1 Overview of **doptim** package

All of the results in Chapter 2 were produced with the R-packages and doptim randon. Both packages were produced by the present author for the purpose of this thesis.

The genObjFun function in the doptim package allows the user to specify an NLME model and, for a given size of population design, generate the corresponding D-optimality objective function, which can then be optimized with any of the available optimizers in R. The user can choose between

- FO and InFO D-optimality
- reduced and full FIM
- returning log[det(FIM)] or the whole FIM. This is of interest when singular FIMs occur
- D or D_A-optimality, i.e., the user can specify a function of θ as the parameter of interest
- Local or pseudo-Bayesian D-optimality

The calc_aPMS function in the randon package allows the user to calculate the aPMS for a given regression function. It is used by the plot_sensBands function to plot sensitivity bands as in REF. calc_aPMS is also used by the genOptimDesnRange function; this function is used to generate the design graphs in REF and calc_aPMS allows us to plot with variance level on the x-axis rather than the magnitude of the variance components.

2 Example: Simple linear regression aka how to specify a model in **doptim**

This example covers the basic usage of the genObjFun function in the doptim package. It is used to generate a D-optimality objective function for a user-specified NLME model: simple linear regression with random effects.

In this model, η is of the form

$$\eta(\boldsymbol{\beta}, t) = \beta_1 + \beta_1 t = (\beta_1 + \beta_2)\begin{pmatrix}1\\ t\end{pmatrix},$$

with $t \in [a, b]$ for a fixed choice of $a, b > 0$ with $a < b$. Since η is linear in $\boldsymbol{\beta}$ this will technically lead to an LME model, which is not representative of the models this thesis is otherwise concerned with. However, an LME model is merely a special case of an NLME model (recall, η is allowed but not required to be nonlinear in $\boldsymbol{\beta}$).

One reason to consider such a simple model to begin with is that it is one of the few cases where explicit results regarding D-optimal designs are available.

In the fixed effects case, and with an even number of samples, it is known (see e.g. page 70 of Atkinson et al. (2007)) that the D-optimal design assigns half of the samples to $t = a$ and the other half of the samples to $t = b$. Therefore, it can be expected that when ω_1 and ω_2 are both small , the D-optimal design for estimating $\boldsymbol{\beta}$ will be of a similar form.

The parameters of the model are assumed to be

$$\boldsymbol{\beta} = (3, 5) \quad \Omega = \mathsf{diag}\{.1, .1\} \quad \sigma^2 = 0.25^2 \ ,$$

and the first task is to find a $D_{\boldsymbol{\beta}}$-optimal design for a single individual ($N = 1$? for whom six samples are taken ($k = 6$? in the time interval $[0, 10]$.

To generate the $D_{\boldsymbol{\beta}}$-optimality objective function run the following R-code:

```
objFun <- genObjFun(
  eta = expression(
    (beta1 + b1) + (beta2+b2) * t
  ),
  Ebeta = c(0, 1),
  Vb = c(Vb1 = .1, Vb2 = .1),
  Veps = .25^2,
  phi = "fixed",
  desvarNames = "t",
  noSamples = 6
  )
```

The arguments to genObjFun clearly specify the model described above as well as some aspects of the design problem; by setting phi = "fixed" it is indicated that a $D_{\boldsymbol{\beta}}$-optimal design is sought and by setting noSamples = c(6) the number of individuals is set to the length of the noSamples vector (i.e. one) and the number of samples (for each individual) is set to the corresponding entry in the noSamples vector (i.e. six for the single individual in the design).

The objective function can now be evaluated in a design of the user's choice, provided it is of the correct length:

```
Xi0 <- c(1, 2, 4, 6, 8, 10)
objFun(Xi0)

# returns #

[1] -4.276033
```

One aspect of the design problem which is not specified above is the experimental region, i.e., that $[a, b] = [0, 10]$. That is addressed in the optimisation step where the user is free to employ any optimisation algorithm they deem appropriate. Given that this is a bounded optimisation problem, the nlminb function (available in the pre-loaded package stats) can be used:

```
nlminb(
    start = Xi0,
    objective = objFun,
    lower = 0,
    upper = 10
)

# returns #

$`par`
[1]  0  0  0  0  0 10

$objective
[1] -4.480053

$convergence
[1] 0

$iterations
[1] 20

$evaluations
function gradient
20      121

$message
[1] "both X-convergence and relative convergence (5)"
```

The list of outputs from nlminb includes 'par' which is the optimum identified by the algorithm. In this case, the optimum identified by the algorithm is different from the theoretical optimum in the fixed effects case. A simple way to check this is to reduce the magnitude of the variance component and re-run the code. Since it takes at least seven arguments to specify a D-optimality problem, it is advisable to save these arguments in a dedicated list so as to avoid duplicating code:

```
args <- list(
  eta = expression(
    (beta1 + b1) + (beta2+b2) * t
  ),
  Ebeta = c(0, 1),
  Vb = c(Vb1 = 1, Vb2 = 1)/10,
  Veps = .25^2,
  phi = "fixed",
  desvarNames = "t",
  noSamples = c(6)
)
```

Now it is simple to reduce the variance components and re-run the problem with the rest of the problem retained:

```
args$Vb = c(Vb1 = 1, Vb2 = 1)/10000

objFun <- do.call(genObjFun, args)

nlminb(
  start = Xi0,
  objective = objFun,
  lower = 0,
  upper = 10
)

# returns #

$`par`
[1]  0  0  0 10 10 10

$objective
[1] -11.94752

$convergence
[1] 0

$iterations
[1] 12

$evaluations
function gradient
12       72

$message
[1] "both X-convergence and relative convergence (5)"
```

With the variance component reduced, nlminb yields the same optimum as the fixed effects version of the problem, i.e., it splits the samples evenly between the two boundaries of the time interval.

3 Example: exponential decay - first nonlinear example

A simple, but important, example where the regression function is nonlinear is exponential decay. This is another case where a general expression for the D-optimal design has been found for the fixed effects case (Kitsos and Kolovos, 2013).

The regression function is

$$\eta(\boldsymbol{\beta}, t) = \beta_1 \exp(\beta_2 t) \ ,$$

the parameters of the model are assumed to be

$$\boldsymbol{\beta} = (1, -1/10) \quad \Omega = \mathsf{diag}\{1/100, 1/100\} \quad \sigma^2 = 1/1000$$

and the experimental region is $\mathcal{X} = [0, 30]$. To specify and solve the problem, again searching for a single elementary D_β-optimal design of length six:

```
args <- list(
  eta = expression(
  (beta1 + b1) * exp((beta2 + b2)*t)
  ),
  Ebeta = c(1, - 1/10),
  Vb = c(1, 1)/100,
  Veps = 1/1000,
  phi = "fixed",
  desvarNames = "t",
  noSamples = c(6)
)


objFun <- do.call(genObjFun, args)

Xi0 = 1:6/10

nlminb(
  start = Xi0,
  objective = objFun,
  lower = 0,
  upper = 30
)

# returns #

$`par`
[1]  0.000000  0.000000  0.000000  3.894444  3.894445 25.562844
```

```
$objective
[1] -14.13597

$convergence
[1] 0

$iterations
[1] 23

$evaluations
function gradient
24      157

$message
[1] "relative convergence (4)"
```

According to Kitsos and Kolovos (2013), the D-optimal design for the fixed effects case has support points $(0, -1/\beta_1) = (0, 10)$. Again, one can check if decreasing the magnitude of the variance component produces designs that are closer to the fixed effects case. In the code below, a decreasing sequence of variance component magnitudes is used to generate a sequence of D_β-optimal designs:

```
var_mags <- seq(1/10^3, 1/10^5, length.out = 20)

optim_desns <- t(sapply(var_mags,
       function(x){
         args$Vb <- c(1, 1) * x
         objFun <- do.call(genObjFun, args)
         res <- try(nlminb(
           start = Xi0,
           objective = objFun,
           lower = 0,
           upper = 30
         )$par)
         if(class(res) == "try-error") return(rep(NA, length(Xi0)))
         return(res)
         }))

# returns #

Error in chol.default(x) :
the leading minor of order 2 is not positive definite
```

Note that in a single case, numerical issues caused the matrix inversion of the variance of the linearised model to fail. Base R can be used to generate a simple design graph with the following code:

```
s
matplot(var_mags, optim_desns)
```

Which produces the following figure for the 19 successfully computed designs.

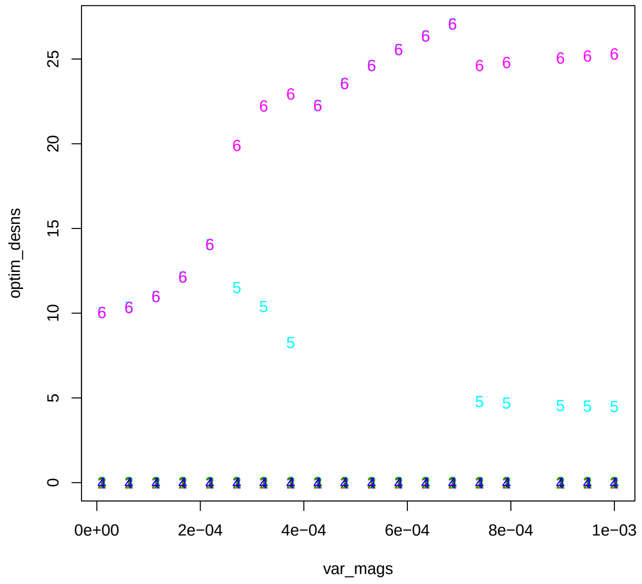


Figure 1: A simple version of the design graphs found in earlier chapters. The plotting symbol is the index of the sampling time in the individual D_β-optimal design being plotted and the x-axis is the variance magnitude used to generate the design.

As can be seen from Figure 1, when the magnitude of the variance component is reduced sufficiently, the D_β-optimal designs have two sampling times, the first at 0 and the second at a time which converges to 10. This is consistent with the above-mentioned result from Kitsos and Kolovos (2013).

4 Example: replicating results from [KBT: REF] wang2012

In Wang et al. (2012), several kinds of D-optimal design are found for a specific NLME model. In this example, some of those results are reproduced using the doptim package. The regression function is motivated by a one-compartment model:

$$\eta(\boldsymbol{\beta}, t) = \frac{\exp(\beta_1 - \beta_3)}{\exp(\beta_1) - \exp(\beta_2)} \{\exp(-\exp(\beta_2)t) - \exp(-\exp(\beta_1)t)\} \; .$$

Note that the model is parametrised so as to ensure positivity of rate constants ($\exp(\beta_1)$ and $\exp(\beta_2)$) and volume of distribution ($\exp(\beta_3)$).

4.1 D_β-optimal design for a single individual

```
args0 <- list(
  eta = expression(
    exp((beta2 + b2) - (beta3 + b3))/
      (exp(beta2+b2)-exp(beta1+b1)) *
      (exp(-exp(beta1+b1)*t) - exp(-exp(beta2 + b2)*t))
  ),
  Ebeta = c(0, 1, 1),
  Vb = c(0.05, 0.05, 0.1),
  Veps = 0.05^2,
  phi = "fixed",
  desvarNames = "t",
  noSamples = c(5),
  reduced = TRUE
)

objFun <- do.call(genObjFun, args0)

Xi0 <- seq(0, 4, length.out = 5)
nlminb(start = Xi0, objective = objFun, lower = 0, upper = 4)

# returns #

$‘par‘
[1] 0.1961514 0.1961512 0.9437235 2.5528683 2.5528691

$objective
[1] -3.170249

$convergence
[1] 0
```

```
$iterations
[1] 16

$evaluations
function gradient
19       107

$message
[1] "relative convergence (4)"
```

The resulting design, along with the value of the objective function, correspond exactly with the first row of Table 1 in Wang et al. (2012). Note the argument $\mathsf{reduced} = \mathsf{TRUE}$ in the code; this indicates that in the approximation of the FIM, the "trace term" should be set to zero. More specifically, recall from Section **??** that the linearised model had FIM

$$\mathsf{FIM}_{lin}(\bar{\mathbf{b}})_{kl} = \sum_{i=1}^{n} \left[\frac{\partial E_i^\top}{\partial \beta_k} V_i^{-1} \frac{\partial E_i}{\partial \beta_l} + \frac{1}{2} \mathsf{trace}\left(\frac{\partial V_i}{\partial \beta_l} V_i^{-1} \frac{\partial V_i}{\partial \beta_k} V_i^{-1} \right) \right].$$

The "reduced" version omits the second term of this approximation. In Wang et al. (2012), this is recommended for cases where the variance component is deemed "small".

4.2 Composite D-optimal design

In Wang et al. (2012), the *composite D-optimality criterion* is introduced, which allows the user to express a weighted interest in the fixed effects and variance components, respectively. Specifically, for a chosen $\lambda \in [0, 1]$, the composite D-optimality criterion is to find the design $\xi^* \in \Xi$ which maximizes

$$\lambda \cdot \log\det \mathsf{FIM}(\xi^*)_\beta + (1-\lambda) \cdot \log\det \mathsf{FIM}(\xi^*)_{\omega,\sigma^2}$$

The following code replicates a result from Wang et al. (2012), second row of Table 2. For the same model as before, the composite D-optimality objective function is created, with $\lambda = 0.5$, and the optimal design is identified:

```
args <- args0
args$returnFIM <- TRUE
objFun <- do.call(genObjFun, args)

f1 <- function(x) - log(det(objFun(x)[[1]][1:3, 1:3]))
f2 <- function(x) - log(det(objFun(x)[[1]][-(1:3), -(1:3)]))
f <- function(x) .5 * f1(x) + .5*f2(x)
```

```
Xi0 <-  seq(0, 4, length.out = 5)
nlminb(start = Xi0, objective = f, lower = 0, upper = 4)[c("par", "objective")]

# returns #

$‘par‘
[1] 0.2331545 0.2331545 1.0261762 2.1961446 2.1961435

$objective
[1] -11.75895
```

The second line of code adds a new argument, returnFIM, which is set to TRUE. This changes the return value of the generated function from the log-determinant of the FIM to simply the FIM approximation itself. The remaining lines of code directly specify the composite D-optimality objective function, followed by the optimisation step.

To replicate the first row of Table 2 in Wang et al. (2012), where $\lambda = 0$, simply run

```
nlminb(start = Xi0, objective = f2, lower = 0, upper = 4)[c("par", "objective")]

# returns #

$‘par‘
[1] 0.2748855 0.2748866 1.6424521 1.6424451 1.6424473

$objective
[1] -20.88068
```

4.3 Population D-optimal design

The genObjFun can also handle population D-optimal designs. In this case, the design problem involves several individuals who may be allocated to different elementary designs. In the code below, the args list is modified to specify that two individuals are to be allocated to two different elementary designs, each with three sampling times.

```
args <- args0
args$noSamples <- c(3,3)
objFun <- do.call(genObjFun, args)
Xi0 <- c(1:3, 1:3)
nlminb(start = Xi0, objective = objFun, lower = 0, upper = 4)[c("par", "objective")]

# returns #

$‘par‘
```

```
[1] 0.2192545 0.9634978 2.4872472 0.2192544 0.9634977 2.4872472

$objective
[1] -4.297543
```

Note that the noSamples argument is used to indicate the number of individuals (by the length of the vector provided, i.e., two) and the sizes of the elementary designs (by the integer valued entries of the vector provided, i.e., both three). The first three values of the optimisation result constitute the first elementary design of the population D-optimal design and the last three values constitute the second elementary design. This example replicates the first row of Table 3 in Wang et al. (2012). They note that while the two elementary designs are identical in the optimal design for this problem, that does not generally have to be the case.

4.4 Population D-optimal design, unbalanced

This case is an extension of the previous case with multiple individuals. Here, three groups of different sizes (with group sizes chosen by the user) are allocated to elementary designs of different lengths (with the design lengths chosen by the user). Specifically, the first group consists of 30 individuals allocated to a design of length three, the second group of 4 individuals allocated to a design of length two and the last group of 2 individuals allocated to a single point design. The design problem is now to optimally choose the six design points that make up the three elementary designs. Specifically, a D_β-optimal design is sought and the trace term is omitted (so reduced = TRUE).

```
args <- args0
args$returnFIM <- TRUE

args1 <- args
args1$noSamples <- 3
f1 <- function(Xi) do.call(genObjFun, args1)(Xi)[[1]][1:3, 1:3]

args2 <- args
args2$noSamples <- 2
f2 <- function(Xi) do.call(genObjFun, args2)(Xi)[[1]][1:3, 1:3]

args3 <- args
args3$noSamples <- 1
f3 <- function(Xi) do.call(genObjFun, args3)(Xi)[[1]][1:3, 1:3]


f <- function(Xi) -log(det(
  30*f1(Xi[1:3]) + 4*f2(Xi[4:5]) + 2*f3(Xi[6])
  ))
```

```
case <- DEoptim::DEoptim(f, lower = rep(0, 6), upper = rep(4, 6))

... output from DEoptim ...

nlminb(
  start = case$optim$bestmem,
  objective = f,
  lower = 0,
  upper = 4)[c("par", "objective")]

# returns #

$`par`
par1      par2      par3      par4      par5      par6
0.2170318 0.9784380 2.4956097 2.4309390 0.7556319 0.2550840

$objective
[1] -12.74138
```

NB: the reason we need to define args1, args2 and args3 is lazy evaluation in R; if we instead kept updating args, the resulting functions would *all* use the last definition of args.

The above results replicate row one of Table 4 in Wang et al. (2012). With this example, the basic capabilities of the doptim package have been covered. In particular, it is worth noting that because the user is able to easily construct objective functions, the package provides a great deal of flexibility: for instance, it allows the user to expand or restrict the design space, customise the optimisation method and create composite objective functions.

5 Overview of randon package

The sensitivity plots and design graphs in Chapter 2 were created using the randon package. The package makes three main functions available to the user:

calc_aPMS	Compute a first order approximation of aPMS for a given nonlinear regression function, parameter and interval. The aPMS is an attempt to measure the average deviation in the regression function over the interval (in NLME terms, the population level profile), caused by a perturbation in a single fixed effects parameter. It can used as a starting point to find realistic priors for marginal variances of (additive) random efffects.
plot_sensBands	Plot regression function with aPMS sensitivity bands
genOptimDesnRange	A wrapper to generate a range of optimal designs for given model, parameter transformation and range of variance levels.

Of these functions, only plot_sensBands and genOptimDesnRange are envisioned as part of the workflow suggested in this thesis; the calc_aPMS function is used by the other two functions in randon to translate between variance level and magnitude of variance components.

5.1 plot_sensBands

This function generates plots of the regression function, with added sensitivity bands, as seen in REF(Figure 2.1 of submitted thesis). The following code partially recreates the plot in the first panel of the Figure, i.e., 10% sensitivity bands for the one-compartment model.

```
m <- list(
  eta = expression(
    (exp(beta2) / exp(beta3)) /
      (exp(beta2) - exp(beta1)) *
      (exp( - exp(beta1) * t) - exp( - exp(beta2) * t))
  ),
  beta = list(beta1 = 0, beta2 = 1, beta3 = 1),
  tInt = c(0, 4)
)

plot_sensBands(model = m, VLrange_variedParam = .1)
```

The model argument must be a list with three named elements

- eta is an expression which defines the regression function, but *without* any random effects specified. The parameters must be named beta1, beta2, etc. The covariate must be named t.
- beta is a list with entry names matching the corresponding fixed effects parameters. The entries are scalars setting the values of the parameters.

- tInt is a vector of length two, defining the (time) interval which the covariate t is restricted to for the purposes of the sensitivity plot.

By supplying the VLrange_variedParam argument, the user specifies which variance levels she wishes to visualise for the parameter being varied. Per default, the parameter being varied is "beta1", because it always exists. If the user wishes to visualise several variance levels in the same plot, she can supply a vector-valued VLrange_variedParam argument, as follows.

```
    plot_sensBands(model = m, VLrange_variedParam = c(.1, .2, .3))
```

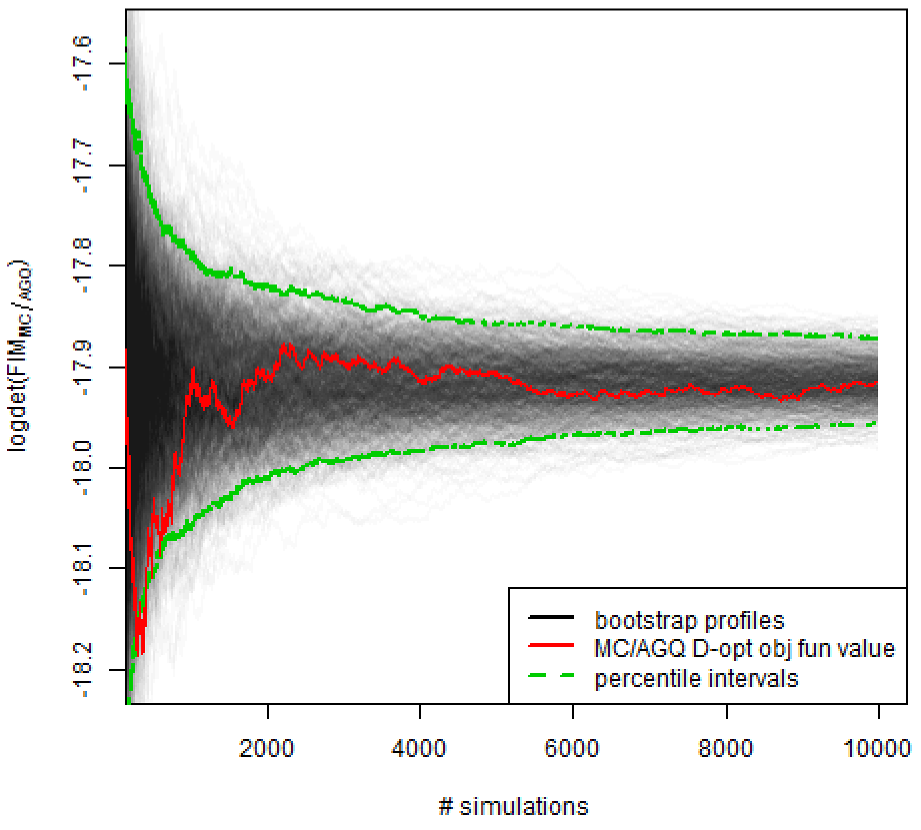
This code recreates the first panel of Figure 2.1. Lastly, to fully recreate Figure 2.1, one can supply a vector of parameter names to the function.

```
plot_sensBands(model = m,
               VLrange_variedParam = c(.1, .2, .3),
               paramName_variedParam = c("beta1", "beta2","beta3"))
```

This generates three plots, one per entry in the vector c("beta1", "beta2","beta3") supplied to paramName_variedParam, each with three sensitivity bands, one per entry in the vector c(.1, .2, .3) supplied to VLrange_variedParam.

5.2 genOptimDesnRange

This function was used to generate the design graphs and efficiency profiles in REF(Chapter 2 in submitted thesis). C:/Users/ketil/OneDrive/Documents/research/Research/Reports



References

A. C. Atkinson, A. N. Donev, and R. D. Tobias. *Optimum Experimental Designs, with SAS*. Oxford University Press, 2007.

C. Kitsos and K. Kolovos. A compilation of the D-optimal designs in chemical kinetics. *Chemical Engineering Communications*, 200:185–204, 2013.

Y. Wang, K. M. Eskridge, and S. Nadarajah. Optimal design of mixed-effects PK/PD models based on differential equations. *Journal of Biopharmaceutical Statistics*, 22:180–205, 2012.