

# Package ‘PopED’

May 7, 2026

**Type** Package

**Title** Population (and Individual) Optimal Experimental Design

**Version** 0.7.0

**Depends** R (>= 2.14)

**Imports** ggplot2, MASS, mvtnorm, dplyr (>= 0.7.0), codetools, stats, utils, magrittr, boot, purrr, stringr, tibble, gtools

**Suggests** testthat, Hmisc, nlme, GA, deSolve, Rcpp, shiny, rhandsontable, knitr, rmarkdown, gridExtra, covr, devtools, mrgsolve

**Description** Optimal experimental designs for both population and individual studies based on nonlinear mixed-effect models. Often this is based on a computation of the Fisher Information Matrix. This package was developed for pharmacometric problems, and examples and predefined models are available for these types of systems. The methods are described in Nyberg et al. (2012) <[doi:10.1016/j.cmpb.2012.05.005](https://doi.org/10.1016/j.cmpb.2012.05.005)>, and Foracchia et al. (2004) <[doi:10.1016/S0169-2607\(03\)00073-7](https://doi.org/10.1016/S0169-2607(03)00073-7)>.

**License** LGPL (>= 3)

**ByteCompile** true

**URL** <https://andrewhooker.github.io/PopED/>,  
<https://github.com/andrewhooker/PopED>

**BugReports** <https://github.com/andrewhooker/PopED/issues>

**Copyright** 2014-2021 Andrew C. Hooker

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**VignetteBuilder** knitr

**Config/Needs/website** mrgsolve, kableExtra, PKPDSim, rxode2

**NeedsCompilation** no

**Author** Andrew C. Hooker [aut, cre, trl, cph] (ORCID:  
<<https://orcid.org/0000-0002-2676-5912>>),  
Marco Foracchia [aut] (O-Matrix version),

Eric Stroemberg [ctb] (MATLAB version),  
 Martin Fink [ctb] (Streamlining code, added functionality, vignettes),  
 Giulia Lestini [ctb] (Streamlining code, added functionality,  
 vignettes),  
 Sebastian Ueckert [aut] (MATLAB version, ORCID:  
<https://orcid.org/0000-0002-3712-0255>),  
 Joakim Nyberg [aut] (MATLAB version)

**Maintainer** Andrew C. Hooker <andrew.hooker@farmaci.uu.se>

**Repository** CRAN

**Date/Publication** 2024-10-07 19:30:02 UTC

## Contents

a_line_search . . . . .	3
build_sfg . . . . .	6
calc_ofv_and_fim . . . . .	7
cell . . . . .	11
create.poped.database . . . . .	12
create_design . . . . .	23
create_design_space . . . . .	25
design_summary . . . . .	29
efficiency . . . . .	29
evaluate.e.ofv.fim . . . . .	31
evaluate.fim . . . . .	35
evaluate_design . . . . .	39
evaluate_fim_map . . . . .	40
evaluate_power . . . . .	42
feps.add . . . . .	44
feps.add.prop . . . . .	46
feps.prop . . . . .	48
ff.PK.1.comp.oral.md.CL . . . . .	50
ff.PK.1.comp.oral.md.KE . . . . .	51
ff.PK.1.comp.oral.sd.CL . . . . .	53
ff.PK.1.comp.oral.sd.KE . . . . .	55
ff.PKPD.1.comp.oral.md.CL.imax . . . . .	57
ff.PKPD.1.comp.sd.CL.emax . . . . .	59
get_rse . . . . .	61
LEDOptim . . . . .	64
mc_mean . . . . .	68
median_hilow_poped . . . . .	70
model_prediction . . . . .	70
ofv_criterion . . . . .	75
ofv_fim . . . . .	77
ones . . . . .	80
optimize_groupsize . . . . .	81
optimize_n_eff . . . . .	82
optimize_n_rse . . . . .	83

optim_ARS . . . . .	85
optim_LS . . . . .	91
pargen . . . . .	95
plot_efficiency_of_windows . . . . .	98
plot_model_prediction . . . . .	101
poped_gui . . . . .	107
poped_optim . . . . .	107
poped_optimize . . . . .	115
RS_opt . . . . .	122
shrinkage . . . . .	126
size . . . . .	128
start_parallel . . . . .	129
summary.poped_optim . . . . .	130
tic . . . . .	131
toc . . . . .	132
zeros . . . . .	133
<b>Index</b>	<b>134</b>

---

a_line_search	<i>Optimize using line search</i>
---------------	-----------------------------------

---

## Description

The function performs a grid search sequentially along design variables. The grid is defined by `ls_step_size`.

## Usage

```
a_line_search(
  poped.db,
  out_file = "",
  bED = FALSE,
  diff = 0,
  fmf_initial = 0,
  dmf_initial = 0,
  opt_xt = poped.db$settings$optsw[2],
  opt_a = poped.db$settings$optsw[4],
  opt_x = poped.db$settings$optsw[3],
  opt_samps = poped.db$settings$optsw[1],
  opt_inds = poped.db$settings$optsw[5],
  ls_step_size = poped.db$settings$ls_step_size
)
```

**Arguments**

poped.db	A PopED database.
out_file	The output file to write to.
bED	If the algorithm should use E-family methods. Logical.
diff	The OFV difference that is deemed significant for changing a design. If, by changing a design variable the difference between the new and old OFV is less than diff the change is not made.
fmf_initial	The initial value of the FIM. If 0 then the FIM is calculated from poped.db.
dmf_initial	The initial value of the objective function value (OFV). If 0 then the OFV is calculated from poped.db.
opt_xt	Should the sample times be optimized?
opt_a	Should the continuous design variables be optimized?
opt_x	Should the discrete design variables be optimized?
opt_samps	Are the number of sample times per group being optimized?
opt_inds	Are the number of individuals per group being optimized?
ls_step_size	Number of grid points in the line search.

**Value**

A list containing:

fmf	The FIM.
dmf	The final value of the objective function value.
best_changed	If the algorithm has found a better design than the starting design.
xt	A matrix of sample times. Each row is a vector of sample times for a group.
x	A matrix for the discrete design variables. Each row is a group.
a	A matrix of covariates. Each row is a group.
poped.db	A PopED database.

**See Also**

Other Optimize: [Doptim\(\)](#), [LEDOptim\(\)](#), [RS\\_opt\(\)](#), [bfgsb\\_min\(\)](#), [calc Autofocus\(\)](#), [calc\\_ofv\\_and\\_grad\(\)](#), [mfea\(\)](#), [optim\\_ARS\(\)](#), [optim\\_LS\(\)](#), [poped\\_optim\(\)](#), [poped\\_optim\\_1\(\)](#), [poped\\_optim\\_2\(\)](#), [poped\\_optim\\_3\(\)](#), [poped\\_optimize\(\)](#)

**Examples**

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin model for optimization)
#####
```

```

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## Optimization using an additive + proportional residual error
## to avoid sample times at very low concentrations (time 0 or very late samples).

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(prop=0.01,add=0.25),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0.01,
                                maxx=120,
                                a=c(DOSE=70),
                                mina=c(DOSE=0.01),
                                maxa=c(DOSE=100))

##### END #####
## Create PopED database
## (warfarin model for optimization)
#####

# very sparse grid to evaluate (4 points for each design variable)
output <- a_line_search(poped.db, opt_xt=TRUE, opt_a=TRUE, ls_step_size=4)

## Not run:

# longer run time
output <- a_line_search(poped.db,opt_xt=TRUE)

# output to a text file

```

```
output <- a_line_search(poped.db,opt_xt=TRUE,out_file="tmp.txt")

## End(Not run)
```

---

 build\_sfg

*Build PopED parameter function from a model function*


---

### Description

Build PopED parameter function from a model function

### Usage

```
build_sfg(
  model = "ff.PK.1.comp.oral.sd.CL",
  covariates = c("dose", "tau"),
  par_names = NULL,
  etas = "exp",
  no_etas = c("F", "Favail"),
  env = parent.frame()
)
```

### Arguments

model	A string of text describing the model function name
covariates	A list of covariate names to be filtered out of the model
par_names	A list of parameter names in the model file. If not supplied then all undefined variables in the model file are extracted and the covariate names are filtered out of that list.
etas	Can be "exp", "prop", "add" or "none". Either one value for all parameters or a list defining the model per parameter.
no_etas	Parameters that should not have etas associated with them.
env	The environment to create the function in.

### Value

A parameter model function to be used as input to PopED calculations.

### Examples

```
build_sfg(model="ff.PK.1.comp.oral.md.CL")

etas <- c(Favail="exp",KA="exp",V="add",CL="exp")
build_sfg(model="ff.PK.1.comp.oral.md.CL",etas = etas)
```

---

calc_ofv_and_fim	<i>Calculate the Fisher Information Matrix (FIM) and the OFV(FIM) for either point values or parameters or distributions.</i>
------------------	---

---

### Description

This function computes the expectation of the FIM and OFV(FIM) for either point values of parameter estimates or parameter distributions given the model, parameters, distributions of parameter uncertainty, design and methods defined in the PopED database.

### Usage

```
calc_ofv_and_fim(
  poped.db,
  ofv = 0,
  fim = 0,
  d_switch = poped.db$settings$d_switch,
  bpopdescr = poped.db$parameters$bpop,
  ddescr = poped.db$parameters$d,
  bpop = bpopdescr[, 2, drop = F],
  d = getfullid(ddescr[, 2, drop = F], poped.db$parameters$covd),
  docc_full = getfullid(poped.db$parameters$docc[, 2, drop = F],
    poped.db$parameters$covdocc),
  model_switch = poped.db$design$model_switch,
  ni = poped.db$design$ni,
  xt = poped.db$design$xt,
  x = poped.db$design$x,
  a = poped.db$design$a,
  fim.calc.type = poped.db$settings$iFIMCalculationType,
  use_laplace = poped.db$settings$iEDCalculationType,
  laplace.fim = FALSE,
  ofv_fun = poped.db$settings$ofv_fun,
  evaluate_fim = TRUE,
  ...
)
```

### Arguments

poped.db	A PopED database.
ofv	The current ofv. If other than zero then this value is simply returned unchanged.
fim	The current FIM. If other than zero then this value is simply returned unchanged.
d_switch	<ul style="list-style-type: none"> <li>• <b>*****START OF CRITERION SPECIFICATION OPTIONS*****</b></li> </ul> D-family design (1) or ED-family design (0) (with or without parameter uncertainty)
bpopdescr	Matrix defining the fixed effects, per row (row number = parameter_number) we should have:

	<ul style="list-style-type: none"> <li>• column 1 the type of the distribution for E-family designs (0 = Fixed, 1 = Normal, 2 = Uniform, 3 = User Defined Distribution, 4 = lognormal and 5 = truncated normal)</li> <li>• column 2 defines the mean.</li> <li>• column 3 defines the variance of the distribution (or length of uniform distribution).</li> </ul>
ddescr	Matrix defining the diagonals of the IIV (same logic as for the bpopdescr).
bpop	<p>Matrix defining the fixed effects, per row (row number = parameter_number) we should have:</p> <ul style="list-style-type: none"> <li>• column 1 the type of the distribution for E-family designs (0 = Fixed, 1 = Normal, 2 = Uniform, 3 = User Defined Distribution, 4 = lognormal and 5 = truncated normal)</li> <li>• column 2 defines the mean.</li> <li>• column 3 defines the variance of the distribution (or length of uniform distribution).</li> </ul> <p>Can also just supply the parameter values as a vector <code>c()</code> if no uncertainty around the parameter value is to be used. The parameter order of 'bpop' is defined in the 'fg_fun' or 'fg_file'. If you use named arguments in 'bpop' then the order of this vector can be rearranged to match the 'fg_fun' or 'fg_file'. See 'reorder_parameter_vectors'.</p>
d	Matrix defining the diagonals of the IIV (same logic as for the fixed effects matrix bpop to define uncertainty). One can also just supply the parameter values as a <code>c()</code> . The parameter order of 'd' is defined in the 'fg_fun' or 'fg_file'. If you use named arguments in 'd' then the order of this vector can be rearranged to match the 'fg_fun' or 'fg_file'. See 'reorder_parameter_vectors'.
docc_full	A between occasion variability matrix.
model_switch	A matrix that is the same size as <code>xt</code> , specifying which model each sample belongs to.
ni	A vector of the number of samples in each group.
xt	A matrix of sample times. Each row is a vector of sample times for a group.
x	A matrix for the discrete design variables. Each row is a group.
a	A matrix of covariates. Each row is a group.
fim.calc.type	<p>The method used for calculating the FIM. Potential values:</p> <ul style="list-style-type: none"> <li>• 0 = Full FIM. No assumption that fixed and random effects are uncorrelated.</li> <li>• 1 = Reduced FIM. Assume that there is no correlation in the FIM between the fixed and random effects, and set these elements in the FIM to zero.</li> <li>• 2 = weighted models (placeholder).</li> <li>• 3 = Not currently used.</li> <li>• 4 = Reduced FIM and computing all derivatives with respect to the standard deviation of the residual unexplained variation (<math>\sqrt{\text{SIGMA}}</math> in NONMEM). This matches what is done in PFIM, and assumes that the standard deviation of the residual unexplained variation is the estimated parameter (NOTE: NONMEM estimates the variance of the residual unexplained variation by default).</li> </ul>



	<ul style="list-style-type: none"> <li>• 5 = Full FIM parameterized with A,B,C matrices &amp; derivative of variance.</li> <li>• 6 = Calculate one model switch at a time, good for large matrices.</li> <li>• 7 = Reduced FIM parameterized with A,B,C matrices &amp; derivative of variance.</li> </ul>
use_laplace	Should the Laplace method be used in calculating the expectation of the OFV?
laplace.fim	Should an E(FIM) be calculated when computing the Laplace approximated E(OFV). Typically the FIM does not need to be computed and, if desired, this calculation is done using the standard MC integration technique, so can be slow.
ofv_fun	User defined function used to compute the objective function. The function must have a popped database object as its first argument and have "." in its argument list. Can be referenced as a function or as a file name where the function defined in the file has the same name as the file. e.g. "cost.txt" has a function named "cost" in it.
evaluate_fim	Should the FIM be calculated?
...	Other arguments passed to the function.

### Value

A list containing the FIM and OFV(FIM) or the E(FIM) and E(OFV(FIM)) according to the function arguments.

### See Also

Other FIM: [LinMatrixH\(\)](#), [LinMatrixLH\(\)](#), [LinMatrixL\\_occ\(\)](#), [ed\\_laplace\\_ofv\(\)](#), [ed\\_mftot\(\)](#), [efficiency\(\)](#), [evaluate.e.ofv.fim\(\)](#), [evaluate.fim\(\)](#), [gradf\\_eps\(\)](#), [mf3\(\)](#), [mf7\(\)](#), [mftot\(\)](#), [ofv\\_criterion\(\)](#), [ofv\\_fim\(\)](#)

Other E-family: [ed\\_laplace\\_ofv\(\)](#), [ed\\_mftot\(\)](#), [evaluate.e.ofv.fim\(\)](#)

Other evaluate\_FIM: [evaluate.e.ofv.fim\(\)](#), [evaluate.fim\(\)](#), [ofv\\_fim\(\)](#)

### Examples

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin model for optimization
## with parameter uncertainty)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## Optimization using an additive + proportional residual error
## to avoid sample times at very low concentrations (time 0 or very late samples).

## find the parameters that are needed to define from the structural model
```

```

ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

# Adding 10% log-normal Uncertainty to fixed effects (not Favail)
bpop_vals <- c(CL=0.15, V=8, KA=1.0, Favail=1)
bpop_vals_ed_ln <- cbind(ones(length(bpop_vals),1)*4, # log-normal distribution
                        bpop_vals,
                        ones(length(bpop_vals),1)*(bpop_vals*0.1)^2) # 10% of bpop value
bpop_vals_ed_ln["Favail",] <- c(0,1,0)
bpop_vals_ed_ln

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                bpop=bpop_vals_ed_ln,
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(0.01,0.25),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0,
                                maxxt=120,
                                a=70,
                                mina=0,
                                maxa=100)

##### END #####
## Create PopED database
## (warfarin model for optimization
## with parameter uncertainty)
#####

calc_ofv_and_fim(poped.db)

## Not run:

calc_ofv_and_fim(poped.db,d_switch=0)
calc_ofv_and_fim(poped.db,d_switch=0,use_laplace=TRUE)
calc_ofv_and_fim(poped.db,d_switch=0,use_laplace=TRUE,laplace.fim=TRUE)

```

```
## End(Not run)
```

---

cell	<i>Create a cell array (a matrix of lists)</i>
------	--

---

### Description

Create a cell array as in MATLAB.

### Usage

```
cell(...)
```

### Arguments

...                   Dimensions for the cell array.

### Value

A list of empty lists.

### Note

This is a modified version of the same function in the matlab R-package.

### See Also

Other MATLAB: [diag\\_matlab\(\)](#), [feval\(\)](#), [fileparts\(\)](#), [isempty\(\)](#), [ones\(\)](#), [rand\(\)](#), [randn\(\)](#), [size\(\)](#), [tic\(\)](#), [toc\(\)](#), [zeros\(\)](#)

### Examples

```
cell(3)
cell(2,3)

## define possible values of 2 categorical design variable
x.space <- cell(1,2)
x.space[1,1] <- list(seq(10,100,10))
x.space[1,2] <- list(seq(10,300,10))
x.space
x.space[1,1]
x.space[1,2]
```

---

create.poped.database *Create a PopED database*

---

## Description

This function takes the input file (a previously created poped database) supplied by the user, or function arguments, and creates a database that can then be used to run all other PopED functions. The function supplies default values to elements of the database that are not specified in the input file or as function arguments. Default arguments are supplied in the Usage section (easiest to use a text search to find values you are interested in).

## Usage

```
create.poped.database(
  popedInput = list(),
  ff_file = NULL,
  ff_fun = poped.choose(popedInput$model$ff_pointer, NULL),
  fg_file = NULL,
  fg_fun = poped.choose(popedInput$model$fg_pointer, NULL),
  fError_file = NULL,
  fError_fun = poped.choose(popedInput$model$ferror_pointer, NULL),
  optsw = poped.choose(popedInput$settings$optsw, cbind(0, 0, 0, 0, 0)),
  xt = poped.choose(popedInput$design[["xt"]], stop("'xt' needs to be defined")),
  m = poped.choose(popedInput$design[["m"]], NULL),
  x = poped.choose(popedInput$design[["x"]], NULL),
  nx = poped.choose(popedInput$design$nx, NULL),
  a = poped.choose(popedInput$design[["a"]], NULL),
  groupsize = poped.choose(popedInput$design$groupsize,
    stop("'groupsize' needs to be defined")),
  ni = poped.choose(popedInput$design$ni, NULL),
  model_switch = poped.choose(popedInput$design$model_switch, NULL),
  maxni = poped.choose(popedInput$design_space$maxni, NULL),
  minni = poped.choose(popedInput$design_space$minni, NULL),
  maxtotni = poped.choose(popedInput$design_space$maxtotni, NULL),
  mintotni = poped.choose(popedInput$design_space$mintotni, NULL),
  maxgroupsize = poped.choose(popedInput$design_space$maxgroupsize, NULL),
  mingroupsize = poped.choose(popedInput$design_space$mingroupsize, NULL),
  maxtotgroupsize = poped.choose(popedInput$design_space$maxtotgroupsize, NULL),
  mintotgroupsize = poped.choose(popedInput$design_space$mintotgroupsize, NULL),
  maxxt = poped.choose(popedInput$design_space$maxxt, NULL),
  minxt = poped.choose(popedInput$design_space$minxt, NULL),
  discrete_xt = poped.choose(popedInput$design_space$xt_space, NULL),
  discrete_x = poped.choose(popedInput$design_space$discrete_x, NULL),
  maxa = poped.choose(popedInput$design_space$maxa, NULL),
  mina = poped.choose(popedInput$design_space$mina, NULL),
  discrete_a = poped.choose(popedInput$design_space$a_space, NULL),
  bUseGrouped_xt = poped.choose(popedInput$design_space$bUseGrouped_xt, FALSE),
```

```

G_xt = poped.choose(popedInput$design_space$G_xt, NULL),
bUseGrouped_a = poped.choose(popedInput$design_space$bUseGrouped_a, FALSE),
G_a = poped.choose(popedInput$design_space$G_a, NULL),
bUseGrouped_x = poped.choose(popedInput$design_space$bUseGrouped_x, FALSE),
G_x = poped.choose(popedInput$design_space[["G_x"]], NULL),
iFIMCalculationType = poped.choose(popedInput$settings$iFIMCalculationType, 1),
iApproximationMethod = poped.choose(popedInput$settings$iApproximationMethod, 0),
iFOCENumInd = poped.choose(popedInput$settings$iFOCENumInd, 1000),
prior_fim = poped.choose(popedInput$settings$prior_fim, matrix(0, 0, 1)),
strAutoCorrelationFile = poped.choose(popedInput$model$auto_pointer, ""),
d_switch = poped.choose(popedInput$settings$d_switch, 1),
ofv_calc_type = poped.choose(popedInput$settings$ofv_calc_type, 4),
ds_index = popedInput$parameters$ds_index,
strEDPenaltyFile = poped.choose(popedInput$settings$strEDPenaltyFile, ""),
ofv_fun = poped.choose(popedInput$settings$ofv_fun, NULL),
iEDCalculationType = poped.choose(popedInput$settings$iEDCalculationType, 0),
ED_samp_size = poped.choose(popedInput$settings$ED_samp_size, 45),
bLHS = poped.choose(popedInput$settings$bLHS, 1),
strUserDistributionFile = poped.choose(popedInput$model$user_distribution_pointer, ""),
nbpop = popedInput$parameters$nbpop,
NumRanEff = popedInput$parameters$NumRanEff,
NumDocc = popedInput$parameters$NumDocc,
NumOcc = popedInput$parameters$NumOcc,
bpop = poped.choose(popedInput$parameters$bpop, stop("bpop must be defined")),
d = poped.choose(popedInput$parameters$d, NULL),
covd = popedInput$parameters$covd,
sigma = popedInput$parameters$sigma,
docc = poped.choose(popedInput$parameters$docc, matrix(0, 0, 3)),
covdocc = poped.choose(popedInput$parameters$covdocc, zeros(1, length(docc[, 2, drop =
  F]) * (length(docc[, 2, drop = F]) - 1)/2)),
notfixed_bpop = popedInput$parameters$notfixed_bpop,
notfixed_d = popedInput$parameters$notfixed_d,
notfixed_covd = popedInput$parameters$notfixed_covd,
notfixed_docc = popedInput$parameters$notfixed_docc,
notfixed_covdocc = poped.choose(popedInput$parameters$notfixed_covdocc, zeros(1,
  length(covdocc))),
notfixed_sigma = poped.choose(popedInput$parameters$notfixed_sigma, t(rep(1,
  size(sigma, 2)))),
notfixed_covsigma = poped.choose(popedInput$parameters$notfixed_covsigma, zeros(1,
  length(notfixed_sigma) * (length(notfixed_sigma) - 1)/2)),
reorder_parameter_vectors = FALSE,
bUseRandomSearch = poped.choose(popedInput$settings$bUseRandomSearch, TRUE),
bUseStochasticGradient = poped.choose(popedInput$settings$bUseStochasticGradient, TRUE),
bUseLineSearch = poped.choose(popedInput$settings$bUseLineSearch, TRUE),
bUseExchangeAlgorithm = poped.choose(popedInput$settings$bUseExchangeAlgorithm, FALSE),
bUseBFGSMinimizer = poped.choose(popedInput$settings$bUseBFGSMinimizer, FALSE),
EACriteria = poped.choose(popedInput$settings$EACriteria, 1),
strRunFile = poped.choose(popedInput$settings$run_file_pointer, ""),

```

```

poped_version = poped.choose(popedInput$settings$poped_version,
  packageVersion("PopED")),
modtit = poped.choose(popedInput$settings$modtit, "PopED model"),
output_file = poped.choose(popedInput$settings$output_file, paste("PopED_output",
  "_summary", sep = "")),
output_function_file = poped.choose(popedInput$settings$output_function_file,
  paste("PopED", "_output_", sep = "")),
strIterationFileName = poped.choose(popedInput$settings$strIterationFileName,
  paste("PopED", "_current.R", sep = "")),
user_data = poped.choose(popedInput$settings$user_data, cell(0, 0)),
ourzero = poped.choose(popedInput$settings$ourzero, 1e-05),
dSeed = poped.choose(popedInput$settings$dSeed, NULL),
line_opta = poped.choose(popedInput$settings$line_opta, NULL),
line_optx = poped.choose(popedInput$settings$line_optx, NULL),
bShowGraphs = poped.choose(popedInput$settings$bShowGraphs, FALSE),
use_logfile = poped.choose(popedInput$settings$use_logfile, FALSE),
m1_switch = poped.choose(popedInput$settings$m1_switch, 1),
m2_switch = poped.choose(popedInput$settings$m2_switch, 1),
hle_switch = poped.choose(popedInput$settings$hle_switch, 1),
gradff_switch = poped.choose(popedInput$settings$gradff_switch, 1),
gradfg_switch = poped.choose(popedInput$settings$gradfg_switch, 1),
grad_all_switch = poped.choose(popedInput$settings$grad_all_switch, 1),
rsit_output = poped.choose(popedInput$settings$rsit_output, 5),
sgit_output = poped.choose(popedInput$settings$sgit_output, 1),
hm1 = poped.choose(popedInput$settings[["hm1"]], 1e-05),
hlf = poped.choose(popedInput$settings[["hlf"]], 1e-05),
hlg = poped.choose(popedInput$settings[["hlg"]], 1e-05),
hm2 = poped.choose(popedInput$settings[["hm2"]], 1e-05),
hgd = poped.choose(popedInput$settings[["hgd"]], 1e-05),
hle = poped.choose(popedInput$settings[["hle"]], 1e-05),
AbsTol = poped.choose(popedInput$settings$AbsTol, 1e-06),
RelTol = poped.choose(popedInput$settings$RelTol, 1e-06),
iDiffSolverMethod = poped.choose(popedInput$settings$iDiffSolverMethod, NULL),
bUseMemorySolver = poped.choose(popedInput$settings$bUseMemorySolver, FALSE),
rsit = poped.choose(popedInput$settings[["rsit"]], 300),
sgit = poped.choose(popedInput$settings[["sgit"]], 150),
inrsit = poped.choose(popedInput$settings$inrsit, 250),
intsgit = poped.choose(popedInput$settings$intsgit, 50),
maxrsnullit = poped.choose(popedInput$settings$maxrsnullit, 50),
convergence_eps = poped.choose(popedInput$settings$convergence_eps, 1e-08),
rslxt = poped.choose(popedInput$settings$rslxt, 10),
rsla = poped.choose(popedInput$settings$rsla, 10),
cfaxt = poped.choose(popedInput$settings$cfaxt, 0.001),
cfaa = poped.choose(popedInput$settings$cfaa, 0.001),
bGreedyGroupOpt = poped.choose(popedInput$settings$bGreedyGroupOpt, FALSE),
EAStepSize = poped.choose(popedInput$settings$EAStepSize, 0.01),
EANumPoints = poped.choose(popedInput$settings$EANumPoints, FALSE),
EAConvergenceCriteria = poped.choose(popedInput$settings$EAConvergenceCriteria, 1e-20),

```

```

bEANOreplicates = poped.choose(popedInput$settings$bEANOreplicates, FALSE),
BFGSConvergenceCriteriaMinStep = NULL,
BFGSProjectedGradientTol = poped.choose(popedInput$settings$BFGSProjectedGradientTol,
1e-04),
BFGSTolerancef = poped.choose(popedInput$settings$BFGSTolerancef, 0.001),
BFGSToleranceg = poped.choose(popedInput$settings$BFGSToleranceg, 0.9),
BFGSTolerancex = poped.choose(popedInput$settings$BFGSTolerancex, 0.1),
ED_diff_it = poped.choose(popedInput$settings$ED_diff_it, 30),
ED_diff_percent = poped.choose(popedInput$settings$ED_diff_percent, 10),
line_search_it = poped.choose(popedInput$settings$ls_step_size, 50),
Doptim_iter = poped.choose(popedInput$settings$iNumSearchIterationsIfNotLineSearch, 1),
iCompileOption = poped.choose(popedInput$settings$parallel$iCompileOption, -1),
iUseParallelMethod = poped.choose(popedInput$settings$parallel$iUseParallelMethod, 1),
MCC_Dep = NULL,
strExecuteName = poped.choose(popedInput$settings$parallel$strExecuteName,
"calc_fim.exe"),
iNumProcesses = poped.choose(popedInput$settings$parallel$iNumProcesses, 2),
iNumChunkDesignEvals = poped.choose(popedInput$settings$parallel$iNumChunkDesignEvals,
-2),
Mat_Out_Pre = poped.choose(popedInput$settings$parallel$strMatFileOutputPrefix,
"parallel_output"),
strExtraRunOptions = poped.choose(popedInput$settings$parallel$strExtraRunOptions, ""),
dPollResultTime = poped.choose(popedInput$settings$parallel$dPollResultTime, 0.1),
strFunctionInputName = poped.choose(popedInput$settings$parallel$strFunctionInputName,
"function_input"),
bParallelRS = poped.choose(popedInput$settings$parallel$bParallelRS, FALSE),
bParallelSG = poped.choose(popedInput$settings$parallel$bParallelSG, FALSE),
bParallelMFEA = poped.choose(popedInput$settings$parallel$bParallelMFEA, FALSE),
bParallelLS = poped.choose(popedInput$settings$parallel$bParallelLS, FALSE)
)

```

## Arguments

popedInput	A PopED database file or an empty list list(). List elements should match the values seen in the Usage section (the defaults to function arguments).
ff_file	<ul style="list-style-type: none"> <li>• <b>*****START OF MODEL DEFINITION OPTIONS*****</b></li> </ul> A string giving the function name or filename and path of the structural model. The filename and the function name must be the same if giving a filename. e.g. "ff.PK.1.comp.oral.md.KE"
ff_fun	Function describing the structural model. e.g. ff.PK.1.comp.oral.md.KE.
fg_file	A string giving the function name or filename and path of the parameter model. The filename and the function name must be the same if giving a filename. e.g. "parameter.model"
fg_fun	Function describing the parameter model. e.g. parameter.model.
fError_file	A string giving the function name or filename and path of the residual error model. The filename and the function name must be the same if giving a filename. e.g. "feps.prop".

fError_fun	Function describing the residual error model. e.g. feps.prop.
optsw	<ul style="list-style-type: none"> <li>• <b>*****WHAT TO OPTIMIZE*****</b></li> </ul> <p>Row vector of optimization tasks (1=TRUE,0=FALSE) in the following order: (Samples per subject, Sampling schedule, Discrete design variable, Continuous design variable, Number of id per group). All elements set to zero =&gt; only calculate the FIM with current design</p>
xt	<ul style="list-style-type: none"> <li>• <b>*****START OF INITIAL DESIGN OPTIONS*****</b></li> </ul> <p>Matrix defining the initial sampling schedule. Each row is a group/individual. If only one vector is supplied, e.g. c(1,2,3,4), then all groups will have the same initial design.</p>
m	Number of groups in the study. Each individual in a group will have the same design.
x	A matrix defining the initial discrete values for the model Each row is a group/individual.
nx	Number of discrete design variables.
a	Matrix defining the initial continuous covariate values. n_rows=number of groups, n_cols=number of covariates. If the number of rows is one and the number of groups > 1 then all groups are assigned the same values.
groupsize	Vector defining the size of the different groups (num individuals in each group). If only one number then the number will be the same in every group.
ni	Vector defining the number of samples for each group.
model_switch	Matrix defining which response a certain sampling time belongs to.
maxni	<ul style="list-style-type: none"> <li>• <b>*****START OF DESIGN SPACE OPTIONS*****</b></li> </ul> <p>Max number of samples per group/individual</p>
minni	Min number of samples per group/individual
maxtotni	Number defining the maximum number of samples allowed in the experiment.
mintotni	Number defining the minimum number of samples allowed in the experiment.
maxgroupsize	Vector defining the max size of the different groups (max number of individuals in each group)
mingroupsize	Vector defining the min size of the different groups (min num individuals in each group) –
maxtotgroupsize	The total maximal groupsize over all groups
mintotgroupsize	The total minimal groupsize over all groups
maxxt	Matrix or single value defining the maximum value for each xt sample. If a single value is supplied then all xt values are given the same maximum value.
minxt	Matrix or single value defining the minimum value for each xt sample. If a single value is supplied then all xt values are given the same minimum value
discrete_xt	Cell array <code>cell</code> defining the discrete variables allowed for each xt value. Can also be a list of values <code>list(1:10)</code> (same values allowed for all xt), or a list of lists <code>list(1:10, 2:23, 4:6)</code> (one for each value in xt). See examples in <a href="#">create_design_space</a> .



discrete_x	Cell array defining the discrete variables for each x value. See examples in <a href="#">create_design_space</a> .
maxa	Vector defining the max value for each covariate. If a single value is supplied then all a values are given the same max value
mina	Vector defining the min value for each covariate. If a single value is supplied then all a values are given the same max value
discrete_a	Cell array <code>cell</code> defining the discrete variables allowed for each a value. Can also be a list of values <code>list(1:10)</code> (same values allowed for all a), or a list of lists <code>list(1:10, 2:23, 4:6)</code> (one for each value in a). See examples in <a href="#">create_design_space</a> .
bUseGrouped_xt	Use grouped time points (1=TRUE, 0=FALSE).
G_xt	Matrix defining the grouping of sample points. Matching integers mean that the points are matched.
bUseGrouped_a	Use grouped covariates (1=TRUE, 0=FALSE)
G_a	Matrix defining the grouping of covariates. Matching integers mean that the points are matched.
bUseGrouped_x	Use grouped discrete design variables (1=TRUE, 0=FALSE).
G_x	Matrix defining the grouping of discrete design variables. Matching integers mean that the points are matched.
iFIMCalculationType	<ul style="list-style-type: none"> <li>• <b>*****START OF FIM CALCULATION OPTIONS*****</b></li> </ul> Fisher Information Matrix type <ul style="list-style-type: none"> <li>• 0=Full FIM</li> <li>• 1=Reduced FIM</li> <li>• 2=weighted models</li> <li>• 3=Loc models</li> <li>• 4=reduced FIM with derivative of SD of sigma as in PFIM</li> <li>• 5=FULL FIM parameterized with A,B,C matrices &amp; derivative of variance</li> <li>• 6=Calculate one model switch at a time, good for large matrices</li> <li>• 7=Reduced FIM parameterized with A,B,C matrices &amp; derivative of variance</li> </ul>
iApproximationMethod	Approximation method for model, 0=FO, 1=FOCE, 2=FOCEI, 3=FOI
iFOCNumInd	Num individuals in each step of FOCE
prior_fim	The prior FIM (added to calculated FIM)
strAutoCorrelationFile	Filename and path, or function name, for the Autocorrelation function, empty string means no autocorrelation.
d_switch	<ul style="list-style-type: none"> <li>• <b>*****START OF CRITERION SPECIFICATION OPTIONS*****</b></li> </ul> D-family design (1) or ED-family design (0) (with or without parameter uncertainty)
ofv_calc_type	OFV calculation type for FIM

	<ul style="list-style-type: none"> <li>• 1 = "D-optimality". Determinant of the FIM: <math>\det(\text{FIM})</math></li> <li>• 2 = "A-optimality". Inverse of the sum of the expected parameter variances: <math>1/\text{trace\_matrix}(\text{inv}(\text{FIM}))</math></li> <li>• 4 = "lnD-optimality". Natural logarithm of the determinant of the FIM: <math>\log(\det(\text{FIM}))</math></li> <li>• 6 = "Ds-optimality". Ratio of the Determinant of the FIM and the Determinant of the uninteresting rows and columns of the FIM: <math>\det(\text{FIM})/\det(\text{FIM\_u})</math></li> <li>• 7 = Inverse of the sum of the expected parameter RSE: <math>1/\text{sum}(\text{get\_rse}(\text{FIM}, \text{poped.db}, \text{use\_percent}=\text{FA}))</math></li> </ul>
ds_index	Ds_index is a vector set to 1 if a parameter is uninteresting, otherwise 0. size=(1,num unfixed parameters). First unfixed bpop, then unfixed d, then unfixed docc and last unfixed sigma. Default is the fixed effects being important, everything else not important. Used in conjunction with ofv_calc_type=6.
strEDPenaltyFile	Penalty function name or path and filename, empty string means no penalty. User defined criterion can be defined this way.
ofv_fun	User defined function used to compute the objective function. The function must have a poped database object as its first argument and have "..." in its argument list. Can be referenced as a function or as a file name where the function defined in the file has the same name as the file. e.g. "cost.txt" has a function named "cost" in it.
iEDCalculationType	<ul style="list-style-type: none"> <li>• <b>*****START OF E-FAMILY CRITERION SPECIFICATION OPTIONS*****</b></li> </ul> ED Integral Calculation, 0=Monte-Carlo-Integration, 1=Laplace Approximation, 2=BFGS Laplace Approximation --
ED_samp_size	Sample size for E-family sampling
bLHS	How to sample from distributions in E-family calculations. 0=Random Sampling, 1=LatinHyperCube --
strUserDistributionFile	Filename and path, or function name, for user defined distributions for E-family designs
nbpop	<ul style="list-style-type: none"> <li>• <b>*****START OF Model parameters SPECIFICATION OPTIONS*****</b></li> </ul> Number of typical values
NumRanEff	Number of IIV parameters. Typically can be computed from other values and not supplied.
NumDocc	Number of IOV variance parameters. Typically can be computed from other values and not supplied.
NumOcc	Number of occasions. Typically can be computed from other values and not supplied.
bpop	Matrix defining the fixed effects, per row (row number = parameter_number) we should have: <ul style="list-style-type: none"> <li>• column 1 the type of the distribution for E-family designs (0 = Fixed, 1 = Normal, 2 = Uniform, 3 = User Defined Distribution, 4 = lognormal and 5 = truncated normal)</li> </ul>

- column 2 defines the mean.
- column 3 defines the variance of the distribution (or length of uniform distribution).

Can also just supply the parameter values as a vector `c()` if no uncertainty around the parameter value is to be used. The parameter order of 'bpop' is defined in the 'fg\_fun' or 'fg\_file'. If you use named arguments in 'bpop' then the order of this vector can be rearranged to match the 'fg\_fun' or 'fg\_file'. See 'reorder\_parameter\_vectors'.

d	Matrix defining the diagonals of the IIV (same logic as for the fixed effects matrix bpop to define uncertainty). One can also just supply the parameter values as a <code>c()</code> . The parameter order of 'd' is defined in the 'fg_fun' or 'fg_file'. If you use named arguments in 'd' then the order of this vector can be rearranged to match the 'fg_fun' or 'fg_file'. See 'reorder_parameter_vectors'.
covd	Column major vector defining the covariances of the IIV variances. That is, from your full IIV matrix <code>covd &lt;- IIV[lower.tri(IIV)]</code> .
sigma	Matrix defining the variances and covariances of the residual variability terms of the model. can also just supply the diagonal parameter values (variances) as a <code>c()</code> .
docc	Matrix defining the IOV, the IOV variances and the IOV distribution as for d and bpop.
covdocc	Column major vector defining the covariance of the IOV, as in covd.
notfixed_bpop	<ul style="list-style-type: none"> <li>• <b>*****START OF Model parameters fixed or not SPECIFICATION OPTIONS*****</b></li> </ul> Vector defining if a typical value is fixed or not (1=not fixed, 0=fixed). The parameter order of 'notfixed_bpop' is defined in the 'fg_fun' or 'fg_file'. If you use named arguments in 'notfixed_bpop' then the order of this vector can be rearranged to match the 'fg_fun' or 'fg_file'. See 'reorder_parameter_vectors'.
notfixed_d	Vector defining if a IIV is fixed or not (1=not fixed, 0=fixed). The parameter order of 'notfixed_d' is defined in the 'fg_fun' or 'fg_file'. If you use named arguments in 'notfixed_d' then the order of this vector can be rearranged to match the 'fg_fun' or 'fg_file'. See 'reorder_parameter_vectors'. .
notfixed_covd	Vector defining if a covariance IIV is fixed or not (1=not fixed, 0=fixed)
notfixed_docc	Vector defining if an IOV variance is fixed or not (1=not fixed, 0=fixed)
notfixed_covdocc	Vector row major order for lower triangular matrix defining if a covariance IOV is fixed or not (1=not fixed, 0=fixed)
notfixed_sigma	Vector defining if a residual error parameter is fixed or not (1=not fixed, 0=fixed)
notfixed_covsigma	Vector defining if a covariance residual error parameter is fixed or not (1=not fixed, 0=fixed). Default is fixed.
reorder_parameter_vectors	If you use named arguments in 'bpop' or 'd' then PopED will try to figure out the order of the parameters based on what is found in the 'fg_fun'. See the resulting 'poped_db\$parameters' and make sure the order matches with 'fg_fun'.

bUseRandomSearch

- **\*\*\*\*\*START OF Optimization algorithm SPECIFICATION OPTIONS\*\*\*\*\***

Use random search (1=TRUE, 0=FALSE)

bUseStochasticGradient

Use Stochastic Gradient search (1=TRUE, 0=FALSE)

bUseLineSearch Use Line search (1=TRUE, 0=FALSE)

bUseExchangeAlgorithm

Use Exchange algorithm (1=TRUE, 0=FALSE)

bUseBFGSMinimizer

Use BFGS Minimizer (1=TRUE, 0=FALSE)

EACriteria Exchange Algorithm Criteria, 1 = Modified, 2 = Fedorov

strRunFile Filename and path, or function name, for a run file that is used instead of the regular PopED call.

poped\_version

- **\*\*\*\*\*START OF Labeling and file names SPECIFICATION OPTIONS\*\*\*\*\***

The current PopED version

modtit The model title

output\_file Filename and path of the output file during search

output\_function\_file

Filename suffix of the result function file

strIterationFileName

Filename and path for storage of current optimal design

user\_data

- **\*\*\*\*\*START OF Miscellaneous SPECIFICATION OPTIONS\*\*\*\*\***

User defined data structure that, for example could be used to send in data to the model

ourzero Value to interpret as zero in design

dSeed The seed number used for optimization and sampling – integer or -1 which creates a random seed as `integer(Sys.time())` or NULL.

line\_opta Vector for line search on continuous design variables (1=TRUE,0=FALSE)

line\_optx Vector for line search on discrete design variables (1=TRUE,0=FALSE)

bShowGraphs Use graph output during search

use\_logfile If a log file should be used (0=FALSE, 1=TRUE)

m1\_switch Method used to calculate M1 (0=Complex difference, 1=Central difference, 20=Analytic derivative, 30=Automatic differentiation)

m2\_switch Method used to calculate M2 (0=Central difference, 1=Central difference, 20=Analytic derivative, 30=Automatic differentiation)

hle\_switch Method used to calculate linearization of residual error (0=Complex difference, 1=Central difference, 30=Automatic differentiation)

gradff\_switch Method used to calculate the gradient of the model (0=Complex difference, 1=Central difference, 20=Analytic derivative, 30=Automatic differentiation)

gradfg\_switch Method used to calculate the gradient of the parameter vector g (0=Complex difference, 1=Central difference, 20=Analytic derivative, 30=Automatic differentiation)

grad_all_switch	Method used to calculate all the gradients (0=Complex difference, 1=Central difference)
rsit_output	Number of iterations in random search between screen output
sgit_output	Number of iterations in stochastic gradient search between screen output
hm1	Step length of derivative of linearized model w.r.t. typical values
hlf	Step length of derivative of model w.r.t. g
hlg	Step length of derivative of g w.r.t. b
hm2	Step length of derivative of variance w.r.t. typical values
hgd	Step length of derivative of OFV w.r.t. time
hle	Step length of derivative of model w.r.t. sigma
AbsTol	The absolute tolerance for the diff equation solver
RelTol	The relative tolerance for the diff equation solver
iDiffSolverMethod	The diff equation solver method, NULL as default.
bUseMemorySolver	If the differential equation results should be stored in memory (1) or not (0)
rsit	Number of Random search iterations
sgit	Number of stochastic gradient iterations
intrsit	Number of Random search iterations with discrete optimization.
intsgit	Number of Stochastic Gradient search iterations with discrete optimization
maxrsnullit	Iterations until adaptive narrowing in random search
convergence_eps	Stochastic Gradient convergence value, (difference in OFV for D-optimal, difference in gradient for ED-optimal)
rslxt	Random search locality factor for sample times
rsla	Random search locality factor for covariates
cfaxt	Stochastic Gradient search first step factor for sample times
cfaa	Stochastic Gradient search first step factor for covariates
bGreedyGroupOpt	Use greedy algorithm for group assignment optimization
EAStepSize	Exchange Algorithm StepSize
EANumPoints	Exchange Algorithm NumPoints
EAConvergenceCriteria	Exchange Algorithm Convergence Limit/Criteria
bEAnoReplicates	Avoid replicate samples when using Exchange Algorithm
BFGSConvergenceCriteriaMinStep	BFGS Minimizer Convergence Criteria Minimum Step
BFGSProjectedGradientTol	BFGS Minimizer Convergence Criteria Normalized Projected Gradient Tolerance

BFGSTolerancef	BFGS Minimizer Line Search Tolerance f
BFGSToleranceg	BFGS Minimizer Line Search Tolerance g
BFGSTolerancex	BFGS Minimizer Line Search Tolerance x
ED_diff_it	Number of iterations in ED-optimal design to calculate convergence criteria
ED_diff_percent	ED-optimal design convergence criteria in percent
line_search_it	Number of grid points in the line search
Doptim_iter	Number of iterations of full Random search and full Stochastic Gradient if line search is not used
iCompileOption	***** <b>START OF PARALLEL OPTIONS</b> ***** Compile options for PopED <ul style="list-style-type: none"> <li>• -1 = No compilation,</li> <li>• 0 or 3 = Full compilation,</li> <li>• 1 or 4 = Only using MCC (shared lib),</li> <li>• 2 or 5 = Only MPI,</li> <li>• Option 0,1,2 runs PopED and option 3,4,5 stops after compilation</li> </ul>
iUseParallelMethod	Parallel method to use (0 = Matlab PCT, 1 = MPI)
MCC_Dep	Additional dependencies used in MCC compilation (mat-files), if several space separated
strExecuteName	Compilation output executable name
iNumProcesses	Number of processes to use when running in parallel (e.g. 3 = 2 workers, 1 job manager)
iNumChunkDesignEvals	Number of design evaluations that should be evaluated in each process before getting new work from job manager
Mat_Out_Pre	The prefix of the output mat file to communicate with the executable
strExtraRunOptions	Extra options send to e\$g. the MPI executable or a batch script, see execute_parallel\$m for more information and options
dPollResultTime	Polling time to check if the parallel execution is finished
strFunctionInputName	The file containing the popedInput structure that should be used to evaluate the designs
bParallelRS	If the random search is going to be executed in parallel
bParallelSG	If the stochastic gradient search is going to be executed in parallel
bParallelMFEA	If the modified exchange algorithm is going to be executed in parallel
bParallelLS	If the line search is going to be executed in parallel

**Value**

A PopED database

**See Also**

Other `poped_input`: [convert\\_variables\(\)](#), [create\\_design\(\)](#), [create\\_design\\_space\(\)](#), [downsizing\\_general\\_design\(\)](#), [poped.choose\(\)](#)

**Examples**

```
## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

library(PopED)

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.md.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

## -- Define initial design and design space
poped.db <- create.poped.database(
  ff_fun=ff.PK.1.comp.oral.sd.CL,
  fg_fun=sfg,
  fError_fun=feps.prop,
  bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
  notfixed_bpop=c(1,1,1,0),
  d=c(CL=0.07, V=0.02, KA=0.6),
  sigma=0.01,
  groupsize=32,
  xt=c( 0.5,1,2,6,24,36,72,120),
  minxt=0,
  maxxt=120,
  a=70)

## evaluate initial design
evaluate_design(poped.db)
```

**Description**

Create design variables to fully describe a design. If variables are supplied then these variables are checked for consistency and, if possible, changed to sizes that make sense if there are inconsistencies. Returns a list of matrices compatible with PopED.

**Usage**

```
create_design(
  xt,
  groupsize,
  m = NULL,
  x = NULL,
  a = NULL,
  ni = NULL,
  model_switch = NULL
)
```

**Arguments**

xt	Matrix defining the sampling schedule. Each row is a group.
groupsize	Vector defining the size of the different groups (number of individuals in each group).
m	A number defining the number of groups. Computed from xt if not defined.
x	A matrix defining the discrete design variables for the model Each row is a group.
a	Matrix defining the continuous design variables. Each row is a group.
ni	Vector defining the number of samples for each group, computed as all elements of xt for each group by default.
model_switch	Matrix defining which response a certain sampling time belongs to. Defaults to one for all elements of xt.

**Details**

If a value (or a vector/list of values) is supplied that corresponds to only one group and the design has multiple groups then all groups will have the same value(s). If a matrix is expected then a list of lists can be supplied instead, each list corresponding to a group.

**See Also**

Other poped\_input: [convert\\_variables\(\)](#), [create.poped.database\(\)](#), [create\\_design\\_space\(\)](#), [downsizing\\_general\\_design\(\)](#), [poped.choose\(\)](#)

**Examples**

```
library(PopED)

xt1 <- list(c(1,2,3),c(1,2,3,4))
```



```

xt4 <- list(c(1,2,3,4,5),c(1,2,3,4))
xt2 <- rbind(c(1,2,3,4),c(1,2,3,4))
xt3 <- c(1,2,3,4)

design_1 <- create_design(xt=xt1,groupsize=20)
design_2 <- create_design(xt=xt4,groupsize=20)
design_3 <- create_design(xt=xt2,groupsize=20)
design_4 <- create_design(xt=xt3,groupsize=20)

design_5 <- create_design(xt=xt3,groupsize=20,m=3)

design_6 <- create_design(xt=xt1,groupsize=20,model_switch=ones(2,4))

design_7 <-create_design(xt=xt1,groupsize=20,a=c(2,3,4))
design_8 <-create_design(xt=xt1,groupsize=20,a=rbind(c(2,3,4),c(4,5,6)))
design_9 <-create_design(xt=xt1,groupsize=20,a=list(c(2,3,4,6),c(4,5,6)))
design_10 <-create_design(xt=xt1,groupsize=20,a=list(c(2,3,4),c(4,5,6)))

design_11 <-create_design(xt=c(0,1,2,4,6,8,24),
                        groupsize=50,
                        a=c(WT=70,DOSE=1000))

design_12 <-create_design(xt=c(0,1,2,4,6,8,24),
                        groupsize=50,
                        a=c(WT=70,DOSE=1000),m=2)

design_13 <-create_design(xt=c(0,1,2,4,6,8,24),
                        groupsize=50,
                        a=list(c(WT=70,DOSE=1000),c(DOSE=90,WT=200,AGE=45)),m=2)

design_14 <-create_design(xt=c(0,1,2,4,6,8,24),
                        groupsize=50,
                        a=list(list(WT=70,DOSE=1000),list(DOSE=90,WT=200,AGE=45)),m=2)

design_15 <-create_design(xt=xt4,
                        groupsize=c(50,20),
                        a=rbind(c("DOSE"=2,"WT"=3,"AGE"=4),
                                c(4,5,6)))

```

---

create\_design\_space     *Create design variables and a design space for a full description of an optimization problem.*

---

## Description

create\_design\_space takes an initial design and arguments for a design space and creates a design and design space for design optimization. Checks the sizes of supplied design space variables and changes them to sizes that make sense if there are inconsistencies. Function arguments can use shorthand notation (single values, vectors, lists of vectors and list of list) or matrices. Returns a list of matrices compatible with PopED.

**Usage**

```

create_design_space(
  design,
  maxni = NULL,
  minni = NULL,
  maxtotni = NULL,
  mintotni = NULL,
  maxgroupsize = NULL,
  mingroupsize = NULL,
  maxtotgroupsize = NULL,
  mintotgroupsize = NULL,
  maxxt = NULL,
  minxt = NULL,
  xt_space = NULL,
  maxa = NULL,
  mina = NULL,
  a_space = NULL,
  x_space = NULL,
  use_grouped_xt = FALSE,
  grouped_xt = NULL,
  use_grouped_a = FALSE,
  grouped_a = NULL,
  use_grouped_x = FALSE,
  grouped_x = NULL,
  our_zero = NULL
)

```

**Arguments**

design	The output from a call to <a href="#">create_design</a> .
maxni	Vector defining the maximum number of samples per group.
minni	Vector defining the minimum number of samples per group.
maxtotni	Number defining the maximum number of samples allowed in the experiment.
mintotni	Number defining the minimum number of samples allowed in the experiment.
maxgroupsize	Vector defining the maximum size of the different groups (maximum number of individuals in each group)
mingroupsize	Vector defining the minimum size of the different groups (minimum num individuals in each group)
maxtotgroupsize	The total maximal groupsize over all groups
mintotgroupsize	The total minimal groupsize over all groups
maxxt	Matrix or single value defining the maximum value for each xt sample. If a single value is supplied then all xt values are given the same maximum value.
minxt	Matrix or single value defining the minimum value for each xt sample. If a single value is supplied then all xt values are given the same minimum value

xt_space	Cell array <code>cell</code> defining the discrete variables allowed for each xt value. Can also be a vector of values <code>c(1:10)</code> (same values allowed for all xt), or a list of lists <code>list(1:10, 2:23, 4:6)</code> (one for each value in xt in row major order or just for one row in xt, and all other rows will be duplicated).
maxa	Vector defining the maximum value for each covariate. IF a single value is supplied then all a values are given the same maximum value
mina	Vector defining the minimum value for each covariate. IF a single value is supplied then all a values are given the same minimum value
a_space	Cell array <code>cell</code> defining the discrete variables allowed for each a value. Can also be a list of values <code>list(1:10)</code> (same values allowed for all a), or a list of lists <code>list(1:10, 2:23, 4:6)</code> (one for each value in a).
x_space	Cell array <code>cell</code> defining the discrete variables for each x value.
use_grouped_xt	Group sampling times between groups so that each group has the same values (TRUE or FALSE).
grouped_xt	Matrix defining the grouping of sample points. Matching integers mean that the points are matched. Allows for finer control than <code>use_grouped_xt</code>
use_grouped_a	Group continuous design variables between groups so that each group has the same values (TRUE or FALSE).
grouped_a	Matrix defining the grouping of continuous design variables. Matching integers mean that the values are matched. Allows for finer control than <code>use_grouped_a</code> .
use_grouped_x	Group discrete design variables between groups so that each group has the same values (TRUE or FALSE).
grouped_x	Matrix defining the grouping of discrete design variables. Matching integers mean that the values are matched. Allows for finer control than <code>use_grouped_x</code> .
our_zero	Value to interpret as zero in design.

### Details

If a value (or a vector or a list of values) is supplied that corresponds to only one group and the design has multiple groups then all groups will have the same value(s). If a matrix is expected then a list of lists can be supplied instead, each list corresponding to a group.

### See Also

Other `poped_input`: [convert\\_variables\(\)](#), [create.poped.database\(\)](#), [create\\_design\(\)](#), [downsizing\\_general\\_design.poped.choose\(\)](#)

### Examples

```
library(PopED)

design_1 <- create_design(xt=list(c(1,2,3,4,5),
                                c(1,2,3,4)),
                        groupsize=c(50,20),
                        a=list(c(WT=70,DOSE=1000),
                               c(DOSE=1000,WT=35)))
```

```

ds_1 <- create_design_space(design_1)

ds_1_a <- create_design_space(design_1,our_zero = 1e-5)

ds_2 <- create_design_space(design_1,maxni=10,maxxt=10,minxt=0)

ds_3 <- create_design_space(design_1,maxni=10,mingroupsize=20,maxxt=10,minxt=0)

ds_4 <- create_design_space(design_1,maxa=c(100,2000))

ds_5 <- create_design_space(design_1,mina=c(10,20))

design_2 <- create_design(xt=list(c(1,2,3,4,5),
                                c(1,2,3,4)),
                        groupsize=c(50,20),
                        a=list(c(WT=70,DOSE=1000),
                               c(WT=35,DOSE=1000)),
                        x=list(c(SEX=1,DOSE_discrete=100),
                               c(SEX=2,DOSE_discrete=200)))

ds_6 <- create_design_space(design_2)

ds_7 <- create_design_space(design_2,
                            x_space=list(SEX=c(1,2),
                                          DOSE_discrete=seq(100,400,by=20)))

ds_8 <- create_design_space(design_2,
                            x_space=list(SEX=c(1,2),
                                          DOSE_discrete=seq(100,400,by=20)),
                            grouped_xt=c(1,2,3,4,5))

ds_9 <- create_design_space(design_2,
                            x_space=list(SEX=c(1,2),
                                          DOSE_discrete=seq(100,400,by=20)),
                            use_grouped_xt=TRUE)

design_3 <- create_design(xt=list(c(1,2,3,4,5),
                                c(1,2,3,4)),
                        groupsize=c(50,20),
                        a=list(c(WT=35,DOSE=1000)),
                        x=list(c(SEX=1,DOSE_discrete=100)))

ds_10 <- create_design_space(design_3,
                             x_space=list(SEX=c(1,2),DOSE_discrete=seq(100,400,by=20)),
                             use_grouped_a=TRUE)

ds_11 <- create_design_space(design_2,
                             x_space=list(SEX=c(1,2),DOSE_discrete=seq(100,400,by=20)),
                             grouped_a=list(c(1,2),c(3,2)))

ds_12 <- create_design_space(design_3,
                             x_space=list(SEX=c(1,2),DOSE_discrete=seq(100,400,by=20)),

```

```

use_grouped_x=TRUE)

ds_13 <- create_design_space(design_3,
                             x_space=list(SEX=c(1,2),DOSE_discrete=seq(100,400,by=20)),
                             grouped_x=list(c(1,2),c(3,2)))

seq_1 <- 1:10
ds_14 <- create_design_space(design_1,maxxt=10,minxt=0,
                             xt_space = list(seq_1,seq_1,seq_1,seq_1,seq_1))
ds_15 <- create_design_space(design_1,maxxt=10,minxt=0,xt_space = list(seq_1))

possible_values <- as.matrix(cbind(list(0:10),list(0:10),list(0:10),list(0:20),list(0:20)))
xt_space <- as.matrix(rbind(possible_values,possible_values))

ds_16 <- create_design_space(design_1,maxxt=10,minxt=0,xt_space = xt_space)

ds_17 <- create_design_space(design_1,a_space = list(1:100,seq(1000,100000,by=1000)))

```

---

design_summary	<i>Display a summary of output from poped_db</i>
----------------	--

---

### Description

Display a summary of output from poped\_db

### Usage

```
design_summary(poped_db, file = "", ...)
```

### Arguments

poped_db	An object returned from <code>create.poped.database</code> to summarize.
file	A file handle to write to. Default is to the R console.
...	Additional arguments. Passed to <code>blockfinal</code> .

---

efficiency	<i>Compute efficiency.</i>
------------	----------------------------

---

### Description

Efficiency calculation between two designs.

**Usage**

```

efficiency(
  ofv_init,
  ofv_final,
  poped_db,
  npar = get_fim_size(poped_db),
  ofv_calc_type = poped_db$settings$ofv_calc_type,
  ds_index = poped_db$parameters$ds_index,
  use_log = TRUE,
  ...
)

```

**Arguments**

ofv_init	An initial objective function
ofv_final	A final objective function.
poped_db	a poped database
npar	The number of parameters to use for normalization.
ofv_calc_type	OFV calculation type for FIM <ul style="list-style-type: none"> <li>• 1 = "D-optimality". Determinant of the FIM: <math>\det(\text{FIM})</math></li> <li>• 2 = "A-optimality". Inverse of the sum of the expected parameter variances: <math>1/\text{trace\_matrix}(\text{inv}(\text{FIM}))</math></li> <li>• 4 = "lnD-optimality". Natural logarithm of the determinant of the FIM: <math>\log(\det(\text{FIM}))</math></li> <li>• 6 = "Ds-optimality". Ratio of the Determinant of the FIM and the Determinant of the uninteresting rows and columns of the FIM: <math>\det(\text{FIM})/\det(\text{FIM}_u)</math></li> <li>• 7 = Inverse of the sum of the expected parameter RSE: <math>1/\text{sum}(\text{get\_rse}(\text{FIM}, \text{poped.db}, \text{use\_percent}=\text{FA}))</math></li> </ul>
ds_index	Ds_index is a vector set to 1 if a parameter is uninteresting, otherwise 0. size=(1,num unfixed parameters). First unfixed bpop, then unfixed d, then unfixed docc and last unfixed sigma. Default is the fixed effects being important, everything else not important. Used in conjunction with ofv_calc_type=6.
use_log	Are the 'ofv' arguments in the log space?
...	arguments passed to <a href="#">evaluate.fim</a> and <a href="#">ofv.fim</a> .

**Value**

The specified efficiency value depending on the ofv\_calc\_type. The attribute "description" tells you how the calculation was made `attr(return_vale, "description")`

**See Also**

Other FIM: [LinMatrixH\(\)](#), [LinMatrixLH\(\)](#), [LinMatrixL\\_occ\(\)](#), [calc\\_ofv\\_and\\_fim\(\)](#), [ed\\_laplace\\_ofv\(\)](#), [ed\\_mftot\(\)](#), [evaluate.e.ofv.fim\(\)](#), [evaluate.fim\(\)](#), [gradf\\_eps\(\)](#), [mf3\(\)](#), [mf7\(\)](#), [mftot\(\)](#), [ofv\\_criterion\(\)](#), [ofv.fim\(\)](#)

---

evaluate.e.ofv.fim	<i>Evaluate the expectation of the Fisher Information Matrix (FIM) and the expectation of the OFV(FIM).</i>
--------------------	---

---

### Description

Compute the expectation of the FIM and OFV(FIM) given the model, parameters, distributions of parameter uncertainty, design and methods defined in the PopED database. Some of the arguments coming from the PopED database can be overwritten; by default these arguments are NULL in the function, if they are supplied then they are used instead of the arguments from the PopED database.

### Usage

```
evaluate.e.ofv.fim(
  poped.db,
  fim.calc.type = NULL,
  bpop = poped.db$parameters$bpop,
  d = poped.db$parameters$d,
  covd = poped.db$parameters$covd,
  docc = poped.db$parameters$docc,
  sigma = poped.db$parameters$sigma,
  model_switch = NULL,
  ni = NULL,
  xt = NULL,
  x = NULL,
  a = NULL,
  groupsize = poped.db$design$groupsize,
  deriv.type = NULL,
  bLHS = poped.db$settings$bLHS,
  ofv_calc_type = poped.db$settings$ofv_calc_type,
  ED_samp_size = poped.db$settings$ED_samp_size,
  use_laplace = poped.db$settings$iEDCalculationType,
  laplace.fim = FALSE,
  ...
)
```

### Arguments

poped.db	A PopED database.
fim.calc.type	The method used for calculating the FIM. Potential values: <ul style="list-style-type: none"> <li>• 0 = Full FIM. No assumption that fixed and random effects are uncorrelated.</li> <li>• 1 = Reduced FIM. Assume that there is no correlation in the FIM between the fixed and random effects, and set these elements in the FIM to zero.</li> <li>• 2 = weighted models (placeholder).</li> <li>• 3 = Not currently used.</li> </ul>

- 4 = Reduced FIM and computing all derivatives with respect to the standard deviation of the residual unexplained variation ( $\sqrt{\text{SIGMA}}$  in NONMEM). This matches what is done in PFIM, and assumes that the standard deviation of the residual unexplained variation is the estimated parameter (NOTE: NONMEM estimates the variance of the residual unexplained variation by default).
- 5 = Full FIM parameterized with A,B,C matrices & derivative of variance.
- 6 = Calculate one model switch at a time, good for large matrices.
- 7 = Reduced FIM parameterized with A,B,C matrices & derivative of variance.

**bpop** Matrix defining the fixed effects, per row (row number = parameter\_number) we should have:

- column 1 the type of the distribution for E-family designs (0 = Fixed, 1 = Normal, 2 = Uniform, 3 = User Defined Distribution, 4 = lognormal and 5 = truncated normal)
- column 2 defines the mean.
- column 3 defines the variance of the distribution (or length of uniform distribution).

Can also just supply the parameter values as a vector `c()` if no uncertainty around the parameter value is to be used. The parameter order of 'bpop' is defined in the 'fg\_fun' or 'fg\_file'. If you use named arguments in 'bpop' then the order of this vector can be rearranged to match the 'fg\_fun' or 'fg\_file'. See 'reorder\_parameter\_vectors'.

**d** Matrix defining the diagonals of the IIV (same logic as for the fixed effects matrix bpop to define uncertainty). One can also just supply the parameter values as a `c()`. The parameter order of 'd' is defined in the 'fg\_fun' or 'fg\_file'. If you use named arguments in 'd' then the order of this vector can be rearranged to match the 'fg\_fun' or 'fg\_file'. See 'reorder\_parameter\_vectors'.

**covd** Column major vector defining the covariances of the IIV variances. That is, from your full IIV matrix `covd <- IIV[lower.tri(IIV)]`.

**docc** Matrix defining the IOV, the IOV variances and the IOV distribution as for d and bpop.

**sigma** Matrix defining the variances and covariances of the residual variability terms of the model. can also just supply the diagonal parameter values (variances) as a `c()`.

**model\_switch** A matrix that is the same size as xt, specifying which model each sample belongs to.

**ni** A vector of the number of samples in each group.

**xt** A matrix of sample times. Each row is a vector of sample times for a group.

**x** A matrix for the discrete design variables. Each row is a group.

**a** A matrix of covariates. Each row is a group.

**groupsize** A vector of the number of individuals in each group.

**deriv.type** A number indicating the type of derivative to use:



	<ul style="list-style-type: none"> <li>• 0=Complex difference</li> <li>• 1=Central difference</li> <li>• 20=Analytic derivative (placeholder)</li> <li>• 30=Automatic differentiation (placeholder)</li> </ul>
bLHS	How to sample from distributions in E-family calculations. 0=Random Sampling, 1=LatinHyperCube –
ofv_calc_type	OFV calculation type for FIM <ul style="list-style-type: none"> <li>• 1 = "D-optimality". Determinant of the FIM: <math>\det(\text{FIM})</math></li> <li>• 2 = "A-optimality". Inverse of the sum of the expected parameter variances: <math>1/\text{trace\_matrix}(\text{inv}(\text{FIM}))</math></li> <li>• 4 = "lnD-optimality". Natural logarithm of the determinant of the FIM: <math>\log(\det(\text{FIM}))</math></li> <li>• 6 = "Ds-optimality". Ratio of the Determinant of the FIM and the Determinant of the uninteresting rows and columns of the FIM: <math>\det(\text{FIM})/\det(\text{FIM\_u})</math></li> <li>• 7 = Inverse of the sum of the expected parameter RSE: <math>1/\text{sum}(\text{get\_rse}(\text{FIM}, \text{poped.db}, \text{use\_percent}=\text{FA}))</math></li> </ul>
ED_samp_size	Sample size for E-family sampling
use_laplace	Should the Laplace method be used in calculating the expectation of the OFV?
laplace.fim	Should an E(FIM) be calculated when computing the Laplace approximated E(OFV). Typically the FIM does not need to be computed and, if desired, this calculation is done using the standard MC integration technique, so can be slow.
...	Other arguments passed to the function.

**Value**

A list containing the E(FIM) and E(OFV(FIM)) and the a poped.db updated according to the function arguments.

**See Also**

Other FIM: [LinMatrixH\(\)](#), [LinMatrixLH\(\)](#), [LinMatrixL\\_occ\(\)](#), [calc\\_ofv\\_and\\_fim\(\)](#), [ed\\_laplace\\_ofv\(\)](#), [ed\\_mftot\(\)](#), [efficiency\(\)](#), [evaluate.fim\(\)](#), [gradf\\_eps\(\)](#), [mf3\(\)](#), [mf7\(\)](#), [mftot\(\)](#), [ofv\\_criterion\(\)](#), [ofv\\_fim\(\)](#)

Other E-family: [calc\\_ofv\\_and\\_fim\(\)](#), [ed\\_laplace\\_ofv\(\)](#), [ed\\_mftot\(\)](#)

Other evaluate\_FIM: [calc\\_ofv\\_and\\_fim\(\)](#), [evaluate.fim\(\)](#), [ofv\\_fim\(\)](#)

**Examples**

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin model for optimization
## with parameter uncertainty)
#####

## Warfarin example from software comparison in:
```

```

## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## Optimization using an additive + proportional residual error
## to avoid sample times at very low concentrations (time 0 or very late samples).

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

# Adding 10% log-normal Uncertainty to fixed effects (not Favail)
bpop_vals <- c(CL=0.15, V=8, KA=1.0, Favail=1)
bpop_vals_ed_ln <- cbind(ones(length(bpop_vals),1)*4, # log-normal distribution
                        bpop_vals,
                        ones(length(bpop_vals),1)*(bpop_vals*0.1)^2) # 10% of bpop value
bpop_vals_ed_ln["Favail",] <- c(0,1,0)
bpop_vals_ed_ln

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                bpop=bpop_vals_ed_ln,
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(0.01,0.25),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0,
                                maxx=120,
                                a=70,
                                mina=0,
                                maxa=100)

##### END #####
## Create PopED database
## (warfarin model for optimization
## with parameter uncertainty)
#####

## ED evaluate (with very few samples)

```

```

output <- evaluate.e.ofv.fim(poped.db,ED_samp_size=10)
output$E_ofv

## API evaluate (with very few samples)
output <- evaluate.e.ofv.fim(poped.db,ED_samp_size=10,ofv_calc_type=4)
output$E_ofv

## ED evaluate using Laplace approximation
tic()
output <- evaluate.e.ofv.fim(poped.db,use_laplace=TRUE)
toc()
output$E_ofv

## Not run:

## ED expected value with more precision.
## Compare time and value to Laplace approximation.
## Run a couple of times to see stochasticity of calculation.
tic()
e_ofv_mc <- evaluate.e.ofv.fim(poped.db,ED_samp_size=500)
toc()
e_ofv_mc$E_ofv

# If you want to get an E(FIM) from the laplace approximation you have to ask for it
# and it will take more time.
output <- evaluate.e.ofv.fim(poped.db,use_laplace=TRUE,laplace.fim=TRUE)
output$E_fim

## End(Not run)

```

---

evaluate.fim

---

*Evaluate the Fisher Information Matrix (FIM)*


---

## Description

Compute the FIM given the model, parameters, design and methods defined in the PopED database. Some of the arguments coming from the PopED database can be overwritten; by default these arguments are NULL in the function, if they are supplied then they are used instead of the arguments from the PopED database.

## Usage

```

evaluate.fim(
  poped.db,
  fim.calc.type = NULL,
  approx.method = NULL,
  FOCE.num = NULL,

```

```

bpop.val = NULL,
d_full = NULL,
docc_full = NULL,
sigma_full = NULL,
model_switch = NULL,
ni = NULL,
xt = NULL,
x = NULL,
a = NULL,
groupsize = NULL,
deriv.type = NULL,
...
)

```

### Arguments

poped.db	A PopED database.
fim.calc.type	The method used for calculating the FIM. Potential values: <ul style="list-style-type: none"> <li>• 0 = Full FIM. No assumption that fixed and random effects are uncorrelated.</li> <li>• 1 = Reduced FIM. Assume that there is no correlation in the FIM between the fixed and random effects, and set these elements in the FIM to zero.</li> <li>• 2 = weighted models (placeholder).</li> <li>• 3 = Not currently used.</li> <li>• 4 = Reduced FIM and computing all derivatives with respect to the standard deviation of the residual unexplained variation (sqrt(SIGMA) in NONMEM). This matches what is done in PFIM, and assumes that the standard deviation of the residual unexplained variation is the estimated parameter (NOTE: NONMEM estimates the variance of the residual unexplained variation by default).</li> <li>• 5 = Full FIM parameterized with A,B,C matrices &amp; derivative of variance.</li> <li>• 6 = Calculate one model switch at a time, good for large matrices.</li> <li>• 7 = Reduced FIM parameterized with A,B,C matrices &amp; derivative of variance.</li> </ul>
approx.method	Approximation method for model, 0=FO, 1=FOCE, 2=FOCEI, 3=FOI
FOCE.num	Number individuals in each step of FOCE approximation method
bpop.val	The fixed effects parameter values. Supplied as a vector.
d_full	A between subject variability matrix (OMEGA in NONMEM).
docc_full	A between occasion variability matrix.
sigma_full	A residual unexplained variability matrix (SIGMA in NONMEM).
model_switch	A matrix that is the same size as xt, specifying which model each sample belongs to.
ni	A vector of the number of samples in each group.
xt	A matrix of sample times. Each row is a vector of sample times for a group.
x	A matrix for the discrete design variables. Each row is a group.



```

bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
# notfixed_bpop=c(1,1,1,0),
notfixed_bpop=c(CL=1,V=1,KA=1,Favail=0),
d=c(CL=0.07, V=0.02, KA=0.6),
sigma=0.01,
groupsize=32,
xt=c( 0.5,1,2,6,24,36,72,120),
minxt=0,
maxxt=120,
a=70)

## evaluate initial design with the reduced FIM
FIM.1 <- evaluate.fim(poped.db)
FIM.1
det(FIM.1)
det(FIM.1)^(1/7)
get_rse(FIM.1,poped.db)

## evaluate initial design with the full FIM
FIM.0 <- evaluate.fim(poped.db,fim.calc.type=0)
FIM.0
det(FIM.0)
det(FIM.0)^(1/7)
get_rse(FIM.0,poped.db)

## evaluate initial design with the reduced FIM
## computing all derivatives with respect to the
## standard deviation of the residual unexplained variation
FIM.4 <- evaluate.fim(poped.db,fim.calc.type=4)
FIM.4
det(FIM.4)
get_rse(FIM.4,poped.db,fim.calc.type=4)

## evaluate initial design with the full FIM with A,B,C matrices
## should give same answer as fim.calc.type=0
FIM.5 <- evaluate.fim(poped.db,fim.calc.type=5)
FIM.5
det(FIM.5)
get_rse(FIM.5,poped.db,fim.calc.type=5)

## evaluate initial design with the reduced FIM with
## A,B,C matrices and derivative of variance
## should give same answer as fim.calc.type=1 (default)
FIM.7 <- evaluate.fim(poped.db,fim.calc.type=7)
FIM.7
det(FIM.7)
get_rse(FIM.7,poped.db,fim.calc.type=7)

## evaluate FIM and rse with prior FIM.1
poped.db.prior = create.poped.database(poped.db, prior_fim = FIM.1)
FIM.1.prior <- evaluate.fim(poped.db.prior)
all.equal(FIM.1.prior,FIM.1) # the FIM is only computed from the design in the poped.db

```

```
get_rse(FIM.1.prior,poped.db.prior) # the RSE is computed with the prior information
```

---

evaluate_design	<i>Evaluate a design</i>
-----------------	--------------------------

---

## Description

This function evaluates the design defined in a poped database.

## Usage

```
evaluate_design(poped.db, ...)
```

## Arguments

poped.db	A poped database
...	Extra parameters passed to <code>calc_ofv_and_fim</code> and <code>get_rse</code>

## Value

A list of elements evaluating the current design.

## See Also

Other `evaluate_design`: [evaluate.fim\(\)](#), [evaluate\\_power\(\)](#), [get\\_rse\(\)](#), [model\\_prediction\(\)](#), [plot\\_efficiency\\_of\\_windows\(\)](#), [plot\\_model\\_prediction\(\)](#)

## Examples

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin example)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
```

```

        KA=bpop[3]*exp(b[3]),
        Favail=bpop[4],
        DOSE=a[1])
    return(parameters)
}

## -- Define model, parameters, initial design
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(prop=0.01),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                a=c(DOSE=70))

##### END #####
## Create PopED database
## (warfarin example)
#####

evaluate_design(poped.db)

```

---

evaluate\_fim\_map

*Compute the Bayesian Fisher information matrix*

---

## Description

Computation of the Bayesian Fisher information matrix for individual parameters of a population model based on Maximum A Posteriori (MAP) estimation of the empirical Bayes estimates (EBEs) in a population model

## Usage

```

evaluate_fim_map(
  poped.db,
  use_mc = FALSE,
  num_sim_ids = 1000,
  use_purrr = FALSE,
  shrink_mat = F
)

```

## Arguments

poped.db      A PopED database



use_mc	Should the calculation be based on monte-carlo simulations. If not then then a first order approximation is used
num_sim_ids	If use_mc=TRUE, how many individuals should be simulated to make the computations.
use_purrr	If use_mc=TRUE then should the method use the package purrr in calculations? This may speed up computations (potentially).
shrink_mat	Should the shrinkage matrix be returned. Calculated as the inverse of the Bayesian Fisher information matrix times the inverse of the omega matrix (variance matrix of the between-subject variability).

## Value

The Bayesian Fisher information matrix for each design group

## References

1. Combes, F. P., Retout, S., Frey, N., & Mentre, F. (2013). Prediction of shrinkage of individual parameters using the Bayesian information matrix in non-linear mixed effect models with evaluation in pharmacokinetics. *Pharmaceutical Research*, 30(9), 2355-67. doi:10.1007/s1109501310793.
2. Hennig, S., Nyberg, J., Fanta, S., Backman, J. T., Hoppu, K., Hooker, A. C., & Karlsson, M. O. (2012). Application of the optimal design approach to improve a pretransplant drug dose finding design for ciclosporin. *Journal of Clinical Pharmacology*, 52(3), 347-360. doi:10.1177/0091270010397731.

## Examples

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin example)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
}
```

```

    return(parameters)
  }

  ## -- Define model, parameters, initial design
  poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                   fg_fun=sfg,
                                   fError_fun=feps.prop,
                                   bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                   notfixed_bpop=c(1,1,1,0),
                                   d=c(CL=0.07, V=0.02, KA=0.6),
                                   sigma=c(prop=0.01),
                                   groupsize=32,
                                   xt=c( 0.5,1,2,6,24,36,72,120),
                                   a=c(DOSE=70))

  ##### END #####
  ## Create PopED database
  ## (warfarin example)
  #####

  shrinkage(poped.db)

```

---

 evaluate\_power

*Power of a design to estimate a parameter.*


---

### Description

Evaluate the power of a design to estimate a parameter value different than some assumed value (often the assumed value is zero). The power is calculated using the linear Wald test and the the design is defined in a poped database.

### Usage

```

evaluate_power(
  poped.db,
  bpop_idx,
  h0 = 0,
  alpha = 0.05,
  power = 0.8,
  twoSided = TRUE,
  find_min_n = TRUE,
  fim = NULL,
  out = NULL,
  ...
)

```

**Arguments**

poped.db	A poped database
bpop_idx	Index for an unfixed population parameter (bpop) for which the power should be evaluated for being different than the null hypothesis (h0).
h0	The null hypothesized value for the parameter.
alpha	Type 1 error.
power	Targeted power.
twoSided	Is this a two-sided test.
find_min_n	Should the function compute the minimum n needed (given the current design) to achieve the desired power?
fim	Provide the FIM from a previous calculation
out	provide output from a previous calculation (e.g., calc_ofv_and_fim, ...)
...	Extra parameters passed to <a href="#">calc_ofv_and_fim</a> and <a href="#">get_rse</a>

**Value**

A list of elements evaluating the current design including the power.

**References**

1. Retout, S., Comets, E., Samson, A., and Mentre, F. (2007). Design in nonlinear mixed effects models: Optimization using the Fedorov-Wynn algorithm and power of the Wald test for binary covariates. *Statistics in Medicine*, 26(28), 5162-5179. doi:10.1002/sim.2910.
2. Ueckert, S., Hennig, S., Nyberg, J., Karlsson, M. O., and Hooker, A. C. (2013). Optimizing disease progression study designs for drug effect discrimination. *Journal of Pharmacokinetics and Pharmacodynamics*, 40(5), 587-596. doi:10.1007/s1092801393313.

**See Also**

Other evaluate\_design: [evaluate.fim\(\)](#), [evaluate\\_design\(\)](#), [get\\_rse\(\)](#), [model\\_prediction\(\)](#), [plot\\_efficiency\\_of\\_windows\(\)](#), [plot\\_model\\_prediction\(\)](#)

**Examples**

```
# Following the examples presented in Retout, 2007

ff <- function(model_switch,xt,parameters,poped.db){
  with(as.list(parameters),{

    lambda1 <- lam1a
    if(TREAT==2) lambda1 <- lam1b

    y=log10(P1*exp(-lambda1*xt)+P2*exp(-lam2*xt))

    return(list(y=y,poped.db=poped.db))
  })
}
```

```

sfg <- function(x,a,bpop,b,bocc){
  parameters=c(P1=exp(bpop[1]+b[1]),
               P2=exp(bpop[2]+b[2]),
               lam1a=exp(bpop[3]+b[3]),
               lam1b=exp(bpop[3]+bpop[4]+b[3]),
               lam2=exp(bpop[5]+b[4]),
               TREAT=a[1])
  return(parameters)
}

poped.db <- create.poped.database(ff_fun = ff,
                                fg_fun = sfg,
                                fError_fun = feps.add,
                                bpop=c(P1=12, P2=8,
                                         lam1=-0.7,beta=0,lam2=-3.0),
                                d=c(P1=0.3, P2=0.3,
                                     lam1=0.3,lam2=0.3),
                                sigma=c(0.065^2),
                                groupsize=100,
                                m=2,
                                xt=c(1, 3, 7, 14, 28, 56),
                                minxt=0,
                                maxx=100,
                                a=list(c(TREAT=1),c(TREAT=2)))

plot_model_prediction(poped.db)
evaluate_design(poped.db)

poped.db_2 <- create.poped.database(poped.db,bpop=c(P1=12, P2=8,
                                                    lam1=-0.7,beta=0.262,lam2=-3.0))

plot_model_prediction(poped.db_2)
evaluate_design(poped.db_2)

evaluate_power(poped.db_2,bpop_idx = 4)

```

---

feps.add

*RUV model: Additive .*


---

### Description

This is a residual unexplained variability (RUV) model function that encodes the model described above. The function is suitable for input to the `create.poped.database` function using the `fError_file` argument.

### Usage

```
feps.add(model_switch, xt, parameters, epsi, poped.db)
```



```

                                sigma=1,
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0,
                                maxxt=120,
                                a=70)

## create plot of model without variability
plot_model_prediction(poped.db)

## evaluate initial design
FIM <- evaluate.fim(poped.db)
FIM
det(FIM)
get_rse(FIM, poped.db)

```

---

feps.add.prop

*RUV model: Additive and Proportional.*


---

## Description

This is a residual unexplained variability (RUV) model function that encodes the model described above. The function is suitable for input to the `create.poped.database` function using the `fError_file` argument.

## Usage

```
feps.add.prop(model_switch, xt, parameters, epsi, poped.db)
```

## Arguments

<code>model_switch</code>	a vector of values, the same size as <code>xt</code> , identifying which model response should be computed for the corresponding <code>xt</code> value. Used for multiple response models.
<code>xt</code>	a vector of independent variable values (often time).
<code>parameters</code>	A named list of parameter values.
<code>epsi</code>	A matrix with the same number of rows as the <code>xt</code> vector, columns match the numbers defined in this function.
<code>poped.db</code>	a poped database. This can be used to extract information that may be needed in the model file.

## Value

A list consisting of:

1. `y` the values of the model at the specified points.
2. `poped.db` A (potentially modified) poped database.

**See Also**

Other models: `feps.add()`, `feps.prop()`, `ff.PK.1.comp.oral.md.CL()`, `ff.PK.1.comp.oral.md.KE()`, `ff.PK.1.comp.oral.sd.CL()`, `ff.PK.1.comp.oral.sd.KE()`, `ff.PKPD.1.comp.oral.md.CL.imax()`, `ff.PKPD.1.comp.sd.CL.emax()`

Other RUV\_models: `feps.add()`, `feps.prop()`

**Examples**

```
library(PopED)

## find the parameters that are needed to define in the structural model
ff.PK.1.comp.oral.md.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c( V=bpop[1]*exp(b[1]),
               KA=bpop[2]*exp(b[2]),
               CL=bpop[3]*exp(b[3]),
               Favail=bpop[4],
               DOSE=a[1],
               TAU=a[2])
  return( parameters )
}

## -- Define design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.md.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                groupsize=20,
                                m=2,
                                sigma=c(0.04,5e-6),
                                bpop=c(V=72.8,KA=0.25,CL=3.75,Favail=0.9),
                                d=c(V=0.09,KA=0.09,CL=0.25^2),
                                notfixed_bpop=c(1,1,1,0),
                                notfixed_sigma=c(0,0),
                                xt=c( 1,2,8,240,245),
                                minxt=c(0,0,0,240,240),
                                maxx=c(10,10,10,248,248),
                                a=cbind(c(20,40),c(24,24)),
                                bUseGrouped_xt=1,
                                maxa=c(200,24),
                                mina=c(0,24))

## create plot of model without variability
plot_model_prediction(poped.db)

## evaluate initial design
FIM <- evaluate.fim(poped.db)
FIM
det(FIM)
```

```
get_rse(FIM,poped.db)
```

---

feps.prop

*RUV model: Proportional.*

---

### Description

This is a residual unexplained variability (RUV) model function that encodes the model described above. The function is suitable for input to the [create.poped.database](#) function using the `fError_file` argument.

### Usage

```
feps.prop(model_switch, xt, parameters, epsi, poped.db)
```

### Arguments

<code>model_switch</code>	a vector of values, the same size as <code>xt</code> , identifying which model response should be computed for the corresponding <code>xt</code> value. Used for multiple response models.
<code>xt</code>	a vector of independent variable values (often time).
<code>parameters</code>	A named list of parameter values.
<code>epsi</code>	A matrix with the same number of rows as the <code>xt</code> vector, columns match the numbers defined in this function.
<code>poped.db</code>	a poped database. This can be used to extract information that may be needed in the model file.

### Value

A list consisting of:

1. `y` the values of the model at the specified points.
2. `poped.db` A (potentially modified) poped database.

### See Also

Other models: [feps.add\(\)](#), [feps.add.prop\(\)](#), [ff.PK.1.comp.oral.md.CL\(\)](#), [ff.PK.1.comp.oral.md.KE\(\)](#), [ff.PK.1.comp.oral.sd.CL\(\)](#), [ff.PK.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.oral.md.CL.imax\(\)](#), [ff.PKPD.1.comp.sd.CL.emax\(\)](#)

Other RUV\_models: [feps.add\(\)](#), [feps.add.prop\(\)](#)



**Examples**

```

library(PopED)

##### START #####
## Create PopED database
## (warfarin example)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

## -- Define model, parameters, initial design
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(prop=0.01),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                a=c(DOSE=70))

##### END #####
## Create PopED database
## (warfarin example)
#####

## create plot of model without variability
plot_model_prediction(poped.db)

## evaluate initial design
FIM <- evaluate.fim(poped.db)
FIM
det(FIM)

```

```
get_rse(FIM,poped.db)
```

---

```
ff.PK.1.comp.oral.md.CL
```

*Structural model: one-compartment, oral absorption, multiple bolus dose, parameterized using CL.*

---

## Description

This is a structural model function that encodes a model that is one-compartment, oral absorption, multiple bolus dose, parameterized using CL. The function is suitable for input to the [create.poped.database](#) function using the `ff_fun` or `ff_file` argument.

## Usage

```
ff.PK.1.comp.oral.md.CL(model_switch, xt, parameters, poped.db)
```

## Arguments

<code>model_switch</code>	a vector of values, the same size as <code>xt</code> , identifying which model response should be computed for the corresponding <code>xt</code> value. Used for multiple response models.
<code>xt</code>	a vector of independent variable values (often time).
<code>parameters</code>	A named list of parameter values.
<code>poped.db</code>	a poped database. This can be used to extract information that may be needed in the model file.

## Value

A list consisting of:

1. `y` the values of the model at the specified points.
2. `poped.db` A (potentially modified) poped database.

## See Also

Other models: [feps.add\(\)](#), [feps.add.prop\(\)](#), [feps.prop\(\)](#), [ff.PK.1.comp.oral.md.KE\(\)](#), [ff.PK.1.comp.oral.sd.CL\(\)](#), [ff.PK.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.oral.md.CL.imax\(\)](#), [ff.PKPD.1.comp.sd.CL.emax\(\)](#)

Other structural\_models: [ff.PK.1.comp.oral.md.KE\(\)](#), [ff.PK.1.comp.oral.sd.CL\(\)](#), [ff.PK.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.oral.md.CL.imax\(\)](#), [ff.PKPD.1.comp.sd.CL.emax\(\)](#)

**Examples**

```

library(PopED)

## find the parameters that are needed to define in the structural model
ff.PK.1.comp.oral.md.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c( V=bpop[1]*exp(b[1]),
               KA=bpop[2]*exp(b[2]),
               CL=bpop[3]*exp(b[3]),
               Favail=bpop[4],
               DOSE=a[1],
               TAU=a[2])
  return( parameters )
}

## -- Define design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.md.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                groupsize=20,
                                m=2,
                                sigma=c(0.04,5e-6),
                                bpop=c(V=72.8,KA=0.25,CL=3.75,Favail=0.9),
                                d=c(V=0.09,KA=0.09,CL=0.25^2),
                                notfixed_bpop=c(1,1,1,0),
                                notfixed_sigma=c(0,0),
                                xt=c( 1,2,8,240,245),
                                minxt=c(0,0,0,240,240),
                                maxx=c(10,10,10,248,248),
                                a=cbind(c(20,40),c(24,24)),
                                bUseGrouped_xt=1,
                                maxa=c(200,24),
                                mina=c(0,24))

## create plot of model without variability
plot_model_prediction(poped.db)

## evaluate initial design
FIM <- evaluate.fim(poped.db)
FIM
det(FIM)
get_rse(FIM,poped.db)

```

---

ff.PK.1.comp.oral.md.KE

*Structural model: one-compartment, oral absorption, multiple bolus dose, parameterized using KE.*

---

**Description**

This is a structural model function that encodes a model that is one-compartment, oral absorption, multiple bolus dose, parameterized using KE. The function is suitable for input to the [create.poped.database](#) function using the `ff_fun` or `ff_file` argument.

**Usage**

```
ff.PK.1.comp.oral.md.KE(model_switch, xt, parameters, poped.db)
```

**Arguments**

<code>model_switch</code>	a vector of values, the same size as <code>xt</code> , identifying which model response should be computed for the corresponding <code>xt</code> value. Used for multiple response models.
<code>xt</code>	a vector of independent variable values (often time).
<code>parameters</code>	A named list of parameter values.
<code>poped.db</code>	a poped database. This can be used to extract information that may be needed in the model file.

**Value**

A list consisting of:

1. `y` the values of the model at the specified points.
2. `poped.db` A (potentially modified) poped database.

**See Also**

Other models: [feps.add\(\)](#), [feps.add.prop\(\)](#), [feps.prop\(\)](#), [ff.PK.1.comp.oral.md.CL\(\)](#), [ff.PK.1.comp.oral.sd.CL\(\)](#), [ff.PK.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.oral.md.CL.imax\(\)](#), [ff.PKPD.1.comp.sd.CL.emax\(\)](#)

Other structural\_models: [ff.PK.1.comp.oral.md.CL\(\)](#), [ff.PK.1.comp.oral.sd.CL\(\)](#), [ff.PK.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.oral.md.CL.imax\(\)](#), [ff.PKPD.1.comp.sd.CL.emax\(\)](#)

**Examples**

```
library(PopED)

## find the parameters that are needed to define in the structural model
ff.PK.1.comp.oral.md.KE

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  ## -- parameter definition function
  parameters=c( V=bpop[1]*exp(b[1]),
               KA=bpop[2]*exp(b[2]),
               KE=bpop[3]*exp(b[3]),
               Favail=bpop[4],
               DOSE=a[1],
```

```

        TAU=a[2])
    return( parameters )
}

## -- Define design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.md.KE,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                groupsize=20,
                                m=2,
                                sigma=c(0.04,5e-6),
                                bpop=c(V=72.8,KA=0.25,KE=3.75/72.8,Favail=0.9),
                                d=c(V=0.09,KA=0.09,KE=0.25^2),
                                notfixed_bpop=c(1,1,1,0),
                                notfixed_sigma=c(0,0),
                                xt=c( 1,2,8,240,245),
                                minxt=c(0,0,0,240,240),
                                maxx=c(10,10,10,248,248),
                                a=cbind(c(20,40),c(24,24)),
                                bUseGrouped_xt=1,
                                maxa=c(200,40),
                                mina=c(0,2))

## create plot of model without variability
plot_model_prediction(poped.db)

## evaluate initial design
FIM <- evaluate.fim(poped.db)
FIM
det(FIM)
get_rse(FIM,poped.db)

```

---

ff.PK.1.comp.oral.sd.CL

*Structural model: one-compartment, oral absorption, single bolus dose, parameterized using CL.*

---

## Description

This is a structural model function that encodes a model that is one-compartment, oral absorption, single bolus dose, parameterized using CL. The function is suitable for input to the [create.poped.database](#) function using the `ff_fun` or `ff_file` argument.

## Usage

```
ff.PK.1.comp.oral.sd.CL(model_switch, xt, parameters, poped.db)
```

**Arguments**

model_switch	a vector of values, the same size as xt, identifying which model response should be computed for the corresponding xt value. Used for multiple response models.
xt	a vector of independent variable values (often time).
parameters	A named list of parameter values.
poped.db	a poped database. This can be used to extract information that may be needed in the model file.

**Value**

A list consisting of:

1. y the values of the model at the specified points.
2. poped.db A (potentially modified) poped database.

**See Also**

Other models: [feps.add\(\)](#), [feps.add.prop\(\)](#), [feps.prop\(\)](#), [ff.PK.1.comp.oral.md.CL\(\)](#), [ff.PK.1.comp.oral.md.KE\(\)](#), [ff.PK.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.oral.md.CL.imax\(\)](#), [ff.PKPD.1.comp.sd.CL.emax\(\)](#)

Other structural\_models: [ff.PK.1.comp.oral.md.CL\(\)](#), [ff.PK.1.comp.oral.md.KE\(\)](#), [ff.PK.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.oral.md.CL.imax\(\)](#), [ff.PKPD.1.comp.sd.CL.emax\(\)](#)

**Examples**

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin example)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}
```

```

}

## -- Define model, parameters, initial design
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(prop=0.01),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                a=c(DOSE=70))

##### END #####
## Create PopED database
## (warfarin example)
#####

## create plot of model without variability
plot_model_prediction(poped.db)

## evaluate initial design
FIM <- evaluate.fim(poped.db)
FIM
det(FIM)
get_rse(FIM,poped.db)

```

---

```
ff.PK.1.comp.oral.sd.KE
```

*Structural model: one-compartment, oral absorption, single bolus dose, parameterized using KE.*

---

## Description

This is a structural model function that encodes a model that is one-compartment, oral absorption, single bolus dose, parameterized using KE. The function is suitable for input to the `create.poped.database` function using the `ff_fun` or `ff_file` argument.

## Usage

```
ff.PK.1.comp.oral.sd.KE(model_switch, xt, parameters, poped.db)
```

## Arguments

`model_switch` a vector of values, the same size as `xt`, identifying which model response should be computed for the corresponding `xt` value. Used for multiple response models.

xt                    a vector of independent variable values (often time).  
 parameters         A named list of parameter values.  
 poped.db            a poped database. This can be used to extract information that may be needed in the model file.

### Value

A list consisting of:

1. y the values of the model at the specified points.
2. poped.db A (potentially modified) poped database.

### See Also

Other models: `feps.add()`, `feps.add.prop()`, `feps.prop()`, `ff.PK.1.comp.oral.md.CL()`, `ff.PK.1.comp.oral.md.KE()`, `ff.PK.1.comp.oral.sd.CL()`, `ff.PKPD.1.comp.oral.md.CL.imax()`, `ff.PKPD.1.comp.sd.CL.emax()`

Other structural\_models: `ff.PK.1.comp.oral.md.CL()`, `ff.PK.1.comp.oral.md.KE()`, `ff.PK.1.comp.oral.sd.CL()`, `ff.PKPD.1.comp.oral.md.CL.imax()`, `ff.PKPD.1.comp.sd.CL.emax()`

### Examples

```
library(PopED)

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.KE

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(KE=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.KE,
                                fg_fun=sfg,
                                fError_fun=feps.prop,
                                bpop=c(KE=0.15/8, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(KE=0.07, V=0.02, KA=0.6),
                                sigma=0.01,
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0,
                                maxx=120,
                                a=70)
```



```
## create plot of model without variability
plot_model_prediction(poped.db)

## evaluate initial design
FIM <- evaluate.fim(poped.db)
FIM
det(FIM)
get_rse(FIM, poped.db)
```

---

```
ff.PKPD.1.comp.oral.md.CL.imax
```

*Structural model: one-compartment, oral absorption, multiple bolus dose, parameterized using CL driving an inhibitory IMAX model with a direct effect.*

---

## Description

This is a structural model function that encodes the model described above. The function is suitable for input to the [create.poped.database](#) function using the `ff_fun` or `ff_file` argument.

## Usage

```
ff.PKPD.1.comp.oral.md.CL.imax(model_switch, xt, parameters, poped.db)
```

## Arguments

<code>model_switch</code>	a vector of values, the same size as <code>xt</code> , identifying which model response should be computed for the corresponding <code>xt</code> value. Used for multiple response models.
<code>xt</code>	a vector of independent variable values (often time).
<code>parameters</code>	A named list of parameter values.
<code>poped.db</code>	a poped database. This can be used to extract information that may be needed in the model file.

## Value

A list consisting of:

1. `y` the values of the model at the specified points.
2. `poped.db` A (potentially modified) poped database.

## See Also

Other models: [feps.add\(\)](#), [feps.add.prop\(\)](#), [feps.prop\(\)](#), [ff.PK.1.comp.oral.md.CL\(\)](#), [ff.PK.1.comp.oral.md.KE\(\)](#), [ff.PK.1.comp.oral.sd.CL\(\)](#), [ff.PK.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.sd.CL\(\)](#).

Other structural\_models: [ff.PK.1.comp.oral.md.CL\(\)](#), [ff.PK.1.comp.oral.md.KE\(\)](#), [ff.PK.1.comp.oral.sd.CL\(\)](#), [ff.PK.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.sd.CL.emax\(\)](#)



```

d=c(V=0.09,KA=0.09,CL=0.25^2,E0=0.09),
sigma=c(0.04,5e-6,0.09,100),
notfixed_sigma=c(0,0,0,0),
xt=c(1,2,8,240,240,1,2,8,240,240),
minxt=c(0,0,0,240,240,0,0,0,240,240),
maxxt=c(10,10,10,248,248,10,10,10,248,248),
G_xt=c(1,2,3,4,5,1,2,3,4,5),
model_switch=c(1,1,1,1,2,2,2,2,2),
a=cbind(c(20,40,0),c(24,24,24)),
bUseGrouped_xt=1,
ourzero=0,
maxa=c(200,40),
mina=c(0,2))

## create plot of model without variability
plot_model_prediction(poped.db, facet_scales="free")

## evaluate initial design
FIM <- evaluate.fim(poped.db)
FIM
det(FIM)
get_rse(FIM, poped.db)

```

---

```
ff.PKPD.1.comp.sd.CL.emax
```

*Structural model: one-compartment, single bolus IV dose, parameterized using CL driving an EMAX model with a direct effect.*

---

## Description

This is a structural model function that encodes the model described above. The function is suitable for input to the [create.poped.database](#) function using the `ff_fun` or `ff_file` argument.

## Usage

```
ff.PKPD.1.comp.sd.CL.emax(model_switch, xt, parameters, poped.db)
```

## Arguments

<code>model_switch</code>	a vector of values, the same size as <code>xt</code> , identifying which model response should be computed for the corresponding <code>xt</code> value. Used for multiple response models.
<code>xt</code>	a vector of independent variable values (often time).
<code>parameters</code>	A named list of parameter values.
<code>poped.db</code>	a poped database. This can be used to extract information that may be needed in the model file.

**Value**

A list consisting of:

1. y the values of the model at the specified points.
2. poped.db A (potentially modified) poped database.

**See Also**

Other models: [feps.add\(\)](#), [feps.add.prop\(\)](#), [feps.prop\(\)](#), [ff.PK.1.comp.oral.md.CL\(\)](#), [ff.PK.1.comp.oral.md.KE\(\)](#), [ff.PK.1.comp.oral.sd.CL\(\)](#), [ff.PK.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.oral.md.CL\(\)](#), [ff.PKPD.1.comp.oral.md.KE\(\)](#), [ff.PKPD.1.comp.oral.sd.CL\(\)](#), [ff.PKPD.1.comp.oral.sd.KE\(\)](#), [ff.PKPD.1.comp.oral.md.CL.imax\(\)](#)

**Examples**

```
library(PopED)

## find the parameters that are needed to define from the structural model
ff.PKPD.1.comp.sd.CL.emax

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  ## -- parameter definition function
  parameters=c(
    CL=bpop[1]*exp(b[1]) ,
    V=bpop[2]*exp(b[2]) ,
    E0=bpop[3]*exp(b[3]) ,
    EMAX=bpop[4]*exp(b[4]) ,
    EC50=bpop[5]*exp(b[5]) ,
    DOSE=a[1]
  )
  return( parameters )
}

feps <- function(model_switch,xt,parameters,epsi,poped.db){
  ## -- Residual Error function
  ## -- Proportional PK + additive PD
  returnArgs <- do.call(poped.db$model$ff_pointer,list(model_switch,xt,parameters,poped.db))
  y <- returnArgs[[1]]
  poped.db <- returnArgs[[2]]

  MS <- model_switch

  prop.err <- y*(1+epsi[,1])
  add.err <- y+epsi[,2]

  y[MS==1] = prop.err[MS==1]
  y[MS==2] = add.err[MS==2]

  return(list( y= y,poped.db =poped.db ))
}
```

```

}

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PKPD.1.comp.sd.CL.emax,
                                fError_fun=feps,
                                fg_fun=sfg,
                                groupsize=20,
                                m=3,
                                sigma=diag(c(0.15,0.15)),
                                bpop=c(CL=0.5,V=0.2,E0=1,EMAX=1,EC50=1),
                                d=c(CL=0.01,V=0.01,E0=0.01,EMAX=0.01,EC50=0.01),
                                xt=c( 0.33,0.66,0.9,5,0.1,1,2,5),
                                model_switch=c( 1,1,1,1,2,2,2,2),
                                minxt=0,
                                maxxt=5,
                                a=rbind(2.75,5,10),
                                bUseGrouped_xt=1,
                                maxa=10,
                                mina=0.1)

## create plot of model without variability
plot_model_prediction(poped.db, facet_scales="free")

## evaluate initial design
FIM <- evaluate.fim(poped.db)
FIM
det(FIM)
get_rse(FIM, poped.db)

```

---

get\_rse

---

*Compute the expected parameter relative standard errors*


---

## Description

This function computes the expected relative standard errors of a model given a design and a previously computed FIM.

## Usage

```

get_rse(
  fim,
  poped.db,
  bpop = poped.db$parameters$bpop[, 2],
  d = poped.db$parameters$d[, 2],
  docc = poped.db$parameters$docc,
  sigma = poped.db$parameters$sigma,
  use_percent = TRUE,

```

```

    fim.calc.type = poped.db$settings$iFIMCalculationType,
    prior_fim = poped.db$settings$prior_fim,
    ...
)

```

### Arguments

<code>fim</code>	A Fisher Information Matrix (FIM).
<code>poped.db</code>	A PopED database.
<code>bpop</code>	A vector containing the values of the fixed effects used to compute the <code>fim</code> .
<code>d</code>	A vector containing the values of the diagonals of the between subject variability matrix.
<code>docc</code>	Matrix defining the IOV, the IOV variances and the IOV distribution as for <code>d</code> and <code>bpop</code> .
<code>sigma</code>	Matrix defining the variances can covariances of the residual variability terms of the model. can also just supply the diagonal parameter values (variances) as a <code>c()</code> .
<code>use_percent</code>	Should RSE be reported as percent?
<code>fim.calc.type</code>	The method used for calculating the FIM. Potential values: <ul style="list-style-type: none"> <li>• 0 = Full FIM. No assumption that fixed and random effects are uncorrelated.</li> <li>• 1 = Reduced FIM. Assume that there is no correlation in the FIM between the fixed and random effects, and set these elements in the FIM to zero.</li> <li>• 2 = weighted models (placeholder).</li> <li>• 3 = Not currently used.</li> <li>• 4 = Reduced FIM and computing all derivatives with respect to the standard deviation of the residual unexplained variation (<math>\sqrt{\text{SIGMA}}</math> in NONMEM). This matches what is done in PFIM, and assumes that the standard deviation of the residual unexplained variation is the estimated parameter (NOTE: NONMEM estimates the variance of the residual unexplained variation by default).</li> <li>• 5 = Full FIM parameterized with A,B,C matrices &amp; derivative of variance.</li> <li>• 6 = Calculate one model switch at a time, good for large matrices.</li> <li>• 7 = Reduced FIM parameterized with A,B,C matrices &amp; derivative of variance.</li> </ul>
<code>prior_fim</code>	A prior FIM to be added to the <code>fim</code> . Should be the same size as the <code>fim</code> .
<code>...</code>	Additional arguments passed to <a href="#">inv</a> .

### Value

A named list of RSE values. If the estimated parameter is assumed to be zero then for that parameter the standard error is returned.

### See Also

Other `evaluate_design`: [evaluate.fim\(\)](#), [evaluate\\_design\(\)](#), [evaluate\\_power\(\)](#), [model\\_prediction\(\)](#), [plot\\_efficiency\\_of\\_windows\(\)](#), [plot\\_model\\_prediction\(\)](#)

## Examples

```

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

library(PopED)

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.md.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun = ff.PK.1.comp.oral.sd.CL,
                                fg_fun = sfg,
                                fError_fun = feps.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                # notfixed_bpop=c(1,1,1,0),
                                notfixed_bpop=c(CL=1,V=1,KA=1,Favail=0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=0.01,
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0,
                                maxx=120,
                                a=70)

## evaluate initial design with the reduced FIM
FIM.1 <- evaluate.fim(poped.db)
FIM.1
det(FIM.1)
det(FIM.1)^(1/7)
get_rse(FIM.1,poped.db)

## evaluate initial design with the full FIM
FIM.0 <- evaluate.fim(poped.db,fim.calc.type=0)
FIM.0
det(FIM.0)
det(FIM.0)^(1/7)
get_rse(FIM.0,poped.db)

```

```

## evaluate initial design with the reduced FIM
## computing all derivatives with respect to the
## standard deviation of the residual unexplained variation
FIM.4 <- evaluate.fim(poped.db, fim.calc.type=4)
FIM.4
det(FIM.4)
get_rse(FIM.4, poped.db, fim.calc.type=4)

## evaluate initial design with the full FIM with A,B,C matrices
## should give same answer as fim.calc.type=0
FIM.5 <- evaluate.fim(poped.db, fim.calc.type=5)
FIM.5
det(FIM.5)
get_rse(FIM.5, poped.db, fim.calc.type=5)

## evaluate initial design with the reduced FIM with
## A,B,C matrices and derivative of variance
## should give same answer as fim.calc.type=1 (default)
FIM.7 <- evaluate.fim(poped.db, fim.calc.type=7)
FIM.7
det(FIM.7)
get_rse(FIM.7, poped.db, fim.calc.type=7)

## evaluate FIM and rse with prior FIM.1
poped.db.prior = create.poped.database(poped.db, prior_fim = FIM.1)
FIM.1.prior <- evaluate.fim(poped.db.prior)
all.equal(FIM.1.prior, FIM.1) # the FIM is only computed from the design in the poped.db
get_rse(FIM.1.prior, poped.db.prior) # the RSE is computed with the prior information

```

---

LEDOptim

*Optimization function for D-family, E-family and Laplace approximated ED designs*


---

## Description

Optimize the objective function for D-family, E-family and Laplace approximated ED designs. Right now there is only one optimization algorithm used in this function

1. Adaptive random search. See [RS\\_opt](#).

This function takes information from the PopED database supplied as an argument. The PopED database supplies information about the the model, parameters, design and methods to use. Some of the arguments coming from the PopED database can be overwritten; if they are supplied then they are used instead of the arguments from the PopED database.

## Usage

```

LEDOptim(
  poped.db,
  model_switch = NULL,

```



```

ni = NULL,
xt = NULL,
x = NULL,
a = NULL,
bpopdescr = NULL,
ddescr = NULL,
maxxt = NULL,
minxt = NULL,
maxa = NULL,
mina = NULL,
ofv_init = 0,
fim_init = 0,
trflag = TRUE,
header_flag = TRUE,
footer_flag = TRUE,
opt_xt = poped.db$settings$optsw[2],
opt_a = poped.db$settings$optsw[4],
opt_x = poped.db$settings$optsw[3],
out_file = NULL,
d_switch = FALSE,
use_laplace = T,
laplace.fim = FALSE,
use_RS = poped.db$settings$bUseRandomSearch,
...
)

```

### Arguments

poped.db	A PopED database.
model_switch	A matrix that is the same size as xt, specifying which model each sample belongs to.
ni	A vector of the number of samples in each group.
xt	A matrix of sample times. Each row is a vector of sample times for a group.
x	A matrix for the discrete design variables. Each row is a group.
a	A matrix of covariates. Each row is a group.
bpopdescr	Matrix defining the fixed effects, per row (row number = parameter_number) we should have: <ul style="list-style-type: none"> <li>• column 1 the type of the distribution for E-family designs (0 = Fixed, 1 = Normal, 2 = Uniform, 3 = User Defined Distribution, 4 = lognormal and 5 = truncated normal)</li> <li>• column 2 defines the mean.</li> <li>• column 3 defines the variance of the distribution (or length of uniform distribution).</li> </ul>
ddescr	Matrix defining the diagonals of the IIV (same logic as for the bpopdescr).
maxxt	Matrix or single value defining the maximum value for each xt sample. If a single value is supplied then all xt values are given the same maximum value.

minxt	Matrix or single value defining the minimum value for each xt sample. If a single value is supplied then all xt values are given the same minimum value
maxa	Vector defining the max value for each covariate. If a single value is supplied then all a values are given the same max value
mina	Vector defining the min value for each covariate. If a single value is supplied then all a values are given the same max value
ofv_init	The initial OFV. If set to zero then it is computed.
fim_init	The initial value of the FIM. If set to zero then it is computed.
trflag	Should the optimization be output to the screen and to a file?
header_flag	Should the header text be printed out?
footer_flag	Should the footer text be printed out?
opt_xt	Should the sample times be optimized?
opt_a	Should the continuous design variables be optimized?
opt_x	Should the discrete design variables be optimized?
out_file	Which file should the output be directed to? A string, a file handle using <code>file</code> or <code>" "</code> will output to the screen.
d_switch	<ul style="list-style-type: none"> <li>• *****<b>START OF CRITERION SPECIFICATION OPTIONS</b>*****</li> </ul> D-family design (1) or ED-family design (0) (with or without parameter uncertainty)
use_laplace	Should the Laplace method be used in calculating the expectation of the OFV?
laplace.fim	Should an E(FIM) be calculated when computing the Laplace approximated E(OFV). Typically the FIM does not need to be computed and, if desired, this calculation is done using the standard MC integration technique, so can be slow.
use_RS	should the function use a random search algorithm?
...	arguments passed to <code>evaluate.fim</code> and <code>ofv.fim</code> .

### See Also

Other Optimize: `Doptim()`, `RS_opt()`, `a_line_search()`, `bfgsb_min()`, `calc Autofocus()`, `calc_ofv_and_grad()`, `mfea()`, `optim_ARS()`, `optim_LS()`, `poped_optim()`, `poped_optim_1()`, `poped_optim_2()`, `poped_optim_3()`, `poped_optimize()`

### Examples

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin model for optimization
## with parameter uncertainty)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
```

```

## Br. J. Clin. Pharm., 2014.

## Optimization using an additive + proportional reidual error
## to avoid sample times at very low concentrations (time 0 or very late samoples).

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

# Adding 10% log-normal Uncertainty to fixed effects (not Favail)
bpop_vals <- c(CL=0.15, V=8, KA=1.0, Favail=1)
bpop_vals_ed_ln <- cbind(ones(length(bpop_vals),1)*4, # log-normal distribution
                        bpop_vals,
                        ones(length(bpop_vals),1)*(bpop_vals*0.1)^2) # 10% of bpop value
bpop_vals_ed_ln["Favail",] <- c(0,1,0)
bpop_vals_ed_ln

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                bpop=bpop_vals_ed_ln,
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(0.01,0.25),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0,
                                maxxt=120,
                                a=70,
                                mina=0,
                                maxa=100)

##### END #####
## Create PopED database
## (warfarin model for optimization
## with parameter uncertainty)
#####

# warfarin ed model

## Not run:

```

```

LEDOptim(poped.db)

LEDOptim(poped.db,opt_xt=T,rsit=10)

LEDOptim(poped.db,opt_xt=T,rsit=10,d_switch=TRUE)

LEDOptim(poped.db,opt_xt=T,rsit=10,laplace.fim=TRUE)

LEDOptim(poped.db,opt_xt=T,rsit=10,use_laplace=FALSE)

## testing header and footer
LEDOptim(poped.db,opt_xt=T,rsit=10,d_switch=TRUE,
         out_file="foobar.txt")

ff <- LEDOptim(poped.db,opt_xt=T,rsit=10,d_switch=TRUE,
              trflag=FALSE)

LEDOptim(poped.db,opt_xt=T,rsit=10,d_switch=TRUE,
         header_flag=FALSE)

LEDOptim(poped.db,opt_xt=T,rsit=10,d_switch=TRUE,
         out_file="")

LEDOptim(poped.db,opt_xt=T,rsit=10,d_switch=TRUE,
         footer_flag=FALSE)

LEDOptim(poped.db,opt_xt=T,rsit=10,d_switch=TRUE,
         footer_flag=FALSE, header_flag=FALSE)

LEDOptim(poped.db,opt_xt=T,rsit=10,d_switch=TRUE,
         footer_flag=FALSE, header_flag=FALSE,out_file="foobar.txt")

LEDOptim(poped.db,opt_xt=T,rsit=10,d_switch=TRUE,
         footer_flag=FALSE, header_flag=FALSE,out_file="")

## End(Not run)

```

---

mc\_mean

---

*Compute the monte-carlo mean of a function*


---

### Description

Function computes the monte-carlo mean of a function by varying the parameter inputs to the function

### Usage

```

mc_mean(
  ofv_fcn,

```

```

    poped.db,
    bpopdescr = poped.db$parameters$bpop,
    ddescr = poped.db$parameters$d,
    doccdescr = poped.db$parameters$d,
    user_distribution_pointer = poped.db$model$user_distribution_pointer,
    ED_samp_size = poped.db$settings$ED_samp_size,
    bLHS = poped.db$settings$bLHS,
    ...
)

```

### Arguments

ofv_fcn	A function with poped.db as the first input
poped.db	A PopED database.
bpopdescr	Matrix defining the fixed effects, per row (row number = parameter_number) we should have: <ul style="list-style-type: none"> <li>• column 1 the type of the distribution for E-family designs (0 = Fixed, 1 = Normal, 2 = Uniform, 3 = User Defined Distribution, 4 = lognormal and 5 = truncated normal)</li> <li>• column 2 defines the mean.</li> <li>• column 3 defines the variance of the distribution (or length of uniform distribution).</li> </ul>
ddescr	Matrix defining the diagonals of the IIV (same logic as for the bpopdescr).
doccdescr	Matrix defining the IOV. per row (row number = parameter_number) we should have: <ul style="list-style-type: none"> <li>• column 1 the type of the distribution for E-family designs (0 = Fixed, 1 = Normal, 2 = Uniform, 3 = User Defined Distribution, 4 = lognormal and 5 = truncated normal)</li> <li>• column 2 defines the mean of the variance.</li> <li>• column 3 defines the variance of the distribution (or length of uniform distribution).</li> </ul>
user_distribution_pointer	Function name for user defined distributions for E-family designs
ED_samp_size	Sample size for E-family sampling
bLHS	How to sample from distributions in E-family calculations. 0=Random Sampling, 1=LatinHyperCube –
...	Other arguments passed to the function.

### Value

The mean of the function evaluated at different parameter values.

---

median_hilow_poped	<i>Wrap summary functions from Hmisc and ggplot to work with stat_summary in ggplot</i>
--------------------	---

---

### Description

Created for back compatibility with older versions of ggplot, and so that PopED does not have to load ggplot when started.

### Usage

```
median_hilow_poped(x, ...)
```

### Arguments

x	A numeric vector
...	Additional arguments passed to Hmisc's smedian.hilow function or ggplot2's median_hilow function, depending on your version of ggplot.

---

model_prediction	<i>Model predictions</i>
------------------	--------------------------

---

### Description

Function generates a data frame of model predictions for the typical value in the population, individual predictions and data predictions. The function can also be used to generate datasets without predictions using the design specified in the arguments.

### Usage

```
model_prediction(
  poped.db = NULL,
  design = list(xt = poped.db$design[["xt"]], groupsize = poped.db$design$groupsize, m =
    poped.db$design[["m"]], x = poped.db$design[["x"]], a = poped.db$design[["a"]], ni =
    poped.db$design$ni, model_switch = poped.db$design$model_switch),
  model = list(fg_pointer = poped.db$model$fg_pointer, ff_pointer =
    poped.db$model$ff_pointer, ferror_pointer = poped.db$model$ferror_pointer),
  parameters = list(docc = poped.db$parameters$docc, d = poped.db$parameters$d, bpop =
    poped.db$parameters$bpop, covd = poped.db$parameters$covd, covdocc =
    poped.db$parameters$covdocc, sigma = poped.db$parameters$sigma),
  IPRED = FALSE,
  DV = FALSE,
  dosing = NULL,
  predictions = NULL,
  filename = NULL,
```

```

models_to_use = "all",
model_num_points = NULL,
model_minxt = NULL,
model_maxxt = NULL,
include_sample_times = T,
groups_to_use = "all",
include_a = TRUE,
include_x = TRUE,
manipulation = NULL,
PI = FALSE,
PI_conf_level = 0.95,
PI_ln_dist = TRUE
)

```

### Arguments

poped.db	A PopED database created by <a href="#">create.poped.database</a> .
design	A list that is passed as arguments to the function <a href="#">create_design</a> to create a design object.
model	A list containing the model elements to use for the predictions
parameters	A list of parameters to use in the model predictions.
IPRED	Should we simulate individual predictions?
DV	should we simulate observations?
dosing	A list of lists that adds dosing records to the data frame (Each inner list corresponding to a group in the design).
predictions	Should the resulting data frame have predictions? Either TRUE or FALSE or NULL in which case the function decides based on other arguments.
filename	A filename that the data frame should be written to in comma separate value (csv) format.
models_to_use	Which model numbers should we use? Model numbers are defined in design below using <code>model_switch</code> . For an explanation see <a href="#">create_design</a> .
model_num_points	How many extra observation rows should be created in the data frame for each group or individual per model. If used then the points are placed evenly between <code>model_minxt</code> and <code>model_maxxt</code> . This option is used by <a href="#">plot_model_prediction</a> to simulate the response of the model on a finer grid than the defined design. If NULL then only the input design is used. Can be a single value or a vector the same length as the number of models.
model_minxt	The minimum time value for extra observation rows indicated by <code>model_num_points</code> . A vector the same length as the number of models
model_maxxt	The maximum time value for extra observation rows indicated by <code>model_num_points</code> . A vector the same length as the number of models
include_sample_times	Should observations rows in the output data frame include the times indicated in the input design?

groups_to_use	Which groups should we include in the output data frame? Allowed values are "all" or a vector of numbers indicating the groups to include, e.g. c(1, 3, 6).
include_a	Should we include the continuous design variables in the output?
include_x	Should we include the discrete design variables in the output?
manipulation	A list of one or more <a href="#">expression</a> arguments. Each expression is evaluated using the code <code>for(i in 1:length(manipulation)){df &lt;- within(df, {eval(manipulation[[i]]})}</code> . Can be used to transform or create new columns in the resulting data frame. Note that these transformations are created after any model predictions occur, so transformations in columns having to do with input to model predictions will not affect the predictions.
PI	Compute prediction intervals for the data given the model. Predictions are based on first-order approximations to the model variance and a log-normality assumption of that variance (by default), if all predictions are positive, otherwise a normal distribution is assumed.
PI_conf_level	The confidence level for the prediction interval computed.
PI_ln_dist	Should the PI calculation be done assuming log-normal or a normal distribution. TRUE is the default and indicates a log-normal distribution. If any of the PRED values from the model are negative then a normal distribution is assumed.

### Value

A dataframe containing a design and (potentially) simulated data with some dense grid of samples and/or based on the input design.

### See Also

Other `evaluate_design`: [evaluate.fim\(\)](#), [evaluate\\_design\(\)](#), [evaluate\\_power\(\)](#), [get\\_rse\(\)](#), [plot\\_efficiency\\_of\\_windows\(\)](#), [plot\\_model\\_prediction\(\)](#)

Other Simulation: [plot\\_efficiency\\_of\\_windows\(\)](#), [plot\\_model\\_prediction\(\)](#)

### Examples

```
## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

library(PopED)

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.md.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
```



```

        Favail=bpop[4],
        DOSE=a[1])
    return(parameters)
}

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=0.01,
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0,
                                maxx=120,
                                a=70)

## data frame with model predictions
df_1 <- model_prediction(poped.db)
head(df_1,n=20)

## data frame with variability
df_2 <- model_prediction(poped.db,DV=TRUE)
head(df_2,n=20)

## data frame with variability (only IPRED, no DV)
df_3 <- model_prediction(poped.db,IPRED=TRUE)
head(df_3,n=20)

## data frame with model predictions, no continuous design variables in data frame
df_4 <- model_prediction(poped.db,include_a = FALSE)
head(df_4,n=20)

## -- 2 groups
poped.db.2 <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                   fg_fun=sfg,
                                   fError_fun=feps.prop,
                                   bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                   notfixed_bpop=c(1,1,1,0),
                                   d=c(CL=0.07, V=0.02, KA=0.6),
                                   sigma=0.01,
                                   groupsize=rbind(3,3),
                                   m=2,
                                   xt=c( 0.5,1,2,6,24,36,72,120),
                                   minxt=0,
                                   maxx=120,
                                   a=rbind(70,50))

df_5 <- model_prediction(poped.db.2,DV=TRUE)
head(df_5,n=20)

```

```

## without a popped database, just describing the design
## Useful for creating datasets for use in other software (like NONMEM)
design_1 <- list(
  xt=c( 0.5,1,2,6,24,36,72,120),
  m=2,
  groupsize=3)

design_2 <- list(
  xt=c( 0.5,1,2,6,24,36,72,120),
  m=2,
  groupsize=3,
  a=c(WT=70,AGE=50))

design_3 <- list(
  xt=c( 0.5,1,2,6,24,36,72,120),
  m=2,
  groupsize=3,
  a=list(c(WT=70,AGE=50),c(AGE=45,WT=60)))

(df_6 <- model_prediction(design=design_1))
(df_7 <- model_prediction(design=design_2))
(df_8 <- model_prediction(design=design_3))
(df_9 <- model_prediction(design=design_3,DV=TRUE))

# generate random deviations in WT for each individual
df_10 <- model_prediction(design=design_3,DV=TRUE,
  manipulation=expression({for(id in unique(ID))
    WT[ID==id] = rnorm(1,WT[ID==id],WT[ID==id]*0.1);id <- NULL}))

head(df_10,n=20)

# generate random deviations in WT and AGE for each individual
df_11 <- model_prediction(design=design_3,DV=TRUE,
  manipulation=list(
    expression(for(id in unique(ID))
      WT[ID==id] = rnorm(1,WT[ID==id],WT[ID==id]*0.1)),
    expression(for(id in unique(ID))
      AGE[ID==id] = rnorm(1,AGE[ID==id],AGE[ID==id]*0.2)),
    expression(id <- NULL)
  ))

head(df_11,n=20)

## create dosing rows
dosing_1 <- list(list(AMT=1000,RATE=NA,Time=0.5),list(AMT=3000,RATE=NA,Time=0.5))
dosing_2 <- list(list(AMT=1000,RATE=NA,Time=0.5))
dosing_3 <- list(list(AMT=1000,Time=0.5))
dosing_4 <- list(list(AMT=c(1000,20),Time=c(0.5,10))) # multiple dosing

(df_12 <- model_prediction(design=design_3,DV=TRUE,dosing=dosing_1))
(df_13 <- model_prediction(design=design_3,DV=TRUE,dosing=dosing_2))
(df_14 <- model_prediction(design=design_3,DV=TRUE,dosing=dosing_3))
(df_15 <- model_prediction(design=design_3,DV=TRUE,dosing=dosing_4))

```

```

model_prediction(design=design_3,DV=TRUE,dosing=dosing_4,model_num_points = 10)
model_prediction(design=design_3,DV=TRUE,dosing=dosing_4,model_num_points = 10,model_minxt=20)

design_4 <- list(
  xt=c( 0.5,1,2,6,24,36,72,120),
  model_switch=c(1,1,1,1,2,2,2,2),
  m=2,
  groupsize=3,
  a=list(c(WT=70,AGE=50),c(AGE=45,WT=60)))

model_prediction(design=design_4,DV=TRUE,dosing=dosing_4)
model_prediction(design=design_4,DV=TRUE,dosing=dosing_4,model_num_points = 10)
model_prediction(design=design_4,DV=TRUE,dosing=dosing_4,model_num_points = 10,
  model_minxt=10,model_maxxt=100)
model_prediction(design=design_4,DV=TRUE,dosing=dosing_4,model_num_points = 10,
  model_minxt=c(20,20),model_maxxt=c(100,100))
model_prediction(design=design_4,DV=TRUE,dosing=dosing_4,model_num_points = c(10,10),
  model_minxt=c(20,20),model_maxxt=c(100,100))

```

---

ofv\_criterion

*Normalize an objective function by the size of the FIM matrix*


---

### Description

Compute a normalized OFV based on the size of the FIM matrix. This value can then be used in efficiency calculations. This is NOT the OFV used in optimization, see [ofv\\_fim](#).

### Usage

```

ofv_criterion(
  ofv_f,
  num_parameters,
  poped.db,
  ofv_calc_type = poped.db$settings$ofv_calc_type
)

```

### Arguments

ofv_f	An objective function
num_parameters	The number of parameters to use for normalization
poped.db	a poped database
ofv_calc_type	OFV calculation type for FIM <ul style="list-style-type: none"> <li>1 = "D-optimality". Determinant of the FIM: <math>\det(\text{FIM})</math></li> <li>2 = "A-optimality". Inverse of the sum of the expected parameter variances: <math>1/\text{trace\_matrix}(\text{inv}(\text{FIM}))</math></li> </ul>

- 4 = "lnD-optimality". Natural logarithm of the determinant of the FIM:  $\log(\det(\text{FIM}))$
- 6 = "Ds-optimality". Ratio of the Determinant of the FIM and the Determinant of the uninteresting rows and columns of the FIM:  $\det(\text{FIM})/\det(\text{FIM}_u)$
- 7 = Inverse of the sum of the expected parameter RSE:  $1/\text{sum}(\text{get\_rse}(\text{FIM}, \text{poped.db}, \text{use\_percent}=\text{FA}))$

## Value

The specified criterion value.

## See Also

Other FIM: [LinMatrixH\(\)](#), [LinMatrixLH\(\)](#), [LinMatrixL\\_occ\(\)](#), [calc\\_ofv\\_and\\_fim\(\)](#), [ed\\_laplace\\_ofv\(\)](#), [ed\\_mftot\(\)](#), [efficiency\(\)](#), [evaluate.e.ofv.fim\(\)](#), [evaluate.fim\(\)](#), [gradf\\_eps\(\)](#), [mf3\(\)](#), [mf7\(\)](#), [mftot\(\)](#), [ofv\\_fim\(\)](#)

## Examples

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin model for optimization)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## Optimization using an additive + proportional residual error
## to avoid sample times at very low concentrations (time 0 or very late samples).

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
```

```

notfixed_bpop=c(1,1,1,0),
d=c(CL=0.07, V=0.02, KA=0.6),
sigma=c(prop=0.01,add=0.25),
groupsize=32,
xt=c( 0.5,1,2,6,24,36,72,120),
minxt=0.01,
maxxt=120,
a=c(DOSE=70),
mina=c(DOSE=0.01),
maxa=c(DOSE=100))

##### END #####
## Create PopED database
## (warfarin model for optimization)
#####

## evaluate initial design
FIM <- evaluate.fim(poped.db) # new name for function needed
FIM
get_rse(FIM,poed.db)

ofv_criterion(ofv_fim(FIM,poed.db,ofv_calc_type=1),
              length(get_unfixed_params(poed.db)[["all"]]),
              poed.db,
              ofv_calc_type=1) # det(FIM)

ofv_criterion(ofv_fim(FIM,poed.db,ofv_calc_type=2),
              length(get_unfixed_params(poed.db)[["all"]]),
              poed.db,
              ofv_calc_type=2)

ofv_criterion(ofv_fim(FIM,poed.db,ofv_calc_type=4),
              length(get_unfixed_params(poed.db)[["all"]]),
              poed.db,
              ofv_calc_type=4)

ofv_criterion(ofv_fim(FIM,poed.db,ofv_calc_type=6),
              length(get_unfixed_params(poed.db)[["all"]]),
              poed.db,
              ofv_calc_type=6)

ofv_criterion(ofv_fim(FIM,poed.db,ofv_calc_type=7),
              length(get_unfixed_params(poed.db)[["all"]]),
              poed.db,
              ofv_calc_type=7)

```

**Description**

Compute a criterion of the FIM given the model, parameters, design and methods defined in the PopED database.

**Usage**

```
ofv_fim(
  fmf,
  poped.db,
  ofv_calc_type = poped.db$settings$ofv_calc_type,
  ds_index = poped.db$parameters$ds_index,
  use_log = TRUE,
  ...
)
```

**Arguments**

fmf	The FIM
poped.db	A poped database
ofv_calc_type	OFV calculation type for FIM <ul style="list-style-type: none"> <li>• 1 = "D-optimality". Determinant of the FIM: <math>\det(\text{FIM})</math></li> <li>• 2 = "A-optimality". Inverse of the sum of the expected parameter variances: <math>1/\text{trace\_matrix}(\text{inv}(\text{FIM}))</math></li> <li>• 4 = "lnD-optimality". Natural logarithm of the determinant of the FIM: <math>\log(\det(\text{FIM}))</math></li> <li>• 6 = "Ds-optimality". Ratio of the Determinant of the FIM and the Determinant of the uninteresting rows and columns of the FIM: <math>\det(\text{FIM})/\det(\text{FIM}_u)</math></li> <li>• 7 = Inverse of the sum of the expected parameter RSE: <math>1/\text{sum}(\text{get\_rse}(\text{FIM}, \text{poped.db}, \text{use\_percent}=\text{FA}))</math></li> </ul>
ds_index	Ds_index is a vector set to 1 if a parameter is uninteresting, otherwise 0. size=(1,num unfixed parameters). First unfixed bpop, then unfixed d, then unfixed docc and last unfixed sigma. Default is the fixed effects being important, everything else not important. Used in conjunction with ofv_calc_type=6.
use_log	Should the criterion be in the log domain?
...	arguments passed to <a href="#">evaluate.fim</a> and <a href="#">ofv_fim</a> .

**Value**

The specified criterion value.

**See Also**

Other FIM: [LinMatrixH\(\)](#), [LinMatrixLH\(\)](#), [LinMatrixL\\_occ\(\)](#), [calc\\_ofv\\_and\\_fim\(\)](#), [ed\\_laplace\\_ofv\(\)](#), [ed\\_mftot\(\)](#), [efficiency\(\)](#), [evaluate.e.ofv.fim\(\)](#), [evaluate.fim\(\)](#), [gradf\\_eps\(\)](#), [mf3\(\)](#), [mf7\(\)](#), [mftot\(\)](#), [ofv\\_criterion\(\)](#)

Other evaluate\_FIM: [calc\\_ofv\\_and\\_fim\(\)](#), [evaluate.e.ofv.fim\(\)](#), [evaluate.fim\(\)](#)

## Examples

```

library(PopED)

##### START #####
## Create PopED database
## (warfarin model for optimization)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## Optimization using an additive + proportional residual error
## to avoid sample times at very low concentrations (time 0 or very late samples).

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(prop=0.01, add=0.25),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0.01,
                                maxx=120,
                                a=c(DOSE=70),
                                mina=c(DOSE=0.01),
                                maxa=c(DOSE=100))

##### END #####
## Create PopED database
## (warfarin model for optimization)
#####

```

```
## evaluate initial design
FIM <- evaluate.fim(poped.db)
FIM
get_rse(FIM, poped.db)

det(FIM)
ofv_fim(FIM, poped.db, ofv_calc_type=1) # det(FIM)
ofv_fim(FIM, poped.db, ofv_calc_type=2) # 1/trace_matrix(inv(FIM))
ofv_fim(FIM, poped.db, ofv_calc_type=4) # log(det(FIM))
ofv_fim(FIM, poped.db, ofv_calc_type=6) # Ds with fixed effects as "important"
ofv_fim(FIM, poped.db, ofv_calc_type=6,
        ds_index=c(1,1,1,0,0,0,1,1)) # Ds with random effects as "important"
ofv_fim(FIM, poped.db, ofv_calc_type=7) # 1/sum(get_rse(FIM, poped.db, use_percent=FALSE))
```

---

ones

*Create a matrix of ones*

---

### Description

Create a matrix of ones of size (dim1 x dim2).

### Usage

```
ones(dim1, dim2 = NULL)
```

### Arguments

dim1	The dimension of the matrix (if square) or the number of rows.
dim2	The number of columns

### Value

A matrix of ones

### See Also

Other MATLAB: [cell\(\)](#), [diag\\_matlab\(\)](#), [feval\(\)](#), [fileparts\(\)](#), [isempty\(\)](#), [rand\(\)](#), [randn\(\)](#), [size\(\)](#), [tic\(\)](#), [toc\(\)](#), [zeros\(\)](#)

### Examples

```
ones(4)
ones(3,4)
```



---

optimize\_groupsize      *Title Optimize the proportion of individuals in the design groups*

---

### Description

Title Optimize the proportion of individuals in the design groups

### Usage

```
optimize_groupsize(
  poped.db,
  props = c(poped.db$design$groupsize/sum(poped.db$design$groupsize)),
  trace = 1,
  ...
)
```

### Arguments

poped.db	A PopED database.
props	The proportions of individuals in each group (relative to the total number of individuals) to start the optimization from.
trace	Should there be tracing of the optimization? Value can be integer values. Larger numbers give more information.
...	Arguments passed to <code>ofv_fim</code> and <code>optim</code>

### Value

A list of the initial objective function value, optimal proportions, the objective function value with those proportions, the optimal number of individuals in each group (with integer number of individuals), and the objective function value with that number of individuals.

### Examples

```
# 2 design groups with either early or late samples
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
  fg_fun=function(x,a,bpop,b,bocc){
    parameters=c(CL=bpop[1]*exp(b[1]),
      V=bpop[2]*exp(b[2]),
      KA=bpop[3]*exp(b[3]),
      Favail=bpop[4],
      DOSE=a[1])
    return(parameters)
  },
  fError_fun=feps.add.prop,
  bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
  notfixed_bpop=c(1,1,1,0),
  d=c(CL=0.07, V=0.02, KA=0.6),
  sigma=c(0.01,0.25),
```

```

xt=list(c(1,2,3),c(4,5,20,120)),
groupsize=50,
minxt=0.01,
maxxt=120,
a=70,
mina=0.01,
maxa=100)

plot_model_prediction(poped.db)

evaluate_design(poped.db)

# what are the optimal proportions of
# individuals in the two groups in the study?
(n_opt <- optimize_groupsize(poped.db))

# How many individuals in the original design are needed to achieve an
# efficiency of 1 compared to the optimized design with n=100?
optimize_n_eff(poped.db,
               ofv_ref=n_opt$opt_ofv_with_n)

```

---

optimize_n_eff	<i>Translate efficiency to number of subjects</i>
----------------	---

---

### Description

optimize HOW MANY n there should be to achieve efficiency=1 compared to a reference OFV

### Usage

```
optimize_n_eff(poped.db, ofv_ref, norm_group_fim = NULL, ...)
```

### Arguments

poped.db	A PopED database.
ofv_ref	A reference OFV value to compare to.
norm_group_fim	The FIM per individual in each design group. If NULL, then these are computed.
...	Arguments passed to <a href="#">evaluate.fim</a> and efficiency.

### Value

The number of individuals needed.

**Examples**

```

# 2 design groups with either early or late samples
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=function(x,a,bpop,b,bocc){
                                  parameters=c(CL=bpop[1]*exp(b[1]),
                                                V=bpop[2]*exp(b[2]),
                                                KA=bpop[3]*exp(b[3]),
                                                Favail=bpop[4],
                                                DOSE=a[1])
                                  return(parameters)
                                },
                                fError_fun=feps.add.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(0.01,0.25),
                                xt=list(c(1,2,3),c(4,5,20,120)),
                                groupsize=50,
                                minxt=0.01,
                                maxx=120,
                                a=70,
                                mina=0.01,
                                maxa=100)

plot_model_prediction(poped.db)

evaluate_design(poped.db)

# what are the optimal proportions of
# individuals in the two groups in the study?
(n_opt <- optimize_groupsize(poped.db))

# How many individuals in the original design are needed to achieve an
# efficiency of 1 compared to the optimized design with n=100?
optimize_n_eff(poped.db,
               ofv_ref=n_opt$opt_ofv_with_n)

```

---

optimize\_n\_rse

*Optimize the number of subjects based on desired uncertainty of a parameter.*

---

**Description**

Optimize the number of subjects, based on the current design and the desired uncertainty of a single parameter

**Usage**

```
optimize_n_rse(
  poped.db,
  bpop_idx,
  need_rse,
  use_percent = TRUE,
  allowed_values = seq(poped.db$design$m, sum(poped.db$design$groupsize) * 5, by =
    poped.db$design$m)
)
```

**Arguments**

poped.db	A PopED database.
bpop_idx	The index number of the parameter, currently only bpop parameters are allowed.
need_rse	The relative standard error (RSE) one would like to achieve (in percent, by default).
use_percent	Should the RSE be represented as a percentage (T/F)?
allowed_values	A vector of the allowed total number of subjects in the study.

**Value**

The total number of subjects needed and the RSE of the parameter.

**Examples**

```
# 2 design groups with either early or late samples
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
  fg_fun=function(x,a,bpop,b,bocc){
    parameters=c(CL=bpop[1]*exp(b[1]),
      V=bpop[2]*exp(b[2]),
      KA=bpop[3]*exp(b[3]),
      Favail=bpop[4],
      DOSE=a[1])
    return(parameters)
  },
  fError_fun=feps.add.prop,
  bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
  notfixed_bpop=c(1,1,1,0),
  d=c(CL=0.07, V=0.02, KA=0.6),
  sigma=c(0.01,0.25),
  xt=list(c(1,2,3),c(4,5,20,120)),
  groupsize=50,
  minxt=0.01,
  maxx=120,
  a=70,
  mina=0.01,
  maxa=100)

# plot of the design
```

```

plot_model_prediction(poped.db)

# the current RSE values
evaluate_design(poped.db)$rse

# number of individuals if CL should have 10% RSE
optimize_n_rse(poped.db,
               bpop_idx=1, # for CL
               need_rse=10) # the RSE you want

```

---

optim\_ARS

*Optimize a function using adaptive random search.*


---

### Description

Optimize an objective function using an adaptive random search algorithm. The function works for both discrete and continuous optimization parameters and allows for box-constraints and sets of allowed values.

### Usage

```

optim_ARS(
  par,
  fn,
  lower = NULL,
  upper = NULL,
  allowed_values = NULL,
  loc_fac = 4,
  no_bounds_sd = par,
  iter = 400,
  iter_adapt = 50,
  adapt_scale = 1,
  max_run = 200,
  trace = TRUE,
  trace_iter = 5,
  new_par_max_it = 200,
  maximize = F,
  parallel = F,
  parallel_type = NULL,
  num_cores = NULL,
  mrgsolve_model = NULL,
  seed = round(runif(1, 0, 1e+07)),
  allow_replicates = TRUE,
  replicates_index = seq(1, length(par)),
  generator = NULL,
  ...
)

```

**Arguments**

par	A vector of initial values for the parameters to be optimized over.
fn	A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.
lower	Lower bounds on the parameters. A vector.
upper	Upper bounds on the parameters. A vector.
allowed_values	A list containing allowed values for each parameter list(par1=c(2,3,4,5,6),par2=c(5,6,7,8)). A vector containing allowed values for all parameters is also allowed c(2,3,4,5,6).
loc_fac	Locality factor for determining the standard deviation of the sampling distribution around the current position of the parameters. The initial standard deviation is normally calculated as (upper - lower)/loc_fac except in cases when there are no upper or lower limits (e.g. when upper=Inf or lower=-Inf).
no_bounds_sd	The standard deviation of the sampling distribution around the current position of the parameters when there are no upper or lower limits (e.g. when upper=Inf or lower=-Inf).
iter	The number of iterations for the algorithm to perform (this is a maximum number, it could be less).
iter_adapt	The number of iterations before adapting (shrinking) the parameter search space.
adapt_scale	The scale for adapting the size of the sampling distribution. The adaptation of the standard deviation of the sampling distribution around the current position of the parameters is done after iter_adapt iteration with no change in the best objective function. When adapting, the standard deviation of the sampling distribution is calculated as (upper - lower)/(loc_fac*ff*adapt_scale) where ff starts at 1 and increases by 1 for each adaptation.
max_run	The maximum number of iterations to run without a change in the best parameter estimates.
trace	Should the algorithm output results intermittently.
trace_iter	How many iterations between each update to the screen about the result of the search.
new_par_max_it	The algorithm randomly chooses samples based on the current best set of parameters. If when drawing these samples the new parameter set has already been tested then a new draw is performed. After new_par_max_it draws, with no new parameter sets, then the algorithm stops.
maximize	Should the function be maximized? Default is to minimize.
parallel	Should we use parallel computations?
parallel_type	Which type of parallelization should be used? Can be "snow" or "multicore". "snow" works on Linux-like systems & Windows. "multicore" works only on Linux-like systems. By default this is chosen for you depending on your operating system. See <a href="#">start_parallel</a> .
num_cores	The number of cores to use in the parallelization. By default is set to the number output from parallel::detectCores(). See <a href="#">start_parallel</a> .

mrgsolve_model	If the computations require a mrgsolve model and you are using the "snow" method then you need to specify the name of the model object created by mread or mcode.
seed	The random seed to use in the algorithm,
allow_replicates	Should the algorithm allow parameters to have the same value?
replicates_index	A vector, the same length as the parameters. If two values are the same in this vector then the parameters may not assume the same value in the optimization.
generator	A user-defined function that generates new parameter sets to try in the algorithm. See examples below.
...	Additional arguments passed to fn and start_parallel.

## References

1. M. Foracchia, A.C. Hooker, P. Vicini and A. Ruggeri, "PopED, a software fir optimal experimental design in population kinetics", Computer Methods and Programs in Biomedicine, 74, 2004.
2. J. Nyberg, S. Ueckert, E.A. Stroemberg, S. Hennig, M.O. Karlsson and A.C. Hooker, "PopED: An extended, parallelized, nonlinear mixed effects models optimal design tool", Computer Methods and Programs in Biomedicine, 108, 2012.

## See Also

Other Optimize: [Doptim\(\)](#), [LEDoptim\(\)](#), [RS\\_opt\(\)](#), [a\\_line\\_search\(\)](#), [bfgsb\\_min\(\)](#), [calc Autofocus\(\)](#), [calc\\_ofv\\_and\\_grad\(\)](#), [mfea\(\)](#), [optim\\_LS\(\)](#), [poped\\_optim\(\)](#), [poped\\_optim\\_1\(\)](#), [poped\\_optim\\_2\(\)](#), [poped\\_optim\\_3\(\)](#), [poped\\_optimize\(\)](#)

## Examples

```
## "wild" function , global minimum at about -15.81515
fw <- function(x) 10*sin(0.3*x)*sin(1.3*x^2) + 0.00001*x^4 + 0.2*x+80

# optimization with fewer function evaluations compared to SANN
res1 <- optim_ARS(50, fw, lower = -50, upper=100)

# often not as good performance when upper and lower bounds are poor
res2 <- optim_ARS(50, fw, lower=-Inf, upper=Inf)

# Only integer values allowed
## Not run:
res_int <- optim_ARS(50, fw, allowed_values = seq(-50,100,by=1))

## End(Not run)

## Not run:
#plot of the function and solutions
require(graphics)
plot(fw, -50, 50, n = 1000, main = "Minimizing 'wild function'")
```

```

points(-15.81515, fw(-15.81515), pch = 16, col = "red", cex = 1)
points(res1$par, res1$ofv, pch = 16, col = "green", cex = 1)
points(res2$par, res2$ofv, pch = 16, col = "blue", cex = 1)

## End(Not run)

# optim_ARS does not work great for hard to find minima on flat surface:
# Rosenbrock Banana function
#  $f(x, y) = (a-x)^2 + b(y-x^2)^2$ 
# global minimum at  $(x, y)=(a, a^2)$ , where  $f(x, y)=0$ .
# Usually  $a = 1$  and  $b = 100$ .
## Not run:
fr <- function(x,a=1,b=100) {
  x1 <- x[1]
  x2 <- x[2]
  b*(x2 - x1*x1)^2 + (a - x1)^2
}

res3 <- optim_ARS(c(-1.2,1), fr,lower = -5, upper = 5)

# plot the surface
x <- seq(-50, 50, length= 30)
y <- x
f <- function(x,y){apply(cbind(x,y),1,fr)}
z <- outer(x, y, f)
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue", ticktype="detailed") -> res
points(trans3d(1, 1, 0, pmat = res), col = 2, pch = 16,cex=2)
points(trans3d(res3$par[1], res3$par[1], res3$ofv, pmat = res), col = "green", pch = 16,cex=2)

## End(Not run)

# box constraints
flb <- function(x){
  p <- length(x)
  sum(c(1, rep(4, p-1)) * (x - c(1, x[-p]))^2)^2)
}
## 25-dimensional box constrained
#optim(rep(3, 25), flb,lower = rep(2, 25), upper = rep(4, 25),method = "L-BFGS-B")
res_box <- optim_ARS(rep(3, 25), flb,lower = rep(2, 25), upper = rep(4, 25))

## Combinatorial optimization: Traveling salesman problem
eurodistmat <- as.matrix(eurodist)

distance <- function(sq) { # Target function
  sq2 <- embed(sq, 2)
  sum(eurodistmat[cbind(sq2[,2], sq2[,1])])
}

genseq <- function(sq) { # Generate new candidate sequence
  idx <- seq(2, NROW(eurodistmat)-1)
  changepoints <- sample(idx, size = 2, replace = FALSE)
  tmp <- sq[changepoints[1]]

```



```

sq[changepoints[1]] <- sq[changepoints[2]]
sq[changepoints[2]] <- tmp
sq
}

sq <- c(1:nrow(eurodistmat), 1) # Initial sequence: alphabetic
res3 <- optim_ARS(sq,distance,generator=genseq) # Near optimum distance around 12842

## Not run:
# plot of initial sequence
# rotate for conventional orientation
loc <- -cmdscale(eurodist, add = TRUE)$points
x <- loc[,1]; y <- loc[,2]
s <- seq_len(nrow(eurodistmat))
tspinit <- loc[sq,]

plot(x, y, type = "n", asp = 1, xlab = "", ylab = "",
     main = paste("Initial sequence of traveling salesman problem\n",
                  "Distance =",distance(sq)), axes = FALSE)
arrows(tspinit[s,1], tspinit[s,2], tspinit[s+1,1], tspinit[s+1,2],
       angle = 10, col = "green")
text(x, y, labels(eurodist), cex = 0.8)

# plot of final sequence from optim_ARS
tspres <- loc[res3$par,]
plot(x, y, type = "n", asp = 1, xlab = "", ylab = "",
     main = paste("optim_ARS() 'solving' traveling salesman problem\n",
                  "Distance =",distance(c(1,res3$par,1))), axes = FALSE)
arrows(tspres[s,1], tspres[s,2], tspres[s+1,1], tspres[s+1,2],
       angle = 10, col = "red")
text(x, y, labels(eurodist), cex = 0.8)

# using optim
set.seed(123) # chosen to get a good soln relatively quickly
(res4 <- optim(sq, distance, genseq, method = "SANN",
              control = list(maxit = 30000, temp = 2000, trace = TRUE,
                             REPORT = 500)))

tspres <- loc[res4$par,]
plot(x, y, type = "n", asp = 1, xlab = "", ylab = "",
     main = paste("optim() 'solving' traveling salesman problem\n",
                  "Distance =",distance(res4$par)), axes = FALSE)
arrows(tspres[s,1], tspres[s,2], tspres[s+1,1], tspres[s+1,2],
       angle = 10, col = "red")
text(x, y, labels(eurodist), cex = 0.8)

## End(Not run)

# one-dimensional function
## Not run:
f <- function(x) abs(x)+cos(x)
res5 <- optim_ARS(-20,f,lower=-20, upper=20)

```

```

curve(f, -20, 20)
abline(v = res5$par, lty = 4,col="green")

## End(Not run)

# one-dimensional function
f <- function(x) (x^2+x)*cos(x) # -10 < x < 10
res_max <- optim_ARS(0,f,lower=-10, upper=10,maximize=TRUE) # sometimes to local maxima

## Not run:
res_min <- optim_ARS(0,f,lower=-10, upper=10) # sometimes to local minima

curve(f, -10, 10)
abline(v = res_min$par, lty = 4,col="green")
abline(v = res_max$par, lty = 4,col="red")

## End(Not run)

# two-dimensional Rastrigin function
#It has a global minimum at  $f(x) = f(0) = 0$ .
## Not run:
Rastrigin <- function(x1, x2){
  20 + x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
}

x1 <- x2 <- seq(-5.12, 5.12, by = 0.1)
z <- outer(x1, x2, Rastrigin)

res6 <- optim_ARS(c(-4,4),function(x) Rastrigin(x[1], x[2]),lower=-5.12, upper=5.12)

# color scale
nrz <- nrow(z)
ncz <- ncol(z)
jet.colors <-
  colorRampPalette(c("#00007F", "blue", "#007FFF", "cyan",
                    "#7FFF7F", "yellow", "#FF7F00", "red", "#7F0000"))
# Generate the desired number of colors from this palette
nbc col <- 100
color <- jet.colors(nbc ol)
# Compute the z-value at the facet centres
zfacet <- z[-1, -1] + z[-1, -ncz] + z[-nrz, -1] + z[-nrz, -ncz]
# Recode facet z-values into color indices
facetcol <- cut(zfacet, nbc ol)
persp(x1, x2, z, col = color[facetcol], phi = 30, theta = 30)
filled.contour(x1, x2, z, color.palette = jet.colors)

## End(Not run)

## Parallel computation
## works better when each evaluation takes longer

```

```

## here we have added extra time to the computations
## just to show that it works
## Not run:
res7 <- optim_ARS(c(-4,4),function(x){Sys.sleep(0.01); Rastrigin(x[1], x[2])},
  lower=-5.12, upper=5.12)
res8 <- optim_ARS(c(-4,4),function(x){Sys.sleep(0.01); Rastrigin(x[1], x[2])},
  lower=-5.12, upper=5.12,parallel = T)
res9 <- optim_ARS(c(-4,4),function(x){Sys.sleep(0.01); Rastrigin(x[1], x[2])},
  lower=-5.12, upper=5.12,parallel = T,parallel_type = "snow")

## End(Not run)

```

---

optim\_LS

*Optimize a function using a line search algorithm.*


---

## Description

optim\_LS performs sequential grid search optimization of an arbitrary function with respect to each of the parameters to be optimized over. The function works for both discrete and continuous optimization parameters and allows for box-constraints (by using the upper and lower function arguments) or sets of allowed values (by using the allowed\_values function argument) for all parameters, or on a parameter per parameter basis.

## Usage

```

optim_LS(
  par,
  fn,
  lower = NULL,
  upper = NULL,
  allowed_values = NULL,
  line_length = 50,
  trace = TRUE,
  maximize = F,
  parallel = F,
  parallel_type = NULL,
  num_cores = NULL,
  mrgsolve_model = NULL,
  seed = round(runif(1, 0, 1e+07)),
  allow_replicates = TRUE,
  replicates_index = seq(1, length(par)),
  ofv_initial = NULL,
  closed_bounds = TRUE,
  ...
)

```

**Arguments**

par	A vector of initial values for the parameters to be optimized over.
fn	A function to be minimized (or maximized), with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.
lower	Lower bounds on the parameters. A vector.
upper	Upper bounds on the parameters. A vector.
allowed_values	A list containing allowed values for each parameter <code>list(par1=c(2,3,4,5,6),par2=c(5,6,7,8))</code> . A vector containing allowed values for all parameters is also allowed <code>c(2,3,4,5,6)</code> .
line_length	The number of different parameter values per parameter to evaluate. The values are selected as an evenly spaced grid between the upper and lower bounds.
trace	Should the algorithm output results intermittently.
maximize	Should the function be maximized? Default is to minimize.
parallel	Should we use parallel computations?
parallel_type	Which type of parallelization should be used? Can be "snow" or "multicore". "snow" works on Linux-like systems & Windows. "multicore" works only on Linux-like systems. By default this is chosen for you depending on your operating system. See <a href="#">start_parallel</a> .
num_cores	The number of cores to use in the parallelization. By default is set to the number output from <code>parallel::detectCores()</code> . See <a href="#">start_parallel</a> .
mrgsolve_model	If the computations require a mrgsolve model and you are using the "snow" method then you need to specify the name of the model object created by <code>mread</code> or <code>mcode</code> .
seed	The random seed to use in the algorithm,
allow_replicates	Should the algorithm allow parameters to have the same value?
replicates_index	A vector, the same length as the parameters. If two values are the same in this vector then the parameters may not assume the same value in the optimization.
ofv_initial	An initial objective function value (OFV). If not NULL then the initial design is not evaluated and the OFV value is assumed to be this number.
closed_bounds	Are the upper and lower limits open (boundaries not allowed) or closed (boundaries allowed) bounds?
...	Additional arguments passed to <code>fn</code> and <code>start_parallel</code> .

**References**

1. M. Foracchia, A.C. Hooker, P. Vicini and A. Ruggeri, "PopED, a software fir optimal experimental design in population kinetics", *Computer Methods and Programs in Biomedicine*, 74, 2004.
2. J. Nyberg, S. Ueckert, E.A. Stroemberg, S. Hennig, M.O. Karlsson and A.C. Hooker, "PopED: An extended, parallelized, nonlinear mixed effects models optimal design tool", *Computer Methods and Programs in Biomedicine*, 108, 2012.

**See Also**

Other Optimize: [Doptim\(\)](#), [LEDOptim\(\)](#), [RS\\_opt\(\)](#), [a\\_line\\_search\(\)](#), [bfgsb\\_min\(\)](#), [calc Autofocus\(\)](#), [calc\\_ofv\\_and\\_grad\(\)](#), [mfea\(\)](#), [optim\\_ARS\(\)](#), [poped\\_optim\(\)](#), [poped\\_optim\\_1\(\)](#), [poped\\_optim\\_2\(\)](#), [poped\\_optim\\_3\(\)](#), [poped\\_optimize\(\)](#)

**Examples**

```
# "wild" function
fw <- function(x) 10*sin(0.3*x)*sin(1.3*x^2) + 0.00001*x^4 + 0.2*x+80

# Global minimum of 67.47 at about -15.81515
(fw_min <- fw(-15.81515))

if (interactive()){
  #plot of the function
  require(graphics)
  plot(fw, -50, 50, n = 10000, main = "Minimizing 'wild function'")

  # Known minimum
  points(-15.81515, fw_min, pch = 21, col = "red", cex = 1.5)
}

# optimization with fewer function evaluations
# compared to SANN: see examples in '?optim'
res1 <- optim_LS(50, fw, lower = -50, upper=50, line_length = 10000)

if (interactive()){
  require(graphics)
  plot(fw, -20, 0, n = 10000, main = "Minimizing 'wild function'")

  # Known minimum
  points(-15.81515, fw_min, pch = 21, col = "red", cex = 1.5)

  #plot of the optimization
  points(res1$par, res1$ofv, pch = 16, col = "green", cex = 1)
}

# Upper and lower bounds and line_length should be considered carefully
res2 <- optim_LS(50, fw, lower=-Inf, upper=Inf, line_length = 10000)

# Only integer values allowed
res_int <- optim_LS(50, fw, allowed_values = seq(-50,50,by=1))

# Rosenbrock Banana function
# f(x, y) = (a-x)^2 + b*(y-x^2)^2
# global minimum at (x, y)=(a, a^2), where f(x, y)=0.
# Usually a = 1 and b = 100 so x=1 and y=1
if (interactive()){
  fr <- function(x,a=1,b=100) {
    x1 <- x[1]
    x2 <- x[2]
```

```

    b*(x2 - x1*x1)^2 + (a - x1)^2
  }

res3 <- optim_LS(c(-1.2,1), fr,lower = -5, upper = 5, line_length = 1000)

# plot the surface
x <- seq(-50, 50, length= 30)
y <- x
f <- function(x,y){apply(cbind(x,y),1,fr)}
z <- outer(x, y, f)
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue", ticktype="detailed") -> res
points(trans3d(1, 1, 0, pmat = res), col = 2, pch = 16,cex=2)
points(trans3d(res3$par[1], res3$par[1], res3$ofv, pmat = res), col = "green", pch = 16,cex=1.5)
}

# box constraints
flb <- function(x){
  p <- length(x)
  sum(c(1, rep(4, p-1)) * (x - c(1, x[-p])^2)^2)
}

## 25-dimensional box constrained
if (interactive()){
  optim(rep(3, 25), flb,lower = rep(2, 25), upper = rep(4, 25),method = "L-BFGS-B")
}
res_box <- optim_LS(rep(3, 25), flb,
                    lower = rep(2, 25),
                    upper = rep(4, 25),
                    line_length = 1000)

# one-dimensional function
if (interactive()){
  f <- function(x) abs(x)+cos(x)
  res5 <- optim_LS(-20,f,lower=-20, upper=20, line_length = 500)

  curve(f, -20, 20)
  abline(v = res5$par, lty = 4,col="green")
}

# one-dimensional function
f <- function(x) (x^2+x)*cos(x) # -10 < x < 10
res_max <- optim_LS(0,f,lower=-10, upper=10,maximize=TRUE,line_length = 1000)

if (interactive()){
  res_min <- optim_LS(0,f,lower=-10, upper=10, line_length = 1000)

  curve(f, -10, 10)
  abline(v = res_min$par, lty = 4,col="green")
  abline(v = res_max$par, lty = 4,col="red")
}

# two-dimensional Rastrigin function

```

```

#It has a global minimum at  $f(x) = f(0) = 0$ .
if (interactive()){
  Rastrigin <- function(x1, x2){
    20 + x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
  }

  x1 <- x2 <- seq(-5.12, 5.12, by = 0.1)
  z <- outer(x1, x2, Rastrigin)

  res6 <- optim_LS(c(-4,4),function(x) Rastrigin(x[1], x[2]),
    lower=-5.12, upper=5.12, line_length = 1000)

  # color scale
  nrz <- nrow(z)
  ncz <- ncol(z)
  jet.colors <-
    colorRampPalette(c("#00007F", "blue", "#007FFF", "cyan",
      "#7FFF7F", "yellow", "#FF7F00", "red", "#7F0000"))
  # Generate the desired number of colors from this palette
  nbcol <- 100
  color <- jet.colors(nbcol)
  # Compute the z-value at the facet centres
  zfacet <- z[-1, -1] + z[-1, -ncz] + z[-nrz, -1] + z[-nrz, -ncz]
  # Recode facet z-values into color indices
  facetcol <- cut(zfacet, nbcol)
  persp(x1, x2, z, col = color[facetcol], phi = 30, theta = 30)
  filled.contour(x1, x2, z, color.palette = jet.colors)
}

## Parallel computation
## works better when each evaluation takes longer
## here we have added extra time to the computations
## just to show that it works
if (interactive()){
  res7 <- optim_LS(c(-4,4),function(x){Sys.sleep(0.01); Rastrigin(x[1], x[2])},
    lower=-5.12, upper=5.12, line_length = 200)
  res8 <- optim_LS(c(-4,4),function(x){Sys.sleep(0.01); Rastrigin(x[1], x[2])},
    lower=-5.12, upper=5.12, line_length = 200, parallel = TRUE)
  res9 <- optim_LS(c(-4,4),function(x){Sys.sleep(0.01); Rastrigin(x[1], x[2])},
    lower=-5.12, upper=5.12, line_length = 200, parallel = TRUE,
    parallel_type = "snow")
}

```

---

pargen

---

*Parameter simulation*


---

## Description

Function generates random samples for a list of parameters

**Usage**

```
pargen(par, user_dist_pointer, sample_size, bLHS, sample_number, poped.db)
```

**Arguments**

par	A matrix describing the parameters. Each row is a parameter and the matrix has three columns: <ol style="list-style-type: none"> <li>1. First column - Type of distribution (0-fixed, 1-normal, 2-uniform, 3-user specified, 4-lognormal, 5-Truncated normal).</li> <li>2. Second column - Mean of distribution.</li> <li>3. Third column - Variance or range of distribution.</li> </ol>
user_dist_pointer	A text string of the name of a function that generates random samples from a user defined distribution.
sample_size	The number of random samples per parameter to generate
bLHS	Logical, indicating if Latin Hypercube Sampling should be used.
sample_number	The sample number to extract from a user distribution.
poped.db	A PopED database.

**Value**

A matrix of random samples of size (sample\_size x number\_of\_parameters)

**Examples**

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin example)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
```



```

}

## -- Define model, parameters, initial design
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(prop=0.01),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                a=c(DOSE=70))

##### END #####
## Create PopED database
## (warfarin example)
#####

# Adding 40% Uncertainty to fixed effects log-normal (not Favail)
bpop_vals <- c(CL=0.15, V=8, KA=1.0, Favail=1)
bpop_vals_ed_ln <- cbind(ones(length(bpop_vals),1)*4, # log-normal distribution
                        bpop_vals,
                        ones(length(bpop_vals),1)*(bpop_vals*0.4)^2) # 40% of bpop value
bpop_vals_ed_ln["Favail",] <- c(0,1,0)

pars.ln <- pargen(par=bpop_vals_ed_ln,
                 user_dist_pointer=NULL,
                 sample_size=1000,
                 bLHS=1,
                 sample_number=NULL,
                 poped.db)

# Adding 10% Uncertainty to fixed effects normal-distribution (not Favail)
bpop_vals_ed_n <- cbind(ones(length(bpop_vals),1)*1, # log-normal distribution
                       bpop_vals,
                       ones(length(bpop_vals),1)*(bpop_vals*0.1)^2) # 10% of bpop value
bpop_vals_ed_n["Favail",] <- c(0,1,0)

pars.n <- pargen(par=bpop_vals_ed_n,
                user_dist_pointer=NULL,
                sample_size=1000,
                bLHS=1,
                sample_number=NULL,
                poped.db)

# Adding 10% Uncertainty to fixed effects uniform-distribution (not Favail)
bpop_vals_ed_u <- cbind(ones(length(bpop_vals),1)*2, # uniform distribution
                       bpop_vals,
                       ones(length(bpop_vals),1)*(bpop_vals*0.1)) # 10% of bpop value

```

```

bpop_vals_ed_u["Favail",] <- c(0,1,0)

pars.u <- pargen(par=bpop_vals_ed_u,
                user_dist_pointer=NULL,
                sample_size=1000,
                bLHS=1,
                sample_number=NULL,
                poped.db)

# Adding user defined distributions
bpop_vals_ed_ud <- cbind(ones(length(bpop_vals),1)*3, # user dfined distribution
                        bpop_vals,
                        bpop_vals*0.1) # 10% of bpop value
bpop_vals_ed_ud["Favail",] <- c(0,1,0)

# A normal distribution
my_dist <- function(...){
  par_vec <- rnorm(c(1,1,1,1),mean=bpop_vals_ed_ud[,2],sd=bpop_vals_ed_ud[,3])
}

pars.ud <- pargen(par=bpop_vals_ed_ud,
                 user_dist_pointer=my_dist,
                 sample_size=1000,
                 bLHS=1,
                 sample_number=NULL,
                 poped.db)

```

---

plot\_efficiency\_of\_windows

*Plot the efficiency of windows*

---

## Description

Function plots the efficiency of windows around the sample time points. The function samples from a uniform distribution around the sample time points for each group (or each individual with `deviate_by_id=TRUE`, with slower calculation times) and compares the results with the design defined in `poped.db`. The maximal and minimal allowed values for all design variables as defined in `poped.db` are respected (e.g. `poped.db$design_space$minxt` and `poped.db$design_space$maxxt`).

## Usage

```

plot_efficiency_of_windows(
  poped.db,
  xt_windows = NULL,
  xt_plus = xt_windows,
  xt_minus = xt_windows,

```

```

    iNumSimulations = 100,
    y_eff = TRUE,
    y_rse = TRUE,
    ofv_calc_type = poped.db$settings$ofv_calc_type,
    mean_line = TRUE,
    mean_color = "red",
    deviate_by_id = FALSE,
    parallel = F,
    seed = round(runif(1, 0, 1e+07)),
    ...
)

```

### Arguments

poped.db	A poped database
xt_windows	The distance on one direction from the optimal sample times. Can be a number or a matrix of the same size as the xt matrix found in poped.db\$design\$xt.
xt_plus	The upper distance from the optimal sample times (xt + xt_plus). Can be a number or a matrix of the same size as the xt matrix found in poped.db\$design\$xt.
xt_minus	The lower distance from the optimal sample times (xt - xt_minus). Can be a number or a matrix of the same size as the xt matrix found in poped.db\$design\$xt.
iNumSimulations	The number of design simulations to make within the specified windows.
y_eff	Should one of the plots created have efficiency on the y-axis?
y_rse	Should created plots include the relative standard error of each parameter as a value on the y-axis?
ofv_calc_type	OFV calculation type for FIM <ul style="list-style-type: none"> <li>• 1 = "D-optimality". Determinant of the FIM: <math>\det(\text{FIM})</math></li> <li>• 2 = "A-optimality". Inverse of the sum of the expected parameter variances: <math>1/\text{trace\_matrix}(\text{inv}(\text{FIM}))</math></li> <li>• 4 = "lnD-optimality". Natural logarithm of the determinant of the FIM: <math>\log(\det(\text{FIM}))</math></li> <li>• 6 = "Ds-optimality". Ratio of the Determinant of the FIM and the Determinant of the uninteresting rows and columns of the FIM: <math>\det(\text{FIM})/\det(\text{FIM}_u)</math></li> <li>• 7 = Inverse of the sum of the expected parameter RSE: <math>1/\text{sum}(\text{get\_rse}(\text{FIM}, \text{poped.db}, \text{use\_percent}=\text{FA}))</math></li> </ul>
mean_line	Should a mean value line be created?
mean_color	The color of the mean value line.
deviate_by_id	Should the computations look at deviations per individual instead of per group?
parallel	Should we use parallel computations (T/F)? Other options can be defined in this function and passed to <a href="#">start_parallel</a> . See especially the options <code>d11s</code> and <code>mrgsolve_model</code> from that function if you have a model defined with compiled code or are using <code>mrgsolve</code> .
seed	The random seed to use.
...	Extra arguments passed to <code>evaluate.fim</code>



```

minxt=0.01,
maxxt=120,
a=c(DOSE=70),
mina=c(DOSE=0.01),
maxa=c(DOSE=100))

##### END #####
## Create PopED database
## (warfarin model for optimization)
#####

# Examine efficiency of sampling windows at plus/minus 0.5 hours from
# sample points in the design
plot_efficiency_of_windows(poped.db,xt_windows=0.5)

if(interactive()){

  plot_efficiency_of_windows(poped.db,
                             xt_plus=c( 0.5,1,2,1,2,3,7,1),
                             xt_minus=c( 0.1,2,5,4,2,3,6,2))

  plot_efficiency_of_windows(poped.db,xt_windows=c( 0.5,1,2,1,2,3,7,1))

  plot_efficiency_of_windows(poped.db,
                             xt_plus=c( 0.5,1,2,1,2,3,7,1),
                             xt_minus=c( 0.1,2,5,4,2,3,6,2),
                             y_rse=FALSE)

  plot_efficiency_of_windows(poped.db,
                             xt_plus=c( 0.5,1,2,1,2,3,7,1),
                             xt_minus=c( 0.1,2,5,4,2,3,6,2),
                             y_eff=FALSE)
}

```

---

plot\_model\_prediction *Plot model predictions*

---

### Description

Function plots model predictions for the typical value in the population, individual predictions and data predictions.

**Usage**

```

plot_model_prediction(
  poped.db,
  model_num_points = 100,
  groupsize_sim = 100,
  separate.groups = F,
  sample.times = T,
  sample.times.IPRED = F,
  sample.times.DV = F,
  PRED = T,
  IPRED = F,
  IPRED.lines = F,
  IPRED.lines.pctls = F,
  alpha.IPRED.lines = 0.1,
  alpha.IPRED = 0.3,
  sample.times.size = 4,
  DV = F,
  alpha.DV = 0.3,
  DV.lines = F,
  DV.points = F,
  alpha.DV.lines = 0.3,
  alpha.DV.points = 0.3,
  sample.times.DV.points = F,
  sample.times.DV.lines = F,
  alpha.sample.times.DV.points = 0.3,
  alpha.sample.times.DV.lines = 0.3,
  y_lab = "Model Predictions",
  facet_scales = "fixed",
  facet_label_names = T,
  model.names = NULL,
  DV.mean.sd = FALSE,
  PI = FALSE,
  PI_alpha = 0.3,
  ...
)

```

**Arguments**

poped.db	A PopED database.
model_num_points	How many extra observation rows should be created in the data frame for each group or individual per model. If used then the points are placed evenly between model_minxt and model_maxxt. This option is used by <a href="#">plot_model_prediction</a> to simulate the response of the model on a finer grid than the defined design. If NULL then only the input design is used. Can be a single value or a vector the same length as the number of models.
groupsize_sim	How many individuals per group should be simulated when DV=TRUE or IPRED=TRUE to create prediction intervals?

separate.groups	Should there be separate plots for each group.
sample.times	Should sample times be shown on the plots.
sample.times.IPRED	Should sample times be shown based on the IPRED y-values.
sample.times.DV	Should sample times be shown based on the DV y-values.
PRED	Should a PRED line be drawn.
IPRED	Should we simulate individual predictions?
IPRED.lines	Should IPRED lines be drawn?
IPRED.lines.pctls	Should lines be drawn at the chosen percentiles of the IPRED values?
alpha.IPRED.lines	What should the transparency for the IPRED.lines be?
alpha.IPRED	What should the transparency of the IPRED CI?
sample.times.size	What should the size of the sample.times be?
DV	should we simulate observations?
alpha.DV	What should the transparency of the DV CI?
DV.lines	Should DV lines be drawn?
DV.points	Should DV points be drawn?
alpha.DV.lines	What should the transparency for the DV.lines be?
alpha.DV.points	What should the transparency for the DV.points be?
sample.times.DV.points	TRUE or FALSE.
sample.times.DV.lines	TRUE or FALSE.
alpha.sample.times.DV.points	What should the transparency for the sample.times.DV.points be?
alpha.sample.times.DV.lines	What should the transparency for the sample.times.DV.lines be?
y_lab	The label of the y-axis.
facet_scales	Can be "free", "fixed", "free_x" or "free_y"
facet_label_names	TRUE or FALSE
model.names	A vector of names of the response model/s (the length of the vector should be equal to the number of response models). It is Null by default.
DV.mean.sd	Plot the mean and standard deviation of simulated observations.
PI	Plot prediction intervals for the expected data given the model. Predictions are based on first-order approximations to the model variance and a normality assumption of that variance. As such these computations are more approximate than using DV=T and groupsize_sim = some large number.
PI_alpha	The transparency of the PI.
...	Additional arguments passed to the <a href="#">model_prediction</a> function.

**Value**

A [ggplot](#) object. If you would like to further edit this plot don't forget to load the [ggplot2](#) library using `library(ggplot2)`.

**See Also**

[model\\_prediction](#)

Other evaluate\_design: [evaluate.fim\(\)](#), [evaluate\\_design\(\)](#), [evaluate\\_power\(\)](#), [get\\_rse\(\)](#), [model\\_prediction\(\)](#), [plot\\_efficiency\\_of\\_windows\(\)](#)

Other Simulation: [model\\_prediction\(\)](#), [plot\\_efficiency\\_of\\_windows\(\)](#)

Other Graphics: [plot\\_efficiency\\_of\\_windows\(\)](#)

**Examples**

```
## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

library(PopED)

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.md.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

## -- Define initial design and design space
poped.db <- create.poped.database(
  ff_fun=ff.PK.1.comp.oral.sd.CL,
  fg_fun=sfg,
  fError_fun=feps.prop,
  bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
  notfixed_bpop=c(1,1,1,0),
  d=c(CL=0.07, V=0.02, KA=0.6),
  sigma=0.01,
  groupsize=32,
  xt=c( 0.5,1,2,6,24,36,72,120),
  minxt=0,
  maxxt=120,
  a=70)
```



```

## create plot of model without variability
plot_model_prediction(poped.db)

## create plot of model with variability by simulating from OMEGA and SIGMA
plot_model_prediction(poped.db,IPRED=TRUE,DV=TRUE)

## create plot of model with variability by
## computing the expected variance (using an FO approximation)
## and then computing a prediction interval
## based on an assumption of normality
## computation is faster but less accurate
## compared to using DV=TRUE (and groupsize_sim = 500)
plot_model_prediction(poped.db,PI=TRUE)

##-- Model: One comp first order absorption + inhibitory imax
## -- works for both mutiple and single dosing
ff <- function(model_switch,xt,parameters,poped.db){
  with(as.list(parameters),{

    y=xt
    MS <- model_switch

    # PK model
    N = floor(xt/TAU)+1
    CONC=(DOSE*Favail/V)*(KA/(KA - CL/V)) *
      (exp(-CL/V * (xt - (N - 1) * TAU)) * (1 - exp(-N * CL/V * TAU))/(1 - exp(-CL/V * TAU)) -
        exp(-KA * (xt - (N - 1) * TAU)) * (1 - exp(-N * KA * TAU))/(1 - exp(-KA * TAU)))

    # PD model
    EFF = E0*(1 - CONC*IMAX/(IC50 + CONC))

    y[MS==1] = CONC[MS==1]
    y[MS==2] = EFF[MS==2]

    return(list( y= y,poped.db=poped.db))
  })
}

## -- parameter definition function
sfg <- function(x,a,bpop,b,bocc){
  parameters=c( V=bpop[1]*exp(b[1]),
               KA=bpop[2]*exp(b[2]),
               CL=bpop[3]*exp(b[3]),
               Favail=bpop[4],
               DOSE=a[1],
               TAU = a[2],
               E0=bpop[5]*exp(b[4]),
               IMAX=bpop[6],
               IC50=bpop[7])
  return( parameters )
}

```

```

## -- Residual Error function
feps <- function(model_switch,xt,parameters,epsi,poped.db){
  returnArgs <- ff(model_switch,xt,parameters,poped.db)
  y <- returnArgs[[1]]
  poped.db <- returnArgs[[2]]

  MS <- model_switch

  pk.dv <- y*(1+epsi[,1])+epsi[,2]
  pd.dv <- y*(1+epsi[,3])+epsi[,4]

  y[MS==1] = pk.dv[MS==1]
  y[MS==2] = pd.dv[MS==2]

  return(list( y= y,poped.db =poped.db ))
}

poped.db <-
create.poped.database(
  ff_fun=ff,
  fError_fun=feps,
  fg_fun=sfg,
  groupsize=20,
  m=3,
  bpop=c(V=72.8,KA=0.25,CL=3.75,Favail=0.9,
        E0=1120,IMAX=0.807,IC50=0.0993),
  notfixed_bpop=c(1,1,1,0,1,1,1),
  d=c(V=0.09,KA=0.09,CL=0.25^2,E0=0.09),
  sigma=c(0.04,5e-6,0.09,100),
  notfixed_sigma=c(0,0,0,0),
  xt=c( 1,2,8,240,240,1,2,8,240,240),
  minxt=c(0,0,0,240,240,0,0,0,240,240),
  maxx=c(10,10,10,248,248,10,10,10,248,248),
  discrete_xt = list(0:248),
  G_xt=c(1,2,3,4,5,1,2,3,4,5),
  bUseGrouped_xt=1,
  model_switch=c(1,1,1,1,1,2,2,2,2,2),
  a=list(c(DOSE=20,TAU=24),c(DOSE=40, TAU=24),c(DOSE=0, TAU=24)),
  maxa=c(DOSE=200,TAU=40),
  mina=c(DOSE=0,TAU=2),
  ourzero=0)

## create plot of model and design
plot_model_prediction(poped.db,facet_scales="free",
                     model.names = c("PK","PD"))

## create plot of model with variability by
## computing the expected variance (using an FO approximation)
## and then computing a prediction interval
## based on an assumption of normality
## computation is faster but less accurate
## compared to using DV=TRUE (and groupsize_sim = 500)
plot_model_prediction(poped.db,facet_scales="free",

```

```

model.names = c("PK", "PD"),
PI=TRUE,
separate.groups = TRUE)

```

---

poped\_gui

*Run the graphical interface for PopED*


---

### Description

Run the graphical interface for PopED

### Usage

```
poped_gui()
```

---

poped\_optim

*Optimize a design defined in a PopED database*


---

### Description

Optimize a design defined in a PopED database using the objective function described in the database (or in the arguments to this function). The function works for both discrete and continuous optimization variables.

### Usage

```

poped_optim(
  poped.db,
  opt_xt = poped.db$settings$optsw[2],
  opt_a = poped.db$settings$optsw[4],
  opt_x = poped.db$settings$optsw[3],
  opt_samps = poped.db$settings$optsw[1],
  opt_inds = poped.db$settings$optsw[5],
  method = c("ARS", "BFGS", "LS"),
  control = list(),
  trace = TRUE,
  fim.calc.type = poped.db$settings$iFIMCalculationType,
  ofv_calc_type = poped.db$settings$ofv_calc_type,
  ds_index = poped.db$parameters$ds_index,
  approx_type = poped.db$settings$iApproximationMethod,
  d_switch = poped.db$settings$d_switch,
  ED_samp_size = poped.db$settings$ED_samp_size,
  bLHS = poped.db$settings$bLHS,
  use_laplace = poped.db$settings$iEDCalculationType,

```

```

out_file = "",
parallel = F,
parallel_type = NULL,
num_cores = NULL,
mrgsolve_model = NULL,
loop_methods = ifelse(length(method) > 1, TRUE, FALSE),
iter_max = 10,
stop_crit_eff = 1.001,
stop_crit_diff = NULL,
stop_crit_rel = NULL,
ofv_fun = poped.db$settings$ofv_fun,
maximize = T,
allow_replicates = TRUE,
allow_replicates_xt = TRUE,
allow_replicates_a = TRUE,
...
)

```

### Arguments

poped.db	A PopED database.
opt_xt	Should the sample times be optimized?
opt_a	Should the continuous design variables be optimized?
opt_x	Should the discrete design variables be optimized?
opt_samps	Are the number of sample times per group being optimized?
opt_inds	Are the number of individuals per group being optimized?
method	A vector of optimization methods to use in a sequential fashion. Options are c("ARS", "BFGS", "LS", "GA"). c("ARS") is for Adaptive Random Search <a href="#">optim_ARS</a> . c("LS") is for Line Search <a href="#">optim_LS</a> . c("BFGS") is for Method "L-BFGS-B" from <a href="#">optim</a> . c("GA") is for the genetic algorithm from <a href="#">ga</a> .
control	Contains control arguments for each method specified.
trace	Should the algorithm output results intermittently.
fim.calc.type	The method used for calculating the FIM. Potential values: <ul style="list-style-type: none"> <li>• 0 = Full FIM. No assumption that fixed and random effects are uncorrelated.</li> <li>• 1 = Reduced FIM. Assume that there is no correlation in the FIM between the fixed and random effects, and set these elements in the FIM to zero.</li> <li>• 2 = weighted models (placeholder).</li> <li>• 3 = Not currently used.</li> <li>• 4 = Reduced FIM and computing all derivatives with respect to the standard deviation of the residual unexplained variation (sqrt(SIGMA) in NONMEM). This matches what is done in PFIM, and assumes that the standard deviation of the residual unexplained variation is the estimated parameter (NOTE: NONMEM estimates the variance of the residual unexplained variation by default).</li> <li>• 5 = Full FIM parameterized with A,B,C matrices &amp; derivative of variance.</li> </ul>

- 6 = Calculate one model switch at a time, good for large matrices.
- 7 = Reduced FIM parameterized with A,B,C matrices & derivative of variance.

ofv_calc_type	<p>OFV calculation type for FIM</p> <ul style="list-style-type: none"> <li>• 1 = "D-optimality". Determinant of the FIM: <math>\det(\text{FIM})</math></li> <li>• 2 = "A-optimality". Inverse of the sum of the expected parameter variances: <math>1/\text{trace\_matrix}(\text{inv}(\text{FIM}))</math></li> <li>• 4 = "lnD-optimality". Natural logarithm of the determinant of the FIM: <math>\log(\det(\text{FIM}))</math></li> <li>• 6 = "Ds-optimality". Ratio of the Determinant of the FIM and the Determinant of the uninteresting rows and columns of the FIM: <math>\det(\text{FIM})/\det(\text{FIM\_u})</math></li> <li>• 7 = Inverse of the sum of the expected parameter RSE: <math>1/\text{sum}(\text{get\_rse}(\text{FIM}, \text{poped.db}, \text{use\_percent}=\text{FA}))</math></li> </ul>
ds_index	<p>Ds_index is a vector set to 1 if a parameter is uninteresting, otherwise 0. size=(1,num unfixed parameters). First unfixed bpop, then unfixed d, then unfixed docc and last unfixed sigma. Default is the fixed effects being important, everything else not important. Used in conjunction with ofv_calc_type=6.</p>
approx_type	<p>Approximation method for model, 0=FO, 1=FOCE, 2=FOCEI, 3=FOI.</p>
d_switch	<ul style="list-style-type: none"> <li>• <b>*****START OF CRITERION SPECIFICATION OPTIONS*****</b></li> </ul> <p>D-family design (1) or ED-family design (0) (with or without parameter uncertainty)</p>
ED_samp_size	<p>Sample size for E-family sampling</p>
bLHS	<p>How to sample from distributions in E-family calculations. 0=Random Sampling, 1=LatinHyperCube –</p>
use_laplace	<p>Should the Laplace method be used in calculating the expectation of the OFV?</p>
out_file	<p>Save output from the optimization to a file.</p>
parallel	<p>Should we use parallel computations?</p>
parallel_type	<p>Which type of parallelization should be used? Can be "snow" or "multicore". "snow" works on Linux-like systems &amp; Windows. "multicore" works only on Linux-like systems. By default this is chosen for you depending on your operating system. See <a href="#">start_parallel</a>.</p>
num_cores	<p>The number of cores to use in the parallelization. By default is set to the number output from <code>parallel::detectCores()</code>. See <a href="#">start_parallel</a>.</p>
mrgsolve_model	<p>If the computations require a mrgsolve model and you are using the "snow" method then you need to specify the name of the model object created by <code>mread</code> or <code>mcode</code>.</p>
loop_methods	<p>Should the optimization methods be looped for <code>iter_max</code> iterations, or until the efficiency of the design after the current series (compared to the start of the series) is less than, or equal to, <code>stop_crit_eff</code>?</p>
iter_max	<p>If line search is used then the algorithm tests if line search (always run at the end of the optimization iteration) changes the design in any way. If not, the algorithm stops. If yes, then a new iteration is run unless <code>iter_max</code> iterations have already been run.</p>

stop_crit_eff	If loop_methods==TRUE, the looping will stop if the efficiency of the design after the current series (compared to the start of the series) is less than, or equal to, stop_crit_eff (if maximize==FALSE then 1/stop_crit_eff is the cut off and the efficiency must be greater than or equal to this value to stop the looping).
stop_crit_diff	If loop_methods==TRUE, the looping will stop if the difference in criterion value of the design after the current series (compared to the start of the series) is less than, or equal to, stop_crit_diff (if maximize==FALSE then -stop_crit_diff is the cut off and the difference in criterion value must be greater than or equal to this value to stop the looping).
stop_crit_rel	If loop_methods==TRUE, the looping will stop if the relative difference in criterion value of the design after the current series (compared to the start of the series) is less than, or equal to, stop_crit_rel (if maximize==FALSE then -stop_crit_rel is the cut off and the relative difference in criterion value must be greater than or equal to this value to stop the looping).
ofv_fun	User defined function used to compute the objective function. The function must have a poped database object as its first argument and have "." in its argument list. Can be referenced as a function or as a file name where the function defined in the file has the same name as the file. e.g. "cost.txt" has a function named "cost" in it.
maximize	Should the objective function be maximized or minimized?
allow_replicates	Should the algorithm allow optimized design components to have the same value? If FALSE then all discrete optimizations will not allow replicates within variable types (equivalent to allow_replicates_xt=FALSE and allow_replicates_a=FALSE).
allow_replicates_xt	Should the algorithm allow optimized xt design components to have the same value? If FALSE then all discrete optimizations will not allow replicates.
allow_replicates_a	Should the algorithm allow optimized a design components to have the same value? If FALSE then all discrete optimizations will not allow replicates.
...	arguments passed to other functions.

### Details

This function takes information from the PopED database supplied as an argument. The PopED database supplies information about the the model, parameters, design and methods to use. Some of the arguments coming from the PopED database can be overwritten; if they are supplied then they are used instead of the arguments from the PopED database.

If more than one optimization method is specified then the methods are run in series. If loop\_methods==TRUE then the series of optimization methods will be run for iter\_max iterations, or until the efficiency of the design after the current series (compared to the start of the series) is less than stop\_crit\_eff.

### References

1. M. Foracchia, A.C. Hooker, P. Vicini and A. Ruggeri, "PopED, a software fir optimal experimental design in population kinetics", Computer Methods and Programs in Biomedicine, 74, 2004.

2. J. Nyberg, S. Ueckert, E.A. Stroemberg, S. Hennig, M.O. Karlsson and A.C. Hooker, "PopED: An extended, parallelized, nonlinear mixed effects models optimal design tool", Computer Methods and Programs in Biomedicine, 108, 2012.

### See Also

Other Optimize: [Doptim\(\)](#), [LEDoptim\(\)](#), [RS\\_opt\(\)](#), [a\\_line\\_search\(\)](#), [bfgsb\\_min\(\)](#), [calc Autofocus\(\)](#), [calc\\_ofv\\_and\\_grad\(\)](#), [mfea\(\)](#), [optim\\_ARS\(\)](#), [optim\\_LS\(\)](#), [poped\\_optim\\_1\(\)](#), [poped\\_optim\\_2\(\)](#), [poped\\_optim\\_3\(\)](#), [poped\\_optimize\(\)](#)

### Examples

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin model for optimization)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## Optimization using an additive + proportional residual error
## to avoid sample times at very low concentrations (time 0 or very late samples).

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(prop=0.01, add=0.25),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0.01,
```

```

maxxt=120,
a=c(DOSE=70),
mina=c(DOSE=0.01),
maxa=c(DOSE=100))

##### END #####
## Create PopED database
## (warfarin model for optimization)
#####

#####
# D-family Optimization
#####

# below are a number of ways to optimize the problem

# ARS+BFGS+LS optimization of dose
# optimization with just a few iterations
# only to check that things are working
out_1 <- poped_optim(poped.db,opt_a =TRUE,
                    control = list(ARS=list(iter=2),
                                   BFGS=list(maxit=2),
                                   LS=list(line_length=2)),
                    iter_max = 1)

# cost function
# PRED at 120 hours
crit_fcn <- function(poped.db,...){
  pred_df <- model_prediction(poped.db)
  return(pred_df[pred_df$Time==120,"PRED"])
}

# maximize cost function
out_2 <- poped_optim(poped.db,opt_a =TRUE,
                    ofv_fun=crit_fcn,
                    control = list(ARS=list(iter=2),
                                   BFGS=list(maxit=2),
                                   LS=list(line_length=2)),
                    iter_max = 2)

# minimize the cost function
out_3 <- poped_optim(poped.db,opt_a =TRUE,
                    ofv_fun=crit_fcn,
                    control = list(ARS=list(iter=2),
                                   BFGS=list(maxit=2),
                                   LS=list(line_length=2)),
                    iter_max = 2,
                    maximize = FALSE,
                    evaluate_fim = FALSE)

```



```

## Not run:

# RS+BFGS+LS optimization of sample times
# (longer run time than above but more likely to reach a maximum)
output <- poped_optim(poped.db,opt_xt=T,parallel = TRUE)

get_rse(output$FIM,output$poped.db)
plot_model_prediction(output$poped.db)

# optimization with only integer times allowed
poped.db.2 <- poped.db
poped.db.2$design_space$xt_space <- matrix(list(seq(1,120)),1,8)
output_2 <- poped_optim(poped.db.2,opt_xt=T,parallel = TRUE)

get_rse(output_2$FIM,output_2$poped.db)
plot_model_prediction(output_2$poped.db)

# Examine efficiency of sampling windows
plot_efficiency_of_windows(output_2$poped.db,xt_windows=0.5)
plot_efficiency_of_windows(output_2$poped.db,xt_windows=1)

# Adaptive Random Search (ARS, just a few samples here)
rs.output <- poped_optim(poped.db,opt_xt=T,method = "ARS",
                        control = list(ARS=list(iter=5)))

get_rse(rs.output$FIM,rs.output$poped.db)

# line search, DOSE and sample time optimization
ls.output <- poped_optim(poped.db,opt_xt=T,opt_a=T,method = "LS",
                        control = list(LS=list(line_length=5)))

# Adaptive random search,
# DOSE and sample time optimization
ars.output <- poped_optim(poped.db,opt_xt=T,opt_a=T,method = "ARS",
                        control = list(ARS=list(iter=5)))

# BFGS gradient search from the stats::optim() function,
# DOSE and sample time optimization
bfgs.output <- poped_optim(poped.db,opt_xt=T,opt_a=T,method = "BFGS",
                        control = list(BFGS=list(maxit=5)))

# genetic algorithm from the GA::ga() function,
# DOSE and sample time optimization
ga.output <- poped_optim(poped.db,opt_xt=T,opt_a=F,method = "GA",parallel=T)

# cost function with GA
# maximize
out_2 <- poped_optim(poped.db,opt_a =TRUE,
                    ofv_fun=crit_fcn,
                    parallel = T,
                    method=c("GA"))

```

```

# cost function with GA
# minimize
out_2 <- poped_optim(poped.db,opt_a =TRUE,
                    ofv_fun=crit_fcn,
                    parallel = T,
                    method=c("GA"),
                    iter_max = 1,
                    maximize = F,
                    evaluate_fim = F)

# optimize distribution of individuals in 3 groups
poped_db_2 <- create.poped.database(
  ff_fun=ff.PK.1.comp.oral.sd.CL,
  fg_fun=sfg,
  fError_fun=feps.add.prop,
  bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
  notfixed_bpop=c(1,1,1,0),
  d=c(CL=0.07, V=0.02, KA=0.6),
  sigma=c(prop=0.01,add=0.25),
  groupsize=32,
  m=3,
  xt=list(c( 0.5,1,2,6,8),c(36,72,120),
          c(10,12,14,16,18,20,22,24)),
  minxt=0.01,
  maxx=120,
  a=c(DOSE=70),
  mina=c(DOSE=0.01),
  maxa=c(DOSE=100))

opt_xt_inds <-
  poped_optim(poped_db_2,
              opt_a =TRUE,
              opt_inds = TRUE,
              control = list(ARS=list(iter=2),
                             BFGS=list(maxit=2),
                             LS=list(line_length=2)),
              iter_max = 1)

#####
# E-family Optimization
#####

# Adding 10% log-normal Uncertainty to fixed effects (not Favail)
bpop_vals <- c(CL=0.15, V=8, KA=1.0, Favail=1)
bpop_vals_ed_ln <- cbind(ones(length(bpop_vals),1)*4, # log-normal distribution
                          bpop_vals,
                          ones(length(bpop_vals),1)*(bpop_vals*0.1)^2) # 10% of bpop value
bpop_vals_ed_ln["Favail",] <- c(0,1,0)
bpop_vals_ed_ln

## -- Define initial design and design space

```

```

poped.db <- create.poped.database(
  ff_fun=ff.PK.1.comp.oral.sd.CL,
  fg_fun=sfg,
  fError_fun=feps.add.prop,
  bpop=bpop_vals_ed_ln,
  notfixed_bpop=c(1,1,1,0),
  d=c(CL=0.07, V=0.02, KA=0.6),
  sigma=c(0.01,0.25),
  groupsize=32,
  xt=c( 0.5,1,2,6,24,36,72,120),
  minxt=0,
  maxxt=120,
  a=70,
  mina=0,
  maxa=100)

# E_ln(D) optimization using Random search (just a few samples here)
output <- poped_optim(poped.db,opt_xt=TRUE,opt_a=TRUE,d_switch=0,
  method = c("ARS","LS"),
  control = list(ARS=list(iter=2),
    LS=list(line_length=2)),
  iter_max = 1)
get_rse(output$FIM,output$poped.db)

# ED with laplace approximation,
# optimization using Random search (just a few iterations here)
ars.output <- poped_optim(poped.db,opt_xt=T,opt_a=T,method = "ARS",
  d_switch=0,use_laplace=TRUE,#laplace.fim=TRUE,
  parallel=T,
  control = list(ARS=list(iter=5)))

## End(Not run)

```

---

poped\_optimize

*Retired optimization module for PopED*


---

## Description

This function is an older version of [poped\\_optim](#). Please use [poped\\_optim](#) unless you have a specific reason to use this function instead.

## Usage

```

poped_optimize(
  poped.db,
  ni = NULL,
  xt = NULL,
  model_switch = NULL,

```

```

x = NULL,
a = NULL,
bpop = NULL,
d = NULL,
maxxt = NULL,
minxt = NULL,
maxa = NULL,
mina = NULL,
fmf = 0,
dmf = 0,
trflag = TRUE,
opt_xt = poped.db$settings$optsw[2],
opt_a = poped.db$settings$optsw[4],
opt_x = poped.db$settings$optsw[3],
opt_samps = poped.db$settings$optsw[1],
opt_inds = poped.db$settings$optsw[5],
cfaxt = poped.db$settings$cfaxt,
cfaa = poped.db$settings$cfaa,
rsit = poped.db$settings$rsit,
rsit_output = poped.db$settings$rsit_output,
fim.calc.type = poped.db$settings$iFIMCalculationType,
ofv_calc_type = poped.db$settings$ofv_calc_type,
approx_type = poped.db$settings$iApproximationMethod,
bUseExchangeAlgorithm = poped.db$settings$bUseExchangeAlgorithm,
iter = 1,
d_switch = poped.db$settings$d_switch,
ED_samp_size = poped.db$settings$ED_samp_size,
bLHS = poped.db$settings$bLHS,
use_laplace = poped.db$settings$iEDCalculationType,
...
)

```

### Arguments

poped.db	A PopED database.
ni	A vector of the number of samples in each group.
xt	A matrix of sample times. Each row is a vector of sample times for a group.
model_switch	A matrix that is the same size as xt, specifying which model each sample belongs to.
x	A matrix for the discrete design variables. Each row is a group.
a	A matrix of covariates. Each row is a group.
bpop	Matrix defining the fixed effects, per row (row number = parameter_number) we should have: <ul style="list-style-type: none"> <li>• column 1 the type of the distribution for E-family designs (0 = Fixed, 1 = Normal, 2 = Uniform, 3 = User Defined Distribution, 4 = lognormal and 5 = truncated normal)</li> <li>• column 2 defines the mean.</li> </ul>

- column 3 defines the variance of the distribution (or length of uniform distribution).

Can also just supply the parameter values as a vector `c()` if no uncertainty around the parameter value is to be used. The parameter order of 'bpop' is defined in the 'fg\_fun' or 'fg\_file'. If you use named arguments in 'bpop' then the order of this vector can be rearranged to match the 'fg\_fun' or 'fg\_file'. See 'reorder\_parameter\_vectors'.

d	Matrix defining the diagonals of the IIV (same logic as for the fixed effects matrix bpop to define uncertainty). One can also just supply the parameter values as a <code>c()</code> . The parameter order of 'd' is defined in the 'fg_fun' or 'fg_file'. If you use named arguments in 'd' then the order of this vector can be rearranged to match the 'fg_fun' or 'fg_file'. See 'reorder_parameter_vectors'.
maxxt	Matrix or single value defining the maximum value for each xt sample. If a single value is supplied then all xt values are given the same maximum value.
minxt	Matrix or single value defining the minimum value for each xt sample. If a single value is supplied then all xt values are given the same minimum value
maxa	Vector defining the max value for each covariate. If a single value is supplied then all a values are given the same max value
mina	Vector defining the min value for each covariate. If a single value is supplied then all a values are given the same max value
fmf	The initial value of the FIM. If set to zero then it is computed.
dmf	The initial OFV. If set to zero then it is computed.
trflag	Should the optimization be output to the screen and to a file?
opt_xt	Should the sample times be optimized?
opt_a	Should the continuous design variables be optimized?
opt_x	Should the discrete design variables be optimized?
opt_samps	Are the number of sample times per group being optimized?
opt_inds	Are the number of individuals per group being optimized?
cfaxt	First step factor for sample times
cfaa	Stochastic Gradient search first step factor for covariates
rsit	Number of Random search iterations
rsit_output	Number of iterations in random search between screen output
fim.calc.type	The method used for calculating the FIM. Potential values: <ul style="list-style-type: none"> <li>• 0 = Full FIM. No assumption that fixed and random effects are uncorrelated.</li> <li>• 1 = Reduced FIM. Assume that there is no correlation in the FIM between the fixed and random effects, and set these elements in the FIM to zero.</li> <li>• 2 = weighted models (placeholder).</li> <li>• 3 = Not currently used.</li> <li>• 4 = Reduced FIM and computing all derivatives with respect to the standard deviation of the residual unexplained variation (<math>\sqrt{\text{SIGMA}}</math> in NONMEM). This matches what is done in PFIM, and assumes that the standard deviation of the residual unexplained variation is the estimated parameter (NOTE: NONMEM estimates the variance of the residual unexplained variation by default).</li> </ul>

- 5 = Full FIM parameterized with A,B,C matrices & derivative of variance.
- 6 = Calculate one model switch at a time, good for large matrices.
- 7 = Reduced FIM parameterized with A,B,C matrices & derivative of variance.

ofv\_calc\_type    OFV calculation type for FIM

- 1 = "D-optimality". Determinant of the FIM:  $\det(\text{FIM})$
- 2 = "A-optimality". Inverse of the sum of the expected parameter variances:  $1/\text{trace\_matrix}(\text{inv}(\text{FIM}))$
- 4 = "lnD-optimality". Natural logarithm of the determinant of the FIM:  $\log(\det(\text{FIM}))$
- 6 = "Ds-optimality". Ratio of the Determinant of the FIM and the Determinant of the uninteresting rows and columns of the FIM:  $\det(\text{FIM})/\det(\text{FIM}_u)$
- 7 = Inverse of the sum of the expected parameter RSE:  $1/\text{sum}(\text{get\_rse}(\text{FIM}, \text{poped.db}, \text{use\_percent}=\text{FA}))$

approx\_type    Approximation method for model, 0=FO, 1=FOCE, 2=FOCEI, 3=FOI.

bUseExchangeAlgorithm    Use Exchange algorithm (1=TRUE, 0=FALSE)

iter    The number of iterations entered into the blockheader\_2 function.

d\_switch    • \*\*\*\*\***START OF CRITERION SPECIFICATION OPTIONS**\*\*\*\*\*  
D-family design (1) or ED-family design (0) (with or without parameter uncertainty)

ED\_samp\_size    Sample size for E-family sampling

bLHS    How to sample from distributions in E-family calculations. 0=Random Sampling, 1=LatinHyperCube –

use\_laplace    Should the Laplace method be used in calculating the expectation of the OFV?

...    arguments passed to other functions. See [Doptim](#).

## Details

This function optimized the objective function. The function works for both discrete and continuous optimization variables. This function takes information from the PopED database supplied as an argument. The PopED database supplies information about the the model, parameters, design and methods to use. Some of the arguments coming from the PopED database can be overwritten; if they are supplied then they are used instead of the arguments from the PopED database.

## References

1. M. Foracchia, A.C. Hooker, P. Vicini and A. Ruggeri, "PopED, a software fir optimal experimental design in population kinetics", *Computer Methods and Programs in Biomedicine*, 74, 2004.
2. J. Nyberg, S. Ueckert, E.A. Stroemberg, S. Hennig, M.O. Karlsson and A.C. Hooker, "PopED: An extended, parallelized, nonlinear mixed effects models optimal design tool", *Computer Methods and Programs in Biomedicine*, 108, 2012.

**See Also**

Other Optimize: [Doptim\(\)](#), [LEDoptim\(\)](#), [RS\\_opt\(\)](#), [a\\_line\\_search\(\)](#), [bfgsb\\_min\(\)](#), [calc Autofocus\(\)](#), [calc\\_ofv\\_and\\_grad\(\)](#), [mfea\(\)](#), [optim\\_ARS\(\)](#), [optim\\_LS\(\)](#), [poped\\_optim\(\)](#), [poped\\_optim\\_1\(\)](#), [poped\\_optim\\_2\(\)](#), [poped\\_optim\\_3\(\)](#)

**Examples**

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin model for optimization)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## Optimization using an additive + proportional residual error
## to avoid sample times at very low concentrations (time 0 or very late samples).

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
}

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(prop=0.01, add=0.25),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0.01,
                                maxx=120,
                                a=c(DOSE=70),
                                mina=c(DOSE=0.01),
                                maxa=c(DOSE=100))
```

```
##### END #####
## Create PopED database
## (warfarin model for optimization)
#####

#####
# D-family Optimization
#####

# below are a number of ways to optimize the problem

# RS+SG+LS optimization of DOSE and sample times
# optimization with just a few iterations
# only to check that things are working
out_1 <- poped_optimize(poped.db,opt_a=TRUE,opt_xt=TRUE,
                        rsit=2,sgit=2,ls_step_size=2,
                        iter_max=1,out_file = "")

## Not run:

# RS+SG+LS optimization of sample times
# (longer run time than above but more likely to reach a maximum)
output <- poped_optimize(poped.db,opt_xt=T)
get_rse(output$fmf,output$poped.db)
plot_model_prediction(output$poped.db)

# MFEA optimization with only integer times allowed
mfea.output <- poped_optimize(poped.db,opt_xt=1,
                              bUseExchangeAlgorithm=1,
                              EAStepSize=1)
get_rse(mfea.output$fmf,mfea.output$poped.db)
plot_model_prediction(mfea.output$poped.db)

# Examine efficiency of sampling windows
plot_efficiency_of_windows(mfea.output$poped.db,xt_windows=0.5)
plot_efficiency_of_windows(mfea.output$poped.db,xt_windows=1)

# Random search (just a few samples here)
rs.output <- poped_optimize(poped.db,opt_xt=1,opt_a=1,rsit=20,
                            bUseRandomSearch= 1,
                            bUseStochasticGradient = 0,
                            bUseBFGSMinimizer = 0,
                            bUseLineSearch = 0)
get_rse(rs.output$fmf,rs.output$poped.db)

# line search, DOSE and sample time optimization
ls.output <- poped_optimize(poped.db,opt_xt=1,opt_a=1,
                            bUseRandomSearch= 0,
                            bUseStochasticGradient = 0,
                            bUseBFGSMinimizer = 0,
                            bUseLineSearch = 1,
                            ls_step_size=10)
```



```

# Stochastic gradient search, DOSE and sample time optimization
sg.output <- poped_optimize(poped.db,opt_xt=1,opt_a=1,
                           bUseRandomSearch= 0,
                           bUseStochasticGradient = 1,
                           bUseBFGSMinimizer = 0,
                           bUseLineSearch = 0,
                           sgit=20)

# BFGS search, DOSE and sample time optimization
bfgs.output <- poped_optimize(poped.db,opt_xt=1,opt_a=1,
                              bUseRandomSearch= 0,
                              bUseStochasticGradient = 0,
                              bUseBFGSMinimizer = 1,
                              bUseLineSearch = 0)

#####
# E-family Optimization
#####

# Adding 10% log-normal Uncertainty to fixed effects (not Favail)
bpop_vals <- c(CL=0.15, V=8, KA=1.0, Favail=1)
bpop_vals_ed_ln <- cbind(ones(length(bpop_vals),1)*4, # log-normal distribution
                        bpop_vals,
                        ones(length(bpop_vals),1)*(bpop_vals*0.1)^2) # 10% of bpop value
bpop_vals_ed_ln["Favail",] <- c(0,1,0)
bpop_vals_ed_ln

## -- Define initial design and design space
poped.db <- create.poped.database(
  ff_fun=ff.PK.1.comp.oral.sd.CL,
  fg_fun=sfg,
  fError_fun=feps.add.prop,
  bpop=bpop_vals_ed_ln,
  notfixed_bpop=c(1,1,1,0),
  d=c(CL=0.07, V=0.02, KA=0.6),
  sigma=c(0.01,0.25),
  groupsize=32,
  xt=c( 0.5,1,2,6,24,36,72,120),
  minxt=0,
  maxx=120,
  a=70,
  mina=0,
  maxa=100)

# ED optimization using Random search (just a few samples here)
output <- poped_optimize(poped.db,opt_xt=1,opt_a=1,rsit=10,d_switch=0)
get_rse(output$fmf,output$poped.db)

# ED with laplace approximation,
# optimization using Random search (just a few samples here)
output <- poped_optimize(poped.db,opt_xt=1,opt_a=1,rsit=10,
                        d_switch=0,use_laplace=TRUE,laplace.fim=TRUE)

```

```
get_rse(output$fmf,output$poped.db)
```

```
## End(Not run)
```

---

RS\_opt

*Optimize the objective function using an adaptive random search algorithm for D-family and E-family designs.*

---

### Description

Optimize the objective function using an adaptive random search algorithm. Optimization can be performed for both D-family and E-family designs. The function works for both discrete and continuous optimization variables. This function takes information from the PopED database supplied as an argument. The PopED database supplies information about the the model, parameters, design and methods to use. Some of the arguments coming from the PopED database can be overwritten; by default these arguments are NULL in the function, if they are supplied then they are used instead of the arguments from the PopED database.

### Usage

```
RS_opt(
  poped.db,
  ni = NULL,
  xt = NULL,
  model_switch = NULL,
  x = NULL,
  a = NULL,
  bpopdescr = NULL,
  ddescr = NULL,
  maxx = NULL,
  minx = NULL,
  maxa = NULL,
  mina = NULL,
  fmf = 0,
  dmf = 0,
  trflag = TRUE,
  opt_xt = poped.db$settings$optsw[2],
  opt_a = poped.db$settings$optsw[4],
  opt_x = poped.db$settings$optsw[3],
  cfaxt = poped.db$settings$cfaxt,
  cfaa = poped.db$settings$cfaa,
  rsit = poped.db$settings$rsit,
  rsit_output = poped.db$settings$rsit_output,
  fim.calc.type = poped.db$settings$iFIMCalculationType,
  approx_type = poped.db$settings$iApproximationMethod,
```

```

iter = NULL,
d_switch = poped.db$settings$d_switch,
use_laplace = poped.db$settings$iEDCalculationType,
laplace.fim = FALSE,
header_flag = TRUE,
footer_flag = TRUE,
out_file = NULL,
compute_inv = TRUE,
...
)

```

### Arguments

poped.db	A PopED database.
ni	A vector of the number of samples in each group.
xt	A matrix of sample times. Each row is a vector of sample times for a group.
model_switch	A matrix that is the same size as xt, specifying which model each sample belongs to.
x	A matrix for the discrete design variables. Each row is a group.
a	A matrix of covariates. Each row is a group.
bpopdescr	Matrix defining the fixed effects, per row (row number = parameter_number) we should have: <ul style="list-style-type: none"> <li>• column 1 the type of the distribution for E-family designs (0 = Fixed, 1 = Normal, 2 = Uniform, 3 = User Defined Distribution, 4 = lognormal and 5 = truncated normal)</li> <li>• column 2 defines the mean.</li> <li>• column 3 defines the variance of the distribution (or length of uniform distribution).</li> </ul>
ddescr	Matrix defining the diagonals of the IIV (same logic as for the bpopdescr).
maxxt	Matrix or single value defining the maximum value for each xt sample. If a single value is supplied then all xt values are given the same maximum value.
minxt	Matrix or single value defining the minimum value for each xt sample. If a single value is supplied then all xt values are given the same minimum value
maxa	Vector defining the max value for each covariate. If a single value is supplied then all a values are given the same max value
mina	Vector defining the min value for each covariate. If a single value is supplied then all a values are given the same max value
fmf	The initial value of the FIM. If set to zero then it is computed.
dmf	The initial OFV. If set to zero then it is computed.
trflag	Should the optimization be output to the screen and to a file?
opt_xt	Should the sample times be optimized?
opt_a	Should the continuous design variables be optimized?
opt_x	Should the discrete design variables be optimized?

cfaxt	First step factor for sample times
cfaa	Stochastic Gradient search first step factor for covariates
rsit	Number of Random search iterations
rsit_output	Number of iterations in random search between screen output
fim.calc.type	The method used for calculating the FIM. Potential values: <ul style="list-style-type: none"> <li>• 0 = Full FIM. No assumption that fixed and random effects are uncorrelated.</li> <li>• 1 = Reduced FIM. Assume that there is no correlation in the FIM between the fixed and random effects, and set these elements in the FIM to zero.</li> <li>• 2 = weighted models (placeholder).</li> <li>• 3 = Not currently used.</li> <li>• 4 = Reduced FIM and computing all derivatives with respect to the standard deviation of the residual unexplained variation (sqrt(SIGMA) in NONMEM). This matches what is done in PFIM, and assumes that the standard deviation of the residual unexplained variation is the estimated parameter (NOTE: NONMEM estimates the variance of the residual unexplained variation by default).</li> <li>• 5 = Full FIM parameterized with A,B,C matrices &amp; derivative of variance.</li> <li>• 6 = Calculate one model switch at a time, good for large matrices.</li> <li>• 7 = Reduced FIM parameterized with A,B,C matrices &amp; derivative of variance.</li> </ul>
approx_type	Approximation method for model, 0=FO, 1=FOCE, 2=FOCEI, 3=FOI.
iter	The number of iterations entered into the blockheader_2 function.
d_switch	<ul style="list-style-type: none"> <li>• *****START OF CRITERION SPECIFICATION OPTIONS*****</li> </ul> D-family design (1) or ED-family design (0) (with or without parameter uncertainty)
use_laplace	Should the Laplace method be used in calculating the expectation of the OFV?
laplace.fim	Should an E(FIM) be calculated when computing the Laplace approximated E(OFV). Typically the FIM does not need to be computed and, if desired, this calculation is done using the standard MC integration technique, so can be slow.
header_flag	Should the header text be printed out?
footer_flag	Should the footer text be printed out?
out_file	Which file should the output be directed to? A string, a file handle using <code>file</code> or <code>""</code> will output to the screen.
compute_inv	should the inverse of the FIM be used to compute expected RSE values? Often not needed except for diagnostic purposes.
...	arguments passed to <code>evaluate.fim</code> and <code>ofv_fim</code> .

## References

1. M. Foracchia, A.C. Hooker, P. Vicini and A. Ruggeri, "PopED, a software fir optimal experimental design in population kinetics", *Computer Methods and Programs in Biomedicine*, 74, 2004.
2. J. Nyberg, S. Ueckert, E.A. Stroemberg, S. Hennig, M.O. Karlsson and A.C. Hooker, "PopED: An extended, parallelized, nonlinear mixed effects models optimal design tool", *Computer Methods and Programs in Biomedicine*, 108, 2012.



```

sigma=c(0.01,0.25),
groupsize=32,
xt=c( 0.5,1,2,6,24,36,72,120),
minxt=0,
maxxt=120,
a=70,
mina=0,
maxa=100)

##### END #####
## Create PopED database
## (warfarin model for optimization
## with parameter uncertainty)
#####

# Just a few iterations, optimize on DOSE and sample times using the full FIM
out_1 <- RS_opt(poped.db,opt_xt=1,opt_a=1,rsit=3,fim.calc.type=0, out_file = "")

## Not run:

RS_opt(poped.db)

RS_opt(poped.db,opt_xt=TRUE,rsit=100,compute_inv=F)
RS_opt(poped.db,opt_xt=TRUE,rsit=20,d_switch=0)
RS_opt(poped.db,opt_xt=TRUE,rsit=10,d_switch=0,use_laplace=T)
RS_opt(poped.db,opt_xt=TRUE,rsit=10,d_switch=0,use_laplace=T,laplace.fim=T)

## Different headers and footers of output
RS_opt(poped.db,opt_xt=TRUE,rsit=10,out_file="foo.txt")
output <- RS_opt(poped.db,opt_xt=TRUE,rsit=100,trflag=FALSE)
RS_opt(poped.db,opt_xt=TRUE,rsit=10,out_file="")
RS_opt(poped.db,opt_xt=TRUE,rsit=10,header_flag=FALSE)
RS_opt(poped.db,opt_xt=TRUE,rsit=10,footer_flag=FALSE)
RS_opt(poped.db,opt_xt=TRUE,rsit=10,header_flag=FALSE,footer_flag=FALSE)
RS_opt(poped.db,opt_xt=TRUE,rsit=10,header_flag=FALSE,footer_flag=FALSE,out_file="foo.txt")
RS_opt(poped.db,opt_xt=TRUE,rsit=10,header_flag=FALSE,footer_flag=FALSE,out_file="")

## End(Not run)

```

---

shrinkage

---

*Predict shrinkage of empirical Bayes estimates (EBEs) in a population model*


---

### Description

Predict shrinkage of empirical Bayes estimates (EBEs) in a population model

### Usage

```
shrinkage(poped.db, use_mc = FALSE, num_sim_ids = 1000, use_purrr = FALSE)
```

**Arguments**

poped.db	A PopED database
use_mc	Should the calculation be based on monte-carlo simulations. If not then then a first order approximation is used
num_sim_ids	If use_mc=TRUE, how many individuals should be simulated to make the computations.
use_purrr	If use_mc=TRUE then should the method use the package purrr in calculations? This may speed up computations (potentially).

**Value**

The shrinkage computed in variance units, standard deviation units and the relative standard errors of the EBES.

**References**

1. Combes, F. P., Retout, S., Frey, N., & Mentre, F. (2013). Prediction of shrinkage of individual parameters using the Bayesian information matrix in non-linear mixed effect models with evaluation in pharmacokinetics. *Pharmaceutical Research*, 30(9), 2355-67. doi:10.1007/s1109501310793.
2. Hennig, S., Nyberg, J., Fanta, S., Backman, J. T., Hoppu, K., Hooker, A. C., & Karlsson, M. O. (2012). Application of the optimal design approach to improve a pretransplant drug dose finding design for ciclosporin. *Journal of Clinical Pharmacology*, 52(3), 347-360. doi:10.1177/0091270010397731.

**Examples**

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin example)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
```

```

        DOSE=a[1])
    return(parameters)
}

## -- Define model, parameters, initial design
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(prop=0.01),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                a=c(DOSE=70))

##### END #####
## Create PopED database
## (warfarin example)
#####

shrinkage(poped.db)

```

---

size

*Function written to match MATLAB's size function*


---

### Description

Function written to match MATLAB's size function

### Usage

```
size(obj, dimension.index = NULL)
```

### Arguments

obj                   An object you want to know the various dimensions of. Typically a matrix.  
dimension.index       Which dimension you are interested in.

### Value

The dimensions of the object or specific dimension you are interested in.

### See Also

Other MATLAB: [cell\(\)](#), [diag\\_matlab\(\)](#), [feval\(\)](#), [fileparts\(\)](#), [isempty\(\)](#), [ones\(\)](#), [rand\(\)](#), [randn\(\)](#), [tic\(\)](#), [toc\(\)](#), [zeros\(\)](#)



**Examples**

```
size(c(2,3,4,5,6))
```

```
size(10)
```

```
size(zeros(4,7))
```

---

start_parallel	<i>Start parallel computational processes</i>
----------------	---

---

**Description**

This tool chooses the type of parallelization process to use based on the computer OS being used. For windows the default is "snow" and for Linux-like systems the default is "multicore"

**Usage**

```
start_parallel(
  parallel = TRUE,
  num_cores = NULL,
  parallel_type = NULL,
  seed = NULL,
  dlls = NULL,
  mrgsolve_model = NULL,
  ...
)
```

**Arguments**

parallel	Should the parallel functionality start up?
num_cores	How many cores to use. Default is <code>parallel::detectCores()-1</code> . See <a href="#">detectCores</a> for more information.
parallel_type	Which type of parallelization should be used? Can be "snow" or "multicore". "snow" works on Linux-like systems & Windows. "multicore" works only on Linux-like systems. By default this is chosen for you depending on your operating system.
seed	The random seed to use.
dlls	If the computations require compiled code (DLL's) and you are using the "snow" method then you need to specify the name of the DLL's without the extension as a text vector <code>c("this_file", "that_file")</code> .
mrgsolve_model	If the computations require a mrgsolve model and you are using the "snow" method" then you need to specify the name of the model object created by <code>mread</code> or <code>mcode</code>
...	Arguments passed to <a href="#">makeCluster</a>

**Value**

An atomic vector (TRUE or FALSE) with two attributes: "type" and "cores".

---

```
summary.poped_optim  Display a summary of output from poped_optim
```

---

**Description**

Display a summary of output from poped\_optim

**Usage**

```
## S3 method for class 'poped_optim'
summary(object, ...)
```

**Arguments**

object            An object returned from `poped_optim` to summarize.  
...                Additional arguments. Passed to `blockfinal`.

**Examples**

```
library(PopED)

##### START #####
## Create PopED database
## (warfarin model for optimization)
#####

## Warfarin example from software comparison in:
## Nyberg et al., "Methods and software tools for design evaluation
## for population pharmacokinetics-pharmacodynamics studies",
## Br. J. Clin. Pharm., 2014.

## Optimization using an additive + proportional residual error
## to avoid sample times at very low concentrations (time 0 or very late samples).

## find the parameters that are needed to define from the structural model
ff.PK.1.comp.oral.sd.CL

## -- parameter definition function
## -- names match parameters in function ff
sfg <- function(x,a,bpop,b,bocc){
  parameters=c(CL=bpop[1]*exp(b[1]),
              V=bpop[2]*exp(b[2]),
              KA=bpop[3]*exp(b[3]),
              Favail=bpop[4],
              DOSE=a[1])
  return(parameters)
```

```

}

## -- Define initial design and design space
poped.db <- create.poped.database(ff_fun=ff.PK.1.comp.oral.sd.CL,
                                fg_fun=sfg,
                                fError_fun=feps.add.prop,
                                bpop=c(CL=0.15, V=8, KA=1.0, Favail=1),
                                notfixed_bpop=c(1,1,1,0),
                                d=c(CL=0.07, V=0.02, KA=0.6),
                                sigma=c(prop=0.01,add=0.25),
                                groupsize=32,
                                xt=c( 0.5,1,2,6,24,36,72,120),
                                minxt=0.01,
                                maxx=120,
                                a=c(DOSE=70),
                                mina=c(DOSE=0.01),
                                maxa=c(DOSE=100))

##### END #####
## Create PopED database
## (warfarin model for optimization)
#####

#####
# D-family Optimization
#####

# ARS+BFGS+LS optimization of dose
# optimization with just a few iterations
# only to check that things are working
out_1 <- poped_optim(poped.db,opt_a =TRUE,
                    control = list(ARS=list(iter=2),
                                   BFGS=list(maxit=2),
                                   LS=list(line_length=2)),
                    iter_max = 1)

summary(out_1)

```

---

tic

*Timer function (as in MATLAB)*


---

## Description

Function to start a timer. Stop with toc().

## Usage

```
tic(gcFirst = FALSE, name = ".poped_savedTime")
```

**Arguments**

gcFirst	Perform garbage collection?
name	The saved name of the time object.

**Note**

This is a modified version of the same function in the matlab R-package.

**See Also**

Other MATLAB: [cell\(\)](#), [diag\\_matlab\(\)](#), [feval\(\)](#), [fileparts\(\)](#), [isempty\(\)](#), [ones\(\)](#), [rand\(\)](#), [randn\(\)](#), [size\(\)](#), [tic\(\)](#), [zeros\(\)](#)

**Examples**

```
tic()
toc()

tic(name="foo")
toc()
tic()
toc()
toc()
tic()
toc(name="foo")
```

---

toc

*Timer function (as in MATLAB)*

---

**Description**

Function to stop a timer. Start with tic().

**Usage**

```
toc(echo = TRUE, name = ".poped_savedTime")
```

**Arguments**

echo	Print time to screen?
name	The saved name of the time object.

**Note**

This is a modified version of the same function in the matlab R-package.

**See Also**

Other MATLAB: [cell\(\)](#), [diag\\_matlab\(\)](#), [feval\(\)](#), [fileparts\(\)](#), [isempty\(\)](#), [ones\(\)](#), [rand\(\)](#), [randn\(\)](#), [size\(\)](#), [tic\(\)](#), [zeros\(\)](#)

**Examples**

```
tic()
toc()

tic(name="foo")
toc()
tic()
toc()
toc()
tic()
toc(name="foo")
```

---

zeros	<i>Create a matrix of zeros.</i>
-------	----------------------------------

---

**Description**

Create a matrix of zeros of size (dim1 x dim2).

**Usage**

```
zeros(dim1, dim2 = NULL)
```

**Arguments**

dim1	The dimension of the matrix (if square) or the number of rows.
dim2	The number of columns

**Value**

A matrix of zeros.

**See Also**

Other MATLAB: [cell\(\)](#), [diag\\_matlab\(\)](#), [feval\(\)](#), [fileparts\(\)](#), [isempty\(\)](#), [ones\(\)](#), [rand\(\)](#), [randn\(\)](#), [size\(\)](#), [tic\(\)](#), [toc\(\)](#)

**Examples**

```
zeros(3)
zeros(0,3)
zeros(4,7)
zeros(1,4)
```

# Index

- \* **E-family**
    - calc\_ofv\_and\_fim, 7
    - evaluate.e.ofv.fim, 31
  - \* **FIM**
    - calc\_ofv\_and\_fim, 7
    - efficiency, 29
    - evaluate.e.ofv.fim, 31
    - evaluate.fim, 35
    - ofv\_criterion, 75
    - ofv\_fim, 77
  - \* **Graphics**
    - plot\_efficiency\_of\_windows, 98
    - plot\_model\_prediction, 101
  - \* **MATLAB**
    - cell, 11
    - ones, 80
    - size, 128
    - tic, 131
    - toc, 132
    - zeros, 133
  - \* **Optimize**
    - a\_line\_search, 3
    - LEDOptim, 64
    - optim\_ARS, 85
    - optim\_LS, 91
    - poped\_optim, 107
    - poped\_optimize, 115
    - RS\_opt, 122
  - \* **RUV\_models**
    - feps.add, 44
    - feps.add.prop, 46
    - feps.prop, 48
  - \* **Simulation**
    - model\_prediction, 70
    - plot\_efficiency\_of\_windows, 98
    - plot\_model\_prediction, 101
  - \* **evaluate\_FIM**
    - calc\_ofv\_and\_fim, 7
    - evaluate.e.ofv.fim, 31
    - evaluate.fim, 35
    - ofv\_fim, 77
  - \* **evaluate\_design**
    - evaluate.fim, 35
    - evaluate\_design, 39
    - evaluate\_power, 42
    - get\_rse, 61
    - model\_prediction, 70
    - plot\_efficiency\_of\_windows, 98
    - plot\_model\_prediction, 101
  - \* **models**
    - feps.add, 44
    - feps.add.prop, 46
    - feps.prop, 48
    - ff.PK.1.comp.oral.md.CL, 50
    - ff.PK.1.comp.oral.md.KE, 52
    - ff.PK.1.comp.oral.sd.CL, 53
    - ff.PK.1.comp.oral.sd.KE, 55
    - ff.PKPD.1.comp.oral.md.CL.imax, 57
    - ff.PKPD.1.comp.sd.CL.emax, 59
  - \* **poped\_input**
    - create.poped.database, 12
    - create\_design, 23
    - create\_design\_space, 25
  - \* **structural\_models**
    - ff.PK.1.comp.oral.md.CL, 50
    - ff.PK.1.comp.oral.md.KE, 52
    - ff.PK.1.comp.oral.sd.CL, 53
    - ff.PK.1.comp.oral.sd.KE, 55
    - ff.PKPD.1.comp.oral.md.CL.imax, 57
    - ff.PKPD.1.comp.sd.CL.emax, 59
- a\_line\_search, 3, 66, 87, 93, 111, 119, 125
- bfgsb\_min, 4, 66, 87, 93, 111, 119, 125
- blockfinal, 29, 130
- build\_sfg, 6
- calc\_autofocus, 4, 66, 87, 93, 111, 119, 125

- calc\_ofv\_and\_fim, [7](#), [30](#), [33](#), [37](#), [39](#), [43](#), [76](#), [78](#)
- calc\_ofv\_and\_grad, [4](#), [66](#), [87](#), [93](#), [111](#), [119](#), [125](#)
- cell, [11](#), [16](#), [17](#), [27](#), [80](#), [128](#), [132](#), [133](#)
- convert\_variables, [23](#), [24](#), [27](#)
- create.poped.database, [12](#), [24](#), [27](#), [29](#), [44](#), [46](#), [48](#), [50](#), [52](#), [53](#), [55](#), [57](#), [59](#), [71](#)
- create\_design, [23](#), [23](#), [26](#), [27](#), [71](#)
- create\_design\_space, [16](#), [17](#), [23](#), [24](#), [25](#)
- design\_summary, [29](#)
- detectCores, [129](#)
- diag\_matlab, [11](#), [80](#), [128](#), [132](#), [133](#)
- Doptim, [4](#), [66](#), [87](#), [93](#), [111](#), [118](#), [119](#), [125](#)
- downsizing\_general\_design, [23](#), [24](#), [27](#)
- ed\_laplace\_ofv, [9](#), [30](#), [33](#), [37](#), [76](#), [78](#)
- ed\_mftot, [9](#), [30](#), [33](#), [37](#), [76](#), [78](#)
- efficiency, [9](#), [29](#), [33](#), [37](#), [76](#), [78](#)
- evaluate.e.ofv.fim, [9](#), [30](#), [31](#), [37](#), [76](#), [78](#)
- evaluate.fim, [9](#), [30](#), [33](#), [35](#), [39](#), [43](#), [62](#), [66](#), [72](#), [76](#), [78](#), [82](#), [100](#), [104](#), [124](#)
- evaluate\_design, [37](#), [39](#), [43](#), [62](#), [72](#), [100](#), [104](#)
- evaluate\_fim\_map, [40](#)
- evaluate\_power, [37](#), [39](#), [42](#), [62](#), [72](#), [100](#), [104](#)
- expression, [72](#)
- feps.add, [44](#), [47](#), [48](#), [50](#), [52](#), [54](#), [56](#), [57](#), [60](#)
- feps.add.prop, [45](#), [46](#), [48](#), [50](#), [52](#), [54](#), [56](#), [57](#), [60](#)
- feps.prop, [45](#), [47](#), [48](#), [50](#), [52](#), [54](#), [56](#), [57](#), [60](#)
- feval, [11](#), [80](#), [128](#), [132](#), [133](#)
- ff.PK.1.comp.oral.md.CL, [45](#), [47](#), [48](#), [50](#), [52](#), [54](#), [56](#), [57](#), [60](#)
- ff.PK.1.comp.oral.md.KE, [45](#), [47](#), [48](#), [50](#), [51](#), [54](#), [56](#), [57](#), [60](#)
- ff.PK.1.comp.oral.sd.CL, [45](#), [47](#), [48](#), [50](#), [52](#), [53](#), [56](#), [57](#), [60](#)
- ff.PK.1.comp.oral.sd.KE, [45](#), [47](#), [48](#), [50](#), [52](#), [54](#), [55](#), [57](#), [60](#)
- ff.PKPD.1.comp.oral.md.CL.imax, [45](#), [47](#), [48](#), [50](#), [52](#), [54](#), [56](#), [57](#), [60](#)
- ff.PKPD.1.comp.sd.CL.emax, [45](#), [47](#), [48](#), [50](#), [52](#), [54](#), [56](#), [57](#), [59](#)
- file, [66](#), [124](#)
- fileparts, [11](#), [80](#), [128](#), [132](#), [133](#)
- ga, [108](#)
- get\_rse, [37](#), [39](#), [43](#), [61](#), [72](#), [100](#), [104](#)
- ggplot, [100](#), [104](#)
- gradf\_eps, [9](#), [30](#), [33](#), [37](#), [76](#), [78](#)
- inv, [62](#)
- isempty, [11](#), [80](#), [128](#), [132](#), [133](#)
- LEOptim, [4](#), [64](#), [87](#), [93](#), [111](#), [119](#), [125](#)
- LinMatrixH, [9](#), [30](#), [33](#), [37](#), [76](#), [78](#)
- LinMatrixL\_occ, [9](#), [30](#), [33](#), [37](#), [76](#), [78](#)
- LinMatrixLH, [9](#), [30](#), [33](#), [37](#), [76](#), [78](#)
- makeCluster, [129](#)
- mc\_mean, [68](#)
- median\_hilow\_poped, [70](#)
- mf3, [9](#), [30](#), [33](#), [37](#), [76](#), [78](#)
- mf7, [9](#), [30](#), [33](#), [37](#), [76](#), [78](#)
- mfea, [4](#), [66](#), [87](#), [93](#), [111](#), [119](#), [125](#)
- mftot, [9](#), [30](#), [33](#), [37](#), [76](#), [78](#)
- model\_prediction, [37](#), [39](#), [43](#), [62](#), [70](#), [100](#), [103](#), [104](#)
- ofv\_criterion, [9](#), [30](#), [33](#), [37](#), [75](#), [78](#)
- ofv\_fim, [9](#), [30](#), [33](#), [37](#), [66](#), [75](#), [76](#), [77](#), [78](#), [81](#), [124](#)
- ones, [11](#), [80](#), [128](#), [132](#), [133](#)
- optim, [81](#), [108](#)
- optim\_ARS, [4](#), [66](#), [85](#), [93](#), [108](#), [111](#), [119](#), [125](#)
- optim\_LS, [4](#), [66](#), [87](#), [91](#), [108](#), [111](#), [119](#), [125](#)
- optimize\_groupsize, [81](#)
- optimize\_n\_eff, [82](#)
- optimize\_n\_rse, [83](#)
- pargen, [95](#)
- plot\_efficiency\_of\_windows, [37](#), [39](#), [43](#), [62](#), [72](#), [98](#), [104](#)
- plot\_model\_prediction, [37](#), [39](#), [43](#), [62](#), [71](#), [72](#), [100](#), [101](#), [102](#)
- poped.choose, [23](#), [24](#), [27](#)
- poped\_gui, [107](#)
- poped\_optim, [4](#), [66](#), [87](#), [93](#), [107](#), [115](#), [119](#), [125](#), [130](#)
- poped\_optim\_1, [4](#), [66](#), [87](#), [93](#), [111](#), [119](#), [125](#)
- poped\_optim\_2, [4](#), [66](#), [87](#), [93](#), [111](#), [119](#), [125](#)
- poped\_optim\_3, [4](#), [66](#), [87](#), [93](#), [111](#), [119](#), [125](#)
- poped\_optimize, [4](#), [66](#), [87](#), [93](#), [111](#), [115](#), [125](#)
- rand, [11](#), [80](#), [128](#), [132](#), [133](#)
- randn, [11](#), [80](#), [128](#), [132](#), [133](#)
- RS\_opt, [4](#), [64](#), [66](#), [87](#), [93](#), [111](#), [119](#), [122](#)

shrinkage, 126  
size, *11, 80, 128, 132, 133*  
start\_parallel, *86, 92, 99, 109, 129*  
summary.poped\_optim, 130  
  
tic, *11, 80, 128, 131, 133*  
toc, *11, 80, 128, 132, 132, 133*  
  
zeros, *11, 80, 128, 132, 133, 133*