PopED Manual

Release version 2.13

Introduction

PopED (Population Experimental Design) is a software tool for computing optimal experimental designs. The software has been developed with an emphasis on drug trials based on population models (non-linear mixed effects models). can

The purpose of this manual is **NOT** to teach Optimal Design but rather to give an introduction and help to the PopED Graphical User Interface GUI and PopED script version. PopED GUI is a Windows based program written in the language C# .NET 2.0 that will wrap around the script version of PopED (written in Matlab) that performs the calculations needed to get an optimal design. The purpose of the GUI is to get an easy way to build up an experiment and to optimize the design variables in that experiment. There are also tools available for interpretation of the outcome of the optimal design and ways to validate models and simulate models prior to the optimal design. All these tools are accessible via the script version of PopED but then the user needs knowledge about the Matlab programming language and how to set up an experiment in the Matlab environment, therefore the GUI was developed to minimize the skills needed by the user in terms of programming.

The PopED GUI provides model templates and examples that will help the user to set up their own experiments. Some knowledge in the Matlab language might be useful but the model templates should give a good introduction to defining PopED models in Matlab.

In Addition to this manual, the Matlab manual and literature in the fields of Population pharmacokinetic and pharmacodynamic (PK-PD) modeling and Optimal Design is suggested.

PopED is a open source program developed by the pharmacometrics research group, department of Pharmaceutical Bioscience, Faculty of Pharmacy at Uppsala University.

The PopED Development Team

Installation

PopED is developed for a Microsoft Windows environment. PopED is mainly tested on Windows XP Professional (Service Pack 2) and Windows 7 platforms.

Requirements

PopED needs the following packages/programs to work perfectly:

1) Microsoft .NET 2.0, use Windows Update or download and install the package from www.microsoft.com. Don't use the SDK for .NET 2.0, (the size is unnecessarily big) instead use the redistributable package.

   or

1) Mono 2.4 (or newer), enables PopED GUI on UNIX-systems / Mac-OS. Download and install it from www.mono-project.com.

2) To run optimization within the Graphical User Interface (GUI), Matlab needs to be installed. The Statistics toolbox and The Symbolic Toolbox is used in some functions. If Matlab is not installed the PopED GUI cancan be used as a help to build up PopED runs. PopED is developed with Matlab version 2007a 7.4 with a registered COM Server from the same version. However, new features in 2.09 should make it possible to run without the Matlab COM Server. To make use of the automatic difference techniques INTLAB ver. >=5.4 should be installed and loaded automatically with Matlab.

   or

2) To run optimization within the GUI, FreeMat version >=4.0 can also be used. FreeMat is an open source version of Matlab and contains most of the fundamental functions in Matlab. However some of the functionality in PopED is not available with FreeMat. Furthermore FreeMat is not automatically optimized and compiled with an optimized BLAS library. To make faster runs with FreeMat a new version should be compiled and linked with an optimized BLAS library for the FreeMat installation environment. Download and install FreeMat from: www.freemat.sf.net. The GUI uses files created in the standard template directory to communicate with FreeMat, therefore make sure that necessary permission to create files in the template directory is set. Finally; make sure that the path to PopED script, e.g. c:\program files\poped\program, is added within FreeMat, I.e. in FreeMat menu Tools/Path tool, add with subfolders and then save.

3) To execute any PopED runs in parallel, either Open MPI (MPI), www.open-mpi.org version >= 1.43 or Matlab Parallel Computing Toolbox (PCT) is

required. PopED works perfectly fine without the parallel execution but having a cluster and/or multi core computers might significantly speed up computer intensive optimal design calculations.

Installation step by step

1) Fulfill the requirements above.

2) Run the setup file (setup.exe) and then walk through the setup wizard.

3) Open PopED GUI from the Start Menu and Open File/Settings. Set the paths to the different directories. PopED GUI can also be started with an input argument specifying other config.xml files than the standard one. This can be useful for multiuser system where each user has there own config.xml.

4) If the PopED program is **NOT** installed in the directory *C:\Program Files\PopED*, and the automatic installation path settings **failed**, change the path in the *getPopEDInstallDir.m* file in the *installation dir\program\xml* directory to your local path.

5) Setup is now completed.

Parallel Installation

If PCT is used (see above) the installation is straight forward and no extra settings, except the toolbox installation, needs to be done to complete the installation. Note that if the PCT is not validated it might be done prior to any PCT runs, see Matlab parallel configuration manager and the validation tool.

For MPI, installation of the MPI system must be completed. In windows it's important that the *\MPI installation folder\bin* is added to the system environment variable *Path*. Furthermore, a C++ compiler that works with MPI must be available (e.g. Visual Studio Compiler, g++ etc). Moreover, the model and some optimal design settings must be compiled into a C-shared library using the Matlab compiler (mcc) and therefore needs a C-compiler that Matlab supports for shared library compilation. The compilation is automatically done by PopED prior to a parallel run by the function *compile_poped.m* which copies important files and compiles them together. The actual compilation is performed in the *compile_fim.m* function and this can be updated for specific systems, installation directories etc. After the C-shared library compilation, *compile_fim.m* continues with a MPI compilation including the C-shared library using a C++ compiler or a wrapped C++ compiler using Open MPI. Two different examples of this is provided in the *compile_fim.m*; A compilation using a wrapped g++ into mpi (mpiCC) which is executed with a make file (*calc_fim_make*). This option is suitable for e.g. UNIX systems. The second option/example is to use a Visual Studio C++ Compiler and link the important MPI-libraries instead of using the MPI wrapped compiler. Change the *iCompilerType* in *compile_fim.m* to change this. Installation directories, compiler paths, MPI paths and needed libraries can all be changed in the *compile_fim.m* and/or in the *calc_fim_make* (if this compiler type is

used). The C++ compilers produce a compiled executable, e.g. calc_fim.exe, that will be executed in parallel using the *mpirun* program.

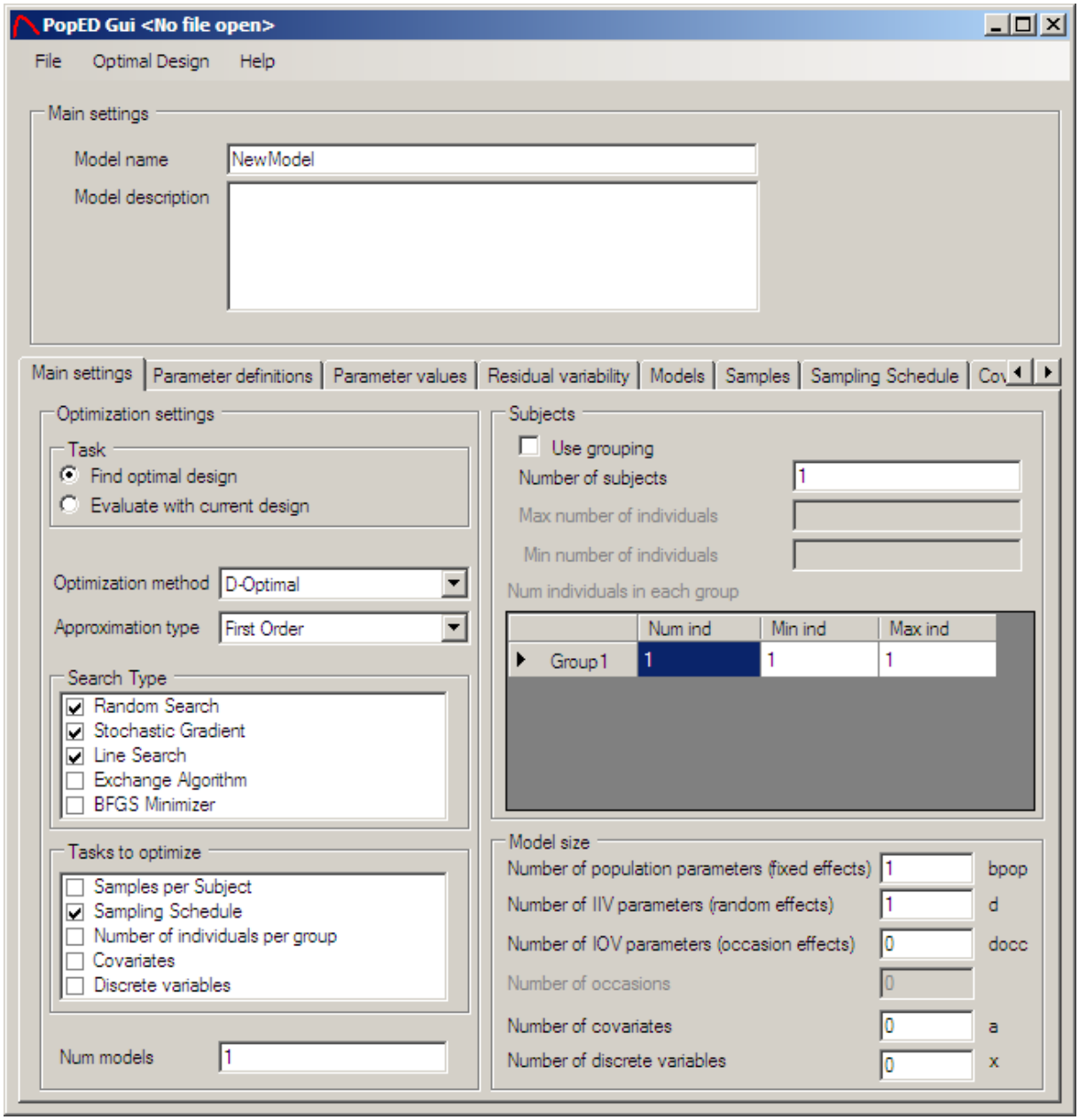
There are three examples on how to start the *mpirun* program in the script folders; running in parallel using a windows batch file (BAT) which is default, running using a shell script file (cluster systems/UNIX) and finally calling *mpirun* directly. These execution ways need to be changed in the compile_fim.m (variable *strAdditionalFiles*) and in the *execute_parallel.m* function where different examples are commented. The files created passing design between the executable and PopED is deleted after each run, to disable this; see *execute_parallel.m,* here settings on how to label the files can be altered. For further information about running parallel examples see the *parallel settings* section below.

If debug or error reporting is needed for the parallel functions using MPI, the debug level can be set by changing the debug flag in *help_functions.h*. Furthermore the batch execution can be stored by redirecting the output, e.g. >> *log.txt* can be added last to the mpirun rows in the *windows_poped_execute.bat*.

Running PopED

Start PopED GUI from the start menu or run the script version (without graphics) of PopED by starting Matlab. In the script version; add the path to the PopED script version (e.g. *C:\Program Files\PopED\program*) to the working directory of Matlab. This can be done by using the *addpath* command or by changing the directory to PopED and then type *poped('')*. A third way is to use the Matlab *Set Path* command under the File menu in the Matlab GUI. In Freemat this can be changed under the Tools/Path tool menu. For more information on how to run the script version with a specific input file see the section *Optimize with PopED*..The GUI can also be started with an input argument specifying the settings file (config.xml). However if no argument is used the config.xml in the installation directory is used. This allows users to specify user specific settings files. To start with a user specified config file, type e.g.:
PopEDGui.exe –conf *user_config_filename.xml*

PopED GUI Main Window



If the PopED GUI version is started the main window will appear. From this window the user can define new PopED GUI settings files, run optimization (if a Matlab COM server is available) and use the diagnostic tools that are available within the GUI.

Creating a new Model with the GUI

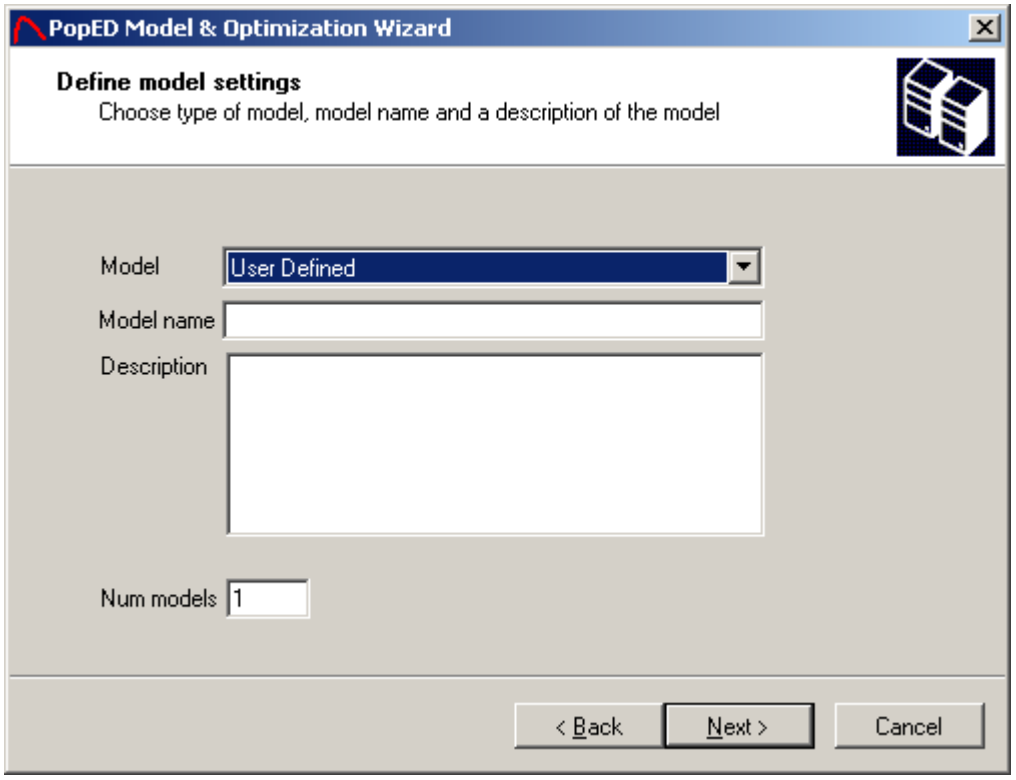
There is several ways to create a new model in the PopED GUI.

1) Under the File Menu choose Wizard and New.
   - Define a user specified model.
   - Use the predefined model templates that are available.

2) Under the File Menu choose New (Ctrl+N).

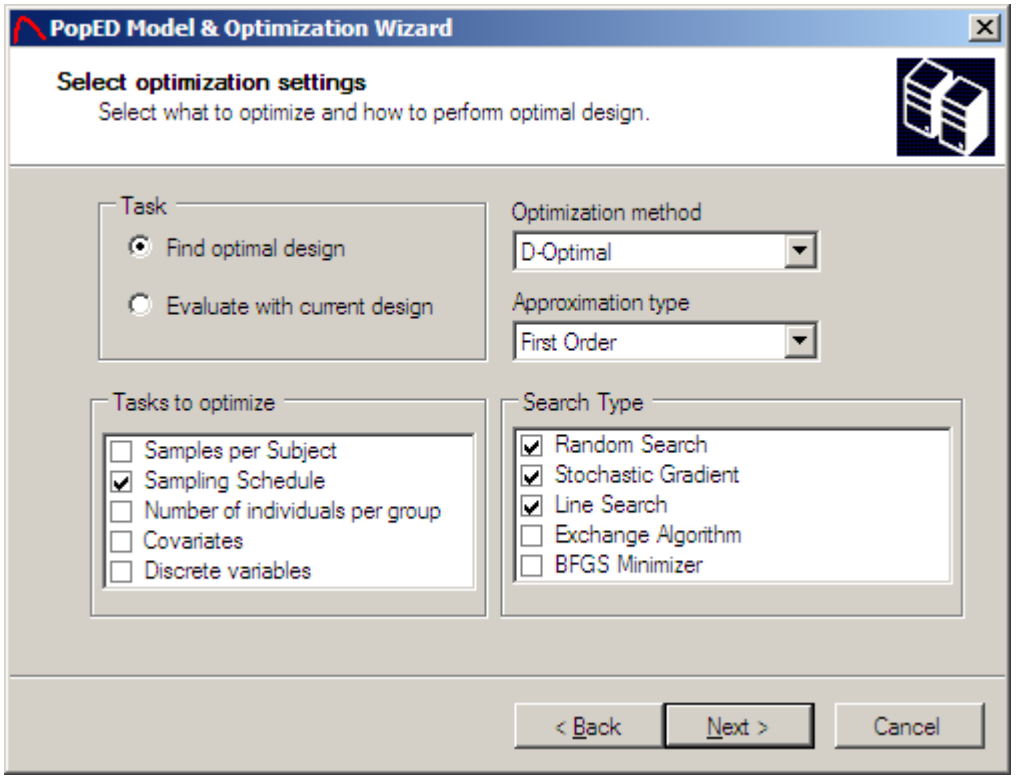
This manual will walk through the creation of a model in the same way as the Wizard does it in PopED.

Define the model settings



Choose a predefined model or choose to create a User Defined model as above. The Model name cancan be any string (containing spaces). The description cancan be any string containing white spaces. To define a new line in the description use the Soft Return, e.g. Ctrl+Return. Num models are the number of sub-models (responses) that are going to be used in the optimal design. E.g. Optimizing on the parameters for both a PK and a PD model will be 2 sub-models, concurrent optimizing on n different drug models will be n sub-models etc.

Optimization Settings



Choose the optimization method:

- D-Optimal Design – Optimization by maximizing the determinant of the Fisher Information Matrix (FIM). A prior model is assumed to be known (the population mean, the Inter Individual Variability and the Residual Variability). Fast but sensitive to model misspecification.

- ln D-Optimal Design – Basically the same at maximizing the determinant of the Fisher Information Matrix but the natural logarithm of the determinant of FIM can be a convex function while the surface of the determinant of FIM is non-convex. In these cases the ln D should be used.

- Ds – Optimization by maximizing the determinant of the Fisher Information matrix (of all un-fixed parameters) divided by the determinant of FIM of the uninterested parameters only. This will allow optimizing for certain parameters and still keeping the correlation to uninterested parameters, instead of assuming that the uninterested parameters are known (fixed). A parameter is considered to be fixed/uninterested in the parameter values wizard window (see below).

- ED-Optimal Design – Optimization by maximizing the expectation value of the determinant of the Fisher Information Matrix. A prior model is assumed to be known but an uncertainty of the model parameters cancan be taken into account. E.g. assuming a normal distribution around all population mean parameters. Slow but can deal with model misspecification.

- In ED-Optimal (API) Design - This criterion is not exactly the same as ED optimal (and are likely to give another optimal design) because it optimizes over the expectation value of the natural logarithm of the determinant of FIM. This criterion can e.g. be useful when convexity is an issue and/or when the value of the determinant is high.

- EDs – Similar to Ds but an expectation over parameters is calculated to allow uncertainty in parameter estimates.

Choose the approximation type

- First Order – The model is linearized around the first derivative evaluated at the typical values (b=0). The residual error model is also linearized around the typical values.

- First Order Conditional – The model is linearized around the first derivative evaluated at the individual mixed effects equal to an individual sample. The number of samples cancan be specified in the search settings. The default is to use 1000 samples per Fisher Information Matrix calculation. The residual variability is linearized around the typical values.

- First Order Conditional Interaction - The model is linearized around the first derivative evaluated at the individual mixed effects equal to an individual sample. The number of samples cancan be specified in the search settings. The default is to use 1000 samples per Fisher Information Matrix calculation. The residual variability is linearized around the individual samples.

- First Order Interaction - The model is linearized around the first derivative evaluated at the typical values (b=0). The residual error model is also linearized around the typical values and an extra interaction terms is used in the linearization. Can be used when e.g. a proportional error structure is used.

Note that if a conditional approximation method is used the calculation of the Fisher Information Matrix can give different results if the individual samples is not the same. It is also notable that the individual samples during an optimization (several calculation of the FIM) will be the same. I.e. the individual samples are **not** re-sampled between each calculation of the FIM.

In most cases the first order approximation linearized around the typical values (the default option) will yield a very similar design compared to the other approximation methods and it is much faster to calculate.

Choose the different design parameters to optimize over:

- Samples per Subject – Number of samples/measurements per design group or individual.

- Sampling Schedule – Optimize when to take the samples.

- Number of individuals per group – Optimizing over the group size.

- Covariates – Optimize for the optimal covariate value in the design. CanCan be any design dependent variable, e.g. Dose, Infusion length, Weight, Age, Study length etc.

- Discrete variables – Optimize over discrete variables (list of discrete values). This can be e.g. list of possible sample times, list of possible doses etc.

The sample times and the covariates are assumed to be continuous, while the Samples per Subject and Number of Individuals per group are discrete variables. A group in Optimal Design is a number of individuals that will get the same design, e.g. the same Sampling Schedule.

Search Types available are:

- Random Search (RS) – Does a random search over the whole search space initially but will collapse to an Adaptive Narrowed Random Search as soon as the Random Search finds a good Objective Function Value (OFV). The Adaptive Random Search will search randomly close to the previous best OFV found by the Random Search. The Random Search is global but it will be local with the Adaptive Random Search. Random Search converges towards an optimal OFV when the number of iterations goes towards infinity. The default number of iterations in the Random Search is 300.

- Stochastic Gradient (SG) – Does a stochastic walk in the direction of the best OFV. In D-Optimal design and ED-optimal design with a discrete user defined distribution there is no stochastic value for the derivative of the OFV and therefore the SG will collapse to a steepest descent algorithm. The SG is a local search method. The SG will converge to a local optimum when the number of iterations reaches infinity. The default number of iterations in SG is 150. When using discrete variables the SG only optimizes with the current best discrete value and therefore calculating the gradient of only the continuous variables.

- Line Search (LS) – Line search does a grid search in one dimension, i.e. over one parameter at a time. Line Search will only work when optimizing over variables like sample times, covariates and/or discrete variables. Line Search is a pure global search method but it will not alone converge toward a global optimum even though the number of grid points reaches infinity. This is because the LS cancan restrict the search space by the order of the design parameter that is used in the LS. The default number of grid points is 50. If optimization is done over a discrete variable the number of grid points will be disregarded for the discrete variables and all possible values will be tested.

- Modified Fedorov Exchange Algorithm (MFEA) – This algorithm searches, in each iteration, the sample/covariate that changes the OFV the most. That sample is exchanged into the current “optimal” design and a new iteration is performed with the current “optimal” design. The possible sample

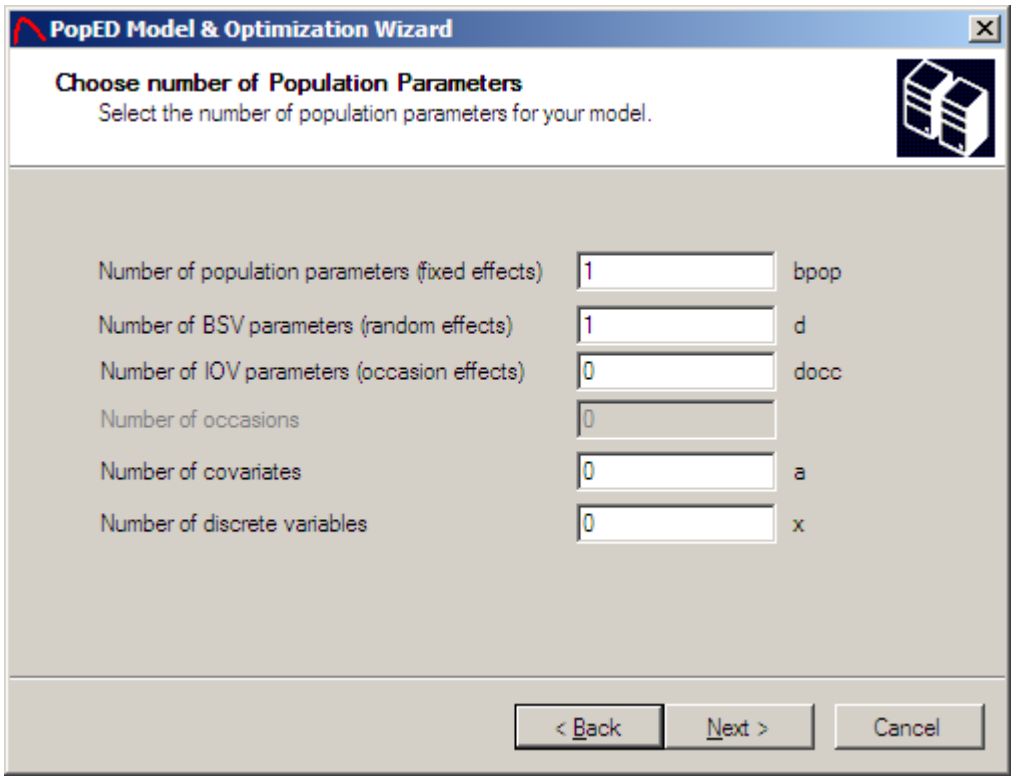
times/covariates to be exchanged are defined by the max/min sample time/covariate value split up with a step length. When the exchange of a sample time/covariate doesn't exceed a threshold value <<1. The threshold value represents the number of percent improvement in the current iteration. If optimization is done over a discrete variable the number of grid points will be disregarded for the discrete variables and all possible values will be tested. See *More search settings* for more information.

- BFGS Minimizer (Broyden-Fletcher-Goldfarb-Shannon) uses a quasi-Newton based method to do a local search. This option might be a faster alternative to use than the standard SG option because it does use the information of the hessian of the search domain while the SG method uses the gradient information. A convergence criteria can be set (see *More search settings* for more information). BGFS can be used together with the SG algorithm, and then the BFGS will be performed before the SG.

All the search types (RS, SG/BFGS and LS) should be used to obtain the best result. The optimization is assumed to have converged if the LS don't change the optimization results of the previous SG/BFGS (or RS if no SG/BFGS is selected). If the Line Search is not selected the method will search with a predefined number of search iterations, i.e. one search iteration run the RS, SG/BFGS or both. See Calculation Settings for more information. The default number of Search Iterations when LS is not selected is 10.

The MFEA algorithm cancan not be used with any of the other algorithms. But it can very well be used for each of the optimization tasks above. It is particularly accurate and fast when optimizing over small lists of possible discrete values (using discrete variables).

Population Parameters



Choose the number of population mean parameters, also called typical values or fixed effects. The number of population mean parameters (bpop) should be the number of typical value parameters for all sub-models. In NONMEM these population parameters are called THETA.

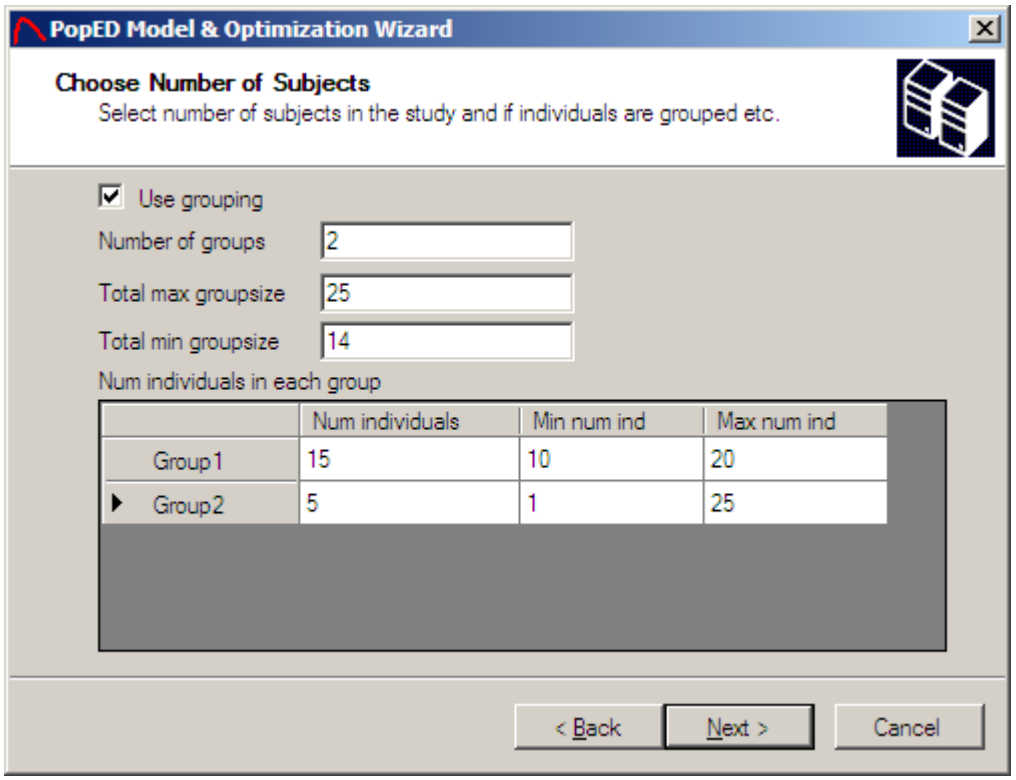
Random effects are the parameters that define the variability between individuals (IIV, BSV). In NONMEM these parameters are called OMEGA. The number of random effects should be the number of random effects for all sub-models.

IOV parameters (docc) are parameters where individual values are different but they come from the same variance within an individual, e.g. an occasion. The number of occasions can be specified and should be the maximum number of occasions for all defined docc.

The numbers of covariates are the number of covariate parameters to have in the model, e.g. Dose, Age, Weight etc. In this version the covariates cancan not have a distribution so they are only continuous variables. The number of covariates should be the number of covariates for all sub-models.

The numbers of discrete variables is the number of discrete variables for all sub-models. This variable can also be used to optimize lists of sample times, in this case, set the number of discrete variables to the number of samples and use *x* in the model instead of *xt* as the time points. See *Define model* and the examples for more information.

Number of Subjects

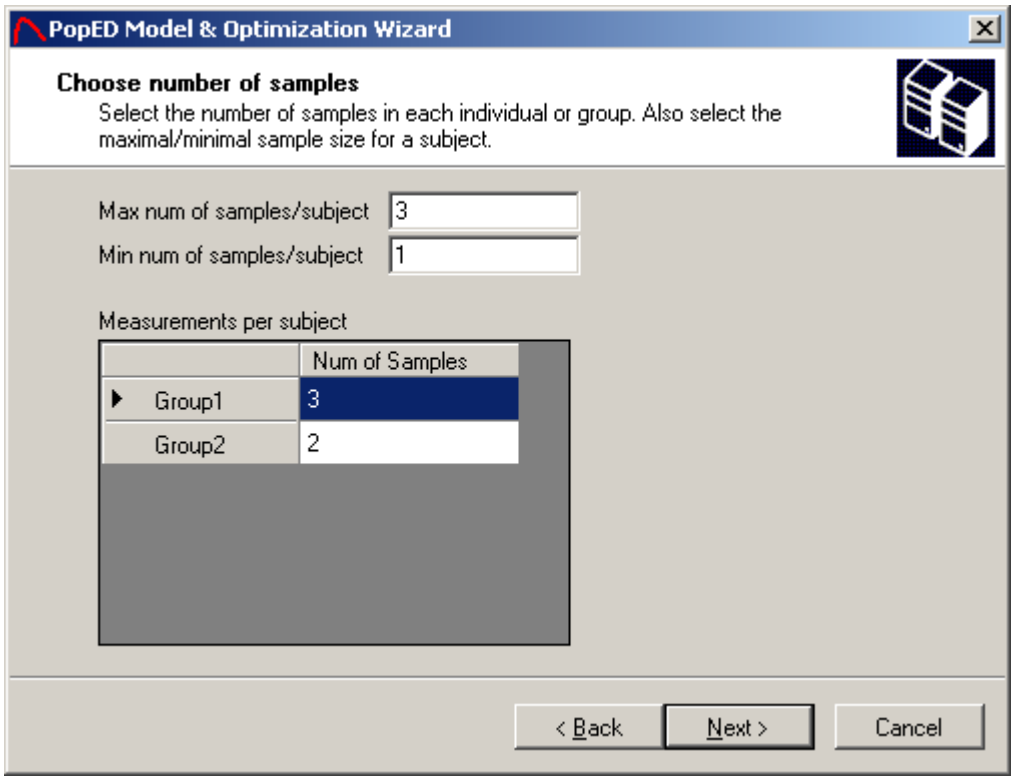


If use grouping are selected there is a possibility to optimize over the number of individuals in each group. In this case it's important to choose the minimal number of individuals in the group and also the maximal number of individuals in the group. If *Use grouping* is not checked, the numbers of individuals in each group are fixed to 1. If *Use grouping* is checked the number of individuals in the group will affect the optimal design even though the optimization procedure only optimize over e.g. the sample times. A group in optimal design is a number of individuals that will have the same design. The *total max groupsize*, i.e. the maximal sum of individuals in all groups and the minimal sum of individuals in all groups must be entered. The sum of initial number of individuals in all groups must be at least equal to the *total min groupsize*. The sum initial number of individuals must also be less (or equal) to the *total max groupsize*.

Example: Optimization of the sampling schedule within a study with at the most, 25 individuals split into 2 groups with initially 15 individuals in group 1 and 5 individuals in group 2. These optimized sampling times will yield one schedule for the 15 individuals and a 2nd sampling schedule for the 5 individuals.

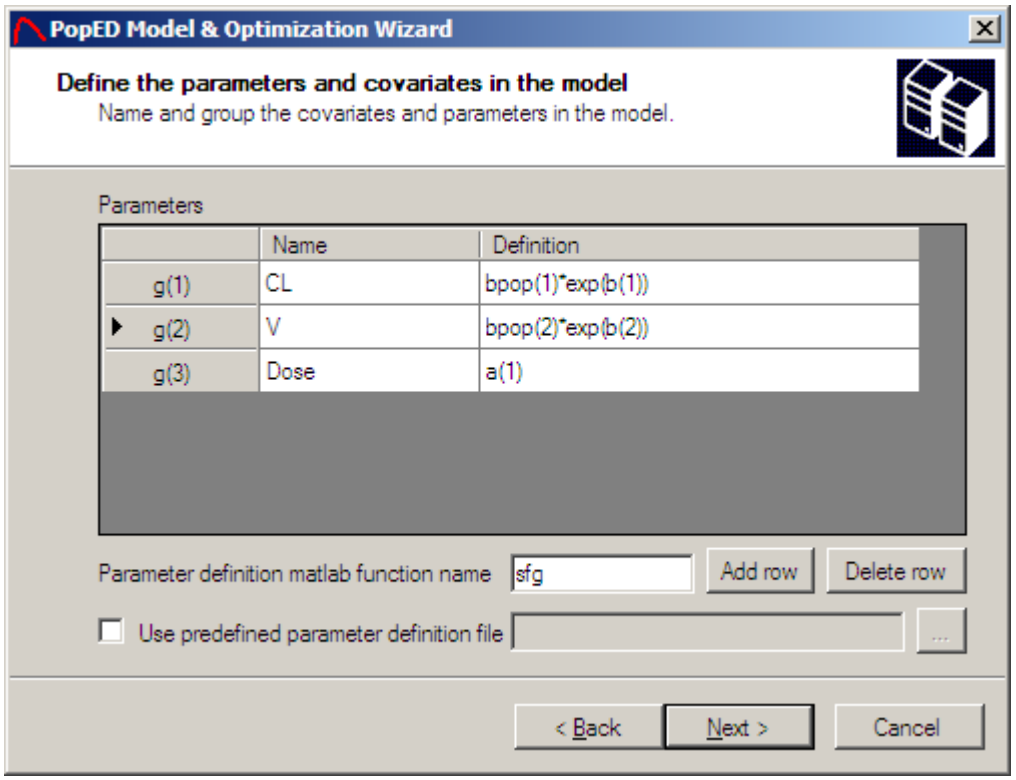
If optimizing on *number of individuals per group*; the optimization procedure will check all possible combinations of individuals divided into the two groups with the restrictions above.

Number of Samples



The number of sample/measurements per subject (group or one individual) cancan be different for each subject. The minimum number of samples per subject should at least be one. The minimum and maximum numbers of samples per subject are only important when optimizing over the number of samples per subject, except that the maximum number of samples per subject should always be larger or equal to the number of samples in each group.

Parameter Definitions



The parameters are defined and stored in a vector called g. The definition of a parameter cancan be of any type, e.g. normal, exponential etc. The typical values (fixed effects) are a vector called **bpop** and the inter-individual variability is a vector called **b,** the occasion parameters is called **bocc**. In NONMEM the b's are called ETA. The different covariates and discrete variables that should be used in the sub-models should also be defined here. The covariates is a vector called **a** and the discrete variables is a vector called **x**. It's perfectly fine to make transformations to cover other IIV distributions than the normal distribution, e.g. the Box-Cox transformation. The Name column (case sensitive) can be used to define the model in the *model definition*. The number of parameters are unlimited but the number of bpop, b, a and x is limited to the numbers specified in population parameters window. The definition is written in the Matlab language and cancan therefore use any Matlab function. *The parameter definition Matlab function file name* cancan be changed to prevent that the GUI overwrites a previous created Matlab parameter definition function. The name should be a valid Matlab function/file name.

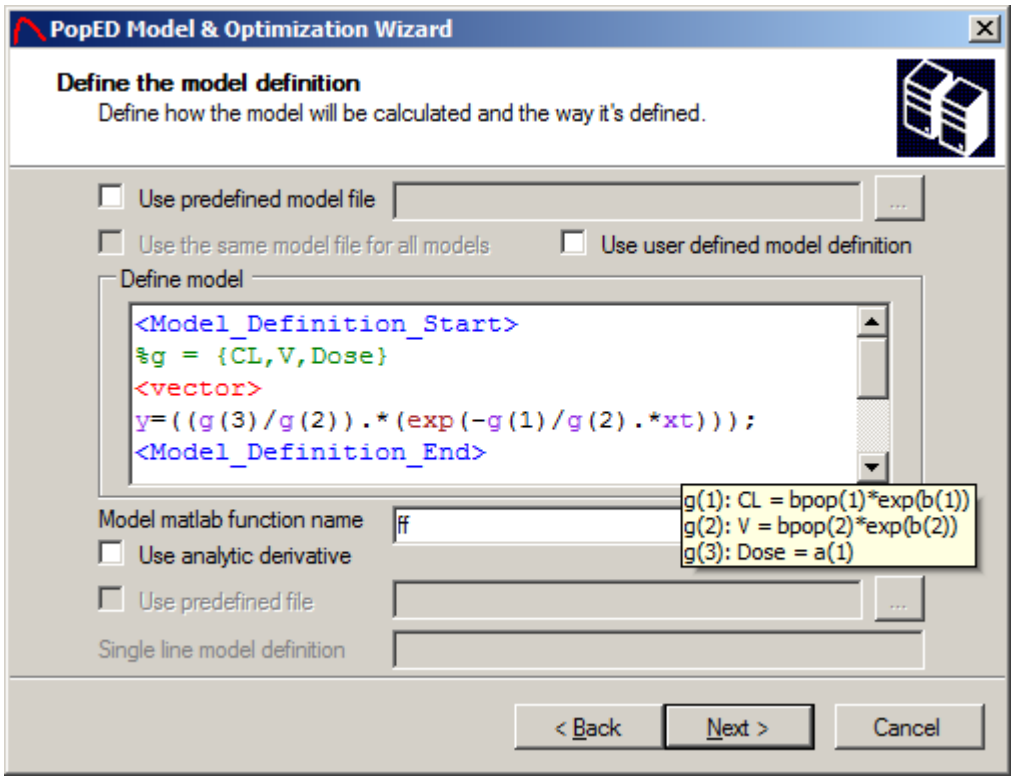
If occasions are used the bocc parameter is index as follows:
bocc(OccasionParameterIndex (1:Number of IOV parameters),OccasionNumber (1:Number of occassions))

It is possible to define a predefined parameter definition file instead of defining the g vector in the GUI. This is done by producing a Matlab function file (*.m) with the following syntax:

```
function pop_params = fg(x,a,bpop,b,bocc)
```

```
% -- Parameter definition file --
% -- { Clearance CL, Volume V, Dose (mg) }

pop_params=[ bpop(1)*exp(b(1)); bpop(2)*exp(b(2)); a(1)];

end
```

Define model



Every sub-model (response) will have a window similar to the one above except that the checkbox *Use the same model file for all models* only will be available at the first sub-model. If this option is checked (or the *Use user defined model definition* is used) no more sub-model definitions are needed even though there is more than one sub-model. In the PopED main window only one tab page for all models will be available. Right click in the model definition will popup a menu that enables to switch between g() and the corresponding names defined in the *parameter definitions*. Hovering over the model definition will show the parameters defined (g's).

The sub-models cancan be defined with a predefined model file or directly in the window by adding Matlab code and tags between the model definition start tag and the model definition end tag. If a differential equation is needed for the current sub-model the code cancan be added within the differential equation start tag and the differential equation end tag.

The *Model Matlab function name* cancan be changed to prevent that the previous model file is overwritten. The name should be a valid Matlab function name.

Analytic derivative can be calculated by checking the *Use analytic derivative* option. To calculate the analytic derivative a single line model definition is needed or a predefined analytic derivative file can be specified. Analytic derivative therefore only works for “simple” models and is very slow to use compared to the numerical derivatives that are used by default. The single line model definition should be written in Matlab code and use the variables mentioned below. Analytic derivatives only work for continuous variable optimization in this version. The predefined analytic derivative file is not fully functional in this version of PopED. Furthermore, there is a restriction that the additive and proportional residual variability must be fixed. The user defined error model can not be used. The Matlab Symbolic Toolbox must be installed to calculate the analytic derivatives.

Predefined model files

The model file can be specified for one sub-model or for all sub-models at the same time (by checking the *Use the same model file for all models* or the *Use user defined model definition* checkbox). If a file is specified only for this sub-model the Matlab function file (*.m) should have this syntax:

```
function [y,globalStructure]=model1(t,g,globalStructure)

% g = {CL, V, E0, Emax, EC50, Dose}
%PK-model
y=g(6)/g(2)*exp(-g(1)/g(2)*t);

end
```

Input should be a scalar value time t and the parameter definition vector g. As output y is set to a scalar value. All sub-models can be defined either by a model file like above or within the GUI independent of the other sub-models.

If a file is specified for all sub-models (even if it’s only one sub-model) the file is again a Matlab function file (*.m) with this syntax:

```
function
[y,globalStructure]=model_example(model_switch,xt,g,globalStructure)

%-- Model: One comp IV bolus with direct Emax effect
%-- {CL,V,E0,EMax,EC50,Dose}

y=xt;
for i=1:length(model_switch)
    t=xt(i);
    if (model_switch(i)==1)
        %g = {CL,V,E0,Emax,EC50,Dose}
        %PK-model
        y(i)=((g(6)/g(2))*(exp(-g(1)/g(2)*t)));
    end
    if (model_switch(i)==2)
```

```
        %g = {CL,V,E0,Emax,EC50,Dose}
        %PD-model
        pkmodel=(g(6)/g(2))*(exp(-g(1)/g(2)*t));
        y(i) = g(3)+pkmodel*g(4)/(g(5)+pkmodel);
    end
end

end
```

The input to this function should be a vector model_switch that define which sub-model a specific time point i belongs to. The length of the model_switch vector is the same as the length of the xt vector containing all the time points for a design subject. The input g is the parameter definition vector g for the current subject. The last input is the *globalStructure* structure that contains information about all design parameters, all the search settings and information about the differential equation solver and differential equation solver options. See advanced settings.

Models defined in PopED GUI

A model can be defined for every sub-model in the PopED GUI. The models are written in an extended version of Matlab code. The models should be defined within the <Model Definition Start> tag and the <Model Definition End> tag. Differential equations for a sub-model can be specified within the <Differential Definition Start> tag and the <Differential Definition End> tag. Three different tags/reserved word are available for the model specification except the regular Matlab code:

- **<vector>** - Specifies that the definition of this sub-model will be vector wise, i.e. the time vector is called xt and the output vector for all the time points xt is called y. If the vector tag is not present the time will be a scalar value called t and the output will be called y(i). The vector tag can only be used in one sub-model; with several sub-models, use t and y(i) as the time and the output.

- **<diff>** - Specifies a call to the differential equation solver, typically used like this: *amount = <diff>;* The amount will be a vector of the amount for every time point xt if the vector tag is used. Otherwise amount will be the amount at time t. If <diff> is specified; a differential equation must be written in the differential definition part of the sub-model. User defined input parameters to the differential equation are stored in the vector *params* that can be of any length. These values are then accessible in the differential equation. If the differential equation contains several compartments, amount will be a matrix with the amount for all compartments and time points. If e.g. compartment two contains the dependent variable/the output it's accessible by typing *amount(:,2)*. This will be a vector if the vector tag is used, otherwise a scalar. For more information about the differential equation solver, see Advanced Settings.

- **<diff_lin>, <diff_lin_inhomc>** - Specify a call to the differential equation solver (see <diff>) but handles only linear homogeneous and linear time-independent constant inhomogeneous odes. Compartment models with simple rate constants are example of linear homogeneous odes. As long as the rate constants are not non-linear (e.g. Michaelis-Menten elimination, feedback

models etc.) this option can be used to speed up the differential equation solver. Look at the typical value plot with <diff> and with <diff_lin>, if they are the same, the <diff_lin> can probably be used. Note that the inhomogeneous solver <diff_lin_inhomc> can only be used with central difference, i.e. not complex differentiation. For an example how to use this, see the distributed example with optimization on 3 drugs.

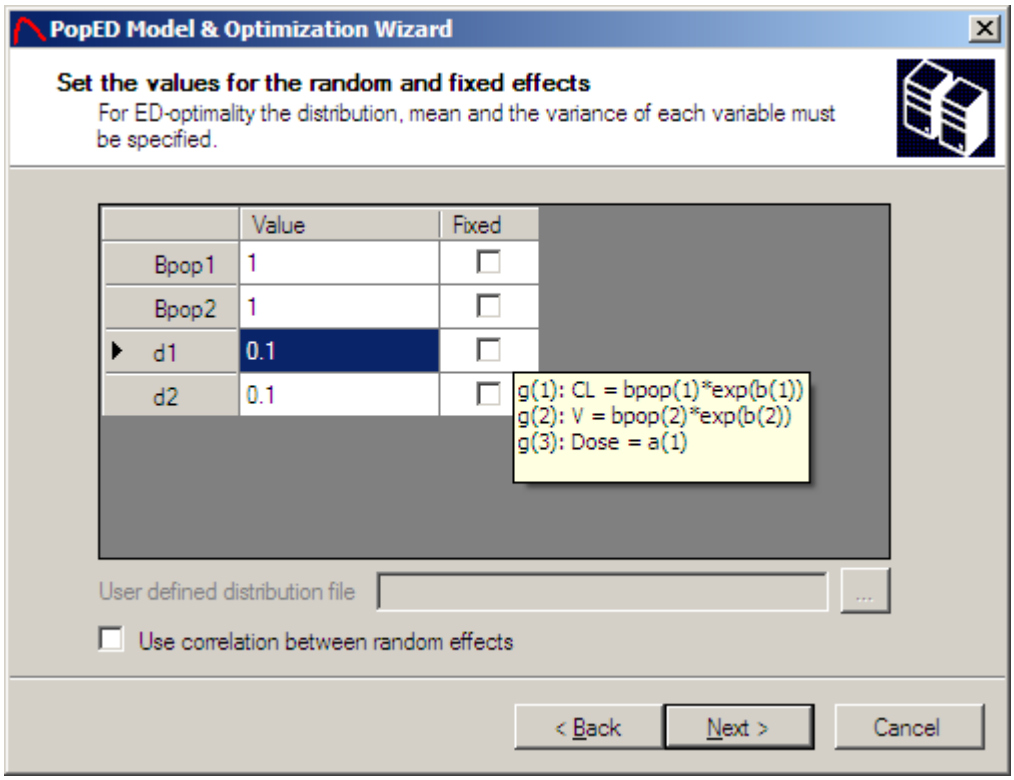
- **<init>** - Specifies a vector of the initial values for the compartments in the differential equation, typically used like this: *<init> = [init1 init2];* where init1 and init2 are the values at compartment one and two at time zero. If the init tag is specified a differential equation definition should be specified.

Example of a GUI defined model:

```
<Model Definition Start>
%Define that operations should be vector wise
<vector>

%g = {ka,ke,V,Dose}
%Store the values needed in the differential equation
params(1) = g(1); %ka
params(2) = g(2); %ke

%The initialization vector for the two compartments (absorption and
central)
<init> = [g(4) 0];

%Call the differential equation
amount = <diff>; %Here the <diff_lin> can be used instead (linear model)
y = amount(:,2)./g(3); %Make the DV a concentration

<Model Definition End>

<Differential Definition Start>

%params(1) = ka
%params(2) = ke
dA(1,:) = -params(1)*A(1);
dA(2,:) = params(1)*A(1) - params(2)*A(2);

<Differential Definition End>
```

See the model templates and the distributed examples for more information about how to code the model files.

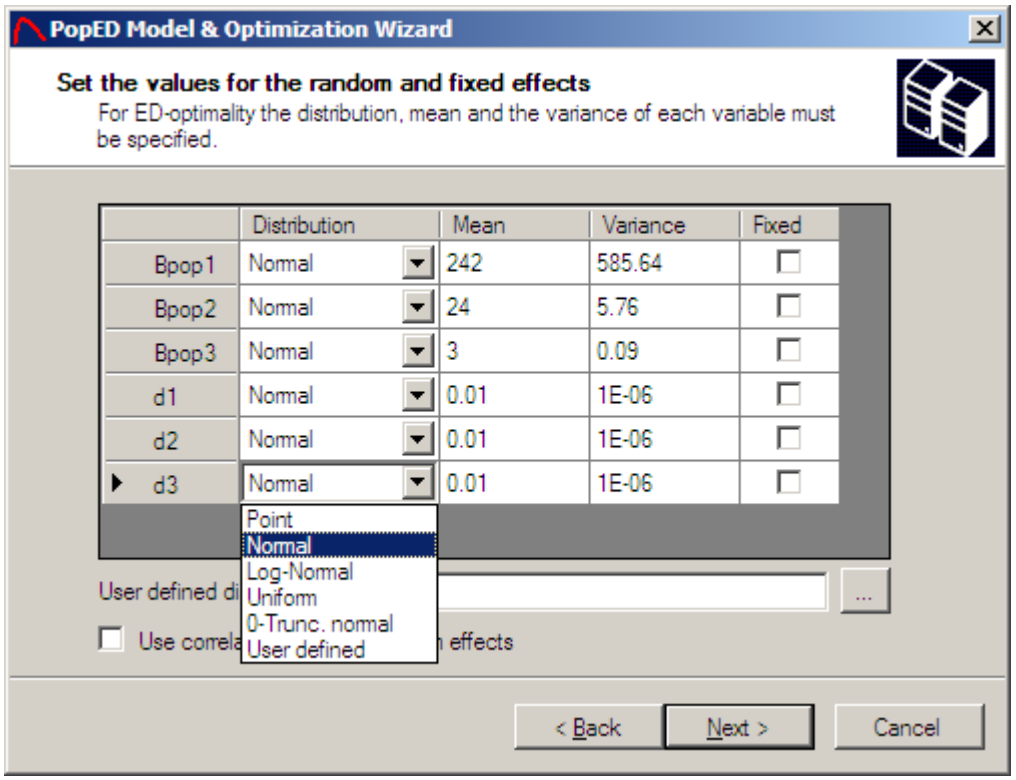
Population parameter values



Enter the model population mean values (bpop, THETA in NONMEM) and the variance of the IIV and IOV normal distribution (d, docc). The d and docc variables are stored as variances, hence a variance 0.01 correspond to 10% inter individual variability for an exponential b or bocc (ETA in NONMEM), i.e. exp(b). The d is interpreted in the same way as the OMEGA in NONMEM.

If there is correlation between the IIV parameters, this can be addressed by clicking the *Use correlation between random effects* checkbox. This will enable a correlation matrix for the random effects. The matrix will be enabled after clicking on the next button.

If ED-optimal design is going to be used the typical values (bpop) and the IIV variance parameters can have a distribution.

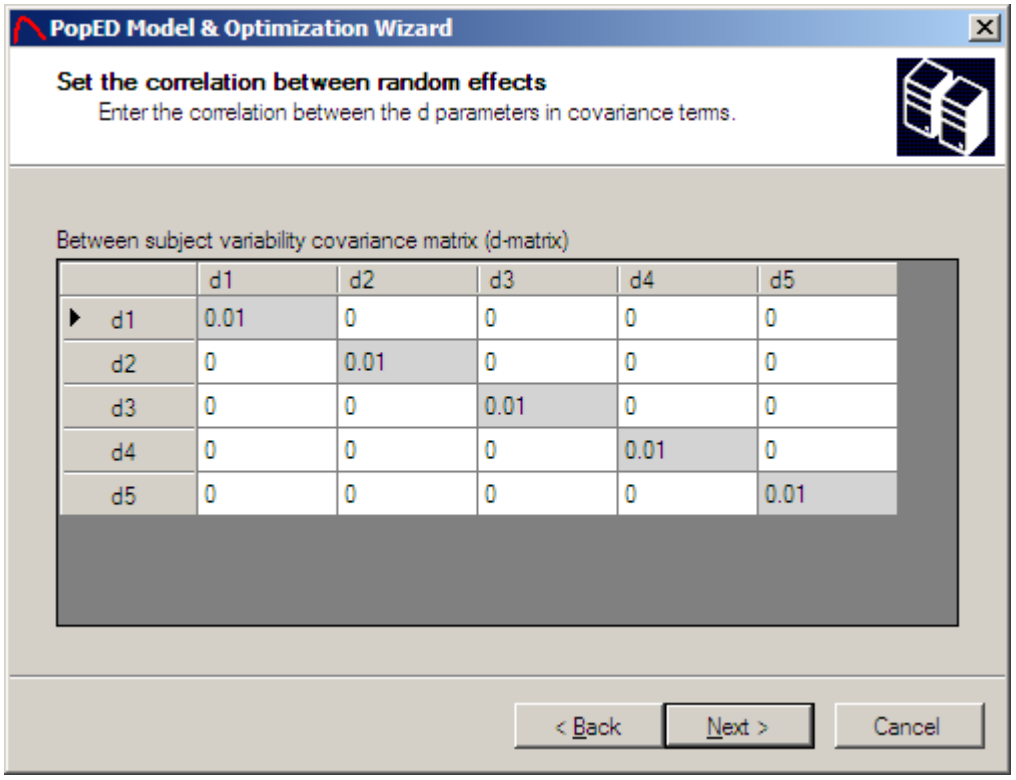


All parameter distributions selected as Point distributions will yield the same result as the D-optimal design. The other available distributions are normal, log-normal, zero-truncated normal and uniformed. There is also possible to define a user specified distribution (stochastic and non-stochastic). To use a user defined distribution; enter the filename pointing to the distribution function. See the examples for more details. In the picture above the variances of the distributions are entered as 10 % but in variance terms. In the uniform distribution the variance is the length of the uniform distribution, **not** the usual definition ($\frac{(length)^2}{12}$).

The zero-truncated normal distribution is left- or right truncated at zero depending on the sign of the mean. I.e. if a negative mean is used the distribution is right truncated at zero and if a positive mean is used the distribution is left truncated at zero.

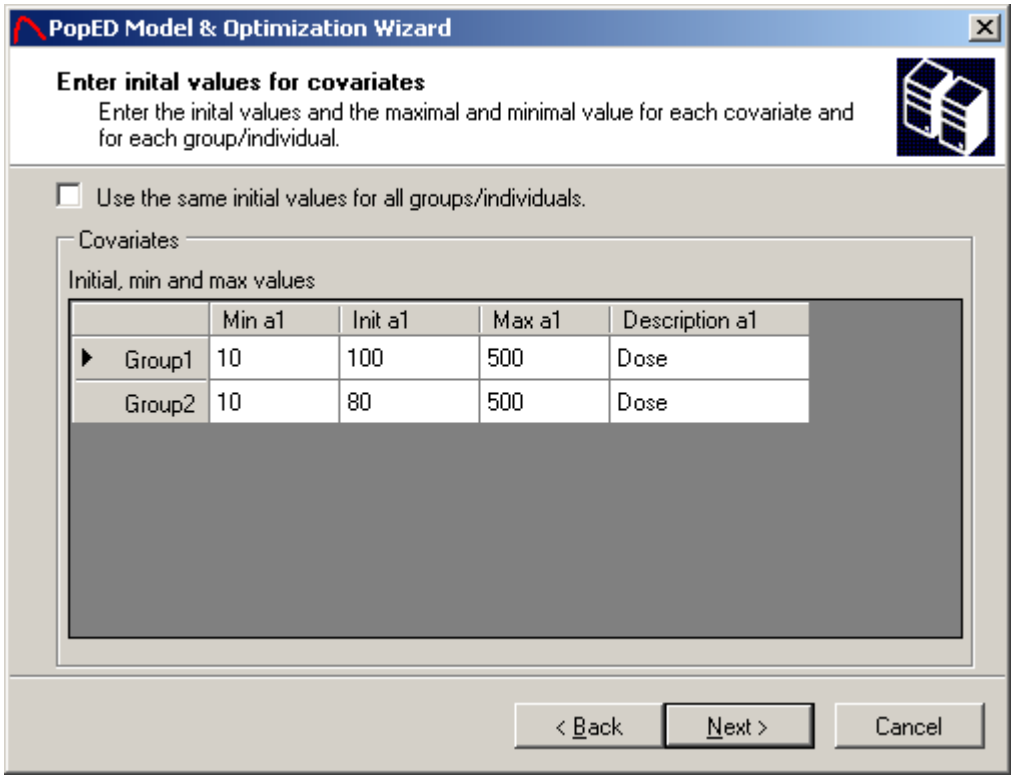
If a parameter is in the model, but is of no interest for estimation, the parameter can be fixed by checking the checkbox corresponding to the variable that is going to be fixed. This means that the variable would not be represented in the Fisher Information Matrix (FIM) and therefore the FIM wouldn't give any expected parameter uncertainty (SE (%)) of that parameter.

Correlation between random effects



Enter the correlation between all random effects expected the diagonal elements, i.e. the actual random effect variance defined in the previous wizard window. The correlation should be entered in covariance terms. A zero means that the random effects are uncorrelated.

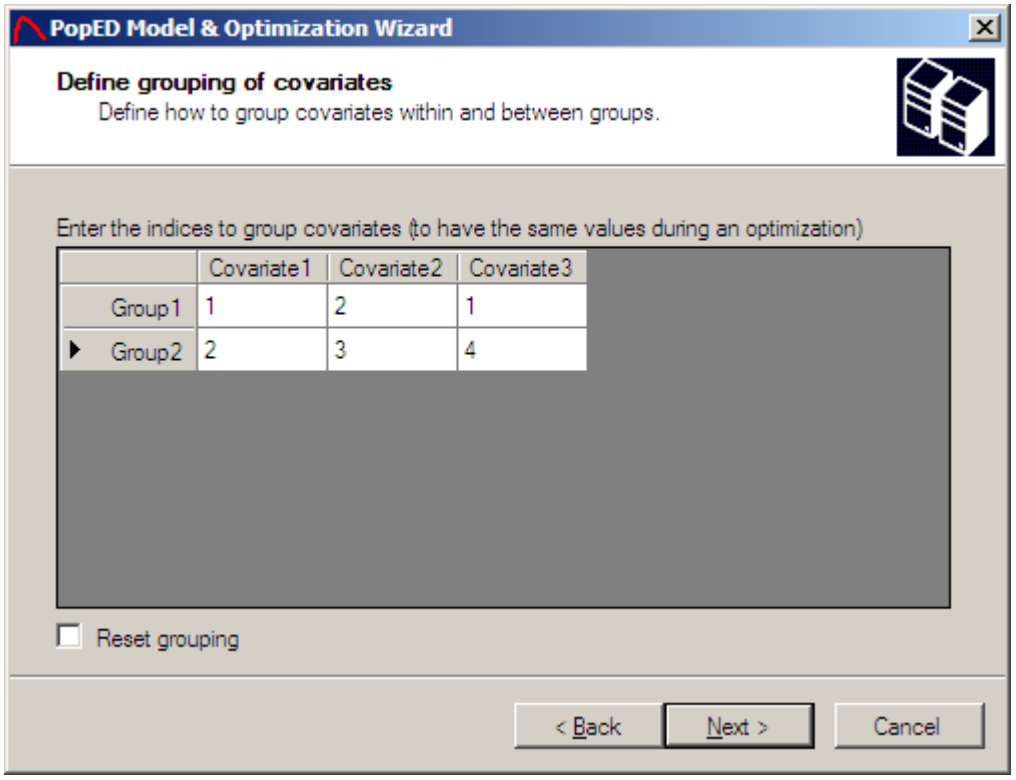
Initial values for covariates



The initial values for the covariates should be entered for all covariates and all subjects (groups/individuals). A minimum value for the covariate and a maximal value for the covariate should be entered as well. The minimum and maximal value is used when optimizing over a covariate. A description for the covariate can be entered as a string. If there are several groups it can be convenient to enter initial, min and max values for all groups at the same time. This is done by checking the *Use the same initial values for all groups/individuals* checkbox. The numbers of covariates are determined by the number specified in the population parameters windows.

A covariate can be fixed even if the optimization method optimizes over the covariate values. To fix a covariate value set the min and max value to the initial value.

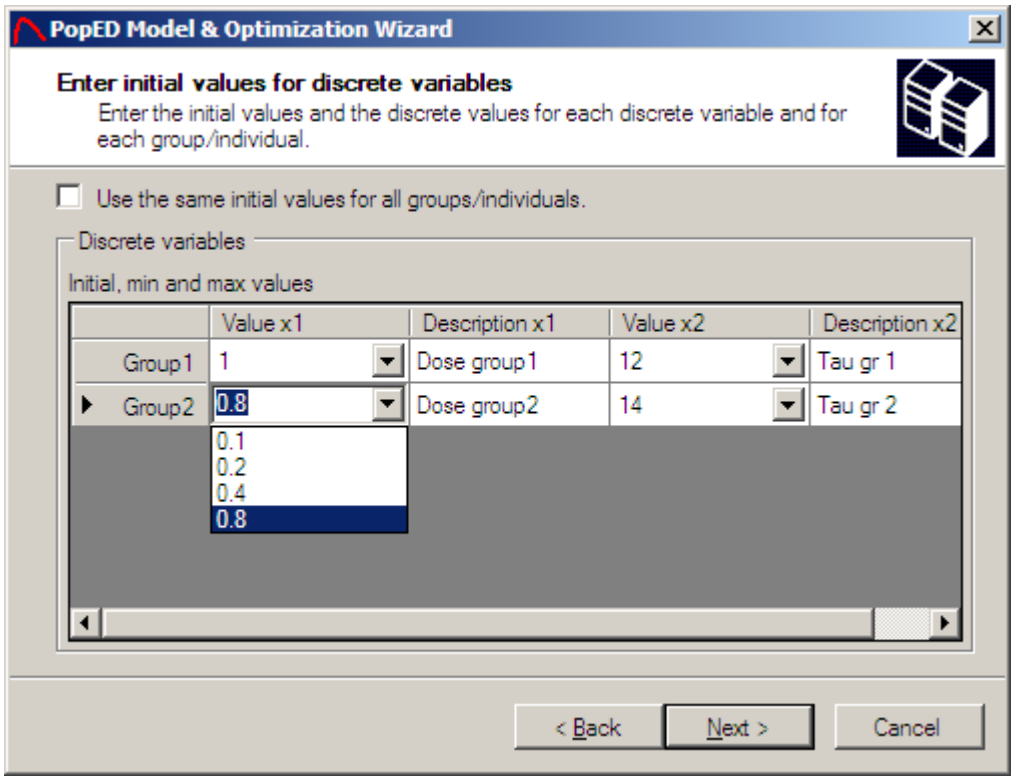
Grouping of covariates



There is a possibility to group covariates between subjects and within a subject.can If covariates are said to be grouped it is similar as they will have the same value during an optimization. In the example above Covariate 1 in group 1 will have the same value as the third covariate in group 1. Also covariate 2 in group 1 will have the same value as the covariate 1 in group 2.

Covariates grouped together should have the same initial, max and min value. The grouping should be defined in an increasing order like the example above. Remove the grouping by clicking *Reset grouping*. Pointing with the mouse on a certain covariate will show the grouping for that covariate.

Initial and Possible Values for Discrete Variables

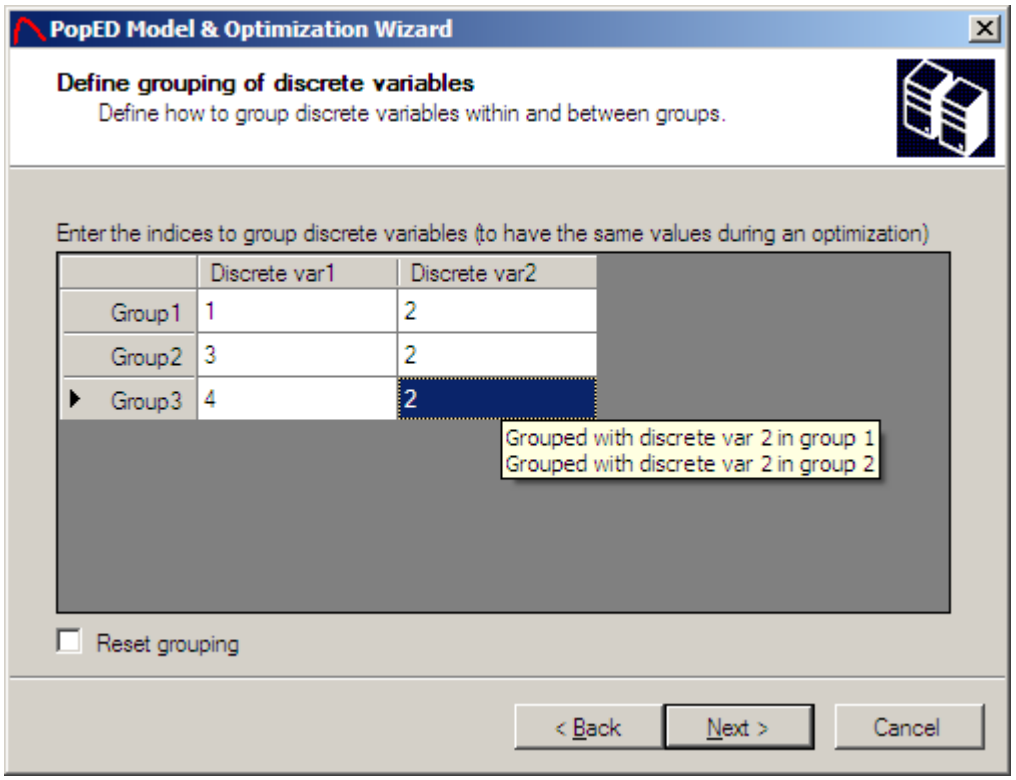


All the possible values for a discrete variable should be entered for each variable and each subject. The initial value for a discrete variable is the value visible in the cell, in the case above 1, 12, 0.8 and 14 are the initial values for the discrete variables. A description for the variable can be entered as a string. If there are several groups it can be convenient to enter initial and possible values for all groups at the same time. This is done by checking the *Use the same initial values for all groups/individuals* checkbox. The numbers of discrete variables are determined by the number specified in the population parameters windows. Note that the number of possible values in each discrete variable doesn't have to be the same, not between different discrete variables within a group and nor between the same discrete variable between two groups (if not grouping of discrete values are used, see below).

A discrete variable can be fixed even if the optimization method optimizes over the discrete variables values. To fix a discrete variable value, have the initial value as the only possible value.

There is a possibility to group discrete variables between subjects and within a subjectcan (see example 1 and below).

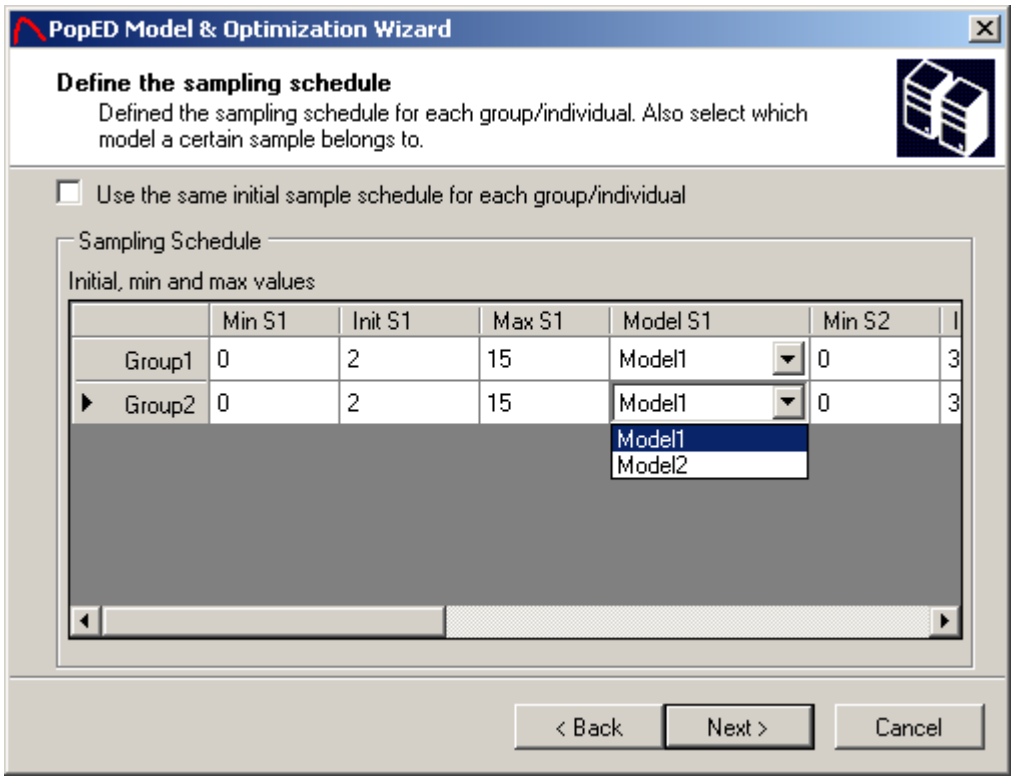
Grouping of Discrete Variables



There is a possibility to group discrete variables between subjects and within a subject. If the discrete variables are said to be grouped it is similar as they will have the same value during an optimization. In the example above the discrete variable 2 is the same for all three groups.

Discrete variables grouped together should have the same initial and possible values. The grouping should be defined in an increasing order like the example above. Remove the grouping by clicking *Reset grouping*. Pointing with the mouse on a certain discrete variable will show the grouping for that discrete variable.

Initial Sampling Schedule

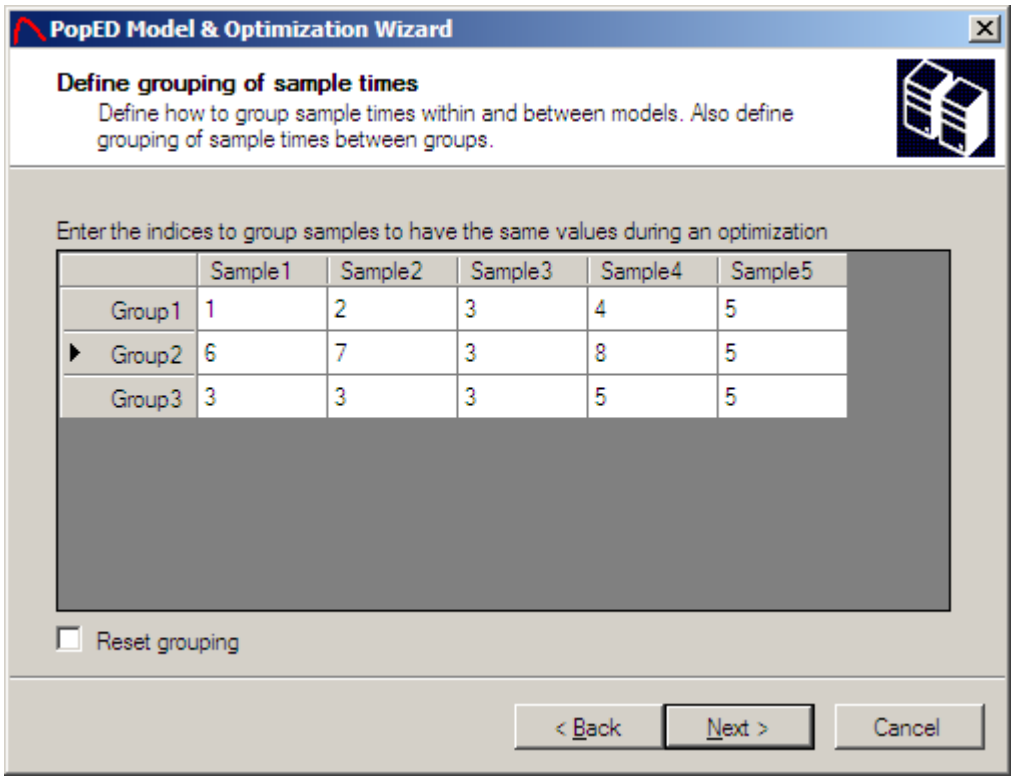


The sampling schedule is defined for all subjects (groups/individuals) and the sample belongs to a sub-model. For all samples; a minimum and maximal sampling time can be specified. This is important when optimizing over the sampling schedule but will not affect the design if the optimization is not performed over the sampling schedule. Number of samples per group will be specified in the number of samples window. If a subject has fewer samples than another subject the subject with fewer samples should define as many samples as the subject with more samples. These samples will not be taken into account when optimizing but are entered to make a consistent matrix of sample times versus subjects. If there are several groups it can be convenient to enter initial, min and max values for all groups at the same time. This is done by checking the *Use the same initial values for all groups/individuals* checkbox.

A sampling time can be fixed even if the optimization method optimizes over the sampling schedule. To fix a sample time set the min and max sample time to the initial sample time.

There is a possibility to group sampling time between subjects and within a subject, e.g. between models (cansee example 1 and below).

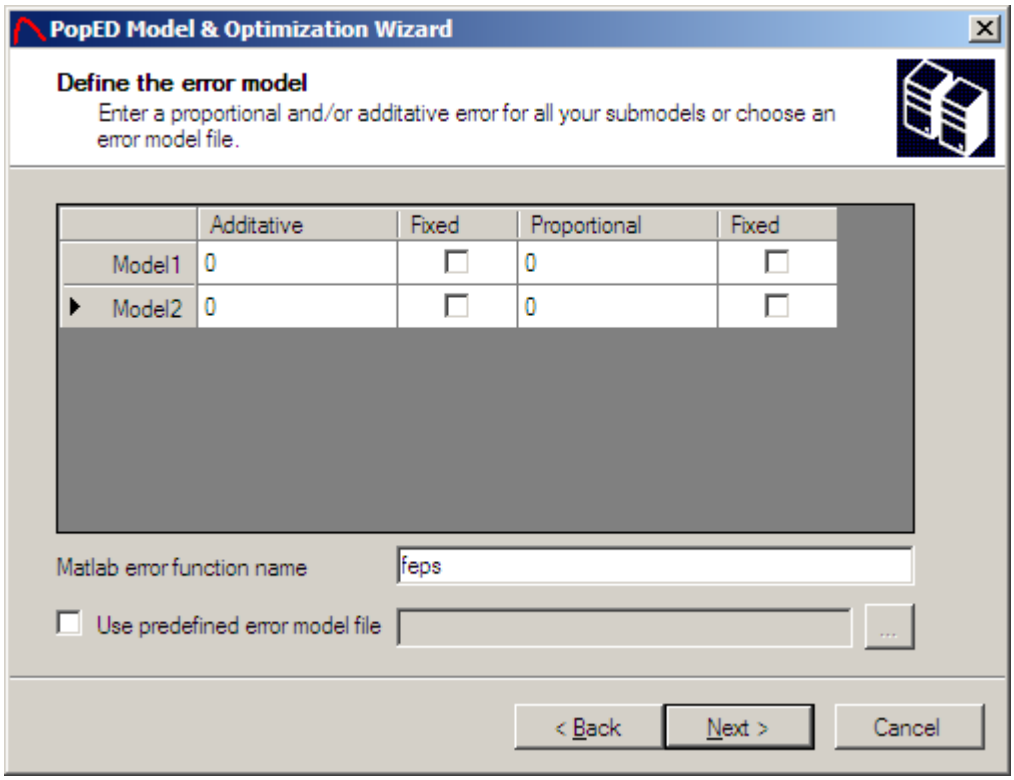
Grouping of samples



There is a possibility to group samples between subjects and within a subject. If the samples are said to be grouped it is similar as they will have the same value during an optimization. In the example above the sample 3 should be the same in all groups. Further the sample 1 and sample 2 in group 3 should be the same as the sample 3 in group 1. Finally sample 5 in all groups and sample 4 in group 3 should be the same as sample 5 in group 1.

Samples grouped together should have the same initial, max and min value. The grouping should be defined in an increasing order like the example above. Remove the grouping by clicking *Reset grouping*. Pointing with the mouse on a certain sample will show the grouping for that sample. It is also possible to group samples that belong to different models.

Define the Residual Variability



The residual variability model can be defined either by an additive part and/or a proportional part or by a predefined error model file. When there is only an additive and/or a proportional part in the error model the error definitions in the PopED GUI works perfectly fine. If more complex error models is going to be used a predefined error model file should be used.

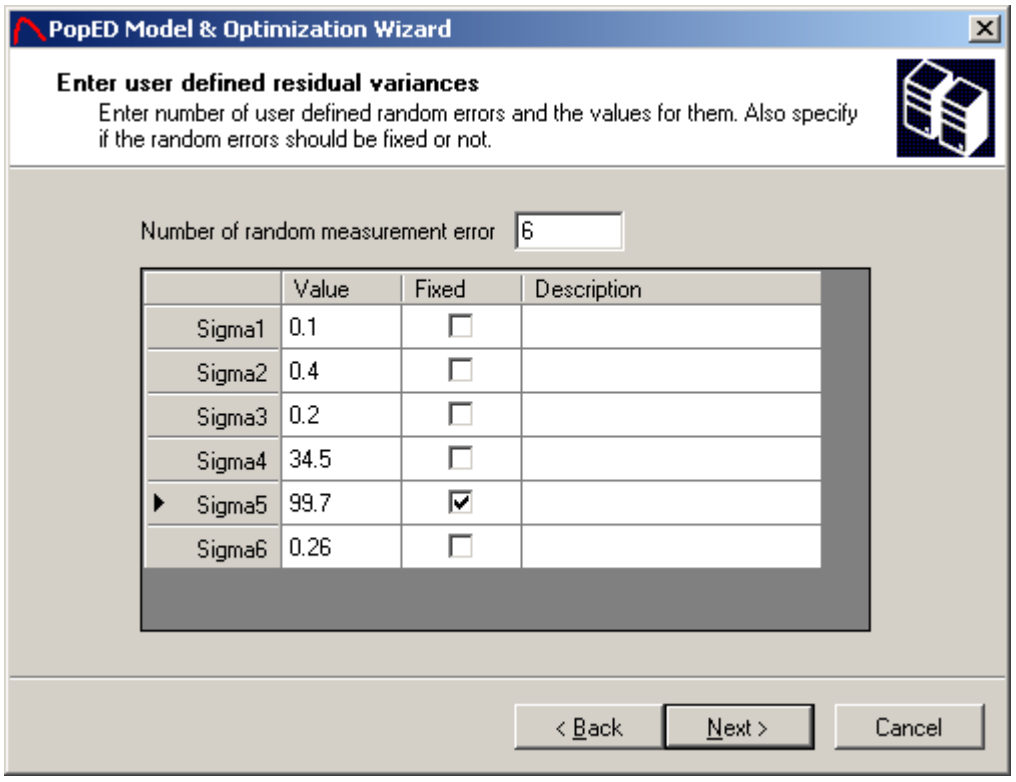
The error values are entered as variances of the residual normal distribution. In NONMEM these variances/variables are called SIGMA.

The *Matlab error function name* can be changed to prevent that the previous error model will be overwritten. This name should be a valid Matlab function name.

If a residual error parameter exists in the model, but is of no interest for estimation, the parameter can be fixed by checking the checkbox corresponding to the variable that is going to be fixed. This means that the variable would not be a row in the Fisher Information Matrix (FIM) and the therefore the FIM wouldn't give any expected parameter uncertainty (SE (%)) of that residual parameter.

In the case with no error model file specified; one additive error and one proportional error can be defined for each sub-model.

If the *Use predefined error model file* is checked a new window were the residual error variances can be entered will be available when pressing the next button.



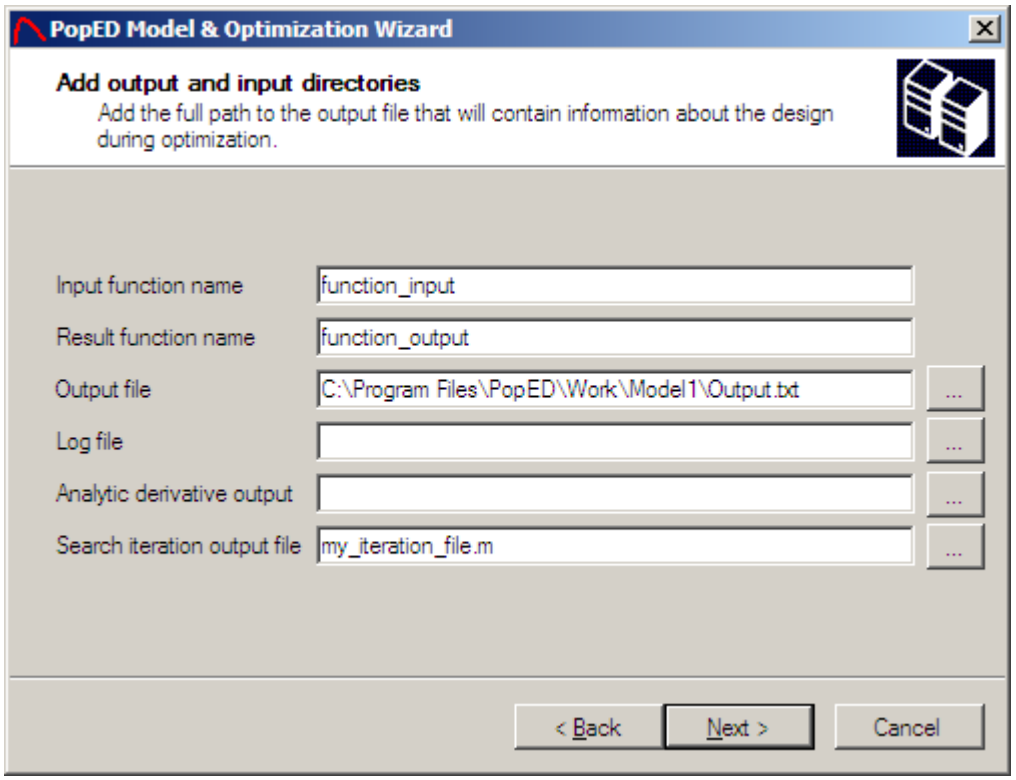
The error model file should be written in Matlab code and have the following syntax:

```
function [y,globalStructure] = feps(model_switch,xt,g,epsi,
globalStructure)

%Error model for model 1 and model 2

[y,globalStructure]=globalStructure.ff_pointer(model_switch,xt,g,globalStru
cture);
for i=1:size(xt,1)
    if (model_switch(i)==1) %The error model for model 1
        y(i) = y(i).*(1+epsi(:,1).*(epsi(:,3).^(epsi(:,4))));
    end
    if (model_switch(i)==2) %The error model for model 2
        y(i) = epsi(:,2) + y(i).*(1+epsi(:,5).*exp(epsi(:,6)));
    end
end

end
```

Input should be a vector with the model switches, the sample time's xt, the parameter definition vector g and the vector with the individual residual error epsi. The last input parameter is a Matlab structure and contains information about the differential equation solver. See Advanced Settings for more information. Epsi is called EPS in NONMEM.

There is also possible to enter a description for the residual error variance parameter. The description can be any string.

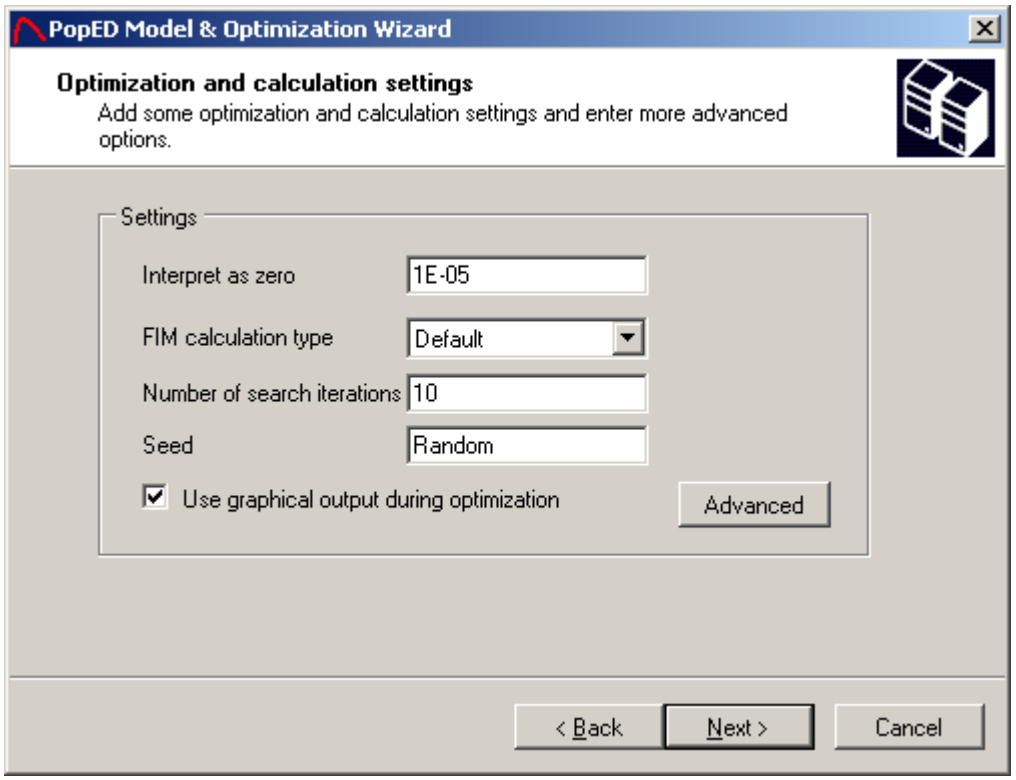
Output and input directories



Enter the name of the Matlab input function and the Matlab result function structure created by Matlab before running PopED. The name should be a valid Matlab function name. The input function will be overwritten each time a new run is started (from the GUI) but the result function will create a new function_output_x where x is the next available number in the running directory.

Also enter the full path to the output file containing information about the current design during optimization. Random Search and Stochastic Gradient will append RS_SG_i.txt to the output path, where i is the search iteration number. If Line Search is used a file with the appendix LS_i.txt, where i is the search iteration, will appear in the output file directory. There must be an output path and file specified to save the optimization settings. The log file is not used in this version of PopED. The Analytic derivate output will store the analytic derivatives for each sub-model in a *.txt file specified by the *Analytic derivative output* path. *Search iteration output file* can be specified to store the best design after each search iteration. This is applied to all search method except the BFGS method very no iteration data is stored. The filename should be a valid m_file. A poped input structure (see script version) can be directly updated from a search iteration file with the `update_popedInput_with_iteration(popedInput,optimalDesign)` function (see *Update input structure from search iteration* for more information).

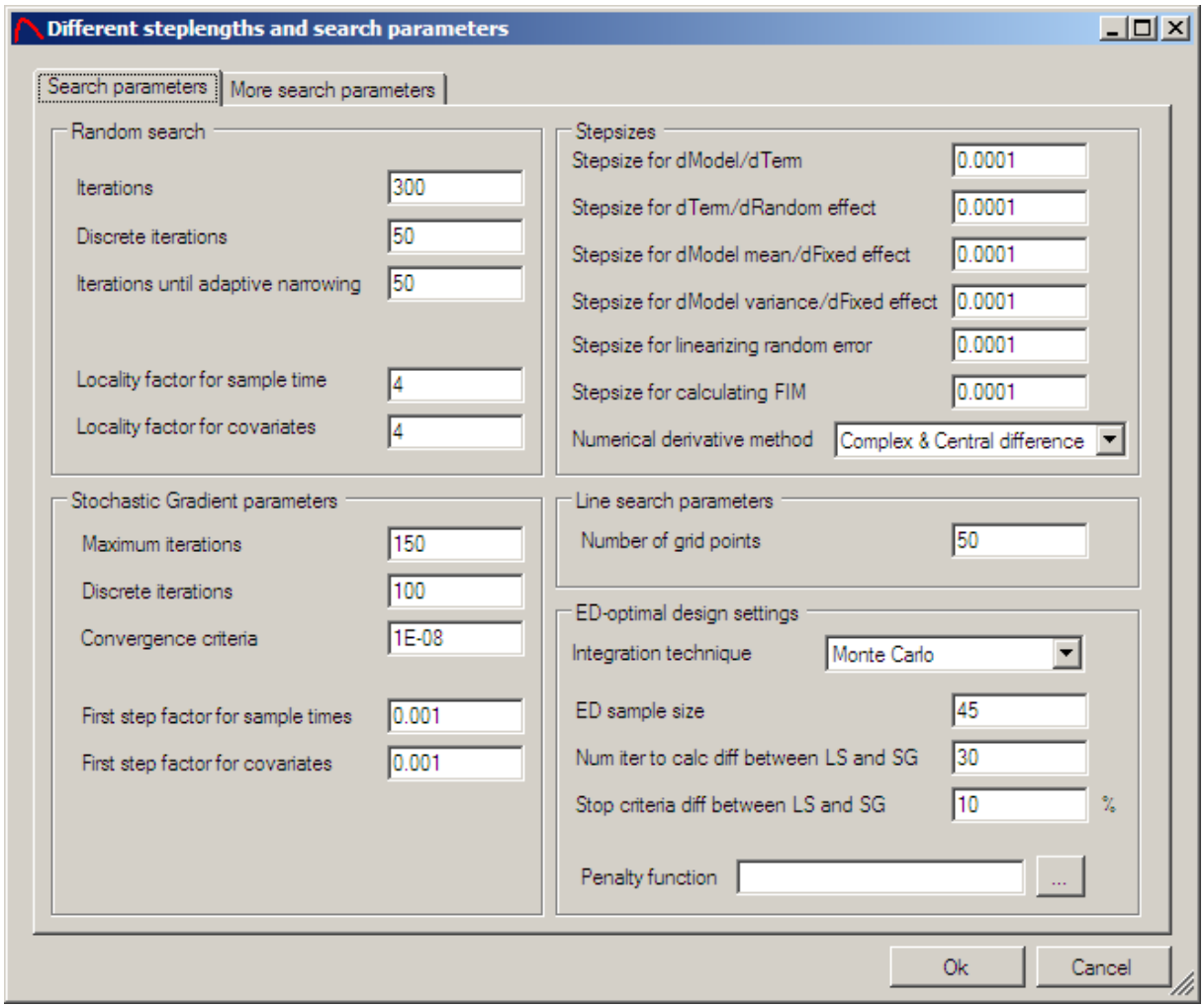
Optimization and calculation settings



When working with numerical problems numerical issues can arise when numbers are exactly zero, therefore PopED needs to define a small number that will be interpret as zero. The Fisher Information Matrix (FIM) calculation type is by default the full FIM calculation but can be changed to a reduced FIM. In the reduced case it is assumed that the derivatives of the variance of the model with respect to the typical values (bpop) are zero. This is a quite ruff approximation but has been shown to give similar design as the full FIM because of the fact that in most models the variance is explained by the inter-individual variability and the residual variability. It's recommended to use the full FIM by default but the reduced FIM can decrease the run time if necessary. The number of search iterations is used when Line Search is not in the search algorithm. See optimization Settings. The seed can be set to a number or bet set to *Random*. A Random seed shouldn't affect the optimization result but can clearly affect the simulation function. See simulate from model. The checkbox *Use graphical output during optimization* will show the current design. In this version it will only show some of the variables, i.e. the sampling times, the covariates and the discrete variables.

Advanced settings for the search algorithms and convergence criterions for ED-optimal design can be entered by clicking the advanced button.

Advanced Settings



A lot of different search settings and step sizes can be entered. Below follows an explanation of the different options.

Random Search Settings

Iterations are the number of random search iterations when optimizing over a continuous variable. The number of discrete iterations is the number of Random Search (RS) iterations to use when optimizing over discrete variables (e.g. Number of samples/Subject, Number of individuals in group). The RS algorithm used is adaptive and therefore it will narrow the search when the number of iterations increases. The adaptive narrowing is started when the Random Search doesn't find a better Objective Function Value (OFV) within the number of iterations entered in *Iterations until adaptive narrowing*. The locality factor for RS will control the localness of the RS algorithm before the adaptive narrowing. If for example a sampling time is in the range of [5-15] hours a locality factor for the sample time of 4 will let the initial RS cover a range of (15-5)/4 = 2.5 hours from the initial sample time. A larger locality factor will produce a better optimization, but then the number of iterations must also be increased.

Stochastic Gradient Settings

Maximum iterations are the maximal number of Stochastic Gradient (SG) search iterations when optimizing over a continuous variable. The number of discrete iterations is the number of SG iterations to use when optimizing over discrete variables (only number of samples per group and group size). The convergence criteria for the SG in ED-optimal design will stop the search when the difference between previous optimal variable value and the variable value after taking a gradient step is less or equal to convergence criteria. In D-optimal design the convergence criteria represent the difference between the current OFV and the OFV of the previous iteration. If the difference is less or equal to the stop criteria the SG search will stop. It's by default set to 1E-08. Stochastic Gradient performs steps in the experiment design variables domain that are guided in direction by the gradient. The magnitude of these steps requires them to be scaled according to iterations history. The first iteration requires a user-defined step, expressed relative to the size of the defined range of the parameters. The required values are the relative values (≤ 1) specifying the size of the first step of the SG algorithm. Increasing the first step factor increases convergence speed, but at the same time increases the probability that the first step moves away from an optimal OFV found by the RS, thus increasing the probability of ending in a local optimum.

Line Search Settings

The number of grid point can be specified in the line-search. That is the number of point each defined range (e.g. a sample time), will be split in. If optimizing over a time point with a range of 5-15 hours and the number of grid points is 50 the OFV will be calculated in 50 points from 5 -15 hours with a step length ≈ 0.2. The best OFV will give the sample time to proceed with. The LS can not be used when optimizing over discrete variables. If LS is used in the optimization the optimization will converge if the LS don't change the OFV from the previous SG or RS. For a discrete variable x the number of points in LS doesn't matter, the number of points will automatically be set to the number of possible values of the discrete variable.

Step sizes

The step sizes are used to define the accuracy when approximating the derivative of different functions numerically. All derivatives used to calculate the Fisher Information Matrix (FIM) are approximated with a central difference approach with a truncation error of $O(h^2)$ where h is the step size. A forward difference approximation is used in the SG algorithm to calculate the derivative of FIM with respect to a continuous variable, e.g. the sample time. This method will have a truncation error of O(h).

ED-Optimal design settings

If the optimization method used is ED-optimal design some ED-optimal settings must be defined. The *integration technique* defines how to solve the expectation integral over the parameter space. The standard way is to use sampling techniques (Monte Carlo) to solve to integral. A second option is to use a second order approximation

using the hessian matrix, i.e. the *Laplace approximation*; finally a Laplace approximation where the hessian is built up with a BFGS optimization algorithm can be used. For the *Laplace* options the *bpop* can be a normal/uniform or log-normal distribution but the *d* distribution has to be a log-normal distribution. The BFGS minimization is not exactly the same as the normal search BFGS (see BFGS algorithm), in the normal case a box-constraint BFGS optimization is performed while in the integral solving an unconstraint BFGS optimization is performed, The sampling technique affects the design optimization and the 3d-plot function but is not used in the efficiency translation nor the sampling windows calculation, instead the Monte Carlo method is always used in those functions. ED sample size is the number of samples that will be drawn from the parameter distribution to calculate the expectation value. One Fisher Information Matrix is calculated for each ED - sample, therefore the execution time will increase linearly with the number of samples. If a user defined distribution is used that are not stochastic this number should be set to the number of parameters in the user defined distribution and the stop criteria below can be set to a very small number e.g. 0.01%. The number of iterations to calculate the difference between SG and LS is the number of calculations of expectation values that will be used to compare the LS and SG. For example an ED sample size of 45 and the number of iterations to calculate difference between SG and LS set to 30 will yield 45*30 = 1350 calculations of the FIM. When using a user defined distribution that is not stochastic the number of iterations should be one. This is only to have an automated way to look if the search has converged. The ED search converges if the LS don't find a better OFV within the previous SG value multiplied with the percentage value.

If a penalty function is used the objective function value to optimize on can be specified in the penalty function. This enables the user to write e.g. user specified cost functions based on the FIM to be optimized over. This can also be used to optimize over D-optimal cost functions by setting the ED sample size to 0 and write the OFV in the penalty function. Other use of the penalty function is to add penalty to different types of ED-samples and to add restrictions in general to the design.

The penalty function should have a function header defined like this:

```
[ED_fim,ED_ofv,globalStructure]=penalty_function(fim_list,bpop_gen,
                      d_gen_list,docc_gen_list,model_switch,groupsize,ni,xt
                      optn,xoptn,aoptn,bpopdescr,ddescr,covd,sigma,docc,
                      globalStructure);
```

fim_list is a vector with OFV values calculated for each ED sample. If the ED sample size is set to 0, this vector will be empty.

bpop_gen is a matrix with each row of the matrix assigned a vector with the typical values used to calculate the OFV. If the ED sample size is set to 0, the matrix will be empty.

d_gen_list is a list with each element of the list assigned a vector with the inter individual variances used to calculate the OFV. If the ED sample size is set to 0, the list will be empty.

docc_gen_list is a list with each element of the list assigned a vector with the occasion variances used to calculate the OFV. If the ED sample size is set to 0, the list will be empty.

model_switch is a vector defining which sub-model a certain sample belong to.

groupsize is a vector with the current optimal group size for each group.

xoptn is a matrix with the current optimal discrete values for each group.

aoptn is a matrix with the current optimal covariate values for each group.

bpopdescr is a matrix with the mean values, the variances and the distributions of each typical value.

ddescr is a matrix with the mean values, the variances and the distributions of each inter individual variance.

covd is a vector defining with the lower triangular matrix of the covariance between each inter individual variance.

sigma is a matrix defining the residual variances.

docc is a matrix with the mean values, the variances and the distributions of each occasion variance.

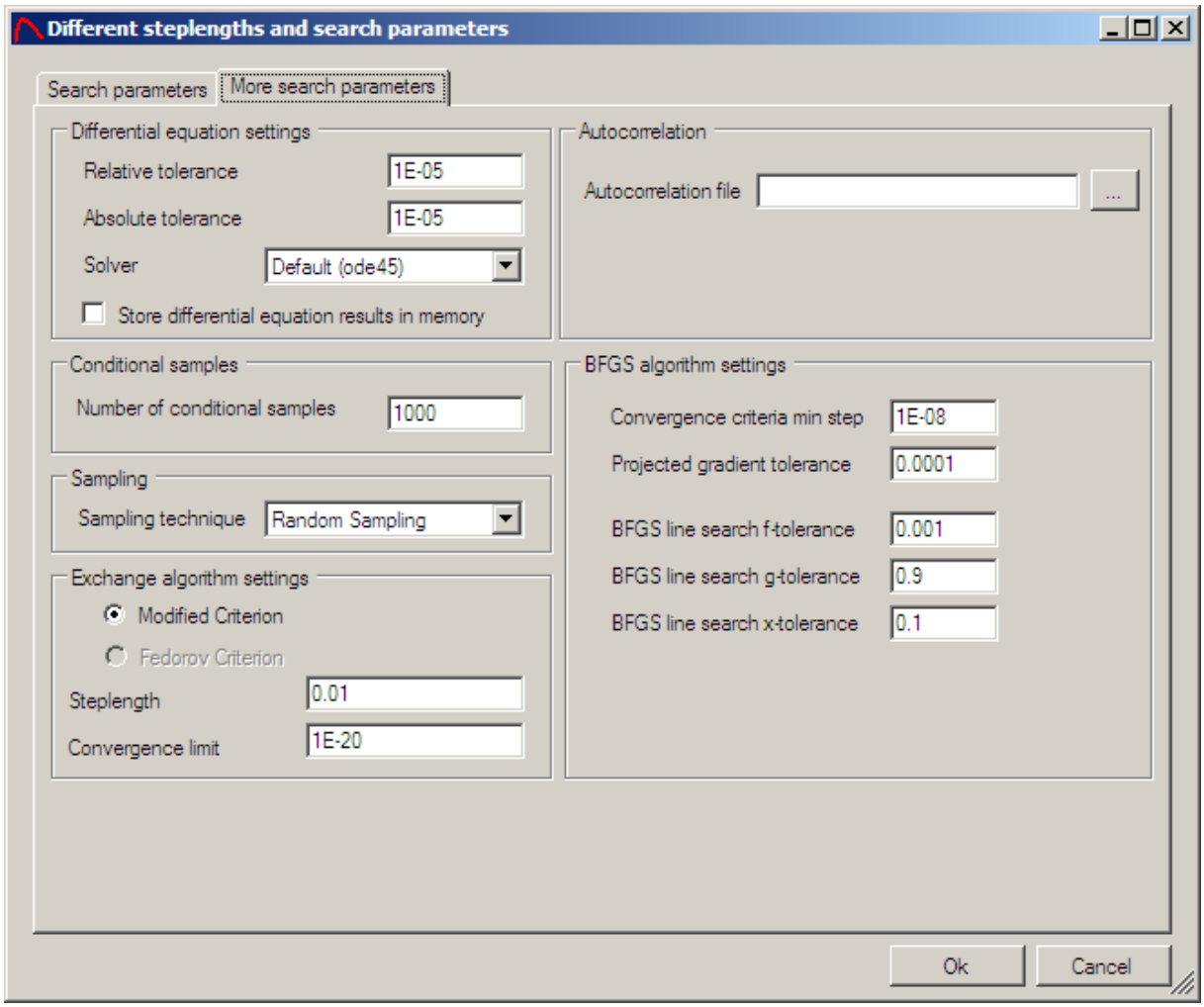
globalStructure is a structure that contains all variable that PopED uses..

ED_fim is a returning value that should contain the Fisher Information Matrix calculated in the penalty function. The matrix should be scaled to the number of ED samples, i.e. it should be the expectation FIM.

ED_ofv is a returning value that should contain the OFV that should be maximized, calculated in the penalty function. The OFV should be scaled to the number of ED samples, i.e. it should be the expectation OFV.

globalStructure is a return structure, see above.

The optimization procedure will try to maximize the OFV based on the penalty function with respect to the design parameters, i.e. the *samples times*, the *discrete variables*, the *covariates*, the *groupsize* and the *number of samples in each group*.



Differential equation settings

For the differential equation solver the relative tolerance (RelTol) and the absolute tolerance (AbsTol) must be set. The solvers will then have an error precision of $e_i = \max\left(relTol \cdot |y_i|, absTol\right)$. The solver method is by default the *ode45* solver but can be changed to a stiff solver (*ode15s*) if the differential equation is stiff. The RelTol, AbsTol and differential equation method can be changed "on the fly" during an optimization. *Store differential equation results in memory* enables the user to store the solutions of the differential equation solver in the memory. This can increase speed a lot when optimizing over number of samples or sampling times. The option should **not** be used when optimizing on continuous covariates.

Numerical derivative method

There are different methods to calculate numerical derivatives. In PopED two different approaches (except analytic derivatives) are available:

- Complex & Central difference – PopED uses complex differentiation when possible, otherwise central difference is used. The complex approach is very robust and accurate for small step lengths and will sometimes be twice as fast as the central

difference approach. The precision for the first order complex difference is $O\left(h^2\right)$ where *h* is the step length.

- Central difference – PopED can use the central difference approach. This can be valid when the function cann't handle complex numbers or when the complex approach is slow due to complex numbers. The precision for the first order central difference is $O\left(h^2\right)$ where *h* is the step length.

- Automatic difference (AD) – PopED can use the automatic differentiation techniques available in INTLAB to derive the derivatives with the precision of the Matlab functions used to define the model. This is almost/the same as doing analytic derivative in terms of precision. However the method is slower than central and complex difference and needs INTLAB to work. To use this option; install INTLAB v>=5.4 (INTerval LABoratory) and make sure that INTLAB is added to the Matlab Path.

Number of conditional samples

If the approximation type is first order conditional or first order conditional with interaction the number of individuals samples can be specified here. A large number of samples will yield a more robust design against different individuals in a coming study but will take more time to compute.

Sampling technique

The sampling technique is used to generate simulations, to generate sample for ED-optimal design and to generate samples for conditional designs. The normal technique uses standard random numbers (see Matlab manual) while the Latin Hypercube Sampling is more evenly distributed within the sampling region. This can have advantages in ED-optimal design in terms of faster convergence and robustness. This sampling technique is used for expectation calculation in robust design if the ED integral technique is set to Monte Carlo.

Exchange Algorithm settings

The Exchange Algorithm settings defined the step length and the convergence criterion when the exchange algorithm is used. In the current version only the modified criterion can be used. This criterion will make the search stop if the percentage change of OFV is less than the criterion. The step length defines how the samples times/covariates should be tested. I.e. if the max and min border of a sample time is t=[0,10] and a step length of 0.1 is used, the samples times tested will be [0, 0.1, 0.2, …., 9.9, 10]. If the script version is used the step length can be changed to number of points instead which prevents the search space to be too large when the borders of the sample times and covariates are unequal. If e.g. 50 points are used the sample times/covariates tested are (max-min)/numpoints. When optimizing over discrete variables, i.e. x, the number of points will be the number of possible values for a certain discrete variable.

Autocorrelation file

PopED enables a possibility to define a user defined variance term of the linearized model. This enables features as e.g. optimizing on models with auto correlation. The filename (*.m) and the path of the user defined variance must be specified.

The user defined variance Matlab function should have a header defined like this:

```
var=user_variance(h,model_switch,xt,x,a,bpop,b,
           bocc,d,sigma,docc,l,locc,interact,globalStructure);
```

h is a matrix containing the linearized residual model with respect to the residual parameters.

model_switch is a vector defining which sub-model a certain sample belong to for individual/group i.

xt is a vector with the current optimal samples for individual/group i.

x is a vector with the current optimal discrete values for individual/group i.

a is a vector with the current optimal covariate values for ind/group i.

bpop is a vector with the typical values.

d is a matrix with the inter individual variances and covariances.

sigma is a matrix defining the residual variances.

docc is a matrix with occasion variances.

l is a matrix containing the linearized model with respect to the individual parameters (b).

locc is a cell vector where each cell contains matrix with the linearized model with respect to the individual occasion parameters for each occasion. The length of the vector is equal to the number of occasions.

interact is a matrix containing the interaction between the residual model and the model.

globalStructure is a structure that contains all variable that PopED uses.

var is a return value that should contain the linearized variance model belonging to the residual error, the size should be num_samples x num_samples for individual/group i. Note, *var* is only the linearized variance with linearization purely done for the residual error model. If the complete variance of the model should be user defined, the var must subtract the linearized model with respect to b, the occasion part and possibly the interaction variance if interaction is used.

BFGS algorithm settings

The BFGS minimizer uses a line search method to calculate the gradient step. The f,x and g –tolerance are settings for the line search algorithm. Check the line_search.m for more information about these parameters. The *Convergence Criteria Min Step* is the smallest allowed difference between two steps in the algorithm; if a step is smaller than this the search will stop. This difference is absolute or relative (%) to the ofv according to: [ofv-ofv_step]<=”Convergence Criteria Min Step)*max(|ofv,ofv_step,1|). The search continues until the normalized projected gradient is larger than the *Projected gradient tolerance* or the min step criteria described above is met.

Prior Fisher Information Matrix

Information from a previous study can be entered to the current design evaluation/optimization by adding prior FIM. This can be done by entering a prior FIM, e.g. the inverse of a covariance matrix for some or all parameters. Note that the size (number of parameters) of the prior FIM should be the size of the current FIM, number of unfixed parameters. If no information is available for a parameter, enter 0 for that parameter and the corresponding covariance’s. The order of the parameters is [bpop1..bpopn,d1..dn,covd1..covdn,docc1…doccn,covdocc1…covdoccn,sigma1,sigman,sigmacov1..sigmacovn]. To get a FIM from an output file and use that as a prior; use the *Update prior FIM from output* menu under the *Optimal Design* menu. Not that a prior FIM can not be entered from the Wizard, only from the main window under the prior FIM tab. Right click with the mouse on the prior FIM tab and to edit the prior FIM. Not that the *Use prior FIM* should be checked to enable a prior FIM.

Parallel Settings

The parallel settings are stored in the model xml file and can therefore be specific for each model/run. Some of the settings are more system specific than run/model specific and thus default settings for each PopED installation can be entered in the *config.xml* file (enter the settings manually in an xml editor). Furthermore a menu option (*Optimal Design/Insert default parallel settings…*) can be used to update the model xml file with the system specific parallel settings.

Two different parallel methods are available, the Matlab PCT run option and the MPI run option. If both are installed correctly any of the two methods can be used. If MPI is used, different *run options* exist. Running without any compilation (*No compilation (only run)*) is suitable if no parallel execution is used or if the calculation of FIM is already compiled with the specific model and settings). The full compilation option does both the mcc compilation and the MPI compilation, with or without running afterwards. For completion, option to only compile the C-shared library (mcc) or only the executable with mpi are available, again with or without running afterwards. *Num cores/processors* is the number of calculation units (n) that will be used; in PCT all the n units are used for FIM calculations (workers) while in MPI, one unit will be a job manager and n-1 units (workers) are used to calculate the FIM. However, for small n, the load for the job manager (JM) is likely to be small and therefore *num*

cores/processors can often be set to n+1, i.e. num physical cores / processors +1). *Parallel search methods* enable to use parallel execution (or not) for the specific search methods. If no methods are chosen the optimization will not be performed in parallel, however compilation can still be performed. Some MCC specific methods are available, the *function_input* name should be specified to use in the C-shared library. Finally, additional mcc compiler options can be set. E.g. the "–C" option can be useful on some system and mcc versions. For the MPI compilation and execution several settings can be added; *Design chunks* tells the JM how many designs that should be execute within each worker before it's available for more work. -2 is the default value which means that the JM will also work and that jobs are only distributed ones (divided equally if possible) to all the workers (including the JM). This might mean that the some workers are doing one more job than others if the number of jobs is not a common divider to the number of workers. Another value is -1 which means that the number of chunks to be sent is dynamically changed depending on how many designs that should be calculated, similar as option -2 except that JM is not doing any work. Another difference to -2 is that jobs might be distributed twice If there are uneven number of jobs compared to the number of workers. -1 is a good option if every node/processor is equally fast and message passing time is similar between different nodes or that the JM is heavily loaded. If the FIM calculation time is long (relative to the startup of the Matlab runtime environment (MCR) and the message passing) it can be wise to chose a small value (e.g. 1 or 2) to optimize the load balance performance. This is especially important if the core/processor speed or load is significantly different. *Additional run options* are added to the *mpirun* command line (see *mpirun* help for option descriptions). *Executable name* is the name of the executable produced by the C++ mpi compilation. *Mat file input prefix* is the prefix of the mat file used to communicate design between the executable and PopED. As default; the next number that results in a non existing file is added to the *Mat file input prefix*. Otherwise, a random number is added to the *Mat file input prefix* before each call to the mpirun. This can be changed in the *execute_parallel.m* file. *Mat file output prefix* is the prefix for the results file after parallel execution with the same number added as for the *Mat file input prefix*. As default, these files are deleted after each parallel execution but they can also be stored, change this in *execute_parallel.m*. *Output poll time* is how often to check (in seconds) for a result from the parallel execution within PopED.

Run file settings

The run file settings are stored in the model xml file and can therefore be used to pre/post process output or input to a PopED run. In principle the run file can e.g. enable adaptive optimal design and multiple calls to PopED and/or other software. An example of how a run file is used can be seen in *Example 2, User defined distributions on model parameters.* A run file can be created either directly in the GUI or a link to a preexisting run file can be used. For the created run file a tag representing the current xml-filename of the PopED xml model file, *<POPED_XML_FILE>* is used. This will automatically be changed when executing PopED to the current file name of the xml-file. A brief definition of how the PopED script functions handle the run file is described below.

When the PopED script version is started with a run file, e.g. poped('*my_xml.xml*') where my_*xml.xml* contains a run file definition, the run file will be automatically

created as a Matlab function file, e.g. a *run.m* file will be created (or the file name specified by *"Run file function name"* . After the creation of the run file, PopED will execute the run file inside PopED (almost as a recursive call). If the run file calls the PopED script version, the PopED script version will acknowledge that this call to PopED came from within the run file and hence the "normal" PopED functions (e.g. finding an optimal design) will be used. When using run files in combination with the GUI and Freemat, the output cannot be automatically parsed within the GUI, i.e. the output file need to be opened manually by the user.

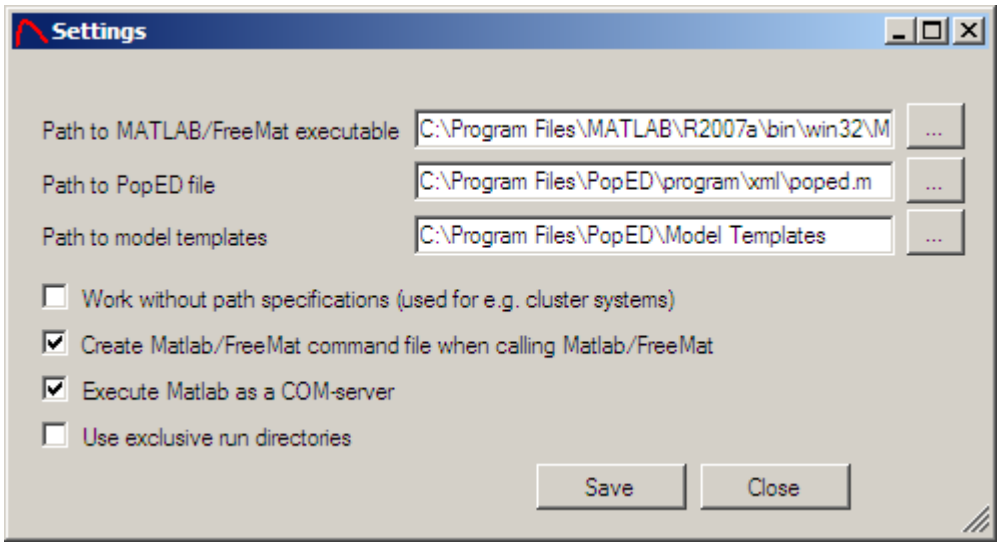
Save the PopED settings

The settings can be saved by using the save function (Ctrl+S) or Save as under the file menu or by walking thru the wizard. If a settings file is saved in the model template directory the model templates will be updated with the new model template. A warning will occur to prevent adding unnecessary files to the model templates directory. The 20 latest saved/open poped files are stored in the File menu / Recent Files.

Open a PopED settings file

A PopED Settings file (*.xml) can be open by using Open under the file menu or by the recent files menu under the file menu.

PopED GUI Directories Settings



A path to the Matlab executable, usually <Matlab dir>\bin\win32\matlab.exe must be entered to run PopED within the GUI, alternatively a path to the FreeMat executable. . A path to the PopED script must also be entered to allow for execution of PopED runs within the GUI. This should be <PopED dir>\programs\poped.m. A path can also be set to the model templates directory. If no path should be used in the xml files, the

checkbox *Work without path specifications* can be checked. This allows the user to specify all input and output files in a relative manner. This is particularly useful when running the GUI on a separate system than the Matlab installation, i.e. when the GUI is used to set up the design and then run on a separate system using e.g. poped('myfile.xml'). The *Create Matlab command file when calling Matlab* stores every call to Matlab in a Matlab function file, usually *poped_cmd.m* (possible to change in the *config.xml*, see below). The option *Execute Matlab as a COM-server* uses the COM server technology to communicate with Matlab, this option only work with Matlab, not FreeMat. If this option is not set, the program execute Matlab via command line options and the Matlab process is closed automatically or need to be closed to get the input back to the GUI. This option doesn't use any COM technique and is therefore suitable for running PopED GUI on e.g. Mono or other third party software that enables .NET implementation. When the COM Server option is not used, live log files are visible during the optimization run. All the settings will be stored in the *config.xml* located in the PopED installation directory when pressing save or a local copy of the *config.xml* if the GUI is started with an input argument specifying the *config.xml* file. The *use exclusive run directories* option Make a new directory for each evaluation/optimization of the design. The name of the directory is the same as the xml file name but with a increasing number added to the name, e.g. a *model.xml* will create a directory *model1* the first time it is run, *model2* the next etc. A run directory is only created when running optimizations or evaluations not for simulations or other diagnostics.

Diagnostics before the optimization

Some diagnostics can be used to see that the model works correctly prior to the optimization procedure. The model typical values can be plot and simulation can be performed to validate the model.

Convert PopED Settings to script version

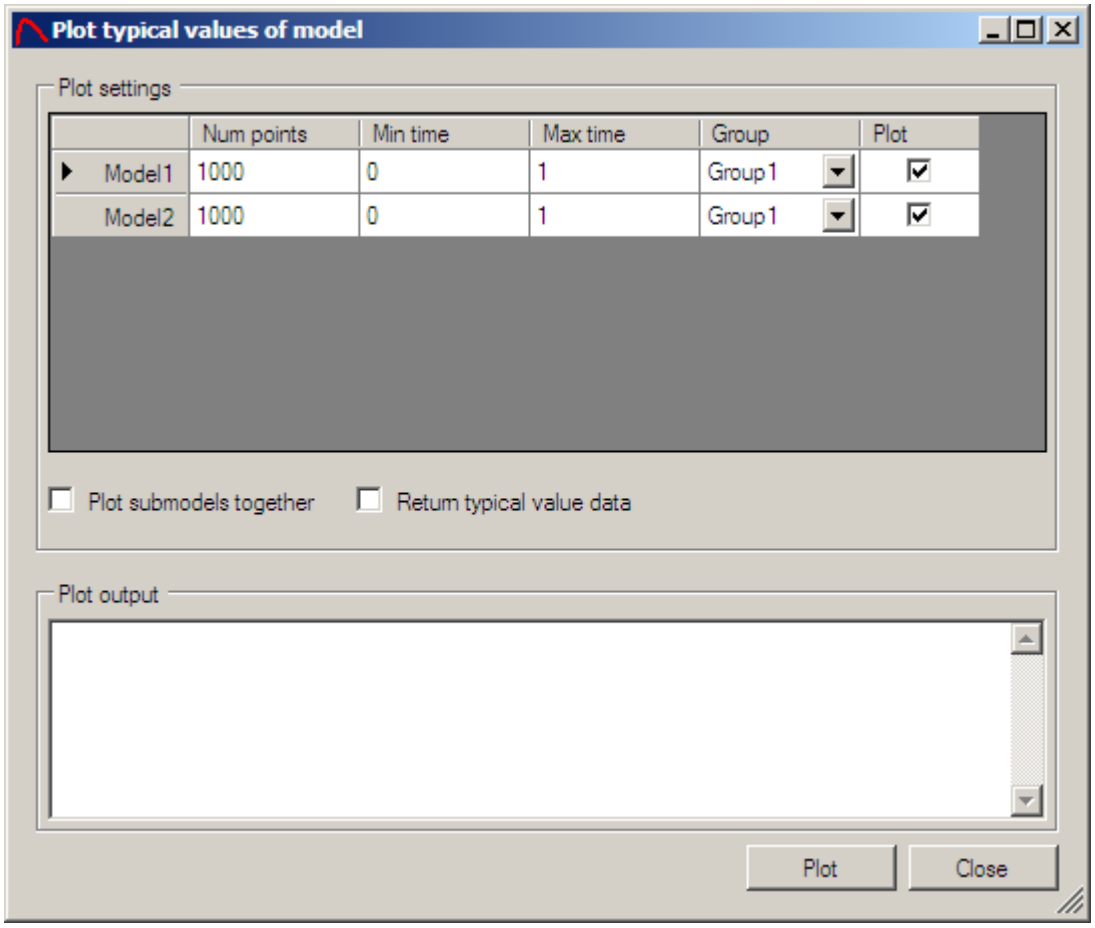
The *poped* function in Matlab can take the *.xml PopED GUI settings as an input or it can take a Matlab function representing the settings. A Matlab function containing the settings information is created when the function *poped* is called or can be created by a call to the function *convert_xml.m* that will take a *.xml filename as input. The Matlab function containing the settings is often one of the inputs to the diagnostic functions.

Run PopED script commands from the GUI

A number of script commands in PopED can be run from the GUI. The GUI will convert the current settings file to a Matlab function (see above) and call the script command via the Matlab COM Server. This is all done in the GUI code but the Matlab COM Server needs to work properly to run PopED script from the GUI. Otherwise the PopED script version must be used (i.e. run optimization and plots

from Matlab command line) or the setting *Use Matlab COM Server* must be unchecked.

Plot Typical Values



PopED GUI has a function that can plot the typical values for all sub-models. The function is available if a PopED setting is saved and open and it's accessible from the Optimal Design menu.

For each sub-model the number of points that will be used to calculate the dependent variable must be entered. For example if the number of time points are 1000 and the time interval are 0-1 the dependent variable (DV) will be calculate every 0.001 time unit. It is also possible to plot the typical value from a certain group (the covariate values can affect the plot, e.g. different doses in different groups). A checkbox indicates whether this sub-model should be plotted or not. If *Plot sub-models together* is checked all the sub-models will be plotted in the same window/figure. The plot output will be the DV for all the sub-models that are plotted and figure windows will show all the sub-models. In the figure window; the data, titles, axis settings etc. can be modified and saved. If the *Return typical value data* is checked the output from the plot (the typical values) will be presented in the Plot output.

The script version of plotting the typical values is:

```
model_values = plot_model(bPlotModel,model_num_points,model_minxt,model,
               maxxt,groups_to_plot,popedInput,bShowGraph,bPlotInSame)
```

bPlotModel is a vector with the length of number sub-models indicating if this sub-model should be plotted or not. 1 equals plot, 0 equals not plot.

model_num_points is the number of time points used to plot the sub-model. This is a vector of the length of the number of sub-models.

model_minxt is a vector with the minimal time point to plot for all sub-models. The length is equal to the number of sub-models.

model_maxxt is a vector with the maximal time point to plot for all sub-models. The length is equal to the number of sub-models.

groups_to_plot is a vector with the group number to plot the sub-model with.

popedInput is the PopED settings function created from the xml file (see above). The function is often accessible by typing *function_input()*.

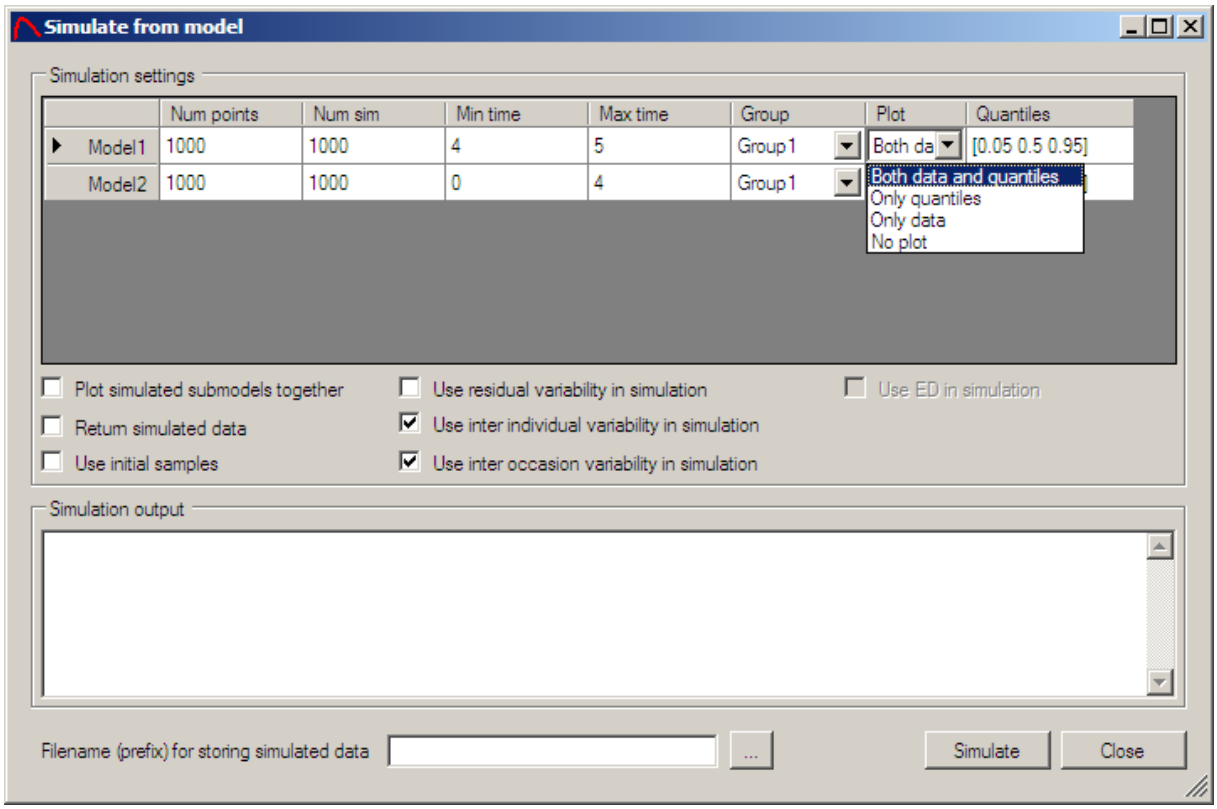
bShowGraph indicates whether a figure will be created or not. If it's set to false or 0 the model_values will be calculated and returned but no figure will be shown. This option can be useful to get the typical values within a script.

bPlotInSame is set to true or 1 if all sub-models should be plotted in the same figure. Otherwise it is set to false or 0.

Example: This will produce the same plot as above:

```
plot_model([1 1],[1000 1000],[0 0],[1 1],[1 1],function_input(),1,0)
```

Simulation of Model



PopED GUI has a function that can simulate data and plot the result for all sub-models. The function is available if a PopED setting is saved and open and it's accessible from the Optimal Design menu.

For each sub-model the number of points that will be used to calculate the dependent variable (DV) is specified, furthermore the num of simulations (number of individual curves to simulate) are specified. All simulations are simulated within a minimum and a maximum time and the group used to simulate can also be specified (the covariates in a group can affect the simulation, e.g. the dose). There is also a plot option that allows for different plots to be created. For each sub-model the plot options are:

- Both data and quantiles – Plot the data from all simulated individual curves and the quantiles specified in the quantiles column.
- Only quantiles – Plot only the quantiles specified in the quantiles column.
- Only data – Plot only the simulated individual curves.
- No plot – Don't plot this sub-model.

In the quantiles column all quantiles that should be plotted from this sub-model can be specified. The quantiles should be specified within brackets ([]) and it should be a value between 0-1. E.g. [0.01 0.5 0.99] will plot the 1 and 99 percentile and the median value of the model given the simulated data. The figure windows produced by the plot can be used to change the plot properties and axis-settings etc. The output returned by the plot is the simulated data but only for the last sub-model that don't have the Plot option – No plot. The data is presented only when the *Return simulated data* is checked. All the sub-models can be plotted in the same window by checking the *Plot simulated submodels together.* The simulation can be done with or without the residual variability by clicking *Use Residual variability in simulation*. If *Use initial samples* is checked the function will only simulate from the current design (the initial samples) and the *Number points* column is ignored. The simulation can also be done with or without inter individual variability by checking the *Use inter individual variability in simulation and* with/without inter occasion variability by checking/un-checking the corresponding check box. It is also possible to simulate with or without ED-variability by checking the *Use ED in simulation* if the design is defined as an ED design (uncertainty around the parameter values). The sample method used in the simulation can be either Latin Hypercube Sampling or normal sampling, see advanced settings. A filename prefix for storing the simulated data can be defined, one file for each sub-model is created with the prefix name followed by the sub-model number, e.g. my_data_1.csv, my_data_2.csv etc. If no name is specified no data file will be created. The data file is stored as a comma separated file (csv) and the first column represent the individual number, the second column the time and the third column the dependent variable, e.g. the concentration or the response.

The script version of the simulate function is:

```
sim_dat =  plot_simulation(bUseResiduals,bUseIIV,bUseED,
                      model_num_simulations,model_num_points,
                      model_minxt,model_maxxt,
                      groups_to_plot,popedInput,sim_quantiles,bShowVector,
                      bPlotInSame,bUseInitialDesign,strFileName,bUseIOV)
```

bUseResiduals is set to true (1) or false (0) if the residuals should be used in the simulation.

bUseIIV is set to true (1) or false (0) if the inter individual variability should be used in the simulation.

bUseED is set to true (1) or false (0) if the ED uncertainty should be used in the simulation.

model_num_simulations is vector with the length of number of sub-models indicating the number of individuals that should be simulated for this sub-model.

model_num_points is the number of time points used to plot the sub-model. This is a vector of the length of the number of sub-models.

model_minxt is a vector with the minimal time point to plot for all sub-models. The length is equal to the number of sub-models.

model_maxxt is a vector with the maximal time point to plot for all sub-models. The length is equal to the number of sub-models.

groups_to_plot is a vector with the group number to plot the sub-model with.

popedInput is the PopED settings function created from the xml file (see above). The function is often accessible by typing *function_input()*.

sim_quantiles is a Matlab cell structure with the length of number of sub-models. The cell structure contains a vector for all sub-models with the quantiles that should be plotted for a sub-model.

bShowVector is vector with the length of the number of sub-models. The show vector will contain information how each sub-model should be plotted and if it should be plotted. The option for the show vector is 0 – Plot data and quantiles, 1 – Plot only quantiles, 2 – Plot only data, 3 – Don't plot this sub-model.

bPlotInSame is set to true or 1 if all sub-models should be plotted in the same figure. Otherwise it is set to false or 0.

bUseInitialDesign is set to true or 1 if the initial design should be used in the simulation. Otherwise it is set to false or 0.

strFileName is set to the path and filename prefix of the csv file where the simulated data should be stored. To skip generate data files, set the value to the empty string ''.

bUseIOV is set to true (1) or false (0) if the inter occasion variability should be used in the simulation.

Example: This will produce the same simulation plot as above:

```
plot_simulation(0,1,0,[1000 1000],[1000 1000],[4 0],[5 1],[1 1],
          function_input(),{[0.05 0.5 0.95] [0.05 0.5 0.95]},[0
0],0,0,'',1)
```

Optimize with PopED

Optimization from the GUI can be performed under the Optimal Design menu, Run. To run the optimization from the script version type poped('file_to_optimize.xml') or call the function with a transformed input function (see Convert PopED settings to script version).

Example:

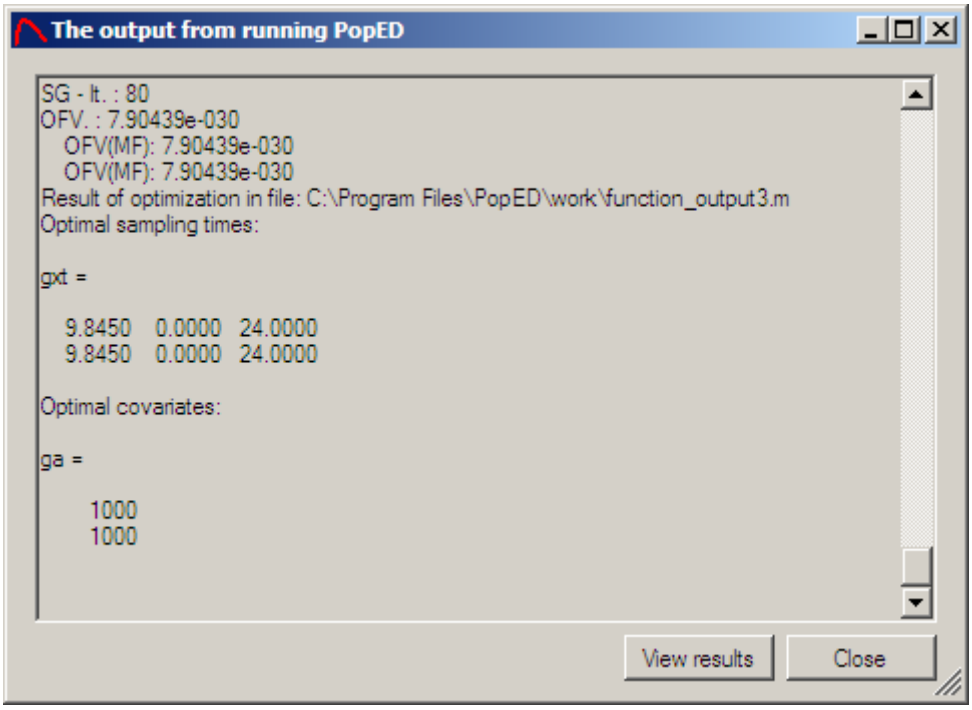
```
[popedOutput,globalStructure,strRunDirectoryName]=poped('MyFile.xml');

[popedOutput,globalStructure,strRunDirectoryName]=poped(function_input());

poped('MyFile.xml','<default>'); %Creates a run directory MyFile
```

```
poped(function_input(),strDirName); %Creates a run directory strDirName
```

During the optimization, Matlab figure windows will show the optimization current status, i.e. the OFV and the continuous variable values in the optimization. If optimizing over discrete variables no figure windows will be shown in this version. Furthermore no figure windows will be shown if the *Use graphical output during optimization* option is unset (see Optimization and calculation settings).

During the optimization, output to the Matlab command window is outputted in the script version of PopED and the output from the different search algorithms are stored in the output file specified in the PopED settings file (*.xml or the input function *function_input.m* (see Output and input directories)). Stochastic Gradient and Random Search will store their result in the output file with an extension _RS_SG_i.txt, where i is the search iteration number. Line Search will have an extension _LS_i.txt where i is the search iteration number.

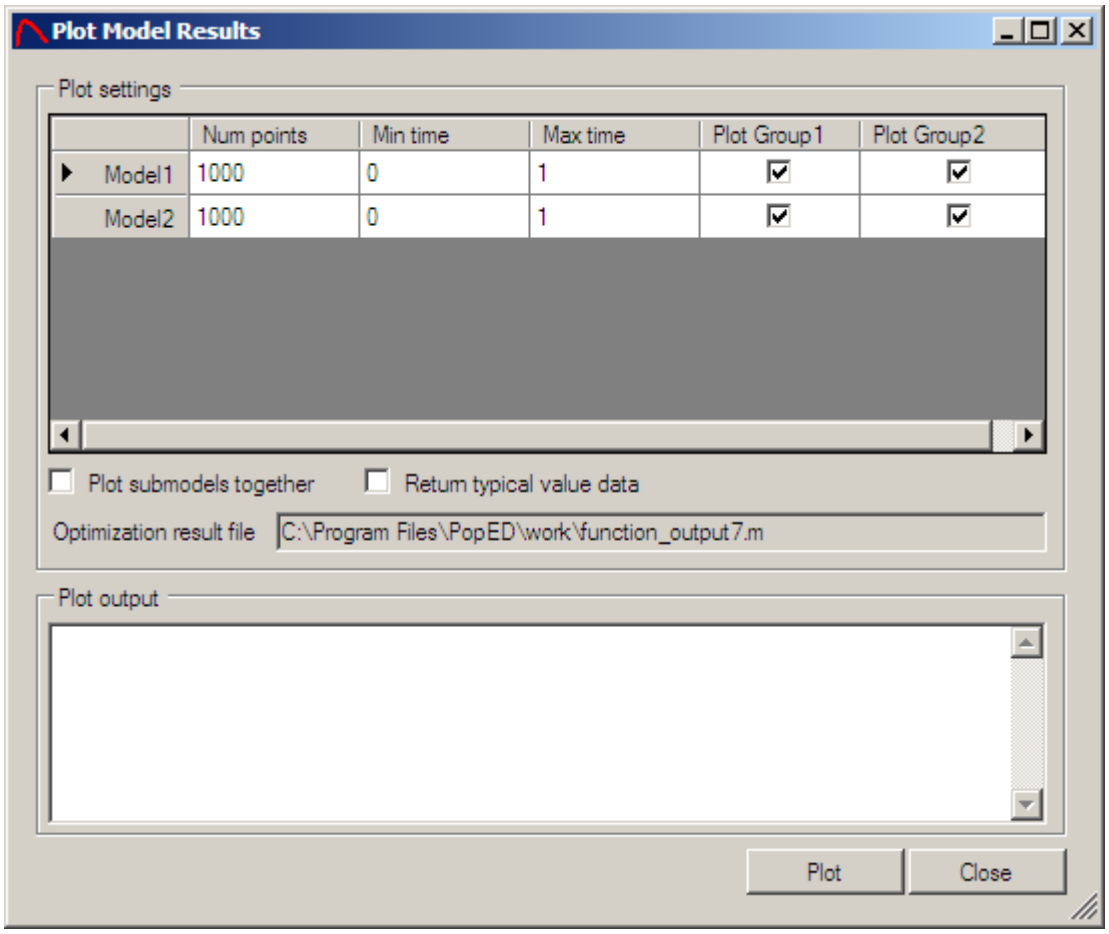
When the optimization are done or if an error occurs, output from the Matlab command window will be shown in an optimization output window for the GUI version and in the Matlab command window in the script version. A Matlab function file with the results from the optimization is also written, the output function is called: *function_outputi.m* (see Output and input directories) where i is the lowest number to create a non-existent filename in the current directory. The same function is also returned from the poped function as a structure with the same form as the function_output.m. Furthermore the globalStructure used in PopED is returned and finally the name of the run directory is returned. If no run directory is created this will be an empty string.



The continuous optimized variables are outputted as text and the result function filename is also visible. If the optimization didn't terminate due to an error the View results button will be visible.

Plot model results

The optimized result for continuous optimization can be viewed with the plot model result function available in the GUI from the Optimal Design menu. To be able to run the function from the GUI an output function corresponding to the optimization of the open PopED GUI Settings file (*.xml) must be specified.



The function works similar to the plot model function and plots the typical values for the different sub-models. The main difference is that the optimal sample times are marked in the plot with red rings. Like in the model plot function the number of time points to calculate the model with is specified in the num point's column. The time interval to plot over is also specified besides which group in each sub-model to plot. If the *Plot submodel together* checkbox are checked the entire plot will appear in the same figure window. If the *Return typical value data* is checked the typical value are presented in the Plot output.

The script version of plot model result function is:

```
[model_values opt_value opt_time opt_group opt_model] =
plot_model_results(bPlotModel,model_num_points,model_minxt,model_maxxt,pope
dInput,bShowGraph,bPlotInSame,bPlotGroups,popedOutput,clustValue)
```

bPlotModel is a vector with the length of number sub-models indicating if this sub-model should be plotted or not. 1 equals plot, 0 equals not plot.

model_num_points is the number of time points used to plot the sub-model. This is a vector of the length of the number of sub-models.

model_minxt is a vector with the minimal time point to plot for all sub-models. The length is equal to the number of sub-models.

model_maxxt is a vector with the maximal time point to plot for all sub-models. The length is equal to the number of sub-models.

popedInput is the PopED settings function created from the xml file (see above). The function is often accessible by typing *function_input()*.

bShowGraph indicates whether a figure will be created or not. If set to false or 0 the model_values will be calculated and returned but no figure will be shown. This option can be useful to get the typical values within a script.

bPlotInSame is set to true or 1 if all sub-models should be plotted in the same figure. Otherwise it is set to false or 0.

bPlotGroups is matrix with the length of number of groups multiplied with the number of sub-models. E.g. the matrix contains a 1 in the first row and 2nd column if the 2 group should be plotted in the first sub-model.

popedOutput should be a PopED output function e.g. function_output1.

clustValue is a value to determine if a point will be considered clustered or not. If the distance between two optimal samples is within the clustValue, a value with the number of points that are clustered will be written in the plot, next to the clustered points. If no clustering should be checked the value should be set to 0.

model_values is a output matrix containing the data from all plotted sub-models. The model_values does not necessarily contain the optimal sampling points. See below.

opt_value, *opt_time*, *opt_group* and *opt_model* are output vectors. *opt_value* contains the dependent variable value, opt_time the time point that the *opt_value* is calculated at, *opt_group* in which group the DV are calculated and *opt_model* has information about which sub-model the sample belong to.

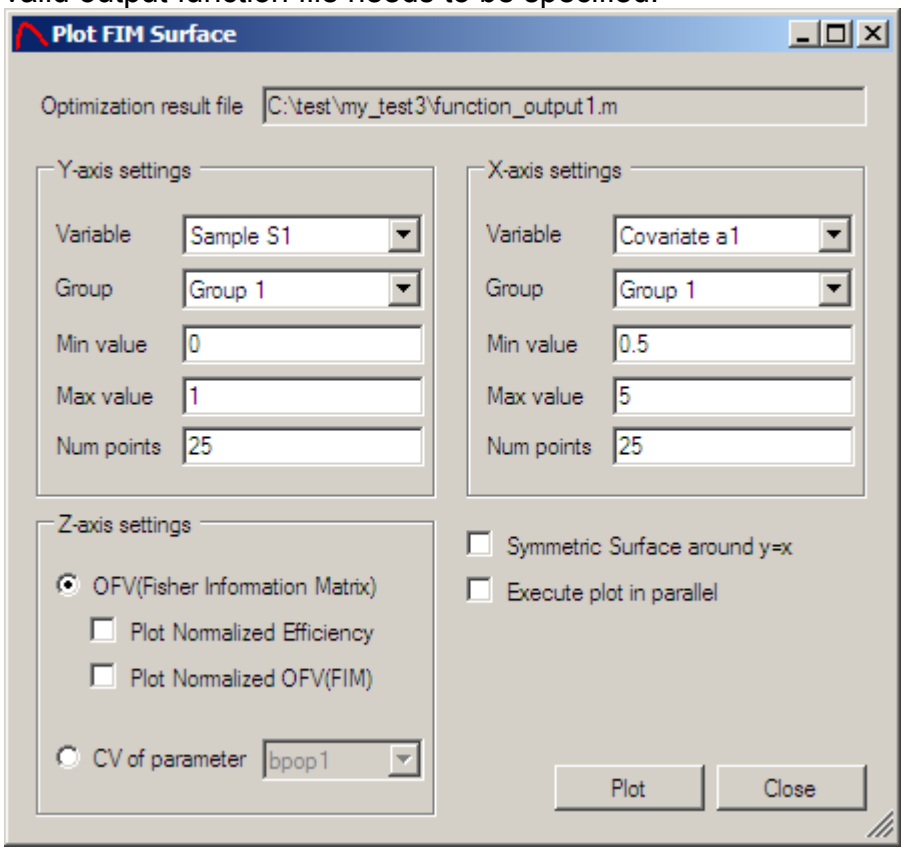
Example:

```
plot_model_results([ 1 1],[ 1000 1000],[ 0 0],
[1 1],function_input(),1,0,[1 1 1; 1 1 1],function_output1(),0)
```

Plot Fisher Information Matrix Surface

The surface of the OFV can be plotted against two of the design variables. This plot

is available in the GUI from the Optimal Design menu. To get access to this plot a valid output function file needs to be specified.



The x and y axis settings are similar and all the variables in the current design are available in the Variable combo box. Choose a group for that design variable and a min/max value of the axis. Num of points to split the axis in and to calculate the OFV should also be specified. The surface will calculate the OFV number of points in the x-axis multiplied by the number of points in the y-axis, i.e. 25x25 = 625 in the example above. The surfaces that can be plotted are the OFV or the normalized OFV, i.e. scaled to the power of one divided by the number of unfixed parameters in the model (FIM^(1/p)). The third alternative is to plot the normalized efficiency, i.e. the value of OFV divided by the optimal OFV scaled with the number of parameters in the FIM. One other option is to plot the SE (%) of an unfixed parameter in the model. The CV will be presented as a percentage value in the plot. If the surface is known to be symmetric, e.g. plotting sample1 and sample2 within the same group the *Symmetric Surface around y=x* can be checked, this will half the run-time. This option can only be used when the num points are the same for the two dimensions. If discrete variables are plotted the 3d plot will disregard the num of points and instead use the number of discrete values of the discrete variable that are within the min and max value. The grouping of the sample times, the covariates and the discrete variables will be considered in the plot. *Execute plot in parallel* can be checked to calculate the surface on several cores/processors to speed up the plot. To enable this, a compiled FIM calculation must be available (for MPI) or Matlab Parallel Computing Toolbox must be installed (see Parallel settings for more information).

This diagnostic must be considered very carefully even though it can be a powerful tool to visualize the surface of the FIM. In most cases the dimensionality of the FIM is

greater than 3 and to look only of 2 of these dimensions and draw conclusions can be misleading. There is also possible to change the min and max values to higher or lower values than the bounds used in the optimization. In this case e.g. the efficiency calculations might be false because the optimal OFV might be in the wider region. Also be very careful to use variables that are not optimized for in the surface, e.g. only optimization on sampling schedule but plotting a sample versus the dose. Again the optimal OFV might be wrong. This option is still available because sequential optimization might have been performed before the last optimization.

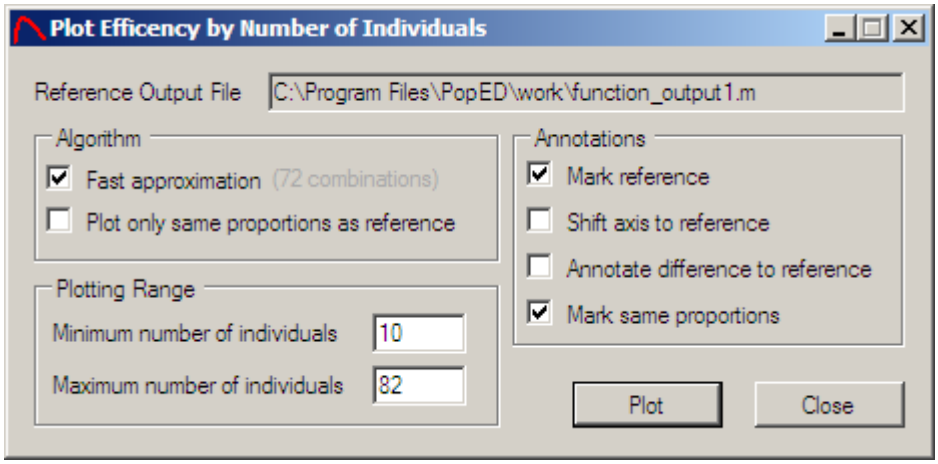
If the plot is not visible the render method might be wrong, this can be the case when the OFV is very large. Change the render from OpenGL to z-buffer in the Matlab figure properties. Another possibility is that the FIM calculation is nearly singular or singular. Try to change the max and min value to be close to the optimal design or use the script version of the surface plot to catch warning messages.

The script version of the FIM surface plot is available from the Matlab command window and the function is named:

```
plot3d_fim
```

The script function has a lot of options and will not be described here but there is a describing help text in the function. Type *plot3d_fim* in the Matlab command window and help text will be printed. For example there is an option to save the plot to a figure file without viewing it that can be useful for cluster/script users.

Relate efficiency to individuals

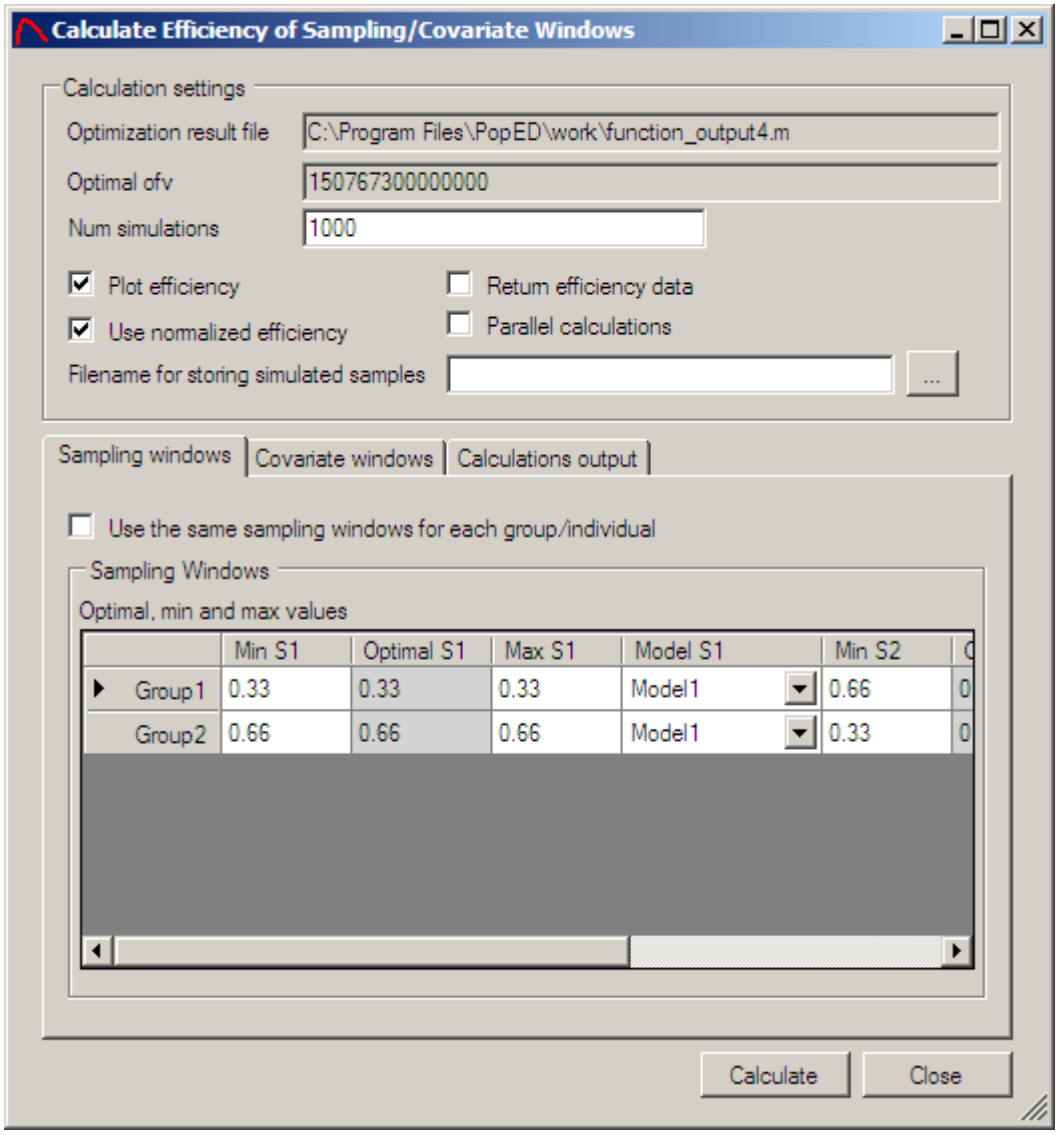


Reporting designs or the relationship between e.g. two designs is often done by reporting efficiency. In general, the efficiency between two designs might not be intuitive to interpret and furthermore the efficiency is related to the size of the model and also calculated differently for each criterion, e.g. D-optimal, A-optimal etc. Instead; a more interesting and intuitive way of comparing designs might be to relate the efficiency to the number of more/less individuals needed to get the same information as the reference design. This is in fact a true optimization problem

because the information one gain when adding an extra individual depends on in which design group the individual will be added. This plot enables the user to see how adding or removing individuals from the study will relate to certain efficiencies. The best efficiency is shown by adding the individuals in the optimal groups and the worst efficiency is shown by placing the individuals in the group with least information.

Fast approximation is an option that allows the calculation of the optimal number of individuals be calculated faster. This option is likely to be exact in most cases but no mathematical proof of that has been provided. A good practice is to always use the *fast approximation* while exploring designs. The range of the number of individuals to show in the plot can be changed and the “maximum” number of individuals in the plot is the *total max groupsize* and the minimum number of individuals is the *total min groupsize*. If the *mark reference* option is set the reference design in the efficiency calculation is indicated by a dashed line. The *shift axis to reference* option allows the user to make the number of individuals axis related to the reference design in the efficiency calculation. Finally; the *annotate difference to reference* option mark the efficiency difference and number of individual difference to the reference design. This mark is only meaningful if the reference design has not been optimized for number of individuals but otherwise the option will show the maximum gain in information by optimizing on group assignment. *Plot only same proportion as reference* will use the same group size proportions as the reference design and hence will not need to optimize to calculate the efficiency. The *mark same proportions* option is used to plot the same proportions curve within the optimized lower and upper bounds efficiencies.

Calculate Design Windows



Often the optimal sampling times, optimal covariates values or optimal discrete variables values are not feasible in practice. Therefore borders around each optimal value can be calculated that will produce a feasible design instead of an optimal design, this is called sampling windows or covariate windows. This option is available in the GUI from the Optimal Design menu and it's also available as a script function.

Depending if covariate and/or discrete variables are in the design, tab pages for covariates windows and discrete variable windows will be enabled. Sampling windows is always available because samples need to be taken in an informative design. In the sampling windows the optimal sampling time is written and the window is defined by changing the min and max values. The optimal value and the model can not be changed. The sampling window cann't be outside of the borders defined before the optimization. If discrete variables is used the value for each discrete variable can be changed from the optimal value to another available discrete value for that particular discrete variable. Grouping will be considered for the samples. The *Filename for storing simulated samples* enabled the user to store the samples for the

sampling/covariate windows as well as the expected CVs and the efficiency for each simulation. The results will be stored in a *.csv file with the number of rows equal to the number of simulations. The columns of the file will start with the sample times for group 1 then the sample times for group2 etc. After the sample times the covariates for group1, group2 etc will be entered followed by the discrete variables for group1, group2 etc. The last columns will contain the CVs (in fractions) for each parameter that is not fixed (in the same order as the output file) and lastly the efficiency (in fractions) is outputted in the last column.

The tool will calculate the efficiency of a sample within a window by randomly (uniformly) take sample times from the window and calculate the OFV and compare the value to the optimal OFV. This can be done by using the normalized OFV (scaled to the number of parameters) or the un-scaled efficiency. The number of simulations indicates how many samples that will be drawn from the windows. If grouping is used for the sampling times, covariates or the discrete variables, the samples will be grouped when calculating the sampling windows. The first sample in the grouping will define the borders for the sampling window, e.g. if sample 1 and sample 3 is grouped, the sampling window for sample1 will be the window for sample 3 regardless of the values for sample 3 and they will of course have the same sampling time sampled from the sampling window. There is also a possibility to plot the efficiency in a Matlab figure window by checking the *Plot efficiency* check box. To calculate the sampling windows in parallel, check the *Parallel calculations* check box. This option needs a compiled version of the FIM calculations with the MPI method or Matlab Parallel Computing Toolbox to be installed, see Parallel Settings for more information.

The windows function will return the efficiency in a vector and it's visible in the calculation output tab after the calculations if the *Return efficiency data* is checked.

The script version of the sampling windows calculation function is:

```
[eff min_eff mean_eff max_eff] =
efficiency_in_windows(iNumSimulations,optdetfim,xt_windows,a_windows,x_wind
ows,bNormalizedEfficiency,bPlot,popedInput,strFileName,bParallel)
```

iNumSimulations is a scalar that specifies how many samples that will be picked from the sampling windows.

optdetfim is a scalar with the optimal OFV from the optimization.

xt_windows is a matrix with the sampling windows for all groups. For each row in the matrix the min and max value of the window for each sample should be entered.

a_windows is a matrix with the covariate windows for all groups. For each row in the matrix the min and max value of the window for each covariate should be entered.

x_windows is a matrix with the discrete variable windows for all groups. For each row in the matrix the min and max value of the window for each discrete variable should be entered. From the GUI, the min and max value for each discrete variable will be the same.

bNormalizedEfficiency if the efficiency should be scaled to the number of unfixed parameters in the model or not. 1 if normalized, 0 otherwise.

bPlot if the efficiency should be plotted, true or false.

popedInput is the PopED settings function created from the xml file (see above). The function is often accessible by typing *function_input()*.

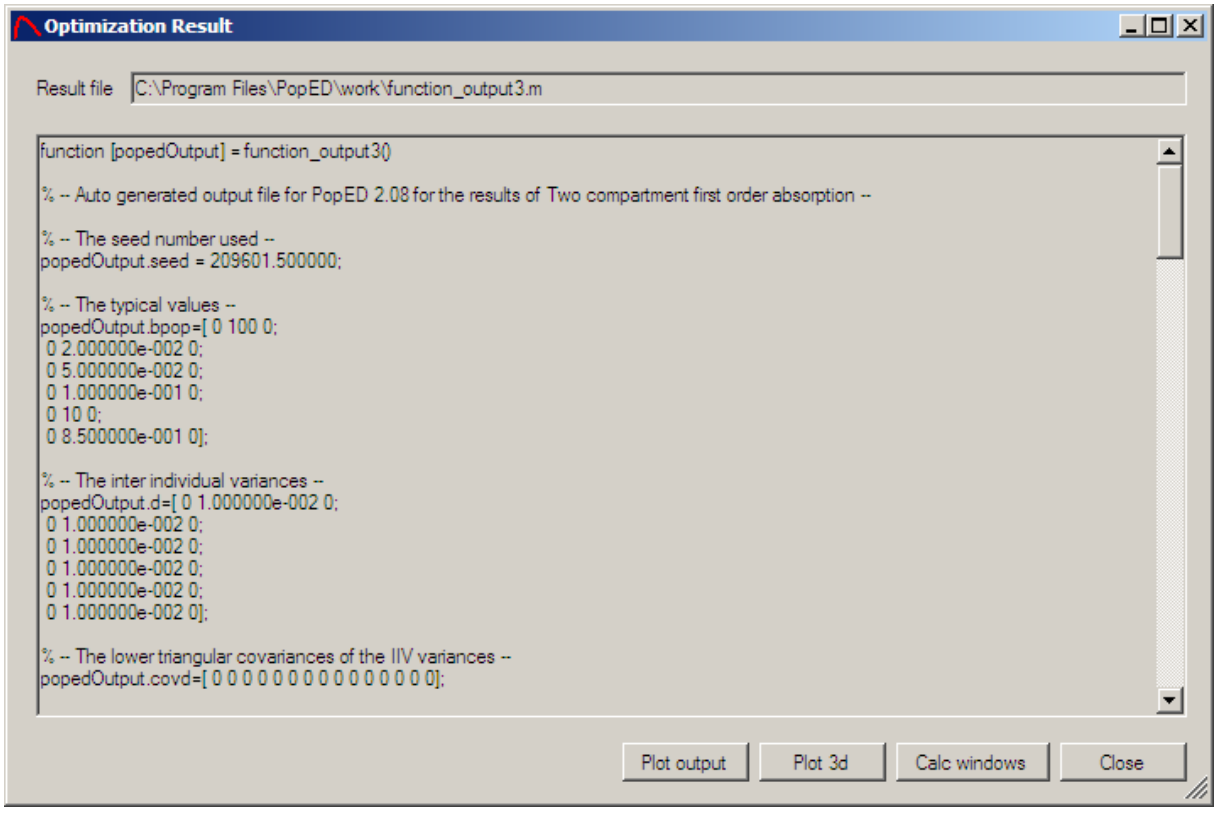
strFileName is the filename and path to the *.csv file where the results should be stored. If *strFileName* is an empty string ‘’, no output file will be created.

bParallel is used to calculate the sampling windows in parallel. Can be set to 1 if calculations in parallel and 0 if regular (serial) calculations.

Example:

```
efficiency_in_windows(1000,1.154813e020,[0 1.2 0 0.5 1 2],[50
150],[],0,1,function_input(),’’,0)
```

View Optimization Output



The optimization result is a Matlab function file that contains a lot of information about the optimization result. The variables in the output function file are.

- seed – The seed number used in the optimization.
- bpop – The typical values, the distributions and variability.

- d – The IIV distribution, the distributions and variability.
- covd – The lower triangular (matrix) vector of covariance for the IIV variances.
- docc – The inter occasion variances, the distributions and variability.
- sigma – The Residual variability.
- NumOcc – The number of occasions.
- xt – The final sampling times.
- x – The final discrete values of the discrete variables.
- a – The final covariates.
- gni – Number of samples per subject.
- groupsize – Group size for each group.
- modelswitch – The vector defining which samples that belong to which sub-model.
- iFIMCalculationType – The full or reduced FIM.
- ApproximationMethod – FO, FOCE or FOCEI.
- FOCENumInd – The number of individual samples if a conditional approximation method was used.
- fmf – The Fisher Information Matrix.
- imf – The inverse of Fisher Information Matrix.
- ofvmf – The objective function value.
- d_switch – If optimized using D or ED-optimal design.
- ofv_calc_type – The criterion used to calculate the OFV. E.g. det(FIM) = 1 and ln(det(FIM)) = 4.
- param_var – A vector with the unfixed parameter variances, the vector start with the unfixed bpop, then the unfixed d, the unfixed docc and last is the unfixed sigma.
- param_cv – A vector with the unfixed parameter CV in fractions, the vector start with the unfixed bpop, then the unfixed d, the unfixed docc and last is the unfixed sigma.

- notfixed_bpop – A vector with a 1 if a bpop is not fixed and a 0 if the bpop is fixed.
- notfixed_d – A vector with a 1 if a d is not fixed and a 0 if the d is fixed.
- notfixed_sigma– A vector with a 1 if a sigma is not fixed and a 0 if the sigma is fixed.
- notfixed_docc – A vector with a 1 if a docc is not fixed and a 0 if the docc is fixed.

- filename – A string containing the filename and path of the output file, used by the GUI if the COM server is not used.
- strengine – A string with the calculation engine used to produce the output function. I.e. a FreeMat version or a Matlab version. The string can e.g. be: “FreeMat v4.0” or “Matlab 7.4.0.287 (R2007a)” or similar.

The function output can also be called within a script by typing function_outputi() where i is the function output number.

Update initial design from the function output

The initial design (sample times, covariates, discrete variables, number of samples per group and the number of individuals per group) can be updated after an optimization. This is done by clicking *Update initial design from output* under the Optimal Design menu. The output file must be consistent with the xml file (i.e. similar number of groups, sample points etc.)

Convert output structure to xml

The output structure, e.g. *function_output.m,* can be converted to an xml file. This feature is used by the GUI but might also be used by third party software that wants to communicate with PopED. To convert an output file; use the following syntax:

```
convert_output_to_xml(function_output10(),strXmlFileName)
```

where *strXMLFileName* is set to the name that the xml file will have. This can also be an empty string and then the file name of the xml file will automatically be the filename and path of the function output but with the extension .xml instead of .m.

Update input structure from search iteration structure

The input structure, e.g. *function_input.m,* can be update by the best optimal design so far from a search iteration file. This file is available if the strIterationFileName in the poped input structure is set and a optimization is performed. If the optimization is terminated (for some reason) the latest search iteration is stored in the strIterationFileName and this function can be used to update the function input structure with this information. To do this; use the following syntax:

```
          popedInput =
update_popedInput_with_iteration(function_input(),iteration_file())
```

the function return a poped input structure that can be used to start poped from the last iteration, e.g. poped(popedInput).

Produce a run summary

A run summary can be produce from the function output structure. The run summary will be a Matlab function file that contains information about the optimal design and the expected precision that is easier to read. The function will also contain symmetric confidence intervals for each parameter. To produce a run summary; use the following syntax:

```
          runSummary=run_summary(function_output1(),strFileName)
```

The strFileName can be omitted and then (in this example above) a file name 'run_summary_function_output1.m' will be created instead. The run summary returns a runSummary structure containing the most important information. If strFileName is not omitted a runSummary structure will be created in the Matlab function file strFileName.

Model Templates

A number of model templates are distributed with the PopED program to illustrate how different models can be implemented. The model templates can be accessed by the Wizard; new option, under the File menu in the GUI. The model templates directory can be extended by own user defined templates by saving a PopED GUI settings file (*.xml) in the template directory. A recommendation is to write user defined templates as general as possible and not to add specific design of models that are already in the template directory. The templates available in the PopED version 2.12 are:

- One compartment IV Bolus dose. Parameterized by CL and V.
- One compartment zero order infusion. Parameterized by CL and V.
- One compartment 1st order absorption. Parameterized by ka, V, F and CL.

- One compartment IV Bolus dose with Michaelis Menten elimination. Parameterized by Vmax, Km and V. Explicit solution.

- One compartment IV Bolus dose with Michaelis Menten elimination. Parameterized by Vmax, Km and V. Differential equation.

- One compartment with transit compartment absorption. Parameterized by ka, CL, V, MTT and n.

- One compartment IV bolus with a direct Emax effect. Parameterized by CL, V, E0, Emax and EC50.

- One compartment IV bolus with a direct Emax effect. Parameterized by CL, V, E0, Emax and EC50, vectorized version.

- One compartment 1st order absorption with repeated dosing. Parameterized by ka, CL, V and tau.

- One compartment 1st order absorption with repeated dosing implemented as differential equation. Parameterized by ka, ke, V and tau.

- Two compartment IV Bolus dose. Parameterized by V, k12, k21 and CL.

- Two compartment zero order infusion. Parameterized by V, k12, k21 and CL.

- Two compartment 1st order absorption. Parameterized by ka, k12, k21, V, F and CL.

- Two compartment 1st order absorption with a lag time. Parameterized by ka, CL, Vc, Vp, Q, F and tlag.

- Two compartment 1st order absorption, multiple dosing. Parameterized by ka, k12, k21, V, CL and F.

- Three compartment IV Bolus Dose, Parameterized by V, k12, k21, k13, k31 and CL.

- Three compartment zero order infusion. Parameterized by V, k12, k21, k13, k31 and CL.

- Three compartment 1st order absorption. Parameterized by ka, k12, k21, k13, k31. V, F and CL.

- Biological rhythm (3 cos rhythms). Each rhythm parameterized by Peak time, Amplitude and the period.

- Linear disease progression model with symptomatic effect. Possibility to optimize on start and stop time of study.

- Linear disease progression model with protective effect. Possibility to optimize on start and stop time of study.

- Linear disease progression model with symptomatic and protective effect. Possibility to optimize on start and stop time of study.

Most of the model templates are analytical solutions of the different compartment models; therefore it can be good to use the templates because the analytic solutions are much faster than the differential equations. Fill free to distribute and share user written templates with other users of PopED.

Examples

The examples are available in the <*PopED Install dir*>*PopED Examples* and will cover more of the script specific options of the PopED program. All examples can be run from the and most of the design options can be changed and viewed by the GUI version of PopED. The examples are not always optimized for speed, instead they are written to be easy to understand and to follow.

To run an example, open the PopED GUI settings file (*.xml) for the example and change the path to the model file. The model file for each example will be in the same directory as the PopED GUI settings file.

Example 1, Multiple Drug optimization and grouping of sample times

In this example optimization will be done in three drugs concurrently. PopED will find an optimal design for parameter estimation of the three drugs at the same time. The drug models are written with analytic solutions but an identical differential equation example is also available by setting the sixth covariate to a value (0=ode, 1=analytic, 2=linear ode solver). The linear solver uses both the inhomogeneous solver as well as the homogeneous solver. Look in the model file for more information. All drugs have different profiles; one is a one-compartment drug with a zero order infusion, the second drug is a one-compartment drug with an IV bolus dose and finally the third drug is a one-compartment drug with 1^{st} order absorption. The design contains four different groups with 10 individuals in each group, all with four sample times. The sampling schedule is coded as 12 different sample times per group but the sample times are grouped in 3, i.e. to cover the measurements of the three drugs at the same time. There is also grouping of the sample times between groups, i.e. the first sample of the first and third group are grouped together meaning that they will be taken at the same time. There is also a grouping of the last sample time between the second and the 4^{th} group. Use the model validation tools from the PopED GUI (plot model and simulate) to look at the different drug profiles.

The grouping forces some of the sample times to be the same within and between groups. The grouping assumes that the residual variability of simultaneous samples of the three drugs is independent. This is probably not a correct assumption but the residual variability correlation in the samples only depends on the difference in the analysis method of the samples.

This example also shows how to implement a multiple response model within the GUI using the *User defined model definition* functionality.

Example 2, User defined distributions on model parameters

In this example a user defined distribution (UDD) is assumed on the typical value parameter for clearance. For a parameter like CL a normal distribution for the typical value or a normal/transformed IIV distribution probably would be better but this example is only an illustration how a user defined distribution can be implemented. The user defined distribution can be discrete or stochastic and in this example a discrete distribution of 25 values for CL is used. In ED-optimal design the expectation value can be approximated by a Laplace integral approximation but this is not used for the user defined distributions. For a user defined distribution that is not stochastic, the ED-sample size variable should be set to the length of the distribution and the stop criteria for ED-optimal design should be small (see ED-optimal design settings). A user defined distribution function should have this syntax:

```
function ret = user_distribution(ret,t,sample_number,globalStructure)
%A user defined distribution function.
%The value from the user defined distribution (UDD) should be returned in
%ret.
%Input:
```

```
%t is a vector that contains all the distributions of the
%parameters in the model. t(i) == 3 is a user defined distribution,
% 0 is point distribution, 1 a normal and 2 a uniform distribution. Ret is
%also an input variable with the values for the other distributions, if
%t(i)~=3 the value ret(i) should not be change, but the user distribution
%value can depend on the other parameters current values. The sample number
%is the ED-sample number; this shouldn't be larger than the distribution
length for a discrete distribution.
typCL = [1 2 4 3 3.2];
IIVCL = [0.01 0.02 0.012 0.011 0.022];
j=1;
for i=1:length(t)
    if (t(i)==3) %This is an user defined distribution parameter
        if (j==1) %If it's the first parameter with a UDD
            ret(i) = typCL(mod(sample_number-1,length(typCL))+1);
        else
            ret(i) = IIVCL(mod(sample_number-1,length(IIVCL))+1);
        end
        j=j+1;
    end
end

end
```

In the example the distribution is defined by a definition outside the distribution function, e.g. in the run.m file defined in the GUI, and then passed to the distribution function via the *globalStructure* (*user_data*). This is to illustrate that if a large discrete distribution is used it might be computationally run-time efficient to declare the distribution outside the distribution function. Look in the run file (defined in the GUI) and the user_distribution.m file to see how the UDD is defined. The model is a one-compartment model with an IV bolus dose. The path to the UDD might need to be changed in the xml file.

This example also illustrates how run files can be used within the GUI.

Example 3, Analytic Fisher Information Matrix of PK-PD model

In this example, analytic derivatives are used/calculated instead of numerical approximations of the FIM. There are two sub-models; a one-compartment IV bolus dose and a direct e-max effect model. Run the example from the GUI. If the run produced an error, it might be that the analytical derivative file has not been flushed to the operating file system before the execution of the FIM. Retry the run or run it from the Matlab command line in the example directory by typing:

```
poped('analytic_derivative.xml');
```

Example 4, Discrete list of sample times

In this example, there is a possibility to optimize the best set of a finite discrete number of samples. The example can be run within the GUI and uses the discrete variables (*x*) as the sample times. This is assigned in the model, settings the sampling time vector (*xt*) to the discrete variables. The number of samples must still be set and entered in order to produce the right dimensions in the Fisher Information Matrix. But the actual sample time (border, initial) doesn't matter; it is the discrete variable border/initial that is used. It is also possible to use the model in simulations and typical value plot because of the if-statement in the model. The model is a one-compartment 1st order absorption model.

Example 5, Model discrimination and penalty functions

This example shows two different optimal design techniques for model discrimination. The regular method used in PopED (*model_discrimination.xml*) and another example that weight the models differently (*model_discrimination_penalty.xml*). The weighted discrimination is weighing the models according to the number of parameters in the model, roughly making the model equally important while the default discrimination in PopED allows the bigger model (in terms of number of parameters) to influence the criterion more than the smaller model. In the weighing example the criterion is defined in a penalty function and can easily be adapted to other models than in the example. The example tries to find the best design that discriminates between a 1-compartment and a 2-compartment model. The penalty function is defined within the function *penalty_function.m*.

Example 6, Serial correlation (autocorrelation using autoregressive of order one, AR(1), model)

This example shows how autocorrelation can be implemented and used within PopED. The autocorrelation variance is defined in the *AutoCorrelation.m* file. In this example the first residual error (additive error) is assumed to be AR(1) correlated

while the second residual error (proportional error) is assumed to be uncorrelated. Note that no correlation between the uncorrelated and correlated residual error variances is allowed. The path to the *AutoCorrelation.m* file needs to be updated to the local path before running the example.

Abbreviations and Variables

FIM – Fisher Information Matrix ~ Inverse of the expected Covariance-Variance matrix of the unknown parameter.

Criterion Function – The function that is optimized in the optimization procedure.

OFV – Objective Function Value – See criterion function.

RS – Random Search – A random search algorithm.

SG – Stochastic Gradient – A stochastic gradient search algorithm.

LS – Line Search – A line search algorithm.

GUI – Graphical User Interface – A graphical tool that communicates with the user.

bpop – Population mean variable.

b – Inter Individual variable.

d - Inter individual distribution variance.

covd – Inter individual covariances

docc – Occasion variability variance.

covdocc – Occasion variability covariances

a – Covariate variable.

x – Discrete variable.

g – Parameter definition vector.

xt – Sample time vector.

epsi – Residual variable for all sample and individuals.

sigma – Residual distribution variance.

covsigma – Residual covariances

IIV – Inter individual Variability – The variability between individuals.

BSV – Between Subject Variability See IIV.

IOV – Inter Occasion Variability – The variability between occasions.

BOV – Between Occasion Variability. See IOV.

NONMEM – NON linear Mixed Effects Modeling.

UDD – User Defined Distributions.

The function input structure

The function input file contains a structure with the following fields:

strPopEDVersion – A string containing the current version, e.g. '2.12'.

ng – The length of the parameter vector g (usually defined in sfg.m).

nbpop – The number of typical values.

nb – The number of Inter Individual Variance parameters.

ndocc – The number of Inter Occasion Variance parameters.

nx – The number of discrete variables for each design group.

na – The number of covariates for each design group.

NumOcc – The number of occasions.

m – The number of groups.

maxni – The maximum number of samples for each group.

minni – The minimum number of samples for each group (must be >0)

d_switch – Set to 1 if local optimal design (e.g. D-optimal) or 0 if global optimal design (e.g. ED-optimal)

iApproximationMethod – The approximation method used to calculate the FIM. (0 = FO, 1 = FOCE, 2 = FOCEI, 3 = FOI).

iFOCENumInd – The number of individuals that are sampled I FOCE or FOCEI approximation method are used.

iEDCalculationType – The calculation technique of expectation integral in robust designs. (0 = Monte Carlo, 1=Laplace Approximation, 2=BFGS Laplace Approx).

bUseRandomSearch – If the search should contain a random search (1 = true, 0=false).

bUseStochasticGradient – If the search should contain a stochastic gradient search (1 = true, 0 = false).

bUseLineSearch – If the search should contain a line search (1 = true, 0 = false).

bUseExchangeAlgorithm – If the search should be the modified exchange algorithm (1= true, 0 = false).

bUseBFGSMinimizer – If the search should contain the BFGS algorithm (1= true, 0 = false).

ofv_calc_type – The calculation criterion used to calculate the objective function value based on the FIM (1 = D-optimal (determinant of FIM), 2 = A-optimal (trace of inverse of FIM), 3 = S-optimal (not implemented), 4 = ln D-optimal (ln determinant of FIM), 5 = C-optimal (not implemented), 6 = Ds – optimal (determinant of FIM with all parameters divided by the determinant of FIM with the uninteresting parameters), 7 = 1/Sum of absolute RSE, (the inverse sum of the absolute relative standard errors).

prior_fim – The prior information matrix (if used; it should be of the same dimensions as the FIM).

optsw – A vector that determine what to optimize on (1 = optimize on this design space, 0 = do not optimize on this design dimension). The vector has this order: [samples per group, sampling schedule, discrete variables, covariates, number of individuals per group].

line_opta – vector determine if a particular covariate should be used in the line search (not implemented), set to 1 if line search should be used for all covariates.

line_optx – vector determine if a particular discrete variable should be used in the line search (not implemented), set to 1 if line search should be used for all discrete variables.

dSeed – the seed number used (-1 = random seed number) .

The substructure *design* contains the following fields:

groupsize – vector defining the initial number of individuals in each design group.

maxgroupsize – vector defining the maximum number of individuals in each design group.

mingroupsize – vector defining the minimum number of individuals in each design group.

maxtotgroupsize – The maximum sum of all individuals in all design groups. (The maximum number of individuals in the study).

mintotgroupsize – The minimum sum of individuals in the study (sum of all design groups).

sigma – A covariance matrix defining the residual variances an their correlations.

bpop – A 3 x number of bpop matrix defining the first column defines the type of the distribution for this bpop (the row number = bpop number). The valid distributions are: 0 = Fixed, 1 = Normal, 2 = Uniform, 3 = UDD, 4 = lognormal and 5 = truncated normal. The second column defines the mean (or value if fixed distribution) of each bpop. The third column defines the variance of the distribution for each bpop.

d – A 3 x number of inter individual variance parameters matrix. It has the same structure as bpop.

covd – A vector (of size n*(n-1)/2, where n is the number of d) with all the lower triangular values of the d covariance matrix (i.e. the correlation between the IIV parameters). If no correlation this is set to a vector of zeros.

docc – A 3 x number of occasion variance parameters matrix. It has the same structure as bpop and d.

covdocc – A vector (of size n*(n-1)/2, where n is the number of docc) with all the lower triangular values of the docc covariance matrix (i.e. the correlation between the IOV parameters). If no correlation this is set to a vector of zeros.

ni – A vector defining the initial number of sample for each group.

xt – A matrix defining the initial sampling schedule for each group (a row) . Size of this matrix is the maxni x number of design groups.

minxt – A matrix defining the minimal sampling time for each design group and sample. Same size as the xt matrix.

maxxt – A matrix defining the maximal sampling time for each design group and sample. Same size as the xt matrix.

x – A matrix defining the initial discrete variable values for each discrete variable in each design group. The size of the matrix is equal to the number of discrete variables x the number of design groups.

discrete_x – A cell matrix where each cell defines the possible values of a discrete variable. The size of the cell matrix is the same a the size of x. Note that each cell contains a vector with at least the initial value of this discrete variable (i.e. the same value as the corresponding element in x),

a – A matrix defining the initial covariate values for each design group and covariate. The size of a is number of covariates x number of design groups.

maxa – A matrix defining the maximum value of each covariate in all design groups. The matrix has the same size as a.

mina – A matrix defining the minimum value of each covariate in all design groups. The matrix has the same size as a.

model_switch – A matrix defining which model a certain sample in a certain design group belongs to. The size of the matrix is the same as the size of *xt*. If only one model this is set to a matrix of ones.

G – A matrix defining how the grouping of sample during an optimization should be done. The same of the matrix is the same as the size of *xt*. If the matrix contains the same number in two or more elements, these samples are grouped together (has always the same value). The grouping must be turned on using the *bUseGrouped_xt* field.

Ga – A matrix defining how the grouping of covariates during an optimization should be done. The same of the matrix is the same as the size of *a*. If the matrix contains the same number in two or more elements, these covariates are grouped together (has always the same value). The grouping must be turned on using the *bUseGrouped_a* field.

Gx – A matrix defining how the grouping of discrete variables during an optimization should be done. The same of the matrix is the same as the size of *x*. If the matrix contains the same number in two or more elements, these disceret variables are grouped together (has always the same value). The grouping must be turned on using the *bUseGrouped_x* field.

The following fields are not in the design structure but directly in the input structure:

notfixed_bpop – A vector of zeros or ones that tells if a bpop is excluded from the FIM or not. 1 = excluded (fixed), 0 = in FIM (not fixed). The length of the vector should be equal to the number of bpop.

notfixed_d – A vector of zeros or ones that tells if a d is excluded from the FIM or not. 1 = excluded (fixed), 0 = in FIM (not fixed). The length of the vector should be equal to the number of d.

notfixed_covd – A vector of zeros or ones that tells if a covd is excluded from the FIM or not. 1 = excluded (fixed), 0 = in FIM (not fixed). The length of the vector should be equal to the number of d.

notfixed_docc – A vector of zeros or ones that tells if a docc is excluded from the FIM or not. 1 = excluded (fixed), 0 = in FIM (not fixed). The length of the vector should be equal to the number of docc.

notfixed_docc – A vector of zeros or ones that tells if a covdocc is excluded from the FIM or not. 1 = excluded (fixed), 0 = in FIM (not fixed). The length of the vector should be equal to the number of covdocc.

notfixed_sigma – A vector of zeros or ones that tells if a sigma is excluded from the FIM or not. 1 = excluded (fixed), 0 = in FIM (not fixed). The length of the vector should be equal to the number of sigma.

notfixed_covsigma – A vector of zeros or ones that tells if a covsigma is excluded from the FIM or not. 1 = excluded (fixed), 0 = in FIM (not fixed). The length of the vector should be equal to the number of covsigma.

bUseGrouped_xt – 1 = Group the sample according to G (see above), 0 = do not group the sample in the optimization.

bUseGrouped_a – 1 = Group the covariate according to Ga (see above), 0 = do not group the covariate in the optimization.

bUseGrouped_x – 1 = Group the discrete variable according to Gx (see above), 0 = do not group the discrete variable in the optimization.

ff_file – The filename and path of the model file. If no path is available poped assumes that the file is available in the running directory (the default values is ff.m)

fg_file – The filename and path of the parameter vector definition file. If no path is available poped assumes that the file is available in the running directory (the default value is sfg.m).

fError_file – The filename and path of the residual error model file. If no path is available poped assumes that the file is available in the running directory (the default value is feps.m).

strUserDistributionFile – The filename and path of the file that contains all the user distribution definitions. If no path is available poped assumes that the file is available in the running directory. Empty string means no UDD file.

strEDPenaltyFile – The filename and path of the file that contains the ED penalty criterion. This file might be used only if *d_switch* is set to 0. If no path is available poped assumes that the file is available in the running directory. Empty string means no ED penalty file. Note that this function can also be used to define user defined local criterions (see model discrimination example).

strAutoCorrelationFile – The filename and path of the file that contains the user defined variance term (only the sigma part). If no path is available poped assumes that the file is available in the running directory. Empty string means no autocorrelation file is defined. Note that this function can also be used to define user defined variance terms.

strRunFile – The filename and path of a run file, if this option is set, i.e. not an empty string, PopED (only) executes the run file instead of any optimization/evaluation tasks. However, PopED can be called within the run file and then regular tasks as optimization/evaluation will be executed instead.

modtit – A string with the model title.

bShowGraphs – 1 if the system should show graphs during the search, 0 otherwise.

use_logfile – 1 if the system should use a log file, 0 otherwise (not implemented).

output_file – The filename and path of the output file that contains the information about the search for the optimal design. This is the suffix of the filename (a prefix will be added dependent on which search method that is used). If no path is available poped assumes that the file is available in the running directory. A typical value can be 'output.txt'

output_function_file – The Matlab function filename that will contain the output from an evaluation or optimization. This should be a valid Matlab function name (**not** containing the extension (.m)).

strIterationFileName – The Matlab function filename and path of the iteration output file that contains the information about the current best design during the search for the optimal design. If no path is available poped assumes that the file is available in the running directory. Set this to an empty string if no iteration file should be created. Note that this file is overwritten for each search method and iteration in the search. The filename **should** contain the extensions (*.m)).

m1_switch – The method used to calculate the derivative of the linearized model with respect to the bpop (M1). 0 = Complex difference, 1 = Central difference, 20 = Analytic derivative and 30 = Automatic derivative.

m2_switch – The method used to calculate the derivative of the variance model with respect to the bpop (M2). 0 = Complex difference, 1 = Central difference, 20 = Analytic derivative and 30 = Automatic derivative.

hle_switch – The method used to calculate the linearized error model (H). 0 = Complex difference, 1 = Central difference and 30 = Automatic derivative. Note that 20 is not available.

gradff_switch – The method used to calculate the gradient of the model with respect to the parameter vector g. 0 = Complex difference, 1 = Central difference, 20 = Analytic derivative and 30 = Automatic derivative.

gradfg_switch – The method used to calculate the gradient of the parameter vector model with respect to the individual values. 0 = Complex difference, 1 = Central difference, 20 = Analytic derivative and 30 = Automatic derivative.

bLHS – The sampling method used to get random samples. If 1, Latin Hyper Cube samples are used otherwise (0) regular random sampling is used.

ourzero – Indicate a (small) value that will replace each 0 value if the sampling schedule; This to get a more stable stochastic gradient. (Can also be set to 0, then it doesn't affect the sampling schedule at all).

rsit_output – The number of random search iteration until a new output to the screen will be visible.

sgit_output – The number of stochastic gradient search iteration until a new output to the screen will be visible.

hm1 – The step length of the derivative to calculate M1. This value will only be used if the derivative option is complex or central difference).

hlf – The step length of the derivative of the model with respect to the g vector. This value will only be used if the derivative option is complex or central difference).

hlg – The step length of the derivative of the parameter vector g with respect to b-vector. This value will only be used if the derivative option is complex or central difference).

hm2 – The step length of the derivative to calculate M2. This value will only be used if the derivative option is complex or central difference).

hgd – The step length of the derivative of the OFV. This method will always use central difference and is used in the stochastic gradient method.

hle – The step length of the derivative of the residual error model with respect to the epsilon. This value will only be used if the derivative option is complex or central difference).

AbsTol – The absolute tolerance of the ordinary differential equation solver.

RelTol – The relative tolerance of the ordinary differential equation solver.

iDiffSolverMethod – The ordinary differential equation solver method used. 0 = ode45, 1 = ode15s (Not available in FreeMat).

bUseMemorySolver – If the solution of the ordinary differential equation should be solved only once and stored in memory. 1 = Use memory solver, 0 = No memory solver.

iFIMCalculationType – The calculation method used to calculate the FIM. 0 = Full FIM (with M1, M2, M3 and M4 matrices), 1 = Reduced FIM (assuming that M2 = 0), 2 = Weighted models (not available), 3 = LOQ models (not available), 4 = Reduced FIM but the variance is calculated as the derivative of the residual standard deviation instead of the residual variance. 5 = Full FIM (with A, B and C matrices instead), should be equal to option 0, 6 = Calculate one model switch at a time (Full FIM and assumes no correlation between responses). 7 = Reduced FIM (with A,B and C matrices instead), should be equal to option 1.

rsit – The number of random search iterations. Only used if *bUseRandomSearch* is set to 1.

sgit – The number of stochastic gradient search iterations. Only used if *bUseStochasticGradient* is set to 1.

intrsit – The number of random search iterations if optimized on sample patterns.

intsgit – The number of stochastic gradient iterations if optimized on samples patterns.

maxrsnullit – The number of iterations until increase the adaptive narrowing in random search. I.e. if the random search has not changed for *maxrsnullit* the search becomes more narrow (more local),

convergence_eps – The convergence value for stochastic gradient (see stochastic gradient for more information),

rslxt – The random search locality factor for samples times.

rsla – The random search locality factor for covariates.

cfaxt – The stochastic gradient first step factor for sample times.

cfaa – The stochastic gradient first step factor for covariates.

bGreedyGroupOpt – Set to zero if an exact (but possibly slow) groupsize optimization method should be use. Set to one if the greedy option, i.e. the fast approximated optimization, should be use.

EACriteria – The exchange algorithm criteria to use (1 = Modified, 2=Fedorov (not available)). This is only used if *bUseExchangeAlgorithm* is set to 1.

EAStepSize – The exchange algorithm step size (i.e. the size of the step for each split design dimension, number of search points in each design variable e.g. *nxt* = (maxxt-minxt)/StepSize). If this is set to zero the *EANumPoints* should be used instead.

EANumPoints – The number of points each design dimension should be split into., e.g. the step size = (maxxt-minxt)/EANumPoints. If this is zero, the *EAStepSize* should be used instead.

EAConvergenceCriteria – The convergence value for the exchange algorithm.

bEANoReplicates – Set to one if no grouped sampling replicates are allowed in the MFEA algorithm. Set to zero if this is allowed.

BFGSConvergenceCriteriaMinStep – The BFGS minimum step convergence criteria (see BFGS algorithm for more information). Typically set to a small number e.g. 1e-08.

BFGSProjectedGradientTol – The BFGS smallest normalized projected gradient convergence criteria. Typically set to a small number e.g. 1e-04.

BFGSTolerancef – The BFGS Line Search tolerance f (see line_search.m for more information). Default = 1E-03.

BFGSToleranceg – The BFGS Line Search tolerance g (see line_search.m for more information). Default = 0.9.

BFGSTolerancex – The BFGS Line Search tolerance x (see line_search.m for more information). Default = 0.1.

ED_samp_size – The number of ED samples that should be sampled. This is only used if *d_switch* = 0. If a penalty function is used to define a local criteria this value can be set to 0 and no sampling will be done. The sampling technique depends on *bLHS* and the distributions are defined in *bpop*, *d* and *docc* (see above).

ED_diff_it – The number of iterations to calculate the convergence criteria between line search and stochastic gradient. If a penalty function with a local criteria is used this can be set to 1. This is only used when *d_switch* = 0.

ED_diff_percent – The difference in percent between line search and stochastic gradient. This is only used when *d_switch*=0.If a penalty function is used with a local criteria the value can be set to a very small number (close to zero).

line_search_it – The number of points each variable in the line search should be split into. E.g. the search will evaluate every value of the sample between *maxxt* and *minxt* with a step length of (*maxxtr-minxt*)/*line_search_it*. This is only used if *bUseLineSearch* is set to 1.

iNumSearchIterationsIfNotLineSearch – The number of search iterations (SG,RS and LS) that is used if no Line search is used, i.e. *bUseLineSearch* is set to 0. The *bUseRandomSearch* and *bUseStochasticGradient* determine if these search methods should be used.

user_data – This can be set to any data structure and is defined as a cell structure. This structure will be available in the model, the error model or any user defined input file except the parameter vector model file (sfg.m). Note that if run directories are used this structure will **not** be copied into the new created input file in the run directory.

The substructure *CriterionOptions* contains the following fields:

ds_index – vector if a parameter is treated as uninteresting or not. The length of the vector should be the number of unfixed parameters (column or row size of FIM). Set the value to 0 if a parameter is interesting or 1 if a parameter is uninteresting. The parameter order is the same as in the FIM (i.e. unfixed: bpop, d, docc and sigma).

The substructure *ParallelSettings* contains the following fields:

iCompileOption – A integer between -1 – 5. This option is used in the MPI parallel execution. -1 means no compilation, 0 – full compilation with a regular run afterwards, 3 – full compilation but no run. 1 – only MCC compilation with a regular run afterwards, 4 – only MCC compilation but no run. 2 – only MPI compilation but with regular run. 5 – only MPI compilation without run.

iUseParallelMethod – Which parallel execution method to use. 0 = Matlab PCT, 1 = Open MPI with mpirun.

strAdditionalMCCCompilerDependencies – Extra option to pass a string (options) to the MCC compiler (only used when *iUseParallelMethod* = 1).

strExecuteName – The name of the executable produced by the MPI compilation. (only used when *iUseParallelMethod* = 1).

iNumProcess – The number of processes/cores the parallel execution should use. In PCT the number of workers = *iNumProcess,* with MPI the number of workers = *iNumProcess* – 1 and one job manager.

iNumChunkDesignEvals – The number of designs (chunks) to calculate for each worker before it can take any new jobs. Set to -2 if the Job manager should work, otherwise; set to -1 to get the theoretical optimal value (however, not very often the same as the best value in practice). Otherwise set to >=1. (only used when *iUseParallelMethod* = 1).

strMatFileInputPrefix – The prefix name of the input file (a mat file) to the executable. A random number will be added to the input file after the prefix. (only used when *iUseParallelMethod* = 1).

strMatFileOutputPrefix – The prefix name of the output file (a mat file) from the executable to the PopED script. A random number will be added to the output file after the prefix. (only used when *iUseParallelMethod* = 1).

strExtraRunOptions – A string with extra run options passed to the bash executable or the mpirun file. (only used when *iUseParallelMethod* = 1).

dPollResultTime – The time (in sec) between checking if a new result file has been created with the evaluated designs. (only used when *iUseParallelMethod* = 1).

strFunctionInputName – The name of the function input that should be used when building a C-shared library with the MCC compiler. (only used when *iUseParallelMethod* = 1).

bParallelRS – Set to 1 if Random Search should be executed in parallel, otherwise 0.

bParallelSG – Set to 1 if Stochastic Gradient should be executed in parallel, otherwise 0.

bParallelLS – Set to 1 if Line Search should be executed in parallel, otherwise 0.

bParallelMFEA – Set to 1 if Modified Fedorov Exchange Algorithm should be executed in parallel, otherwise 0.