DiPS Tutorial

October 15, 2020

Contents

Introduction

In the first part of this tutorial, we describe the components of the Graphical User Interface (GUI), its six tabs, in more detail.

Secondly, we introduce a running example. We use parametric discrete-time Markov chain (pMC) population model of stinging behaviour of two bees and Probabilistic Computation Tree Logic (PCTL) [4] properties of reaching the respective number of stinging - zero, one, or two - as presented in [3]. We reuse presented data point representing the experimentally observed frequency of respective property - 0.04, 0.02, 0.94.

Next, we show the main workflow of DiPS using the running example. It consists of nine use-cases introducing separate functionality of each tab of GUI:

1. Load model and properties
2. Synthesise parameters
3. Load data
4. Optimise parameters
5. Sample functions
6. Calculate constraints
7. Sample space
8. Refine space
9. Metropolis-Hastings

with a discussion on the obtained results.

Further, ideas how to combine respective methods are drawn. Finally, we present visualisations for models with more than two parameters.

Graphical User Interface

The GUI is organised in six tabs providing respective functionality.

1 - Model & properties tab allows to load and view pMC model in `.pm` format and PCTL properties in `.pctl` format.

2 - Synthesise functions tab serves to run the parameter synthesis of selected model and properties. PRISM [5] and Storm [1] output file as a result of parameter synthesis are loaded automatically. For each property, a symbolic representation of its satisfaction in the form of a rational function is read by the *parser*. Rational functions can be factorised, saved as a pickled list in `.p` format, and loaded. Note that more general functions can be applied here.

3 - Sample functions tab visualises synthesised/loaded functions in a given parameter point or one-by-one point of the sampled grid as a barplot. In the presence of data, the respective data value is visualised along with the value of the function. If the data intervals are present as well, their values are visualised as error bars of the respective data bar.

4 - In the *Data & Intervals* tab, data can be loaded, edited, and saved. At this point, the distance of synthesised/loaded functions and data can be optimised. The results, optimised parameter point, respective function values, and the distance, are shown as text and can be saved to a file. Data intervals are computed, loaded, saved, and shown in this tab. Moreover, data-informed properties, which can be again used in PRISM or Storm, are shown after intervals are computed.

5 - In the *Constraints* tab, the inequalities combining the rational functions and lower/upper bounds of the intervals are computed. List of constraints can also be loaded or saved as a pickled list. Note that more general constraints can be applied here.

6 - Sample & Refine space tab is the main results part in which the results of space sampling, space refinement (Figure 2), and Metropolis-Hastings inference are shown (Figure 3). The results can be saved and later loaded. A textual representation of the obtained space (Figure 1) is updated after

sampling or refinements finishes or after loading space. Features and settings are provided with hover over help text.

Above the tabs frame, GUI shows loaded files. GUI provides autosave of results in `tmp` folder. The *configuration file* - `config.ini` sets the paths for external tools, input file, output files, and settings for analysis.

DiPS uses console to provide you with intermediate results or more detailed progress info, although all necessary information can be perceived via GUI.

```
params: ['p', 'q']
region: [[0.0, 1.0], [0.0, 1.0]]
types: ['Real', 'Real']
coverage: 0
rectangles_sat: []
rectangles_unsat: []
rectangles_unknown: [[[0.0, 1.0], [0.0, 1.0]]]
sat_samples: [[0.7777777777777777, 0.8888888888888888], [0.777777777777
7777, 1.0], [0.8888888888888888, 0.7777777777777777], [0.88888888888888
88, 0.8888888888888888], [0.8888888888888888, 1.0]]
unsat_samples: [[0.0, 0.0], [0.0, 0.1111111111111111], [0.0, 0.2222222222222
222], [0.0, 0.3333333333333333], [0.0, 0.4444444444444444]] ... 90 more
quantitative_samples: {}
true_point: None
```

Figure 1: Textual representation of space.

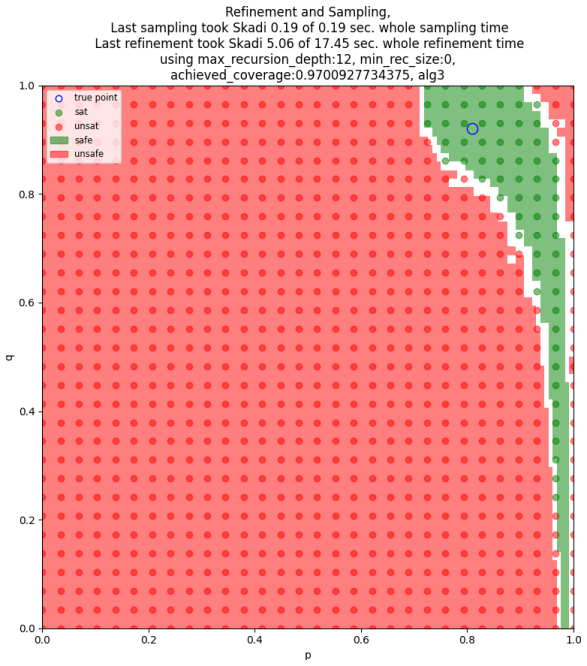

Figure 2: Space sampling and refinement visualised.

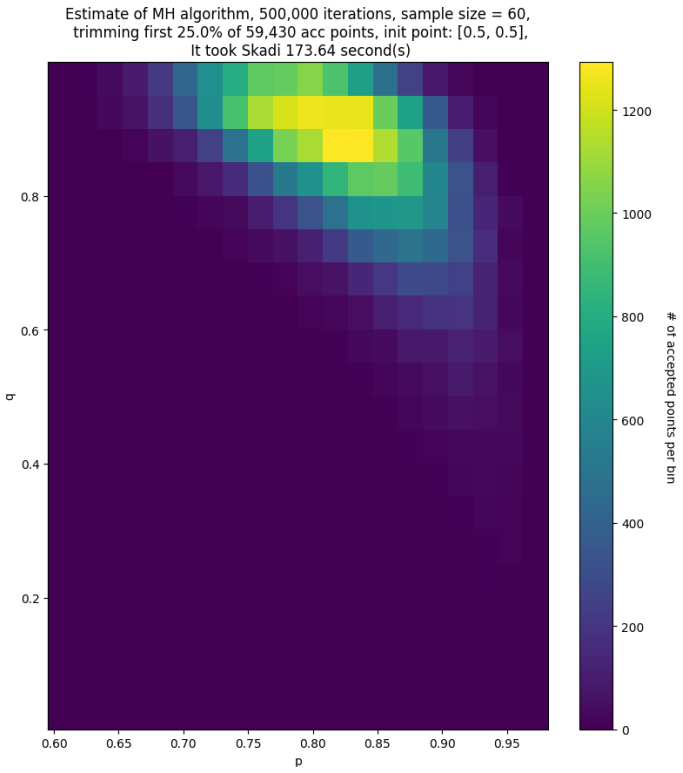

Figure 3: Metropolis-Hastings results visualised.

Explaining the running example

To help to understand the results using the provided example of honeybee stinging [3] we briefly explain the model, properties, and data here - for more detail explanation, please follow the paper [3]. For the sake of a simple presentation of the tool functionality, we use this trivial model of two bees for this purpose. Moreover, for the purpose of discussion of the results, we use synthetic data in this tutorial.

Model

Honeybees protect their hive against an attack by mass stinging. A stinging bee releases an alarm pheromone promoting other bees to sting. We created a pMC population model to track the number of stinging in a population of n bees. With discrete time we do not argue about the exact timing of an individual stinging rather the order of them. In each state of the model a state of each bee (see a_i) is tracked: -1 initial - before the attack, 1 stinging, 2 a bee decided not to sting before smelling the alarm pheromone, and 0 a bee not stinging even regarding the alarm pheromone. b is a flag indicating a BSCC state in the model (a state in which all the bees decided to sting). We do not distinguish individuals; hence we lump the states with the same population up to permutation together. We consider uniform probability of a bee to sting equal to p when no alarm pheromone present, q when alarm pheromone present conditioned by that the bee would not sting without the alarm pheromone being present. p and q are the only parameters in the pMC model.

Properties

In a population of n bees ($n = 2$ in our running example) each bee can decide to sting or not. Counting the number of stinging bees in steady state (each bee already decided) one can observe $n + 1$ outcomes - $0, 1, \ldots, n$ stinging. We encode the outcomes with $n + 1$ PCTL formulae [4], querying for the respective number of stinging:

$$
\begin{aligned}
&P_{=?}[F(a_0 = 0)\&(a_1 = 0)\& \ldots \&(a_n = 0)\&(b = 1)] \\
&P_{=?}[F(a_0 = 1)\&(a_1 = 0)\& \ldots \&(a_n = 0)\&(b = 1)] \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots \\
&P_{=?}[F(a_0 = 1)\&(a_1 = 1)\& \ldots \&(a_n = 1)\&(b = 1)]
\end{aligned}
$$

where $P =?$ is the probability operator asking what is the probability of a CTL formula in square brackets, F is future operator asking for reachability, in our case a steady-state ($b = 1$) in which the count of $a_i = 1$ encodes the number of stinging bees. Note that one can use FG without the necessity of encoding BSCC/steady state. We opted for this for easy encoding of rewards of reaching respective BSCC in the model.

Data

In each run of the real experiment, a population of n bees is introduced to a visual-tactile stimulus of a rotating feather. The number of stinging bees is written down. This is repeated N times. The output is the list of frequencies of respective numbers of stinging bees in all runs.

For comparison of the results, we have synthesised the data, by running the model with parametrisation p = 0.81; q = 0.92. The data have been obtained from 100 runs, in which we observe the event of no stings 4 times, one of two bees stinging twice, and 94 times both of the bees are stinging. Hence we obtain data list $[4/100, 2/100, 94/100] = [0.04, 0.02, 0.94]$ which is in standard setting[1], used as the experimental estimation of probability of satisfaction of the respective property.

[1] using experimental data instead of synthetic

First use in 9 steps

0. For detailed instructions on how to download, install, set-up, and run DiPS follow the README - `https://github.com/xhajnal/DiPS`.

1. When you open DiPS GUI, an autoload window pops-out.

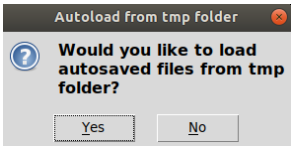

Figure 4: Load automatically saved files.

As we do not have any saved files yet, press `No`. As you run an analysis, we automatically store this result in *tmp* folder. This feature will allow you to reload progress where you have left it. Let's make these files.

As the first of all tabs, you will see - Model and properties - tab. Click the `Open model` button and select *example/semisyn_2_bees.pm* file to load the pMC model in PRISM format. Then click the `Open properties` button and select *example/prop_2_bees.pctl* file to load the PCTL properties. Loaded model is shown on the left text box and loaded properties on the right text box.

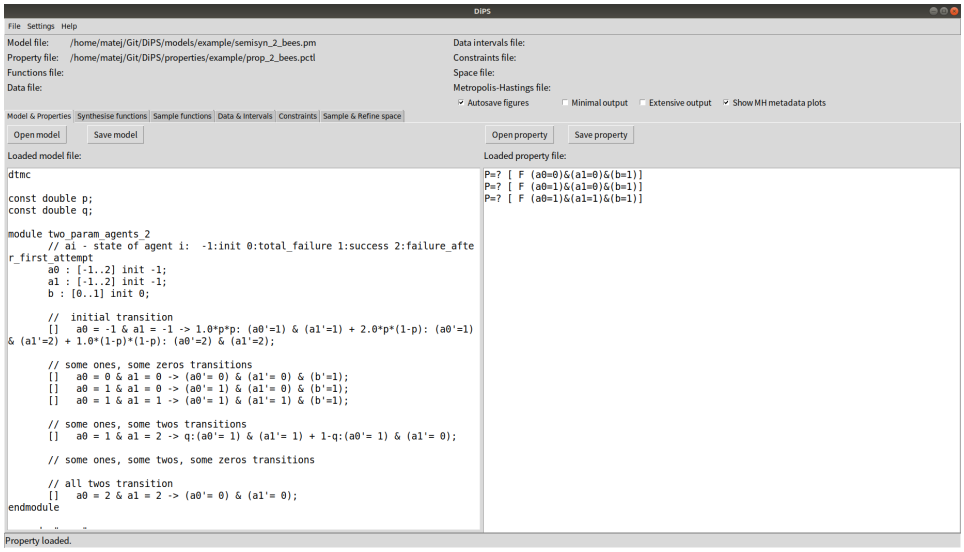

Figure 5: Load model and properties.

2. After the model and properties are loaded, move to the second tab, *Synthesise functions*. Here you can run verify whether the model satisfies the given properties. The standard parameter synthesis procedures provide a symbolic representation of satisfaction probability in the form of rational functions over model parameters. We leverage existing tools, PRISM or Storm. In this case, we use PRISM. To read more, you can follow `https://www.prismmodelchecker.org/manual/RunningPRISM/ParametricModelChecking`.

   To simplify the result rational functions and hence to improve performance of the further analysis click on `Factorised` Radiobutton in `Show function(s)` section (right). To launch parameter synthesis itself click on the `Run parameter synthesis` button (left). In the appeared dialog (Fig 6), press `OK` to select default bounds of the respective model parameter - $[[0, 1], [0, 1]]$.

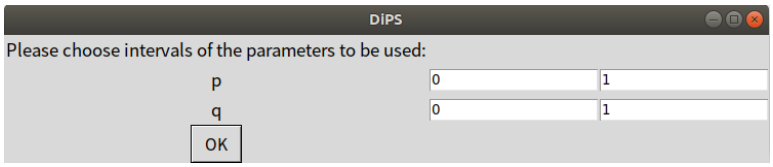

Figure 6: Select model parameter bounds.

(Note:) These parameter bounds will be used for further analysis.

When finished, PRISM output file is shown on the left text box and rational functions parsed from the output on the right text box:

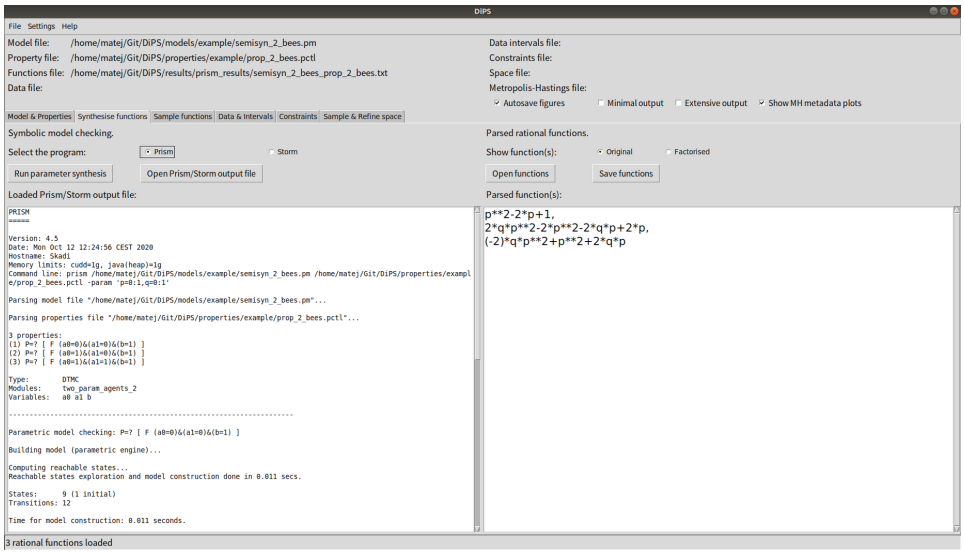

Figure 7: PRISM output (left) and parsed rational functions (right).

(Tip:) To skip the parameter synthesis step next time you open DiPS, you can save the rational functions.

(Note:) With large models, PRISM can run out of memory easily or DiPS can have problems to simplify the output. In this case, you can run parameter synthesis with Storm manually and load the result `.txt` file. To obtain commands on how to call Storm, choose Storm in the `Select the program` section and press `Run parameter synthesis` button. The commands will appear on the left text box.

3. Now proceed to Tab 4, *Data & Intervals* (skipping the third one). Here press `Open data file` button and select *example/data_n=2.csv* file[2] to load data. Loaded data are shown in the upper left text box.

   To compute the data intervals, we need to set two parameters: number of samples and confidence level. As the running example observations consist of hundred runs, we set `Number of samples` textbox to 100 and let us use confidence level 0.95 by setting `C, confidence level` textbox. Now press `Compute intervals` button (in left bottom part). Computed intervals are shown in the lower-left text box. The upper right text box shows loaded properties. Now you have loaded all necessary inputs.

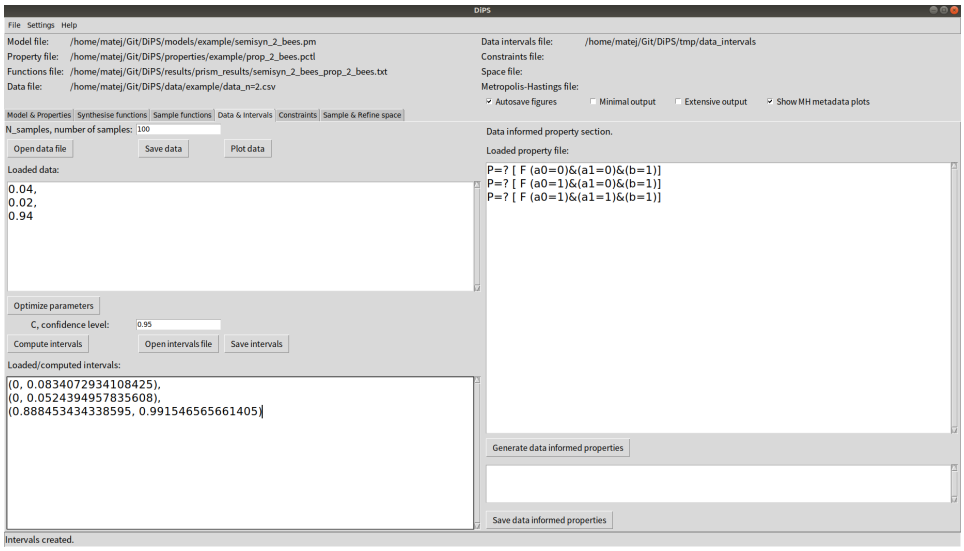

Figure 8: Loaded data (left upper), respective property (right upper), computed intervals (left bottom) with a selected sample size and confidence level.

   In the case you want to use data estimating expected value of reward properties, please edit the Interval textbox manually using your own intervals.

[2] to see .csv file in explorer, please filter .csv or all files

To visualise individual data points with the respective intervals click on the `Plot data` button. A windows containing a bar plot with error bars will be displayed:

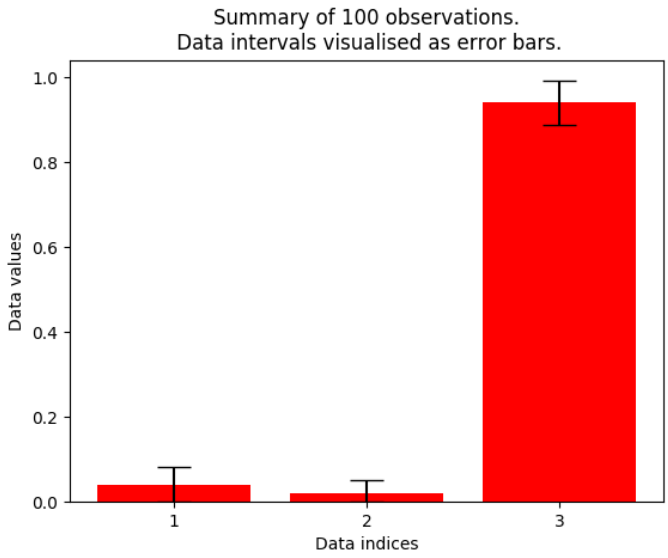

Figure 9: Individual data value as bar plot with respective data interval shown as error bar.

4. While still on Tab 4, *Data & Intervals*, you can optimize parameters of the rational functions for minimal distance to data. Click on `Optimize parameters` button.

   After the optimisation is complete, another dialog showing the results: parameter values, values of rational functions in this point, and the distance between functions and data pops out:

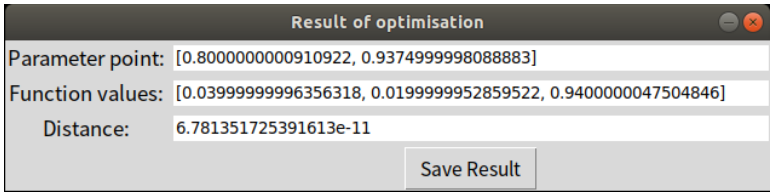

Figure 10: Optimisation result, parameter points, function values, and distance.

(Tip:) You can store the results by clicking on the `Save Results` button.

5. After finishing step 3, you can compare values of computed rational functions with the loaded data. To do so, move to the third tab, *Sample functions*, and click on the `Plot functions in the given point` button. A dialog asking for the parameter point to show the functions in appears. To use the default point [0.5,0.5] press `OK`.

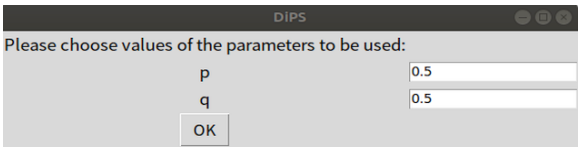

Figure 11: Select parameter point to be visualised.

A plot showing the respective values is visualised on the right:

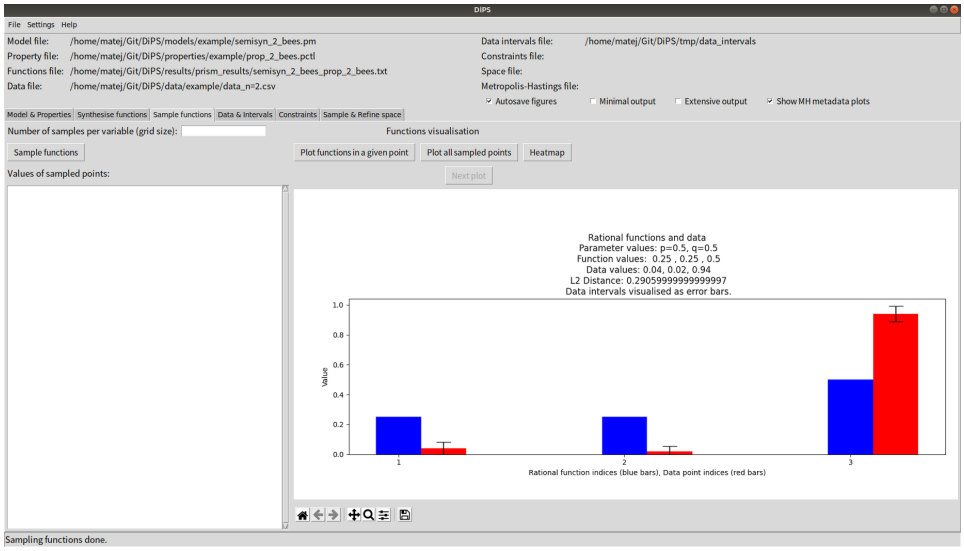

Figure 12: Sample functions in point [0.5, 0.5] comparing with data and intervals. Blue bars show the values of an individual rational function, red bars show the respective data value, and the error bars show the interval values computed from the data. In the title, L2 distance of rational function in the given point and data is computed, ≈ 0.2906.

(Tip:) You can sample functions by selecting the grid size (upper left Textbox) and clicking on the `Sample functions` button (bellow it).

(Tip:) After you select the grid size (upper left Textbox), you can visualise the functions value in the sampled points by clicking on the `Plot all sampled points` button. To move between the points press `Next plot` button.

(Tip:) In the case of 1 or 2 parameters, you can visualise respective functions values as a heatmap in the sampled points. To move between the functions press `Next plot` button.

6. Before you run space sampling or space refinement combine rational functions and intervals to create constraints. Go to Tab 5, *Constraints*, and press `Calculate constraints` button (in the left upper part) to see the results displayed in the text box below:

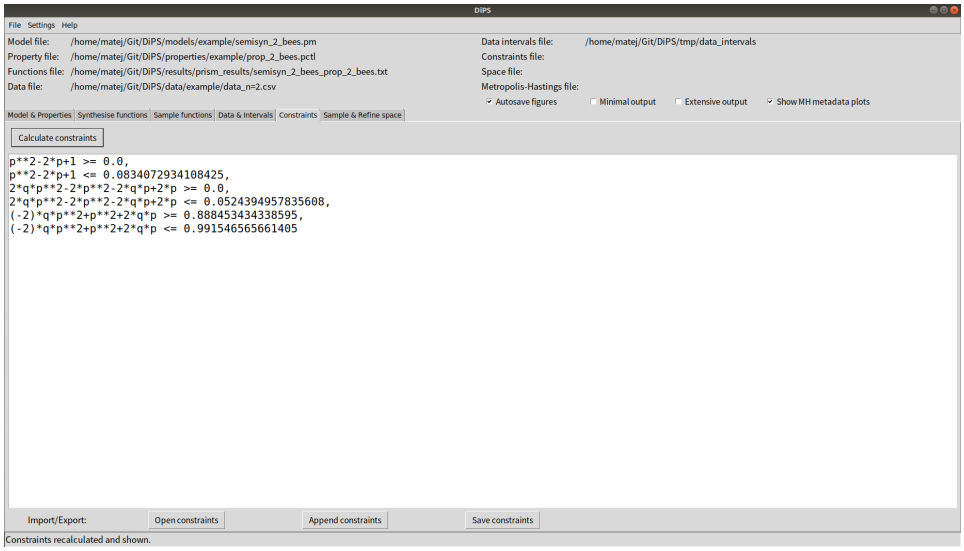

Figure 13: Calculate constraints from the functions and data intervals.

In each pair of lines, there is an inequality created as a coupling of the respective rational function and data interval. The first inequality of a pair indicates that the rational function is greater or equal than the lower bound of the respective interval, while the second inequality indicates that the rational function is lower or equal than the upper bound of the interval.

(Note:) You can either edit calculated constraints or load arbitrary ones. You can use these constraints in the next step to sample or refine space. Metropolis-Hastings is using the functions and data, hence will not respond to change of constraints.

7. Now we are ready to run the rest of the analysis. We move to the last tab, *Sample & Refine Space*. To search for points in parameter space which are within the calculated intervals select `Grid size` textbox, number of samples in each dimension, to e.g. 10 and press `Grid sampling` button (on the first column of the left side). The result is shown in the upper figure on the right:

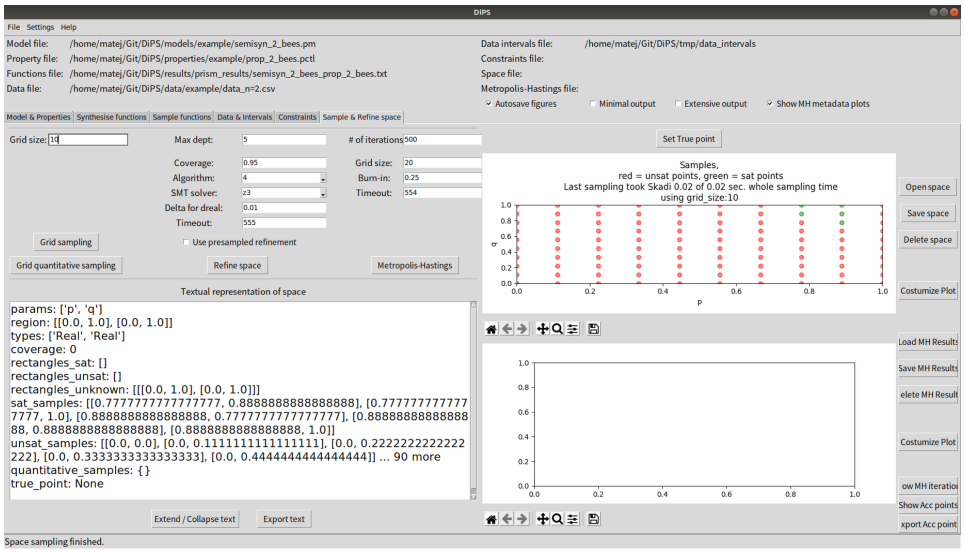

Figure 14: Results of space sampling. Visualisation (right upper) - a phase space with a grid of points, where the green points represent sat points, the parameter points which satisfies all the constraints (calculated in the previous step). The red points represent unsat parameter points for which at least one on the constraints is not satisfied. Textual representation (left below) with the list of sat and unsat points.

(Note: Experimental feature) We are working on the quantitative sampling, one button bellow `Grid Sampling`, which provides the sum of L1 distance to dissatisfy the constraints. Currently, this feature may provide bad results or cause an unhandled error.

(Tip:) Features and settings are provided with over about help text.

8. Moreover, you can obtain global results by clicking on the `Refine Space` button (in the middle of the left side). The result is shown in the same figure on the upper right side:

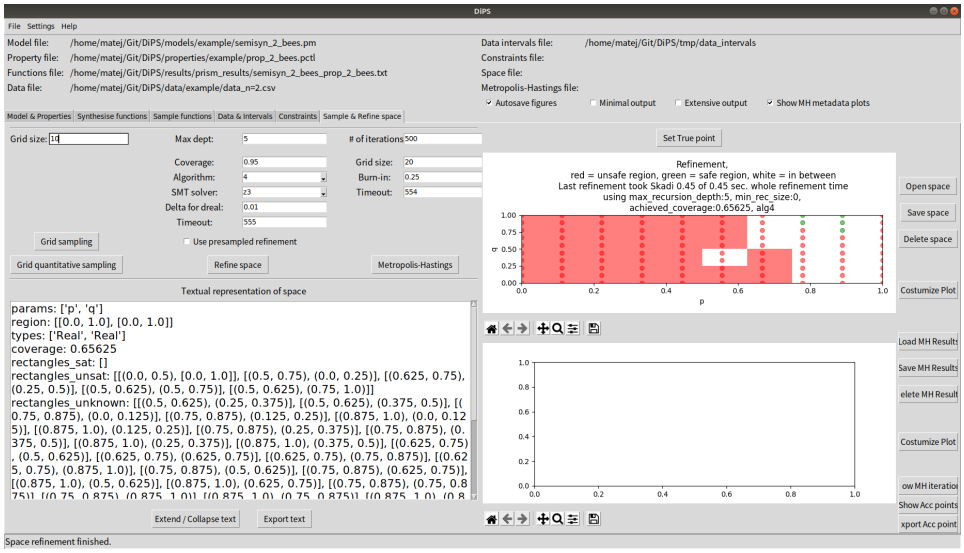

Figure 15: Results of space refinement. Visualisation (right upper). Textual representation (left below) with a list of safe, unsafe, and unknown rectangles.

Space refinement partitions the parameter space in a CEGAR-like manner into 3 types of subregions: *safe*, *unsafe*, and *unknown*. While for each point in:

- safe region all the constraints are satisfied,
- unsafe region at least one of the constraints are **not** satisfied,

Unknown regions, visualised by the white colour, are either not checked (eg. the whole space before refinement) or contains both, a sat and an unsat point. In the later case, the subregion has to be split[3].

This process of splitting and checking continues until *coverage*, proportion of nonwhite space, *maximum depth* of recursion of splitting[4], or *timeout* is reached. One can change the coverage by choosing the value

[3] we split region into two size-equal regions using the longest axis
[4] how many times a single rectangle can be split

of `Coverage` textbox, maximum depth by altering `Max Depth` textbox, and timeout by changing `Timeout` textbox[5].

You can start refinement again from this point by simply clicking on the `Refine Space` button again:

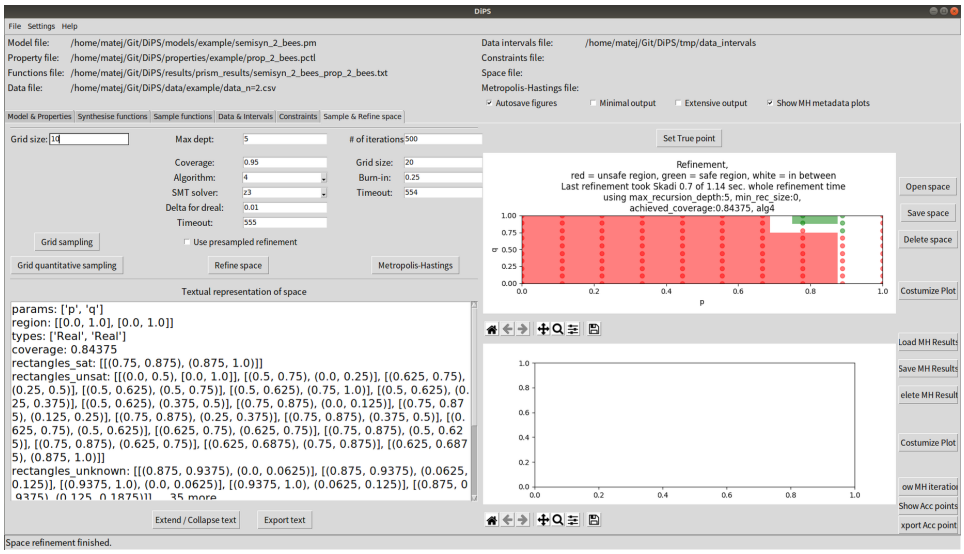

Figure 16: Updated results of space refinement obtained by clicking on the `Refine space` button for the second time.

Solver checking the safety/unsafety of individual hyperrectangle can be chosen within `Algorithm` SelectionBox: for SMT solver, Z3 [6]/dreal[6] [2], choose 1-4 and for interval arithmetic, using `mpmath` library, choose 5. You can choose the way how to select the next region to be checked by SMT solver. Pick algorithm 1-4, where:

- 1 implements depth-first search, where the region which has been just split is being checked until the max recursion depth is reached,
- 2 implements level-order traversal, which provides splitting the subregions with the bigger size first,
- 3 implements level-order traversal, with passing information in which of the two subregions the sat point (a point satisfying the constraints) is located,

[5] set 0 for no timeout
[6] dreal is not available for Windows

- 4 implements level-order traversal, with passing the location of both, sat and unsat point.

We strongly recommend using algorithm 4 or algorithm 5 for the best performance.

(Note:) For problems which are difficult for z3 or interval arithmetic, one can use dreal solver by choosing it in the `SMT solver` SelectionBox. Dreal provides δ-complete decision procedure while the δ value can be chosen in `Delta for dreal` textbox.

(Tip:) To speed up the computation and neglect the partial results of refinement printed in the command line you can click on the `Minimal output` checkbox which is located over the 6 tabs on the right side.

(Tip:) To show a single point in 2D space as a result of optimisation, Metropolis-Hastings or the actual true point you can click on the `Set true point` button and input the desired point - see the result as a blue circle in Figure 2.

(Tip:) Features and settings are provided with hover over help text.

9. In the last part of this tutorial, we present Metropolis-Hastings. It employs Markov chain Monte Carlo approach of sampling parameter space by walking in it for a given number of iterations (set in `# of iterations` textbox). While the Bayesian inference is used to decide whether to accept or to reject proposed new point wrt. to data, D. In more detail, in each iteration, the decision is made by comparing the posterior probability of current and new point, θ and θ' respectively.

$$P(\theta' \mid D) > P(\theta \mid D) \tag{1}$$

Using Bayes rule, we can rewrite the inequality into:

$$\frac{P(D \mid \theta') \cdot P(\theta')}{P(D)} > \frac{P(D \mid \theta) \cdot P(\theta)}{P(D)} \tag{2}$$

As the probability of the data is the same on both sides, we can strike it out. Moreover, as we currently support the basic case of no prior knowledge of parameter values implemented by uniform distribution, which is equal in each point, we can strike that out as well:

$$P(D \mid \theta') > P(D \mid \theta) \tag{3}$$

In the end, we compare the likelihoods. If the likelihood of the new point θ' is greater than the likelihood of current point θ we accept θ'. Otherwise, there is a small probability of accepting the point anyway:

$$P(D \mid \theta') - P(D \mid \theta) > rand.unif(0, 1) \tag{4}$$

Finally, the set of accepted points serves as an approximation of the posterior distribution over model parametrisations wrt. available data.

To run the method press `Metropolis-Hastings` button (on the right of left side). A dialogue asking for a point, in the parameter space from which the search starts, appears. Input a desired point and press OK.

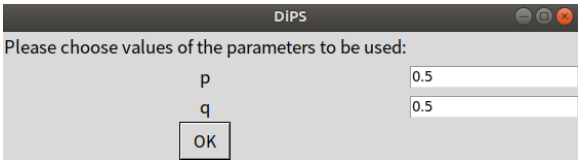

Figure 17: Choose the initial point for Metropolis-Hastings.

After Metropolis-Hastings finishes, the final result is shown as a figure in the right bottom part (with a zoom on it - Figure 3):

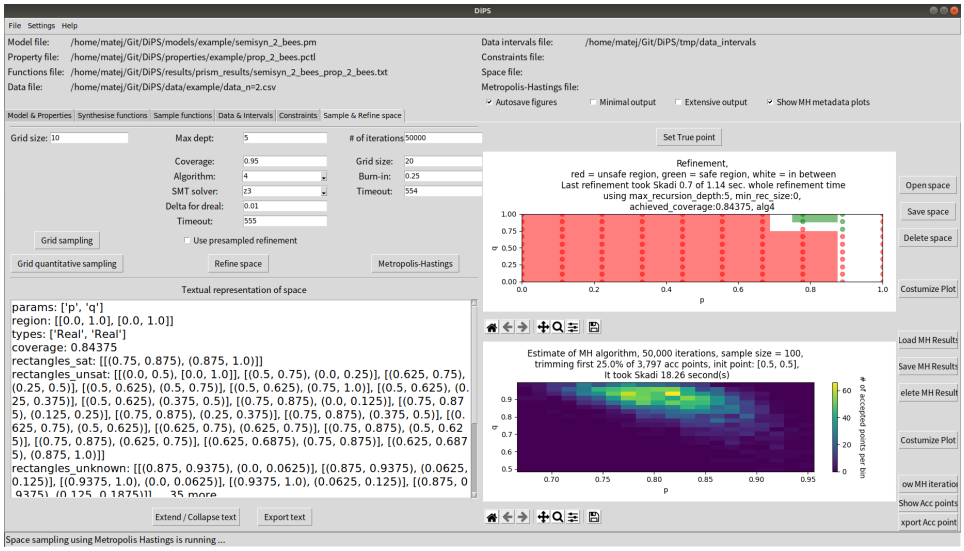

Figure 18: Metropolis-Hastings result: heatmap (lower plot). The colour gradient black-blue-green-yellow visualises the number of accepted points in each bin of heatmap. The label on the right quantifies this gradient.

where the set of accepted points is visualised by putting it into a respective bin. While the number of bins per dimension to be created can be set in `Grid size` textbox in Metropolis-Hastings settings. The grid size of a present plot can be altered by clicking on the `Customize plot` button to the right of the plot.

Before the method converges to the true distribution, the set of accepted points is more dependent on the initial point. This phase is called *burn-in* period and a proportion of accepted points can be trimmed from the beginning by setting `Burn-in` textbox to the desired proportion. The burn-in period is altered by clicking on the `Customize plot` button to the right of the result plot.

(Tip:) Similarly to Space Refinement, a timeout option can be set by altering `Timeout` textbox[7].

[7] set 0 for no timeout

Including this result, two windows with meta-plots pop-out:

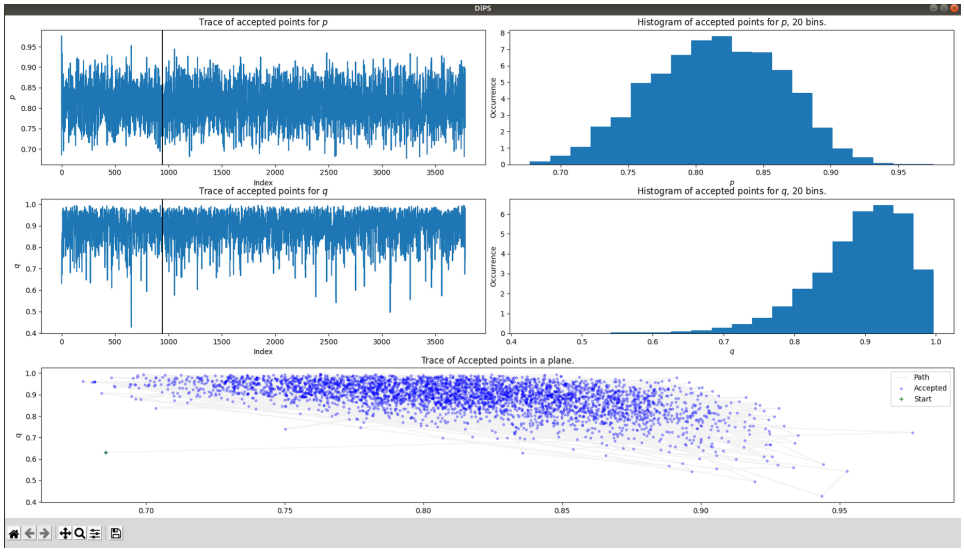

Figure 19: Metropolis-Hastings meta result: trace of accepted points per parameter (left) - for each iteration on the x-axis, the value of the parameter on the y-axis of the respective plot, with the vertical black line showing burn-in period - section of accepted points to be discarded. Marginal histogram projected onto the respective parameter (right). The whole trace of accepted points in phase space (below) - available in case of up to two parameters.

(Note:) Based on this plot, you can visually adjust the burn-in period. E.g. if the trace is still not "converged" after the threshold (the black vertical line) probably the burn-in period is set too short. If the trace is converged long before the threshold, the burn-in period is probably set too long. However, this plot does not provide the information that the trace itself converged within the given number of iterations. You can do this by comparing multiple plots, preferably using different initial points.

(Tip:) To speed up computation, you can select to skip the metaplots you can check out `Show MH metadata plots` checkbox which is located over the 6 tabs on the right side.

In the second plot, we add information about the rejected points:

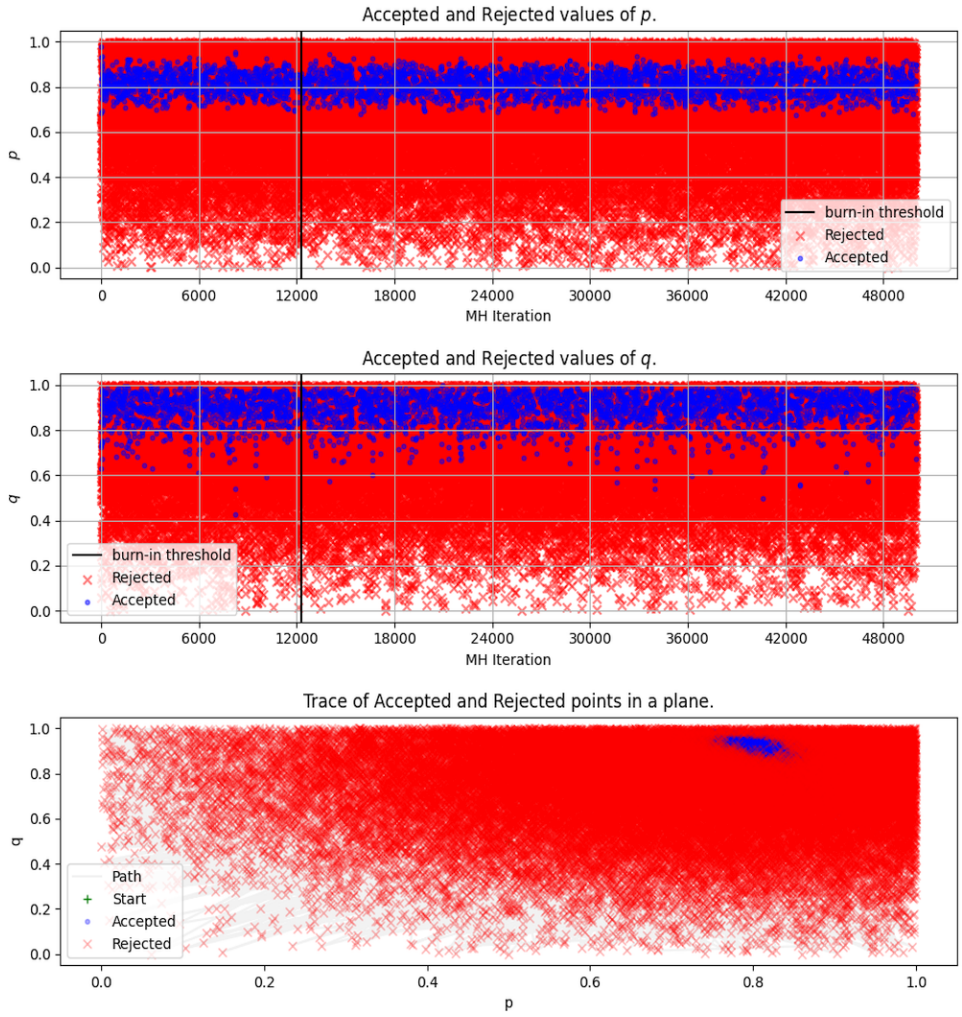

Figure 20: Metropolis-Hastings meta result: trace of both, accepted (blue) and rejected (red), points per parameter (left) - for each iteration on the x-axis, the value of the parameter on the y-axis of the respective plot, with the vertical black line showing burn-in period - section of accepted points to be discarded. The last plot, available in case of up to two parameters, is the whole trace of accepted points in phase space.

Explaining the results

DiPS provides four results - *optimization*, *space sampling*, *Metropolis-Hastings*, and *space refinement*. Here we explain and compare the results obtained from our running example - for further information please follow [3]. Note that we have obtained the data synthetically by running the model with the parametrisation p = 0.81; q = 0.92.

- *Optimizing Parameters.* In point 4 we have used the raw data to minimize the distance of the rational function to the data. More precisely, we search parameter space for a point in which the rational function has minimal (L2) distance to the data. The result is a single point in parameter space - which is $p, q \approx 0.8, 0.9375$ in our example. Evaluating the rational functions in this point we obtain $[0.0399999, 0.0199999, 0.940000]$ with very low distance from the data $[0.04, 0.02, 0.94]$ equal to $6.78e-11$, and close value to the true point.

- *Space Sampling.* As stochastic systems are intrinsically not deterministic, we cannot consider a limited number of observations as exact a precise estimation. Hence we create confidence intervals around each data point - in our example data point $[0.04, 0.02, 0.94]$ results with confidence intervals $\approx [0, 0.089945]$,$[0, 0.058066]$,$[0.881236, 0.99876]$ using confidence level 95% and 100 samples. Now we can sample parameter space and check whether a function (a symbolic result of model checking of the respective property) in the sampled point is within the respective interval. If all the functions are evaluated within the respective interval, ($p^2 - 2p + 1 \in [0, 0.089945], 2qp^2 - 2p^2 - 2qp + 2p \in [0, 0.058066], -2qp^2 + p^2 + 2qp \in [0.881236, 0.99876]$), the point in the parameter space is visualised green, otherwise red.

  Few points around the true point p = 0.81; q = 0.92 were marked sat. Besides that, we have partial information about the landscape of satisfaction of constraints in other points. We can improve this estimate by increasing the grid size or running later methods.

- *Metropolis-Hastings.* To run a more sophisticated search in parameter space one can use this option. As we jump in the parameter space, we estimate the likelihood of the points to observe the data. It drives the search to points with higher likelihood by comparing current and new point in each jump. If the point exceeds the threshold, the point is being accepted. We jump in parameter space for 50 000 iterations and then we discard the first 25 % of the accepted points (burn-in period). Parameter space is divided into a grid and each accepted point is assigned to its rectangle. By visual comparison, the posterior distribution converged to area close to the true point. Especially close is the most yellow rectangle, which is around the point p,q = 0.825,0.92. Besides this result, we have obtained the distribution predicting the landscape quantitatively.

  To obtain better results, one can simply increase the number of iterations or run space refinement for global results. Note that a part of this method is sampling from distributions and hence is not deterministic. Each finite run will provide different results, although converging to a single distribution.

- *Space Refinement.* Here we used the same constraints of the function based on confidence intervals of data as we have used in the sampling of the space. Global results in the coloured (red and green) subregions are provided in the cost of higher computational time.

  In the first refinement, we did not obtain any safe space and the true point remained in the white, unknown, area. Hence we can continue refining space (white regions) by simply pressing `Refine Space` button again. We have obtained a green area enclosing the true point p = 0.81; q = 0.92. Besides that, we have obtained global results on the landscape of satisfaction of the constraints. To obtain better results, one can continue refining the space until sufficient coverage is reached, such as provided in Figure 2.

Combining methods

In the previous chapter, we have shown four different methods for data-informed parameter synthesis. Each of them has different computational demand. In practice, you can use the results of a faster method to tweak the setting of a more demanding method, e.g.:

- Optimisation with Metropolis-Hastings.
  Instead of trimming the first iterations of Metropolis-Hastings you can start in the point obtained by the optimisation.
- Sampling and Refinement.
  - Presampled refinement. We have implemented refinement which starts with the splitting of the space based on the previous sampling results.
  - By sampling, you can obtain area which probably does not satisfy constraints as only unsat points are in this area. Hence, you can focus refinement in the other area of sat points or on the area in between. Simply delete the space and start refinement in the desired area.
  - Moreover, you can use sampling after refinement. When obtained narrow area of white space which is difficult to further refine you can delete space and start sampling in this are.
- Sampling and Metropolis-Hastings.
  Similarly, as in the sampling followed by refinement, you can bound search of Metropolis-Hastings to the area which was observed by the previous sampling, focusing on the unsat, sat, or the area in between.
- Metropolis-Hastings and Refinement.
  To select subregion of parameter space to start refinement, you can use Metropolis-Hastings results in a similar way as the sampling results.

Simply, an area which does not contain accepted points has a lower chance to contain safe region.

We will look more deeply into this field in the next release. Namely, we will provide an option to bound the methods to search in a selected area without the necessity to delete previously obtained results.

Further option on how to supplement your analysis is to use visualisation of the functions in the third tab, *Sample functions*. Functions values in any given point or as a heatmap of respective function can be obtained there.

Multidimensional models

In the running example, we have presented a model with two parameters. Here we show and describe the difference of visualisation for models with more parameters - multidimensional models. The results plot differs namely in the case of space sampling, space refinement, and Metropolis-Hastings. In the previous case, these plots use phase space. In multidimensional case, the phase space is not plane; hence we use a different way to visualise the results.

On the following 3 pages, visualisations of space sampling, Metropolis-Hasting, and space refinement are shown. Results itself, input files, nor the settings do not have any particular meaning and served only the purpose of showing the following plots.

Multidimensional space sampling is a line scatter plot showing the sat parameter point. On the x-axis, there are indices of parameters in lexicographic order, $1, 2, \ldots$. In this order the parameter bounds were inserted - see Figure 6. You can observe this order in the textual representation of space - see Figure 1, first line. On the y-axis the is the value of the respective parameter. Dots of a single point share the colour and are connected with a line.

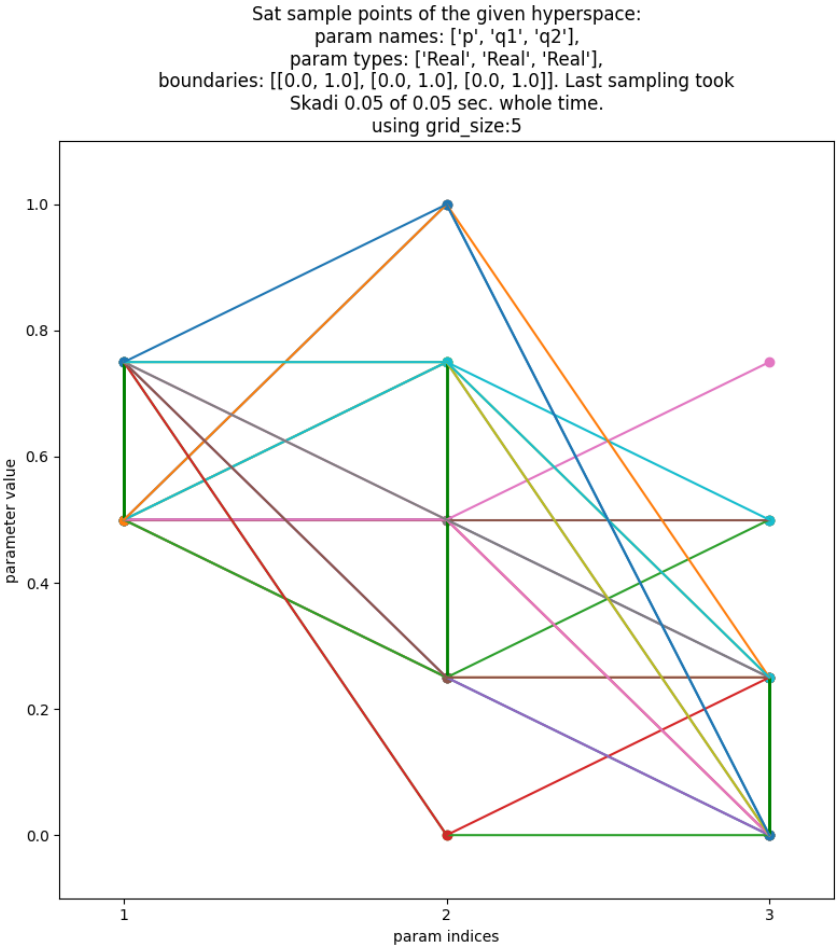

Figure 21: Mutidimensional Metropolis-Hastings results. With green lines parameter values of respective the parameter as a result of projection onto this parameter.

Multidimensional Metropolis-Hastings has the same output with one exception: instead of sat point there is a set of accepted points.

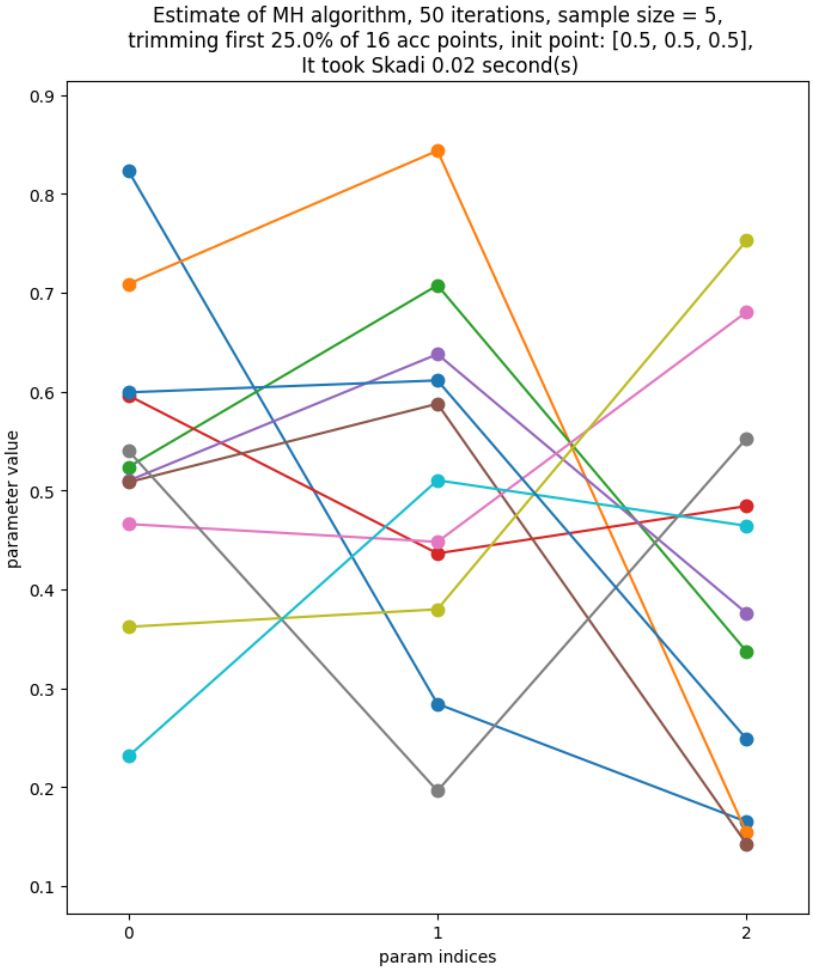

Figure 22: Mutidimensional Metropolis-Hastings results. With green lines parameter values of the respective parameter as a result of projection onto this parameter.

Multimensional space refinement provides an over approximation of the safe space by projections into the respective parameter. It is an overaprroximation as for each parameter the domain showing whether there is **any** safe subspace in this interval/single value.

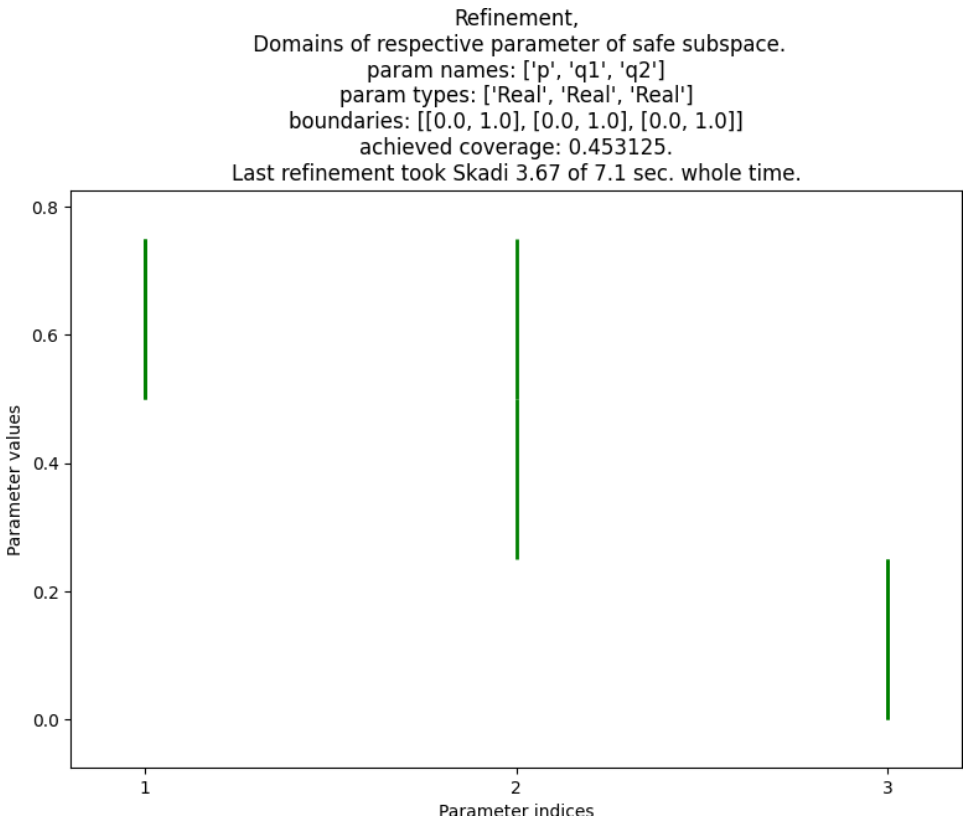

Figure 23: Mutidimensional space refinement results. With green lines parameter values of the respective parameter as a result of projection onto this parameter.

(Tip:) You can switch to show projections of unsafe space instead of safe space. Click on `Customize plot` button on the right of the plot and check the first checkbox. Then press `OK` and the plot will be redrawn automatically.

Bibliography

[1] Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A STORM is coming: A modern probabilistic model checker. In: Computer Aided Verification. pp. 592–600. Springer (2017)

[2] Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT solver for nonlinear theories over the reals. In: CADE-24. LNCS, vol. 7898, pp. 208–214. Springer (2013)

[3] Hajnal, M., Nouvian, M., Šafránek, D., Petrov, T.: Data-informed parameter synthesis for population markov chains. In: Češka, M., Paoletti, N. (eds.) Hybrid Systems Biology. pp. 147–164. Springer International Publishing, Cham (2019)

[4] Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (Sep 1994), `https://doi.org/10.1007/BF01211866`

[5] Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: International conference on computer aided verification. pp. 585–591. Springer (2011)

[6] de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS'08. LNCS, vol. 4963, pp. 337–340. Springer (2008)