A Tutorial on DPP and GuiDPP

Contents

1 Requirements and Installation

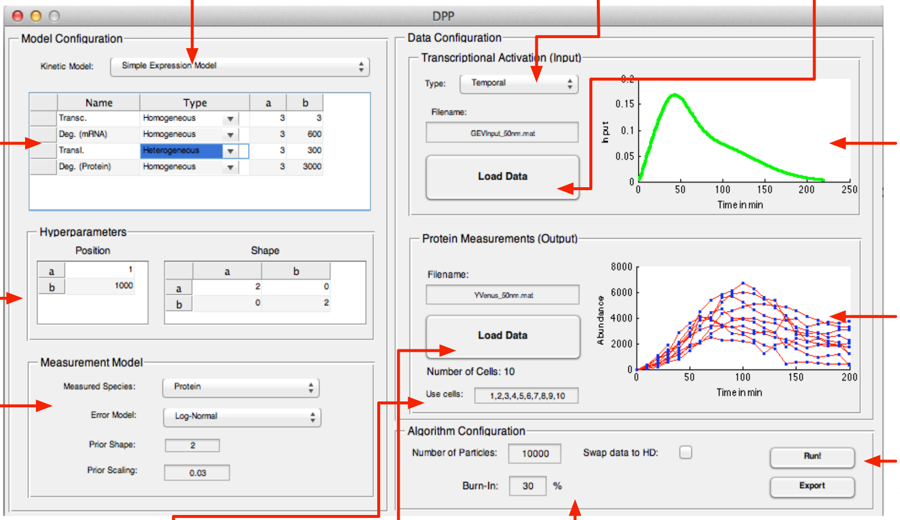
- Working MATLAB installation (toolbox tested with MATLAB2012b).
- Working mex/gcc configuration.
- MacOS, Windows or Linux.
- Extract the zip-archive that you have downloaded to a desired location *PATH*.

2 GuiDPP

GuiDPP is a MATLAB application based on the DPP toolbox with a graphical user interface (GUI). While DPP applies to arbitrary biochemical systems, GuiDPP was specifically designed for calibrating heterogeneous gene expression systems that are regulated by a transient stimulus (e.g., such as the translocation of a transcription factor).

2.1 Starting GuiDPP

Start MATLAB and open the script *PATH/GUI/StartGUI.m*. Upon execution of the script, GuiDPP pops up and is ready to use. A quick guide with the most important properties and features can be found in Figure 1.



DPP

Model Configuration

Kinetic Model: Simple Expression Model

	Name	Type	a	b
	Transc.	Homogeneous	3	3
	Deg. (mRNA)	Homogeneous	3	600
	Transl.	Heterogeneous	3	300
	Deg. (Protein)	Homogeneous	3	3000

Hyperparameters

Position

a	1
b	1000

Shape

	a	b
a	2	0
b	0	2

Measurement Model

Measured Species: Protein

Error Model: Log-Normal

Prior Shape: 2

Prior Scaling: 0.03

Data Configuration

Transcriptional Activation (Input)

Type: Temporal

Filename: GEVInput_50nm.mat

Protein Measurements (Output)

Filename: YVenus_50nm.mat

Number of Cells: 10

Use cells: 1,2,3,4,5,6,7,8,9,10

Algorithm Configuration

Number of Particles: 10000

Swap data to HD: □

Burn-In: 30 %

Kinetic model: Select the kinetic mass-action model from the drop-down menu.

Input profile: Choose whether you want to use a temporal input profile (e.g., measured TF-abundance).

Load input data: Press button to select and load an input data file.

Parameter table: Adjust priors for reaction rates by changing *a* and *b*. One reaction can be assumed to be depending on extrinsic factors by choosing *Heterogeneous* in the column *Type*.

Hyperparameters: Set mean and covariance of the log-normal prior distribution of the extrinsic statistics *a* and *b*.

Measurement model: **(a)** Specify the species that was measured in the experiment. **(b)** Select measurement noise density. **(c)** The measurement noise is estimated from the data. Provide parameters for the inverse-Gamma prior distribution.

Input data plot: Visualizes the loaded input data. In case no input data is selected, a constant value of 1 will be displayed.

Measurement plot: Plots the imported single-cell trajectories.

Run / Stop DPP: Starts the inference algorithm with the given configuration. Once running, the algorithm can be stopped by pressing the button again.

Select cells to consider: You can specify which cells shall be used for inference. Enter integer-numbers between 1 and the number of cells separated by comma.

Load measurements: Press button to select and load the single-cell measurements.

Algorithm configuration: Choose number of particles and the fraction of discarded particles at each time-step (*Burn-In*). Check Swap data to HD in order to save intermediate results to hard drive. This will reduce the amount of memory needed during computation.

Figure 1: Quick guide for GuiDPP .

2.2 How to use GuiDPP

Kinetic model and parameters: GuiDPP automatically loads all models from *PATH/Models/* and displays them in the corresponding drop-down menu. Per default, two gene expression models are available:

1. **Simple Expression Model:** A simple model of transcription and translation given by the reactions:

$$\emptyset \underset{c_2}{\overset{c_1 u(t)}{\rightleftharpoons}} \text{mRNA}$$

$$\text{mRNA} \xrightarrow{c_3} \text{mRNA} + P$$

$$P \xrightarrow{c_4} \emptyset.$$

2. **Two-State Model:** Canonical two-state model including gene-activation and deactivation:

$$\begin{aligned}
G^{off} &\underset{c_2}{\overset{c_1u(t)}{\rightleftharpoons}} G^{on}\\
G^{on} &\xrightarrow{c_3} G^{on} + \text{mRNA}\\
mRNA &\xrightarrow{c_4} \emptyset\\
\text{mRNA} &\xrightarrow{c_5} \text{mRNA} + P\\
P &\xrightarrow{c_6} \emptyset.
\end{aligned}$$

Additional models can be included into GuiDPP by copying the corresponding **.mat* files to the *PATH/Models/* folder. The definition of custom models is described in Section 4.1.

Once a model has been selected, the parameter table will be automatically updated. Each row corresponds to a particular kinetic parameter (i.e., stochastic rate constant) and its properties. In the column *Type*, you can select a reaction that is assumed to be driven by extrinsic factors. In this case, choose *Heterogeneous*[1]. The columns a and b are used to specify the Gamma-type prior distributions for the individual parameters. For practical hints on the choice of the prior distributions see Section 4.4.

Hyperparameters: If a reaction is chosen to be heterogeneous, the section *Hyperparameters* becomes visible. These parameters will be used to specify the bivariate (hyper)-prior distribution for (a, b) corresponding to the heterogeneous parameter. In particular, a log-normal distribution will be constructed as

$$p(a, b) \propto \mathcal{N}(\ln \mu, \Sigma),$$

with

$$\mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}$$

and

$$\Sigma = \begin{pmatrix} s_a & s_{a,b} \\ s_{a,b} & s_b \end{pmatrix}$$

as the parameter values provided by the user.

Measurement model: The user can select the chemical species which was measured in the experiment and chose among a normal and log-normal measurement error density. In both cases, the shape parameter of the respective density (e.g., the standard deviation σ in case of a normal distribution) is assumed to be unknown and estimated from the data. The prior distribution for the shape parameters is assumed to be inverse-Gamma, which is a conjugate prior for both the normal and the log-normal density. The inverse-Gamma distribution is specified by

$$I\mathcal{G}(p, q) = \mathcal{G}(p, q^{-1}), \tag{1}$$

with p and q as the provided shape and scaling parameters.

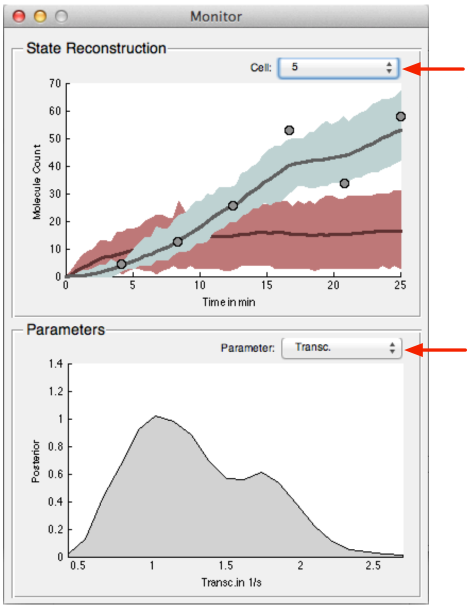
Transcriptional Activation (Input): As described above, the expression of the target gene is assumed to be transiently regulated by a certain *input* stimulus $u(t)$. In case there is no such input (e.g., for constitutively expressed genes), choose *Constant* in the drop-down menu *Type*. If you choose *Temporal*, you can specify and load an external input file. Exemplary input files are provided in the folder *PATH/ExampleData/*. Section 3.1 contains further information on how custom input profiles can be created within MATLAB.

Protein Measurements (Output): The measurement file can be located and loaded using the *Load Data* button. The individual single-cell traces will be automatically plotted. The text box *Use cells* can be used to select only a subset of the cells. Enter the desired cell-numbers separated by comma. Exemplary measurement files are provided in the folder *PATH/ExampleData/*. Section 3.1 contains further information on how to generate valid measurement files from your custom experimental data.

Algorithm Configuration: Specify the number of particles used during each time iteration. Note that this value directly influences the execution time and allocated memory. For a quick test run, we recommend to use a small number of particles (e.g., around 500-1000), but keep in mind that the resulting posterior distributions are likely to be degenerate. The parameter *Burn-In* defines the relative amount of particles to be discarded in the underlying Markov Chain Monte Carlo (MCMC) scheme. Typically, a longer burn-in period will be needed if many cells are in place. For large simulations, we recommend to select the *Swap data to HD* options since the data created at runtime can easily reach several gigabytes. In this case, the intermediate posterior distributions at the individual time points will be swapped to your hard drive (i.e., *PATH/GUI/DistTmp_**). These files can subsequently be used to reconstruct to full posterior distribution over all time points (see Section 3.1).

[1] Note that in the current version, only one heterogeneous reaction is supported.

Starting DPP : Once you have finished your configuration you can press *Run!* in order to start the inference algorithm. The caption of the button will now change to *Stop!*, which you can now use to cancel the running simulations. Be aware that both starting and stopping the simulations will require a certain amount of time, depending on your configuration and the number of particles (memory pre-allocation, cleanup, etc.). After you have started the algorithm, a new window will pop up, which you can use to monitor a running simulation. In particular, it will display the most recent posterior distributions over the states and parameters (see Figure 2). Note that the posterior distribution over the states will not change (i.e., will be empty) until the first time iteration has completed.



Cell: 5

Parameter: Transc.

Cell menu: Select the cell that you want to monitor. If the algorithm is still running, it may take a while until the figure is updated.

Parameter menu: Select the parameter you would like to monitor. Again, during execution of the algorithm, the update may be delayed by a few seconds.

Figure 2: Monitoring the simulations.

Export: After the algorithm has finished, you can export the inferred parameter statistics. Press the *Export* button to specify and save the destination file. The exported file can be easily loaded into MATLAB for further analysis. More specifically, it will contain a structure *Stat*, containing the prior and posterior statistics for all parameters (i.e., the mean values and the 0.05 and 0.95 quantiles).

Example Data: The toolbox comes with two exemplary datasets:

1. **Synthetic data:** A synthetic dataset generated using the *Simple Expression Model* and a double-pulse input stimulus. The corresponding files can be found in the folder *PATH/ExampleData/SimpleModel/*. Additionally, you can find the associated MATLAB scripts that were used to generate the data.
2. **GAL1 data:** An experimental dataset from the study in [1]. It consists of measurements of a fluorescent reporter that is expressed upon translocation of the transcription factor *GEV*. The *GEV* translocation was also experimentally measured and acts as the temporal input to the stochastic model. The corresponding files can be found in the folder *PATH/ExampleData/GAL1/*.

Step-by-step Example:

1. Execute the file *PATH/GUI/StartGUI.m*. The main window will pop up.
2. Per default, the kinetic model drop-down menu will be set to *Simple Expression Model* and the parameter table will show the corresponding parameters. Keep all parameter values as initialized.
3. In the panel *Transcriptional Activation (Input)* select *Temporal* and load the file *PATH/ExampleData/SimpleModel/InputSimpleModel.mat*. The input profile will be displayed in the corresponding figure.
4. In the panel *Protein Measurements (Output)* select and load the file *PATH/ExampleData/SimpleModel/MeasurementsSimpleModel.mat*. The measurements will be displayed in the corresponding figure.
5. For a quick illustration, set the number of particles to 200. This will result in a short simulation time but keep in mind that the resulting posterior distributions are likely to be degenerate (see section 4.3). Keep all other parameters as initialized.
6. Press *Run!*. The algorithm will start and the monitor window pops up. You can use the corresponding drop-down menus to choose between different cells and parameters.

7. In case you want to abort the simulations, press *Stop!*.
8. Once the algorithm has finished, press *Export* to save your results.

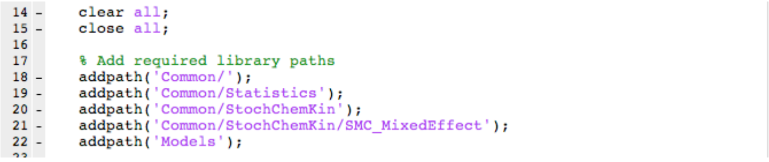
3 The DPP Toolbox

DPP is a MATLAB toolbox for the inference of stochastic chemical reaction networks using time-lapse single-cell data. In contrast to GuiDPP , it is fully general and applies to arbitrary biochemical systems.

3.1 How to Use DPP

In the following, we explain the functionality of DPP using an exemplary MATLAB script. A similar script is delivered with the toolbox, which is located in the root directory (i.e., *PATH/DPP_Example.m*). The goal of the script is to infer stochastic gene expression dynamics from heterogeneous single-cell data using DPP .

Setting path dependencies: DPP requires certain library paths to be set. In particular, it will use functions located in the folder *PATH/Common/** and the model specifications in *PATH/Models/*. Any additional libraries can be added analogously. Figure 3 shows the corresponding code lines.



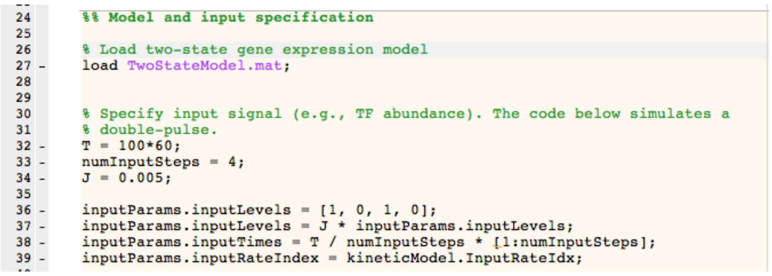
```
clear all;
close all;

% Add required library paths
addpath('Common/');
addpath('Common/Statistics');
addpath('Common/StochChemKin');
addpath('Common/StochChemKin/SMC_MixedEffect');
addpath('Models');
```

Figure 3: Setting the path dependencies.

Initialize model and input: The script is based on the *Two-State Model* described in Section 2.2. In line 27 of Figure 4, the model specification is imported. Subsequently, a structure *kineticModel* will be available in MATLAB's *Workspace*. Since the example code is based on synthetic data, a temporal input profile needs to be synthesized. In this case, we simulate a double-pulsed induction of the target gene. The input profile is represented by a piece-wise constant function. The corresponding structure *inputParams* is characterized by three fields:

- *inputTimes:* Specifies the time points where the input profile changes. The initial time point (i.e., $t = 0$) is not included.
- *inputLevels:* Specifies the magnitude of the input profile. At time $t = 0$, the magnitude of the input profile is set to the first value in *inputLevels*. At the time point corresponding to the first entry in *inputTimes*, the input profile changes to the second value in *inputLevels* and so on. Make sure that the timespan of your input profile is at least as large as the duration of the experiment.
- *inputRateIndex:* Defines the index of the reaction that is modulated by the input profile. In particular, the effective propensity will computed as $h(x, t) = c_i u(t) g_i(x)$, with $i = inputRateIndex$.

Lines 36-39 show the corresponding code from the example script. Note that *inputRateIndex* is part of the model specification and hence, the corresponding field in the *inputParams* structure is set to this value.



```
24     %% Model and input specification
25
26     % Load two-state gene expression model
27 -   load TwoStateModel.mat;
28
29
30     % Specify input signal (e.g., TF abundance). The code below simulates a
31     % double-pulse.
32 -   T = 100*60;
33 -   numInputSteps = 4;
34 -   J = 0.005;
35
36 -   inputParams.inputLevels = [1, 0, 1, 0];
37 -   inputParams.inputLevels = J * inputParams.inputLevels;
38 -   inputParams.inputTimes = T / numInputSteps * [1:numInputSteps];
39 -   inputParams.inputRateIndex = kineticModel.InputRateIdx;
```

Figure 4: Model and input initialization.

Constructing a general state-space model: DPP combines the kinetic model specification with further properties of the experiment to construct a general mixed-effect state-space model. These properties include specifications of the population heterogeneity, prior distributions or the measurement noise model. The corresponding code can be found in Figure 5. In line 44, the output-weight matrix is constructed. This defines the set of species (or linear combinations thereof) that were measured during the experiment. Hence, the measured quantities at a certain time point are given by $W^T X_l$. Since the given code example is based on synthetic data, the measurement time points need to be defined (lines 52-53). In contrast, if experimental data is used, those have to be set to the underlying acquisition time points.

In some cases it might be desired to assume certain kinetic parameters to be known a priori. Hence, the rate constants that are to be estimated from the data need to be specified (line 58). All other rate constants will be set to the values of model specification (i.e., the structure *kineticModel*).

For the remaining parameters, prior distributions need to be specified. The current implementation supports Gamma-type prior distributions $\mathcal{G}(a, b)$. Lines 67-68 specify the prior distributions corresponding to the previously defined reactions. Accordingly, the heterogenous reaction (in this case the translation rate) is selected to be heterogeneous. Note that in the current release, only one reaction can be heterogenous. If you wish the model to be entirely homogenous, set all values in *randomEffect* to zero. Note that for the heterogeneous reaction, the corresponding values a and b of the prior distribution are ignored during the inference. However, when using synthetic data, they are used for simulating the heterogeneity. Hence, the values a and b correspond to the ground-truth and accordingly, each cell's heterogeneous parameter will be randomly drawn from $\mathcal{G}(a, b)$. If a heterogenous reaction was selected, the bivariate log-normal hyperprior distribution over (a, b) needs to be specified. The two parameters *MuHyper* and *SigmaHyper* are defined in lines 76-78.

If morphological features are used for the inference, the morphological parameters ρ and ϕ need to be specified. The morphological parameters are assumed to be noisy readouts of the extrinsic factors specified by the distribution $\mathcal{G}(\rho, \phi z)$, with z as the extrinsic factor. Note that in this case, the morphological readout scales inversely with z, and hence, if the morphological feature is hypothesized to be positively correlated with the extrinsic factor, it should be inverted beforehand (i.e., such as might be the case for the cell-size). In case no morphological features are used, set *MorphParams = [];*, *MuMorph = [];* and *SigmaMorph = [];*. If you would like to use the morphological features but you already know ρ and ϕ (maybe from a previous experiment), set *MorphParams* to the desired values and *MuMorph = [];* and *SigmaMorph = [];*. In order to estimate the morphological parameters from the measurements, *MuMorph* and *SigmaMorph* define the log-normal hyperprior distribution over ρ and ϕ (see lines 87-89). During inference, the values specified in *MorphParams* will be ignored but will be used to simulate the morphological readouts in case of synthetic data.

Lines 95-96 define the measurement noise model. DPP supports either normal or log-normal measurement densities. In case of normal distribution (*mDens = 'normal';*), the measurements are assumed to stem from a Gaussian distribution centered around the true value with a certain standard deviation, i.e., $Y_l \sim N(W^T X_l, \sigma)$. For a log-normal error model (*mDens = 'logn';*), the measurements are assumed to be given by $Y_l \sim N(W^T X_l, s)$, with s as a scaling parameter. The vector *measurementParams* must contain either one or three values, depending on if the scaling parameter (either σ or s) of the noise shall be estimated from the data. If the scaling parameter is known, set *measurementParams* to the respective value. The scaling parameter is estimated, if *measurementParams* is a 1-by-3 vector, where the latter two entries specify the inverse-Gamma prior distribution (such as described in Section 2.2). Again, the first entry will then be ignored during inference, but will be used for generating the synthetic data (i.e., specifies the ground-thruth value).

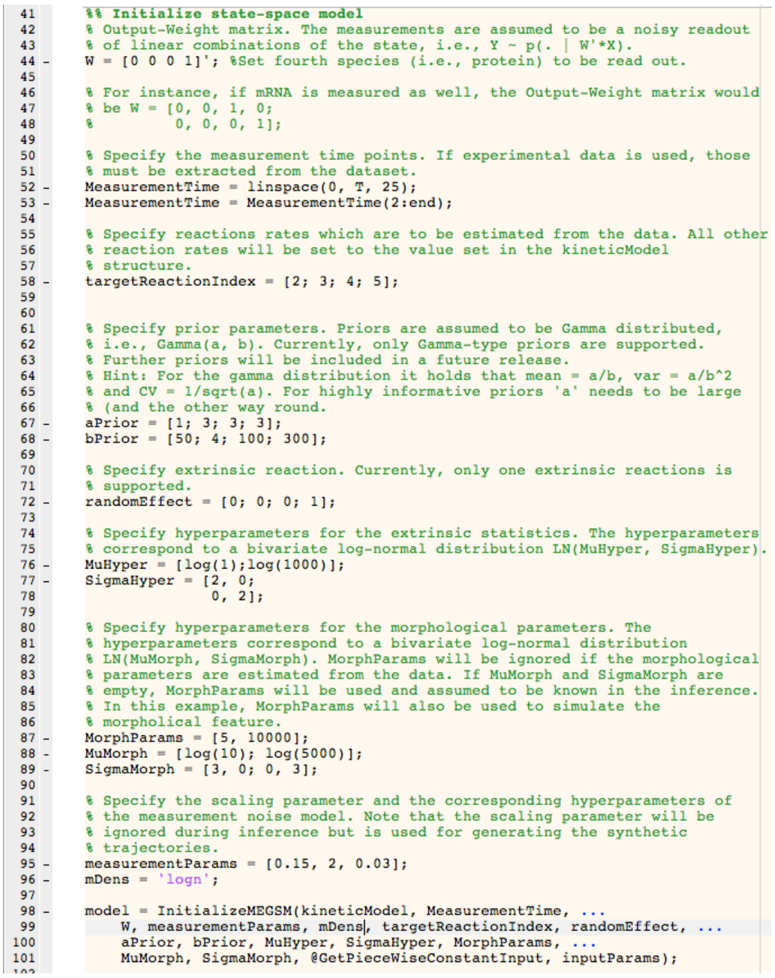
Given the previous definitions, the state-space model structure can be instantiated by calling the function *InitializeMEGSM* (see lines 98-101).

Load / simulate experimental data: If synthetic data shall be used for the inference, you can call the *SimulateCells* function (see lines 101-102 in Figure 6). The resulting trajectories and morphological features are simulated according to the defined state-space model and returned in the cell-array *cells*. Each entry of that array corresponds to one cell and includes the corresponding measurement trajectory, the morphological readout as well as the exact realization of the underlying Markov chain. In case you want to use experimental data, you need to construct the cell-array accordingly. In particular, a valid cell-structure contains the fields:

1. *MeasurementTime:* The vector containing the measurement time points.
2. *Measurement:* The measured protein abundance at the measurement time points defined above.
3. *Simulated:* Zero if the data was measured experimentally and one if it was simulated.
4. *MorphologicalFeatureValue:* If morphological features are used, this field stores the value of the morphological feature.
5. *xPath:* Synthetic data only. Stores the exact species abundance of the underlying Markov chain.
6. *tPath:* Synthetic data only. Contains the jump times of the underlying Markov chain.
7. *rPath:* Synthetic data only. Contains the reaction indices that happened at the jump times.

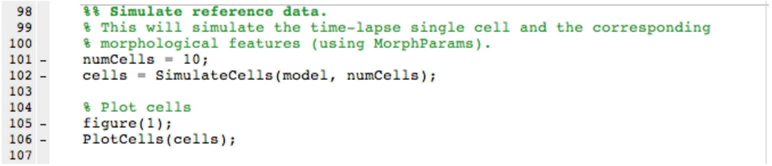
In the folder *PATH/ExampleData/* you find exemplary *.mat* files containing valid cell structures.

Once loaded, the cells can be visualized using the function *PlotCells* (see Figure 6)



```
%% Initialize state-space model
% Output-Weight matrix. The measurements are assumed to be a noisy readout
% of linear combinations of the state, i.e., Y ~ p(. | W'*X).
W = [0 0 0 1]'; %Set fourth species (i.e., protein) to be read out.

% For instance, if mRNA is measured as well, the Output-Weight matrix would
% be W = [0, 0, 1, 0;
%          0, 0, 0, 1];

% Specify the measurement time points. If experimental data is used, those
% must be extracted from the dataset.
MeasurementTime = linspace(0, T, 25);
MeasurementTime = MeasurementTime(2:end);

% Specify reactions rates which are to be estimated from the data. All other
% reaction rates will be set to the value set in the kineticModel
% structure.
targetReactionIndex = [2; 3; 4; 5];


% Specify prior parameters. Priors are assumed to be Gamma distributed,
% i.e., Gamma(a, b). Currently, only Gamma-type priors are supported.
% Further priors will be included in a future release.
% Hint: For the gamma distribution it holds that mean = a/b, var = a/b^2
% and CV = 1/sqrt(a). For highly informative priors 'a' needs to be large
% (and the other way round.
aPrior = [1; 3; 3; 3];
bPrior = [50; 4; 100; 300];

% Specify extrinsic reaction. Currently, only one extrinsic reactions is
% supported.
randomEffect = [0; 0; 0; 1];

% Specify hyperparameters for the extrinsic statistics. The hyperparameters
% correspond to a bivariate log-normal distribution LN(MuHyper, SigmaHyper).
MuHyper = [log(1);log(1000)];
SigmaHyper = [2, 0;
              0, 2];

% Specify hyperparameters for the morphological parameters. The
% hyperparameters correspond to a bivariate log-normal distribution
% LN(MuMorph, SigmaMorph). MorphParams will be ignored if the morphological
% parameters are estimated from the data. If MuMorph and SigmaMorph are
% empty, MorphParams will be used and assumed to be known in the inference.
% In this example, MorphParams will also be used to simulate the
% morpholical feature.
MorphParams = [5, 10000];
MuMorph = [log(10); log(5000)];
SigmaMorph = [3, 0; 0, 3];

% Specify the scaling parameter and the corresponding hyperparameters of
% the measurement noise model. Note that the scaling parameter will be
% ignored during inference but is used for generating the synthetic
% trajectories.
measurementParams = [0.15, 2, 0.03];
mDens = 'logn';

model = InitializeMEGSM(kineticModel, MeasurementTime, ...
    W, measurementParams, mDens, targetReactionIndex, randomEffect, ...
    aPrior, bPrior, MuHyper, SigmaHyper, MorphParams, ...
    MuMorph, SigmaMorph, @GetPieceWiseConstantInput, inputParams);
```

Figure 5: Constructing a general state-space model.



```
%% Simulate reference data.
% This will simulate the time-lapse single cell and the corresponding
% morphological features (using MorphParams).
numCells = 10;
cells = SimulateCells(model, numCells);

% Plot cells
figure(1);
PlotCells(cells);
```

Figure 6: Load / simulate experimental data.

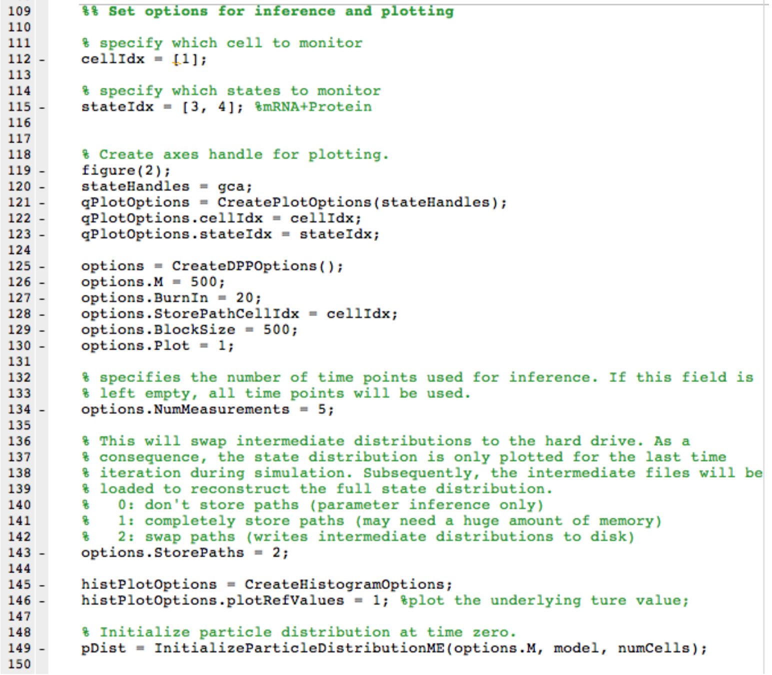
Set options for inference and plotting: DPP requires several properties to be specified before it can be run.There are three different structures associated with different properties of the algorithm.

The algorithm configuration is defined in the structure *options* (lines 125-143). A default options structure can be created using the function *CreateDPPOptions*. The structure consists of the following fields:

1. *M:* The number of particles used for inference.
2. *BurnIn:* The burn-in period of the MCMC sampler. During each time-iteration, the first K samples will be discarded, where K corresponds to the value of *BurnIn*.
3. *StorePathCellIdx:* Specifies the indices of the cells whose state trajectories shall be stored. Note that the required memory increases with the number of cells specified in *StorePathCellIdx*.
4. *BlockSize:* In order to save memory, the algorithm implements a block-processing mode, where one block of size *BlockSize* is processed at once.
5. *Plot:* *Plot=0* won't produce any monitoring plots. If *Plot=1*, figures with the most recent parameter and state distributions will be shown after each time-iteration.
6. *NumMeasurements:* By setting this variable, the algorithm will only consider the first *NumMeasurements* measurement time points. If the variable is not specified, all time points will be used.
7. *StorePathNumPoints:* Set to zero in order to store the full sample paths of the Markov chain. Set to a value greater than zero to down-sample the paths to the desired number of values.

The parameters for plotting the reconstructed state distributions are specified in the structure *qPlotOptions* (lines 119-123). A default options structure can be created using the function *CreatePlotOptions*. You can pass an array of Figures-handles to the function in order to plot the reconstruction into custom windows Otherwise, a figure is created automatically during runtime. The structure consists of the following fields:

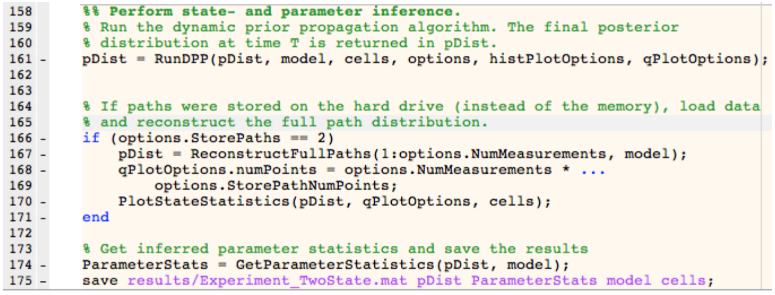
1. *quantile:* A vector containing two quantiles that are plotted as credible intervals. The default value is $[0.05, 0.95]$.
2. *plotReferenceCells:* Set to one if you wish to display the measured cells and to zero otherwise.
3. *stateIdx:* Specify a vector of indices of chemical species you want to visualize.
4. *cellIdx:* Provide the indices of the cells you want to visualize. Note that those each of those indices must also be provided in the *options.StorePathCellIdx* vector.
5. *FigureHandlesCells:* A set of figure handles, each of them corresponding to one of the cells in *cellIdx*. The reconstructed states will be plotted using the given handles.
6. *plotConfidence:* Specify whether you wish to visualize the credible intervals given in *quantile*.
7. *numPoints:* The number of sampling points along the full measurement time interval. Note that for large *numPoints*, plotting of the reconstructed paths may take a while.



```
109     %% Set options for inference and plotting
110
111     % specify which cell to monitor
112 -   cellIdx = [1];
113
114     % specify which states to monitor
115 -   stateIdx = [3, 4]; %mRNA+Protein
116
117
118     % Create axes handle for plotting.
119 -   figure(2);
120 -   stateHandles = gca;
121 -   qPlotOptions = CreatePlotOptions(stateHandles);
122 -   qPlotOptions.cellIdx = cellIdx;
123 -   qPlotOptions.stateIdx = stateIdx;
124
125 -   options = CreateDPPOptions();
126 -   options.M = 500;
127 -   options.BurnIn = 20;
128 -   options.StorePathCellIdx = cellIdx;
129 -   options.BlockSize = 500;
130 -   options.Plot = 1;
131
132     % specifies the number of time points used for inference. If this field is
133     % left empty, all time points will be used.
134 -   options.NumMeasurements = 5;
135
136     % This will swap intermediate distributions to the hard drive. As a
137     % consequence, the state distribution is only plotted for the last time
138     % iteration during simulation. Subsequently, the intermediate files will be
139     % loaded to reconstruct the full state distribution.
140     %   0: don't store paths (parameter inference only)
141     %   1: completely store paths (may need a huge amount of memory)
142     %   2: swap paths (writes intermediate distributions to disk)
143 -   options.StorePaths = 2;
144
145 -   histPlotOptions = CreateHistogramOptions;
146 -   histPlotOptions.plotRefValues = 1; %plot the underlying ture value;
147
148     % Initialize particle distribution at time zero.
149 -   pDist = InitializeParticleDistributionME(options.M, model, numCells);
150
```

Figure 7: Constructing a general state-space model.

Run inference: The main inference algorithm can be started using the function *RunDPP* (see Figure 8). The function is called using the previously defined model and options and returns the inferred posterior distribution in *pDist* (line 161). Note that the execution time of the algorithm strongly depends on the number of particles and cells. Subsequently, if the intermediate data were swapped to the hard-drive, the full posterior distribution over the paths is reconstructed and plotted (lines 166-171). You can then use the posterior distribution for further analysis. For instance you can use the function *GetParameterStatistics* to compute the prior and posterior statistics for the inferred parameters. The results are return in the structure *ParametersStats* (line 174).



```
158      %% Perform state- and parameter inference.
159      % Run the dynamic prior propagation algorithm. The final posterior
160      % distribution at time T is returned in pDist.
161 -    pDist = RunDPP(pDist, model, cells, options, histPlotOptions, qPlotOptions);
162
163
164      % If paths were stored on the hard drive (instead of the memory), load data
165      % and reconstruct the full path distribution.
166 -    if (options.StorePaths == 2)
167 -        pDist = ReconstructFullPaths(1:options.NumMeasurements, model);
168 -        qPlotOptions.numPoints = options.NumMeasurements * ...
169              options.StorePathNumPoints;
170 -        PlotStateStatistics(pDist, qPlotOptions, cells);
171 -    end
172
173      % Get inferred parameter statistics and save the results
174 -    ParameterStats = GetParameterStatistics(pDist, model);
175 -    save results/Experiment_TwoState.mat pDist ParameterStats model cells;
```

Figure 8: Run inference.

4 Practical Hints

4.1 Generating new models

You can easily add your own kinetic models to DPP and GuiDPP using a simple MATLAB script. Models are defined by a MATLAB structure *kineticModel* with the following fields

1. *Name:* A string specifying the name of the model. This name will be shown in the corresponding drop-down menu in GuiDPP .
2. *Pre:* A $m \times n$ matrix defining the left-hand-side of your reaction system (i.e., the reactants). m corresponds to the number of reactions and n to the number of chemical species.
3. *Post:* A $m \times n$ matrix defining the right-hand-side of your reaction system (i.e., the products). m corresponds to the number of reactions and n to the number of chemical species.
4. *S:* The stoichometry matrix defined by $Post - Pre$.
5. *c:* An $m \times 1$ vector containing nominal rate constants for each reaction.
6. *X0:* An $n \times 1$ vector containing the initial conditions of the reaction system.
7. *NumSpecies:* The number of species n.
8. *NumReactions:* The number of reactions m.
9. *SpeciesNames:* A $n \times 1$ list of strings containing the species names. Note that if used with GuiDPP , the species *mRNA* and *Protein* need to be part of the model.
10. *ReactionNames:* A $m \times 1$ list of reaction names.
11. *InputRateIdx:* The index of the reaction that is modulated by an external stimulus. This field is required even if no (i.e., a constant) input is used.

Once the model has been created in MATLAB, save it to the *PATH/Models/* folder using the MATLAB command *save*. Please consult the example scripts *CreateSimpleModel.m* and *CreateTwoStateModel.m* in the folder *PATH/Models/* for further information.

4.2 Static time-delays between induction and expression

In many applications involving fluorescent reporter proteins, a static time-delay between induction and expression is observed. While such delays can in principle be achieved by including multiple sequential first-order events in the kinetic model, the resulting inference problems becomes challenging due to the increased parameter- and state-space. A simple and pragmatic approach is to detect the time-point at which the increase of the protein abundance is statistically significant and to discard the measurements before that time-point. For instance, a one-sided Kolmogorov-Smirnoff test can be performed to detect that time-point at a given significance level [1].

4.3 Sample degeneracy

DPP and GuiDPP rely on a sequential Monte-Carlo inference scheme, which - like all sampling-based methods - produce reasonable results only if *enough* samples (i.e., particles) are used. It is generally hard to provide guidelines on how to choose an appropriate number of particles since that number strongly depends on several factors such as the system dynamics, the prior knowledge or the measurement noise model – to name a few. However, there are certain indicators that may tell you if the number of particles is too small. For instance, sequential Monte-Carlo procedures will in such case yield *degenerate* solutions, i.e., the final distribution consists of only very few different particles. An exemplary degenerate posterior distribution is shown in Figure 9. It can be seen that the for the early time points, the posterior distribution consists of only a single sample path, because all other particles were lost during the subsequent time iterations. Be aware that very narrow posterior distributions do not necessarily mean accurate inference results. In many cases they rather indicate sample degeneracy such that the number of particles must be increased.

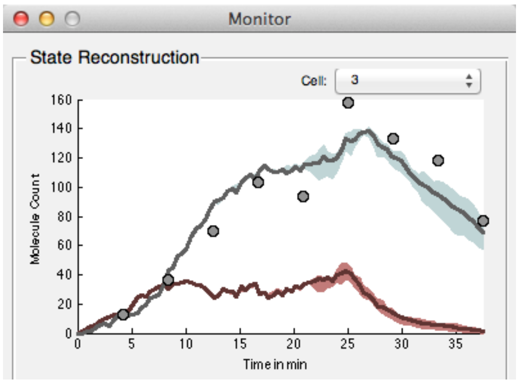


Figure 9: Example of a degenerate posterior distribution.

4.4 Choosing your priors

In order to run DPP you have to provide prior distributions over the unknown parameters. According to the Bayesian philosophy, such priors should not be considered as tuning variables, but should rather reflect what is known about the parameters a priori. Nevertheless, the algorithm performance strongly depends on such prior knowledge which should therefore be incorporated wherever possible. For instance, rate constants are generally constrained by biophysical laws. While exact limits are typically unknown, it is often straight-forward to roughly define plausible ranges. As discussed above, DPP assumes Gamma-type prior distributions over the kinetic parameters. On the one hand, the Gamma distribution turns out to be very useful for biological applications since it represents a very versatile distribution for the positive orthant - ranging from highly over-dispersed and right-tailed to under-dispersed and symmetric distributions. On the other hand, the Gamma distribution has convenient mathematical properties which allow to run DPP without approximations.

The Gamma distribution is fully specified by a shape parameter a and a scaling parameter b. Since those values are generally hard to interpret, it is more convenient to transform them into a set of standard statistics. For instance, mean and variance of a Gamma distribution are given by $m = \frac{a}{b}$ an $s = \frac{a}{b^2}$ respectively. Furthermore, the coefficient of variation (CV) is given by $CV = \frac{1}{\sqrt{a}}$. Hence, if a parameter is well-known a-priori, the variance (and CV) will take very small values. In contrast, if only few information is available, the variance and CV will be large. Hence, for a given mean and variance (or CV) you can specify a and b by

$$a = \frac{m^2}{s} \qquad b = \frac{m}{s} \tag{2}$$

or

$$a = \frac{1}{CV^2} \qquad b = \frac{1}{CV^2 m}. \tag{3}$$

References

[1] C Zechner, M Unger, S Pelet, M Peter, and H Koeppl (2014). Scalable inference of heterogeneous reaction kinetics from pooled single-cell recordings. Nature Methods. In press.