Implementation Information:
MEMO - Multi-experiment mixture model analysis of censored data

E.-M. Geissen, J. Hasenauer, S. Heinrich, S. Hauf, F.J. Theis and N. Radde

Contents

1 Implementation of MEMO

MEMO is implemented in Matlab and has been tested under Version R2014a under Mac OS, Linux and Windows 7.

1.1 Requirements

In addition to the provided Matlab functions, the Parameter EStimation Toolbox (PESTO) developed by Jan Hasenauer (Institute of Computational Biology, Helmholtz Zentrum München, Germany) is required. PESTO is included in the MEMO download. Furthermore the Symbolic Math Toolbox and the Statistics and Machine Learning Toolbox of Matlab are required. For MCMC sampling the MATLAB toolbox DRAM (Haario et al., 2006) is required.

1.2 Model formulation and data format

MEMO uses a single definition file, an m-file, for each finite mixture model. This definition file is provided by the user and contains the specification of the model $\mathcal{M}$ and the data $\mathcal{D}$. An example is shown in Figure I1. The model setup starts by defining the struct `parameters` that specifies the parameters and their range for optimization, sampling and profile likelihood estimation. Since MEMO uses symbolic computation, the parameters have to be defined symbolically and then stacked in a parameter vector. Furthermore, for each parameter a name has to be specified that is used for plots and other visualizing routines and should therefore comply to a LaTeX interpretable format (Figure I1, lines 3 - 21).
In addition, the definition file provides the structure of the finite mixture model for the quantity of interest. The type of distribution can be chosen and the unit of the modeled data can be specified. The latter will be used as label in visualization routines. Furthermore, `M` contains the specification of the different experimental conditions that are subject to analysis. For every experimental condition the subpopulation structure, namely the number of subpopulations and their distribution type, can to specified. The symbolic parameters are assigned to their respective experimental condition and subpopulations. If modeling of the censoring quantity is chosen, the same is performed for the censoring quantity in a second struct `Mc` (Figure I1, lines 23 - 52).
The definition file also links the data to the model. All information on the data is stored in a cell array `D`, whose length is the number of different experimental conditions. For every dataset a title specifying the experimental condition and the vectors containing the data grouped by the type of data (uncensored / interval censored (`Tm`), right censored (`Tc`)) has to be provided. Therefor, data for the different experimental conditions can be loaded from separate data files (Figure I1, lines 34, 53-57) or data and the associated information can be specified in the definition file itself. An exemplary data file is shown in Figure I2. Subsequent to the definition, model and data structure are compiled by the MEMO function `getMixtureModel.m` that creates functional expressions of parameters and derivatives (Figure I1, lines 69 + 70).

1.3 Likelihood calculation

To calculate the likelihood for a certain model given the data, MEMO provides the functions `logLikelihoodMM.m` and `logLikelihoodMMc.m` for model setups without and with model of the censoring distribution, respectively. These functions take the parameters struct, the model struct(s) and the data struct as an input. If two outputs are specified also the gradient of the likelihood is evaluated. Examples for the function call are shown in Figure I3.

1.4 Optimization

To find the optimal parameters of the finite mixture models, given the data, the function `optimizeMultiStart.m` is called (Figure I4). This function performs a multi-start local optimizations, as explained above. The first input argument is the `parameter` struct. The second argument is a function handle to the respective likelihood function (`logLikelihoodMM.m` or `logLikelihoodMMc.m`) and the last argument is a struct with options, e.g. for `fmincon` (see m-file of `optimizeMultiStart.m` for further information). The default options are shown in Figure I5.

```
%% model illustration MEMO model setup for a model with log-normally distributed events ...
    of interest and Johnson SU distributed censoring events.
%%
%% SPECIFY MODEL PARAMETERS
syms     mu_exp1_sub1 mu_exp1_sub2 log_sigma_exp1_sub1 log_sigma_exp1_sub2 w_exp1_sub1  ...
         gamma_cen  esigma_cen  elambda_cen  exi_cen;

parameters.sym  = ...
                    [mu_exp1_sub1; mu_exp1_sub2; log_sigma_exp1_sub1; ...
                        log_sigma_exp1_sub2; w_exp1_sub1;  ...
                                        gamma_cen; esigma_cen; elambda_cen; exi_cen];

% parameter names for plotting and other vizualization
parameters.name = {'\mu_{exp1,sub1}'; 'log(\sigma_{exp1,sub1})'; '\mu_{exp1,sub2}'; ...
    'log(\sigma_{exp1,sub2})'; 'w_{exp1,sub1}'; ...
                                    '\gamma_{cen}'; 'log(\sigma_{cen})'; ...
                                       'log(\lambda_{cen})'; 'log(\xi_{cen})' };

parameters.number = length(parameters.sym);

parameters.min  = [ log(5); log(1e-1); 0; ...
                                        -20    ; log(1e-4); log(5); log(5) ];

parameters.max  = [  log(2e3); log(1e1); 1; ...
                                        20; log(1e4); log(2e3); log(2e3)];

%% SPECIFY MODEL AND DATA
M.mixture.type = 'log-normal'; % specification of distribution type
M.label.x = 'time [min]'; % x-label for model-data-comparison plots, in units of ...
    measurement data
M.label.y = 'probability density'; % y-label for model-data-comparison plots

Mc.mixture.type = 'Johnson SU';
Mc.label.x = 'time [min]';
Mc.label.y = 'probability density';

%% Experimental condition No. 1
% File were data for Exp.1 is stored
data_file_exp_1;
i = 1 ;

M.experiment(i).name  = tit; % title defined in data file data_file_exp_1.m
M.experiment(i).size  = 2;  % number of mixture components=subpopulations
M.experiment(i).w     = {1-w_exp1_sub1,w_exp1_sub1}; % weights
M.experiment(i).mu    = {mu_exp1_sub2,mu_exp1_sub1};  %  one μ per subpopulations
M.experiment(i).sigma = {exp(log_sigma_exp1_sub2),exp(log_sigma_exp1_sub1)}; % one σ per ...
    subpopulations, exp(sigma) for parmeter estimation in logarithmic space

%% model for censoring times of exp.1
Mc.experiment(i).name   = tit;
Mc.experiment(i).size   = 1;
Mc.experiment(i).w      = {            1 };
Mc.experiment(i).gamma  = {       gamma_cen };
Mc.experiment(i).sigma  = { exp(esigma_cen)};
Mc.experiment(i).lambda = {exp(elambda_cen)};
Mc.experiment(i).xi     = {     exp(exi_cen)};


D{i}.name = tit;  % title defined in data file data_file_exp_1.m
D{i}.description = []; % optional
D{i}.data.uncensored = Tm;
D{i}.data.censored = Tc;
D{i}.observation_interval = 5; % in this example data is interval censored and inter ...
    observation time is 5 min

%% Experimental condition No. 2
i = i+1 ;

% and so on


% Compile model
% (This generates the functional expression of parameters and derivatives.)
[M, parameters.constraints] = getMixtureModel(M, parameters.sym);
Mc = getMixtureModel(Mc, parameters.sym);
```

Figure I1: Example model definition in MEMO.

```
% data measured in Mad3 knockout strain
tit=('Delta Mad3'); % title of experimental condition

% experimental condition: normal amount of Mad1 and Mad2, no expression of Mad3
Mad1 = 1;
Mad2 = 1;
Mad3 = 0;

% data of single cells for event of iterest
% here interval censored but not right censord data points
Tm=[
55
55
55
45
60
50
];

% data of cenoring event. here right censored data points, in this example there is no ...
    right censored data
Tc=[];
```

Figure I2: Example MEMO data file.

```
% without model for censoring event
[logL,grad] = logLikelihoodMM(theta,M,D,options)

% with model for censoring event Mc
[logL,grad] = logLikelihoodMMc(theta,M,Mc,D,options)
```

Figure I3: Function calls to obtain the likelihood and its gradient.

```
%% OPTIMIZATION
% Options
options_fit.n_starts = 200;
options_fit.plot = plot_opt;
options_fit.proposal = 'latin hypercube';
options_fit.fmincon = optimset('algorithm','interior-point',...
                               'display','off',...
                               'GradObj','on',...
                               'MaxIter',4000,...
                               'MaxFunEvals',4000*parameters.number);
% Run estimation
[parameters,M.fh.fit] = optimizeMultiStart(parameters,@(theta,opt) ...
    logLikelihoodMMc(theta,M,Mc,D,opt),options_fit);
```

Figure I4: Exemplary call of function `optimizeMultiStart.m`.

```
options.fmincon = optimset('algorithm','interior-point',...
                           'display','off',...
                           'GradObj','on',...
                           'MaxIter',3000,...
                           'MaxFunEvals',3000*parameters.number);
options_MS.linprog = optimset('algorithm','active-set');
options.n_starts = 20;
options.proposal = 'latin hypercube';
options.plot = 'true';
options.mode = 'normal'; % 'silent';
options.logPost_options.sign = 'negative';
options.logPost_options.grad_ind = [1:parameters.number]';
options.fh = [];
```

Figure I5: Default values for the options of multi-start optimization.

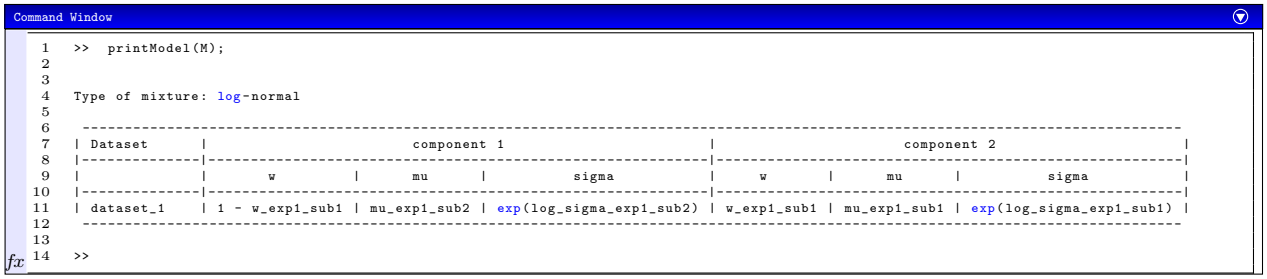

Figure I6: Output of function `printModel.m` to matlab command window for the example in Figure I1.

1.5 Automated backward model selection

The automated model backward selection is performed by calling the function `performModelSelection.m` which takes the model with the maximum number of components per experimental condition `M`, the parameters, the data struct `D` and options and returns the most likely model consisting of `M_red` and `Mc_red`, and the reduced parameter struct `parameters_red`. For details on the method please see Section S.3 of the Supplementary Material.

1.6 Goodness of fit analysis

The Goodness of fit of a model can be assessed via the function `analyzeLogPostDistribution.m`. The function estimates the expected distribution of the likelihood function values, assuming that the model `M` with parametrized with `parameters` describes the data generating distribution. To estimate the distribution of the likelihood function values, artificial measurement data are generated from the mixture model, and the log-posterior function is evaluated.

1.7 Uncertainty analysis

MEMO provides two different approaches to assess the uncertainty of the estimated parameters. There are inbuilt functions for profile likelihood estimation and MCMC sampling. For details we refer to Section S.2 of the Supplementary Material. These function can be called by the functions `computeProfiles.m` and `computeMCMCsample_DRAM.m` included in PESTO.

1.8 Visualization

MEMO provides the user with a variety of visualization functions. The function `printModel.m` prints the specified model in the Matlab command window either with symbolic names of the parameters or, if the `parameter` struct with optimization results is provided as input, the parameter values of the best fit (see Figure I6 for example output). Parameter values shown of `printModel.m` are the parameters θ (not $\xi = \log(\theta)$, see Section S.2.1 of the Supplementary Material.
The function `plotMixtureModel.m` visualizes the model data fit graphically by creating a Matlab figure with subplots for every experimental condition (output similar to Figure **??**). There are several possible options, including visualization of parameter uncertainties obtained by MCMC sampling.

References

Haario, H., Laine, M., Mira, A., & Saksman, E. (2006). DRAM: Efficient adaptive MCMC. Stat. Comp., 16(4), 339–354.