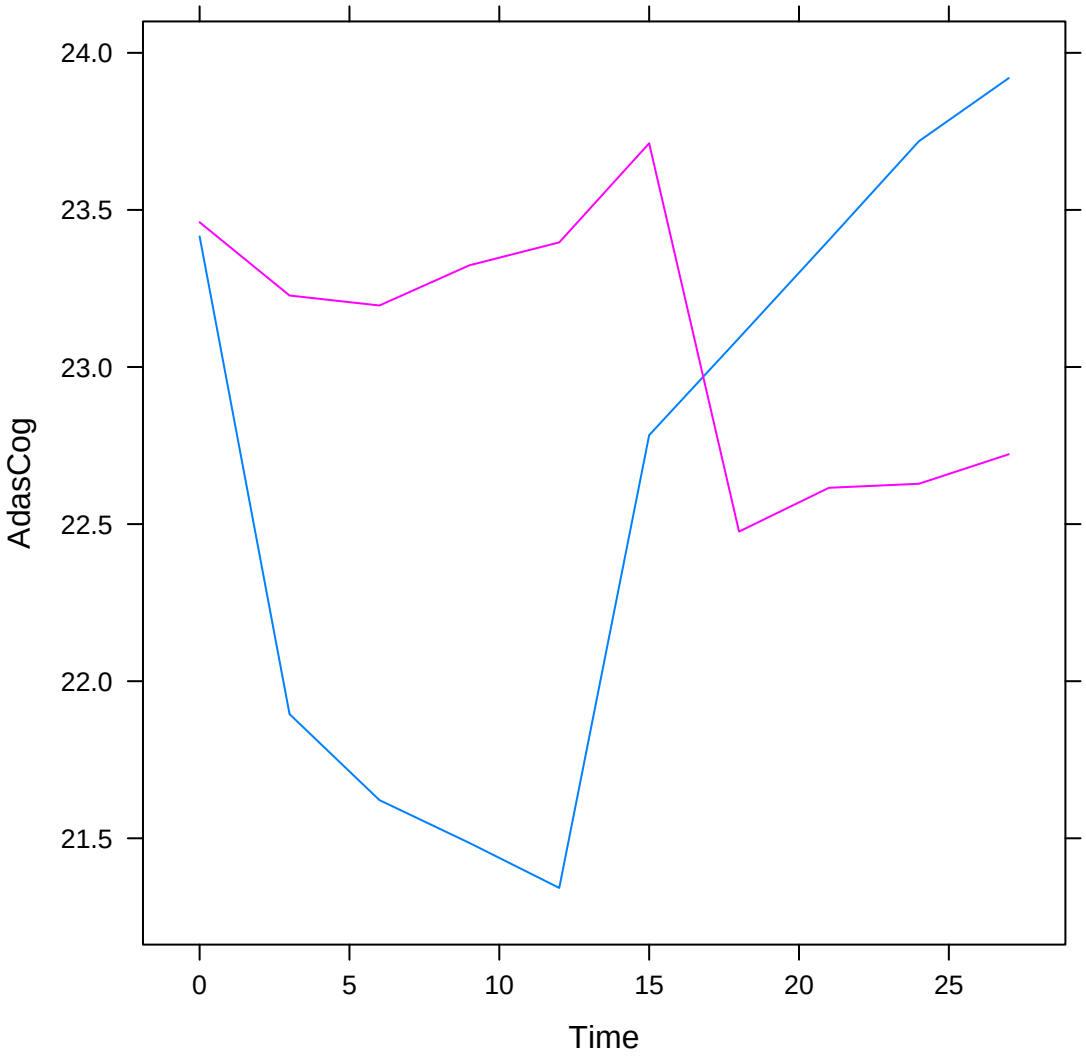


It is also insightful to plot the means from an experiment with the same design but a much larger sample size, just to make sure we understand what is going on:

Listing 4.15:

```
library(reshape)
n.per.arm <- 10^4
n.enrolled <- n.per.arm * 2
patientTable <- acRecruit(n.enrolled, postMed, lbMmse = lbMmse.use
   , ubMmse = ubMmse.use)
baseTable <- acRandomize(patientTable, TrtSeqTable, TrtParTable)
t.global <- seq(0, 27, 3)
xOverDat <- acRun(postMed, baseTable, t.global)
```

```
xOverDat$dropT <- dropTime(xOverDat, postMed)
xOverDat <- xOverDat[ xOverDat$Time < xOverDat$dropT, ] ##
   Observable
xOverDat$Period <- 1 + with(xOverDat, Time >= 15)
xOverDat$Stratum <- factor(ifelse(xOverDat$Bmmse < 20, 'mild', '
   moderate'))
xOverDat$TimeWithin <- with(xOverDat, Time %% 15)
nSites <- 6
site <- sample(1:nSites, length(unique(xOverDat$Patient)), replace
   =TRUE)
xOverDat$Site <- factor(site[xOverDat$Patient])
mns <- as.data.frame(sparseby(xOverDat, list(xOverDat$Arm,
   xOverDat$Time), function(dati) mean(dati$AdasCog)))
names(mns) <- c("Arm", "Time", "AdasCog")
print(
      xyplot(AdasCog  ~  Time, data = mns,
             groups = Arm,
             type = "l"
             )
      )
```



Now we are ready for the real meat of the exercise: creating many trial replicates and summarizing across the replicates. There are a number of ways we could do this, and for a large simulation we might need to think carefully about execution time, memory management, etc. For now we will just do 1000 replicates and we proceed as follows:

Listing 4.16:

```
nSim <- 1000
n.per.arm <- 30
n.enrolled <- n.per.arm*2
system.time(
  datList <- lapply(1:nSim, function(simNum){
    patientTable <- acRecruit(n.enrolled, postMed, lbMmse = lbMmse
```

```
        .use, ubMmse = ubMmse.use)
    baseTable <- acRandomize(patientTable, TrtSeqTable,
        TrtParTable)
    baseTable$dropT <- dropTime(baseTable, postMed)
    xOverDat <- acRun(postMed, baseTable, t.global)
    xOverDat <- xOverDat[ xOverDat$Time < xOverDat$dropT, ] ##
        Observable
    xOverDat$Period <- 1 + with(xOverDat, Time >= 15)
    xOverDat$Stratum <- factor(ifelse(xOverDat$Bmmse < 20, 'mild',
        'moderate'))
    xOverDat$TimeWithin <- with(xOverDat, Time %% 15)
    nSites <- 6
    site <- sample(1:nSites, length(unique(xOverDat$Patient)),
        replace=TRUE)
    xOverDat$Site <- factor(site[xOverDat$Patient])
    xOverDat
  })
)
```

```
   user  system elapsed
 15.981   0.539  16.527
```

Now we can analyze each one of these trial replicates and stack the results in a data frame:

Listing 4.17:

```
system.time(resList <- lapply(datList, pAnalyzeXover, ep.time =
    12))
```

```
   user  system elapsed
171.226   1.030 172.660
```

Listing 4.18:

```
res <- do.call('rbind', resList)
```

We are now able to assess operating characteristics by summarizing across all of the different replicates. A basic operating characteristic of interest is the bias of the usual estimator. This estimator is captured in the `Contrast` column of `res`, so the bias is:

Listing 4.19:

```
mean(res[, 'Contrast']) - (trueDiff)
```

```
[1] 0.3137994
```

Another basic operatic characteristic of interest is power. Assuming our significance threshold is 5% two-sided, the estimated power is:

Listing 4.20:

```
sum(res[, 'Pvalue'] < 0.05) / nrow(res)
```

```
[1] 0.607
```

Although we won't do the calculations here, we note that if we had taken the same delta value and the same variance component estimates and used conventional power calculations for a cross-over design, we would have arrived at a higher estimated power. The conventional calculations, because they do not reflect the treatment-period interaction that would arise from a partially disease modifying effect, over-estimate the true power.

To investigate to what degree this true, we repeat the same exercise with a drug whose effect is entirely symptomatic, but whose net effect at 12 weeks is the same as drug hypothesized above.

Listing 4.21:

```
## Again, let's take our approximate Emax and improve on it a bit:
eDelta.use <-  gLogitDelta(deltaAtRefTime = trueDiff,
                           refTime = 12, # (weeks)
                           refAdas = 22.5, # this is the
                              approximate baseline ADAS
                           et50 = 1
                           )
n.per.arm <- 30
n.enrolled <- n.per.arm*2
treatments <- c("Placebo", "Test")
lbMmse.use <- 14
ubMmse.use <- 26
TrtSeqTable <- data.frame(Arm = LETTERS[1:2],
                           DrugDose = treatments,
                           DoseBegin = c(0, 0),
                           DoseEnd = c(12, 12)
                           )
TrtParTable <- data.frame(
                           DrugDose = treatments,
                           emaxSx = c(0, eDelta.use$logitEmax),
                           et50Sx = 1,
                           et50SxWash = 1,
                           eDm = c(0, 0)  ## Just tuning Emax now
                           )
minutes <- 5
nSims <- minutes*10^4
trueDelta <- lapply( 1:nSims, function(i){
  patientTable <- acRecruit(n.enrolled, postMed, lbMmse=lbMmse.use
     , ubMmse=ubMmse.use)
```

```
  baseTable <- acRandomize(patientTable, TrtSeqTable, TrtParTable)
  bigDat <- acRun(p=postMed, baseTable, timevec=c(0,12))
  mns <- with(bigDat, tapply(AdasCog, list(DrugDose, Time), mean)
     )
  apply(mns, 2, diff)["12"]
})
trueDelta <- unlist(trueDelta)
fudgeFactor <- trueDiff / mean(trueDelta)
eDelta.use <-  gLogitDelta(deltaAtRefTime = trueDiff * fudgeFactor
   ,
                           refTime = 12, # (weeks)
                           refAdas = 22.5, # this is the
                              approximate baseline ADAS
                           et50 = 1
                           )
TrtParTable$emaxSx <- c(0, eDelta.use$logitEmax)
trueDelta <- lapply( 1:nSims, function(i){
  patientTable <- acRecruit(n.enrolled, postMed, lbMmse=lbMmse.use
     , ubMmse=ubMmse.use)
  baseTable <- acRandomize(patientTable, TrtSeqTable, TrtParTable)
  bigDat <- acRun(p=postMed, baseTable, timevec=t.global)
  mns <- with(bigDat, tapply(AdasCog, list(DrugDose, Time), mean)
     )
  apply(mns, 2, diff)["12"]
})
trueDelta <- unlist(trueDelta)
trueDiff2 <- mean(trueDelta)
## Now we simulate trials
## HERE
nSim <- 1000
n.per.arm <- 30
n.enrolled <- n.per.arm*2
treatments <- c("Test")  ## Both Arms will receive the treatment
TrtParTable <- data.frame(
                           DrugDose = treatments,
                           emaxSx = eDelta.use$logitEmax,
                           et50Sx = 1,
                           et50SxWash = 1,
                           eDm = 0
                           )
TrtSeqTable <- data.frame(Arm = LETTERS[1:2],
                           DrugDose = treatments,
                           DoseBegin = c(0, 15),
                           DoseEnd = c(12, 27)
```

```
                            )
system.time(
  datList <- lapply(1:nSim, function(simNum){
    patientTable <- acRecruit(n.enrolled, postMed, lbMmse = lbMmse
       .use, ubMmse = ubMmse.use)
    baseTable <- acRandomize(patientTable, TrtSeqTable,
       TrtParTable)
    baseTable$dropT <- dropTime(baseTable, postMed)
    xOverDat <- acRun(postMed, baseTable, t.global)
    xOverDat <- xOverDat[ xOverDat$Time < xOverDat$dropT, ] ##
       Observable
    xOverDat$Period <- 1 + with(xOverDat, Time >= 15)
    xOverDat$Stratum <- factor(ifelse(xOverDat$Bmmse < 20, 'mild',
        'moderate'))
    xOverDat$TimeWithin <- with(xOverDat, Time %% 15)
    nSites <- 6
    site <- sample(1:nSites, length(unique(xOverDat$Patient)),
       replace=TRUE)
    xOverDat$Site <- factor(site[xOverDat$Patient])
    xOverDat
  })
)
```

```
   user  system elapsed
 16.229   0.300  16.531
```

Listing 4.22:

```
system.time(resList <- lapply(datList, pAnalyzeXover, ep.time =
   12))
```

```
   user  system elapsed
159.388   1.538 160.956
```

Listing 4.23:

```
res <- do.call('rbind', resList)
mean(res[, 'Contrast']) - (trueDiff2)
```

```
[1] 0.03416877
```

Listing 4.24:

```
sum( res[ ,'Pvalue'] < 0.05 ) / nrow(res)
```

```
[1] 0.72
```

This time the bias is nearly zero. We would expect this because there is no treatment-by-period interaction in this scenario. (Strictly speaking, the model still implies a small degree of treatment-by-period interaction because the drug effect, though fixed on the logit scale, varies with "baseline" on the original scale; this type of treatment-by-period interaction appears to be negligible in this context). As mentioned before, the power estimate for this strictly symptomatic compound should be at least as large as the estimated power from the dose modifying compound.

4.2 Simulation of a Parallel Group Design

Exercise: Use simulation to determine the power of a trial with the following characteristics:

- Parallel group design with 100 patients per group
- Assessments every 12 weeks for 84 weeks
- Primary analysis based on the drug effect at 84 weeks, using analysis methodology as specified in the **adsim** R function `pAnalyzeParallel()`.
- Suppose that the drug under investigation is hypothesized to have no symptomatic benefit, but to provide a 40% reduction in the rate of natural progression.

Note: the "tricks" to use to get `AdasCogTwoArmSim()` to simulate a parallel group design were discussed in the preceding section.

4.3 Simulation of a Delayed Start Design

Exercise: Use simulation to determine the power of a trial with the following characteristics:

- Delayed start design with 600 patients per group
  - Two arms with two phases each: placebo-treament and treatment-treament
  - A 91 week trial, assessment periods are weeks 26-52 and weeks 65-91
  - The placebo-treatment arm is switched to treatment at week 52
- Primary analysis based on the effects in both periods, using analysis methodology as specified in the **adsim** R function `pAnalyzeDelayedStart()`. Consider a treatment successful when it differs significantly from the placebo at the end of both periods.

- Suppose that the drug under investigation is hypothesized to have no symptomatic benefit, but to provide a 40% reduction in the rate of natural progression.

Note: If you are stuck, the help file for `acRandomize()` should get you on the right track and the solutions are posted in the appendix.

Appendix A

Approximate Transformations to Make Parameters Interpretable

Table A.1: Approximate transformations between parametric scale and original scale. (*) Population averages for slope and intercept refer to a reference sub-population with MMSE = 21. (**) Use of the Beta distribution for residuals implies that the residual standard deviation varies as a function of expected value; therefore, for purposes of approximate conversion to the original scale, we select a reference expected value of ADAS-cog = 25.

Description of Estimand	**Approximation as Function of Parameters**
Population average	
intercept (points) (*)	$70 \times \exp(\nu_\eta)/(1+\exp(\nu_\eta))$
slope (points per year) (**)	$52 \times 70 \times \nu_\alpha/4$
Covariate adjustment per point MMSE	
for intercepts (points ADAS-cog per point MMSE)	$70 \times \lambda_\eta/4$
for slopes (points ADAS-cog per year per point MMSE)	$52 \times 70 \times \lambda_\alpha/4$
Placebo (incremental) effect	
area under curve (point-weeks)	$70 \times \int E_{\text{PBO}}/4$
constant for elimination (weeks^{-1})	k_{el}
constant for onset of placebo effect (weeks^{-1})	k_{eq}
Drug effects (same prior for all AChE inhibitors)	
difference from placebo at reference dose at 12 weeks (points)	$70 \times E_{\text{drug}}(12, D^*)/4$
time to 50% of maximum drug effect (weeks)	ET_{50}
shape of dose response	γ
Inter-study SD	
of intercepts (points)	$70 \times \psi_\eta/4$
of slopes (points per year)	$70 \times \psi_\alpha/4$
Cross-study (population) average	
of inter-individual SD of intercepts (points)	$70 \times \phi_\eta/4$
of inter-individual SD of intercepts (points per year)	$52 \times 70 \times \phi_\alpha/4$
of residual standard deviation (points) (**)	$70 \times \sqrt{\left(\frac{25}{70}\right) \times \left(1-\frac{25}{70}\right) \times \frac{\phi_\varepsilon^2}{(1+\phi_\varepsilon^2)}}$
Between-study variation (CV%)	
in inter-individual precisions of intercepts	$100 \times 1/\sqrt{\kappa_\eta}$
in inter-individual precisions of slopes	$100 \times 1/\sqrt{\kappa_\alpha}$
in residual precision	$100 \times 1/\sqrt{\kappa_\varepsilon}$

A.1 Exercise Solution: Parallel Group Design

We have seen the parallel design before in tuning the `Emax` parameter, so we proceed without explanation aside from stating that there is no need to tune `Emax` in this example as the desired effect is due completely to dose modification.

Listing A.1:

```
library(adsim)
data(posterior)
postMed <- apply(posterior, 2, median)
n.per.arm <- 100
n.enrolled <- n.per.arm * 2
t.global <- seq(0, 84, 12)
treatments <-  c("Placebo", "Test")
lbMmse.use <- 14
ubMmse.use <- 26
TrtSeqTable <- data.frame(Arm = LETTERS[1:2],
                          DrugDose = treatments,
                          DoseBegin = c(0, 0),
                          DoseEnd = c(84, 84)
                          )
## No need to tune this
TrtParTable <- data.frame(
                          DrugDose = treatments,
                          emaxSx = c(0, 0),
                          et50Sx = 1,
                          et50SxWash = 1,
                          eDm = c(0, 0.4)
                          )
nSim <- 1000
datList <- lapply(1:nSim, function(simNum){
  patientTable <- acRecruit(n.enrolled, postMed, lbMmse = lbMmse.
     use, ubMmse
                          = ubMmse.use)
  baseTable <- acRandomize(patientTable, TrtSeqTable, TrtParTable)
  baseTable$dropT <- dropTime(baseTable, postMed)
  simDat <- acRun(postMed, baseTable, t.global)
  simDat[ simDat$Time < simDat$dropT, ]
})
resList <- lapply(datList, pAnalyzeParallel, ep.time=84)
res <- do.call( rbind, resList )
sum( res[ ,'Pvalue.1'] < 0.05 ) / nrow(res)  ## Predictive power
```

A.2 Exercise Solution: Delayed Start

In the delayed start exercise we are again considering a treatment without symptomatic components, so we can proceed to the predictive power simulations without tuning `Emax`. The design begins with a placebo controlled phase which is followed by a phase where all subjects are on the active treatment. (NOTE: A word of warning, the analysis step here is not fast. This step could take over an hour and a half depending upon your computational environment. Unix users (Mac + Linux) can replace the `lapply` calls with `mclapply` calls from the `multicore` package to speed up the process.)

Listing A.2:

```
library(adsim)
data(posterior)
postMed <- apply(posterior, 2, median)
n.per.arm <- 600
n.enrolled <- n.per.arm * 2
treatments <-  c("Test")
t.global <- c(26 * (0:2), 26 * 2 + 13 + 26 * (0:1))
lbMmse.use <- 14
ubMmse.use <- 26
TrtSeqTable <- data.frame(Arm = LETTERS[1:2],
                          DrugDose = treatments,
                          DoseBegin = c(0, 52),
                          DoseEnd = max(t.global)
                          )
TrtParTable <- data.frame(
                          DrugDose = treatments,
                          emaxSx = 0,
                          et50Sx = 1,
                          et50SxWash = 1,
                          eDm = c(0.4) # proportional decrease (*
                             not* proportion preserved)
                          )
nSim <- 1000
datList <- lapply(1:nSim, function(simNum){
  patientTable <- acRecruit(n.enrolled, postMed, lbMmse = lbMmse.
     use, ubMmse = ubMmse.use)
  baseTable <- acRandomize(patientTable, TrtSeqTable, TrtParTable)
  baseTable$dropT <- dropTime( baseTable, postMed )
  simdat <- acRun(postMed, baseTable, timevec = t.global, sim.
     interstudy = TRUE)
  simdat <- simdat[ simdat$Time < simdat$dropT, ]
  simdat
})
```

```
resList <- lapply(datList, pAnalyzeDelayedStart,
                  ep.time = c(52, 91), stab.time = c(65, 91) )
res <- do.call( rbind, resList )
sum( (res[,'Pvalue.1.52'] < 0.05) & (res[,'Pvalue.1.91'] < 0.05) )
    /
  nrow(res)
```