

```

1 % Generates a file 'calculatePropensities.m' that will contain a function able to calculate
the necessary propensities for that system.
2 function GeneratePropensityCalculatorFile(SBMLModel, VHolder)
3
4 % determine number of reactions and species present in the model
5 numReactions = length(SBMLModel.reaction);
6 numSpecies = length(SBMLModel.species);
7
8 [cNames, cValues] = GetParameters(SBMLModel);
9
10 % matrices to hold propensity values
11 A = zeros(numReactions,1);
12
13 V = VHolder.V;
14 vNumOfReactant = VHolder.vNumOfReactant;
15 vReactant = VHolder.vReactant;
16 vDimerMap = VHolder.vDimerMap;
17
18 fid = fopen('calculatePropensities.m','w');
19 fprintf(fid,'function A = calculatePropensities(X)\n\n');
20
21 for i = 1:numReactions
22
23     % set initially to that reaction's parameter
24     format long;
25     fprintf(fid, 'A(%d) = (%d)', i, cValues(i) );
26
27     % multiply current value by each reactant's value if applicable, and account for
dimerisation reactions
28     for k = 1:(vNumOfReactant(i));
29
30         curSpeciesIndex = vReactant(i,k);
31
32         fprintf(fid, '*X(%d)', curSpeciesIndex);
33
34         % determine if reactant is part of a dimerisation reaction using dimerisation map,
then alter propensity accordingly
35         dimer_number = vDimerMap(curSpeciesIndex, i);
36         if dimer_number > 1
37             count = 1;
38             while (count < dimer_number)
39                 fprintf(fid, '*(X(%d)-%d)', curSpeciesIndex, count);
40                 count = count + 1;
41             end
42
43             fprintf(fid, '/%d', factorial(dimer_number) );
44         end
45     end
46
47     fprintf(fid, ';\n', i, cValues(i) );
48 end
49
50 fprintf(fid, '\nend', numReactions);
51
52 fclose(fid);
53
54 end

```

```
1 function totalsIndecies = GenerateSpecifiedTotalsCalculatorFile(SBMLModel)
2
3 numReactions = length(SBMLModel.reaction);
4 numSpecies = length(SBMLModel.species);
5
6 [speNames, speValues] = GetSpecies(SBMLModel);
7
8 totalsIndecies = zeros(numSpecies,1);
9
10 fid = fopen('calculateSpecifiedTotals.m','w');
11 fprintf(fid,'function X = calculateSpecifiedTotals(X)\n\n');
12
13 count = 0;
14 for i = 1:length(SBMLModel.rule)
15     curRule = SBMLModel.rule(i);
16
17     for j = 1:length( speNames )
18         if strcmp( speNames(j), curRule.variable )
19
20             totalsIndecies(count+1) = j;
21             count = count + 1;
22             fprintf(fid, 'X(%d) = 0', j);
23
24             % token string array
25             ruleToks = strsplit( curRule.formula ,'+');
26
27             for l = 1:length( ruleToks )
28                 for m = 1:length(speNames)
29                     if strcmp( speNames(m), ruleToks(l) )
30                         fprintf(fid, ' + X(%d)', m);
31                     end
32                 end
33             end
34
35             fprintf(fid, ';\n', m);
36
37             break;
38         end
39     end
40 end
41
42 fprintf(fid,'\nend',numReactions);
43
44 fclose(fid);
45
46 totalsIndecies = totalsIndecies(1:count);
47
48 end
```

```
1 function speConstIndecies = GetConstantSpeciesIndecies(SBMLModel)
2
3 numSpecies = length(SBMLModel.species);
4
5 speConstIndecies = zeros(numSpecies,1);
6
7 count = 0;
8 for i = 1:numSpecies
9     if SBMLModel.species(i).constant
10         speConstIndecies(count + 1) = i;
11         count = count + 1;
12     end
13 end
14
15 speConstIndecies = speConstIndecies(1:count,1);
16
17 end
```

```

1 function [cNames, cValues] = GetParameters(SBMLModel)
2
3 numReactions = length(SBMLModel.reaction);
4
5 % get all parameter names, values
6 [allCNames, allCValues] = GetAllParameters(SBMLModel);
7
8 % create matrices to hold parameter names, values
9 cValues = zeros(numReactions,1);
10 cNames = cell(numReactions,1);
11
12 % get parameters for each individual reaction as they may not be in order
13 for i = 1:numReactions
14
15     % attempt to get parameters from local reaction context
16     [cNamesTemp, cValuesTemp] = GetParameterFromReaction(SBMLModel.reaction(i));
17
18     % if parameter values are NOT embedded in each local reaction context (return values will be NULL)
19     if ( length(cNamesTemp) == 0 ) && ( length(cValuesTemp) == 0 )
20
21         % declare parameter found flag, get string from reaction formula field, tokenize it by operator
22         done = 0;
23         kinLawString = SBMLModel.reaction(i).kineticLaw.formula;
24         kinLawStringToks = strsplit( kinLawString , '*' );
25
26         % searches for each token in the list of all parameters in the model, assigns those values to the correct reaction if found
27         for j = 1:length(kinLawStringToks)
28             curTok = kinLawStringToks(j);
29             for k = 1:length(allCNames);
30                 if strcmp( allCNames{k} , curTok )
31                     cNames{i} = allCNames{k};
32                     cValues(i) = allCValues(k);
33                     done = 1;
34                     break;
35                 end
36
37                 if done == 1
38                     break;
39                 end
40             end
41
42             if done == 1
43                 break;
44             end
45         end
46         % if parameters were embedded in the local reaction contexts, assign them
47     else
48         cNames(i) = cNamesTemp;
49         cValues(i) = cValuesTemp;
50     end
51 end
52
53 end

```

```
1 function GraphHist(Y, speciesToGraph, speNames, split_flag, filename, method_name)
2
3
4 num_bins = length(Y)^(1/3);
5
6 specific_species_flag = 0;
7 speIndLength = length(speciesToGraph);
8
9 if speIndLength == 0
10     end_point = min ( size(Y) );
11 else
12     end_point = speIndLength;
13     specific_species_flag = 1;
14 end
15
16 if ~split_flag
17     figure
18     hold all
19 end
20
21 for i = 1:end_point
22
23     if specific_species_flag
24         current_species = speciesToGraph(i);
25     else
26         current_species = i;
27     end
28
29
30     min_val = min( Y(:,current_species) );
31     max_val = max( Y(:,current_species) );
32
33     % determine optimal number of bins (educated guess)
34     space = max_val - min_val + 1;
35     if space > num_bins
36         bins = linspace(min_val, max_val, num_bins);
37         h = hist( Y(:,current_species) , num_bins );
38     else
39         bins = linspace(min_val, max_val, space);
40         h = hist( Y(:,current_species) , space );
41     end
42
43     if split_flag
44         figure
45     end
46
47     plot( bins, h/length(Y) , 'o-' );
48
49     if split_flag
50         legend( speNames(current_species) );
51         xlabel('Number of Species','FontSize',12, 'FontName', 'Helvetica');
52         ylabel('Frequency','FontSize',12,'FontName', 'Helvetica');
53         title('Histogram of number of species at end of simulation','FontSize',16,'FontName', 'Helvetica');
54     end
55 end
56
57 if ~split_flag
```

```
58     hold off
59
60     if specific_species_flag
61         legend( speNames(speciesToGraph) );
62     else
63         legend( speNames )
64     end
65
66     xlabel('Number of Species','FontSize',12, 'FontName', 'Helvetica');
67     ylabel('Frequency','FontSize',12,'FontName', 'Helvetica');
68
69     title_string = ['Histogram of number of species at end of simulation from model source ' filename
filename ' ' using ' method_name];
70     title(title_string, 'Fontsize', 16,'FontName', 'Helvetica');
71 end
72
73 end
```

```
1 function GraphResults(Y, time, speciesToGraph, speNames, split_flag, filename, method_name)
2
3 specific_species_flag = 0;
4 speIndLength = length(speciesToGraph);
5
6 if speIndLength == 0
7     dims = size(Y);
8     end_point = dims(1);
9 else
10     end_point = speIndLength;
11     specific_species_flag = 1;
12 end
13
14 if ~split_flag
15     figure
16     hold all
17 end
18
19 for i = 1:end_point
20
21     if specific_species_flag
22         current_species = speciesToGraph(i);
23     else
24         current_species = i;
25     end
26
27     if split_flag
28         figure
29     end
30
31     plot( time , Y(current_species,:) );
32
33     if split_flag
34         legend( speNames(current_species) );
35         xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
36         ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
37         title('Species vs time using SSA','FontSize',16,'FontName', 'Helvetica');
38     end
39 end
40
41 if ~split_flag
42     hold off
43
44     if specific_species_flag
45         legend( speNames(speciesToGraph) );
46     else
47         legend( speNames )
48     end
49
50     xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
51     ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
52
53     title_string = ['Species vs time from model source ' filename ' using ' method_name];
54     title(title_string,'FontSize',16,'FontName', 'Helvetica');
55 end
56
57 end
```

```

1 function varargout = MARS(filename, varargin)
2
3 % Options:
4 %
5 % 'Hist'      - generate a histogram of the results of 10,000 trajecories
6 % 'Verbose'   - enable diagnostic outputs
7 % 'Time'      - max time to run the simulation for
8 % 'Record'    - step size between recording simulation state information
9 % 'Tau'       - use tau-leaping
10 % 'CLE'       - use Chemical Langevin Equation
11 % 'RRE'       - use raction rate equations
12 % 'GPU'       - use the system's CUDA-supported GPU for multiple trajectory generation
13 % 'MLMC'      - generate histogram using Multi-level Monte-Carlo simulation (experimental)
14 % 'Error'     - error to use for MLMC method
15 % 'Steps'     - number of data points to generate over the integration interval for all non-GPU
parallel methods
16 % 'Graph'     - plot a graph of the results
17
18 % default values for user-provided arguments
19 hist_flag     = 0;
20 verbose_flag  = 0;
21 tau_flag      = 0;
22 cle_flag      = 0;
23 rre_flag      = 0;
24 stiff_flag    = 0;
25 split_flag    = 0;
26 keep_flag     = 0;
27 record_flag   = 0;
28 gpu_flag      = 0;
29 graph_flag    = 0;
30 mlmc_flag     = 0;
31 m_flag        = 0;
32 err_flag      = 0;
33 steps_flag    = 0;
34 tfinal        = 50;
35 recordStep    = 20;
36 numSteps      = 100;
37 err           = 0;
38 speciesToGraph = [];
39
40 i = 1;
41 while (1+i) <= nargin
42     switch varargin{i}
43         case 'Hist'
44             hist_flag = 1;
45
46         case 'Verbose'
47             verbose_flag = 1;
48
49         case 'Time'
50             if (i+2) > nargin
51                 disp( sprintf('\nSimulation time argument missing.\n') );
52                 return;
53             else
54                 i = i + 1;
55                 time_arg = varargin{i};
56                 if ~isnumeric(time_arg)
57                     disp( sprintf('\nSimulation time argument must be a number.\n') );

```



```
58         return;
59     else
60         tfinal = time_arg;
61     end
62 end
63
64 case 'Record' % get record step argument, check for validity (integer)
65     if (i+2) > nargin
66         disp( sprintf('\nRecord step size argument missing.\n') );
67         return;
68     else
69         i = i + 1;
70         record_arg = varargin{i};
71         if ~isnumeric(record_arg) || mod(record_arg,1) ~= 0
72             disp( sprintf('\nRecord step size argument must be an integer.\n') );
73             return;
74         else
75             recordStep = record_arg;
76             record_flag = 1;
77         end
78     end
79
80 case 'Tau' % use tau-leaping, get value to use for tau
81     if (i+2) > nargin
82         tau = 0;
83     else
84         next_arg = varargin{i+1};
85         if isnumeric(next_arg)
86             tau = next_arg;
87             i = i + 1;
88         else
89             tau = 0;
90         end
91     end
92     tau_flag = 1;
93
94 case 'CLE' % use Langevin leaping algorithm, get value to use for tau
95     if (i+2) > nargin
96         tau = 0;
97     else
98         next_arg = varargin{i+1};
99         if isnumeric(next_arg)
100             tau = next_arg;
101             i = i + 1;
102         else
103             tau = 0;
104         end
105     end
106     cle_flag = 1;
107
108 case 'MLMC' % use MLMC method, get value to use for M, defaults to 100 steps
109     if (i+2) > nargin
110         M = 0;
111     else
112         m_arg = varargin{i+1};
113         if isnumeric(m_arg) && mod(m_arg,1) ~= 0
114             disp( sprintf('\nMLMC M-value argument must be an integer.\n') );
115             return;
```

```

116         elseif isnumeric(m_arg) && mod(m_arg,1) == 0
117             M = m_arg;
118             i = i + 1;
119             m_flag = 1;
120         else
121             M = 0;
122         end
123     end
124     mlmc_flag = 1;
125
126     case 'Steps' % get specific number of steps to generate for parallel methods not on a GPU
127         if (i+2) > nargin
128             disp( sprintf('\nNumber of steps size argument missing.\n') );
129             return;
130         else
131             step_arg = varargin{i+1};
132             if isnumeric(step_arg) && mod(step_arg,1) ~= 0
133                 disp( sprintf('\nNumber of steps argument must be an integer.\n') );
134                 return;
135             elseif isnumeric(step_arg) && mod(step_arg,1) == 0
136                 numSteps = step_arg;
137                 i = i + 1;
138             else
139                 disp( sprintf('\nNumber of steps size argument missing or invalid.\n') );
140                 return;
141             end
142         end
143
144     case 'Error' % get error to use for MLMC method, overrides default
145         if (i+2) > nargin
146             disp( sprintf('\nError argument missing.\n') );
147             return;
148         else
149             i = i + 1;
150             err_arg = varargin{i};
151             if ~isnumeric(err_arg)
152                 disp( sprintf('\nError argument missing.\n') );
153                 return;
154             elseif isnumeric(err_arg) && err_arg < 0
155                 disp( sprintf('\nError must be a positive number.\n') );
156             else
157                 err = err_arg;
158             end
159         end
160
161     case 'Graph' % whether or not to graph the results, and if so for which species (default is all)
162         if (i+2) > nargin
163             speciesToGraph = [];
164         else
165             next_arg = varargin{i+1};
166             if isvector(next_arg) && isnumeric(next_arg)
167                 speciesToGraph = next_arg;
168                 i = i + 1;
169             else
170                 speciesToGraph = [];
171             end

```

```

172         end
173         graph_flag = 1;
174
175         case 'RRE'
176             rre_flag = 1;
177
178         case 'Stiff'
179             stiff_flag = 1;
180
181         case 'Split'
182             split_flag = 1;
183
184         case 'Keep'
185             keep_flag = 1;
186
187         case 'GPU'
188             gpu_flag = 1;
189
190         otherwise
191             disp( sprintf('\nInvalid option detected at argument %d.\n',i) );
192             return;
193     end
194     i = i + 1;
195 end
196 if (tau_flag + cle_flag + rre_flag + mlmc_flag + gpu_flag) > 1
197     disp( sprintf('\nInvalid options: multiple methods selected. You may only pick one of
CLE, Tau-Leaping, RRE, MLMC, or GPU\n') );
198     return
199 end
200
201 if tfinal == 0
202     disp( sprintf('\nInvalid final time argument\n') );
203     return
204 end
205
206 if recordStep == 0
207     disp( sprintf('\nInvalid record step size argument\n') );
208     return
209 end
210
211 if numSteps == 0
212     disp( sprintf('\nInvalid number of steps argument\n') );
213     return
214 end
215
216 if m_flag && M < 2
217     disp( sprintf('\nMLMC M-value must be an integer greater than 1\n') );
218     return
219 end
220
221 addpath(genpath('./toolbox/SBMLToolbox'));
222
223 platform_str = computer;
224
225 % get type of platform so proper libraries can be added to path, currently only PC, Mac
supported
226 switch platform_str
227     case 'MACI64'

```

```

228     addpath(genpath('./toolbox/libSBML/mac'));
229     case 'PCWIN'
230         addpath(genpath('./toolbox/libSBML/win32'));
231     case 'PCWIN64'
232         addpath(genpath('./toolbox/libSBML/win64'));
233     case 'GLNXA64'
234         addpath(genpath('./toolbox/libSBML/linux'));
235     otherwise
236         disp('Platform not supported');
237         return;
238 end
239
240 % create files to be filled, then close all
241 file_list = {'calculatePropensities.m';...
242             'calculateSpecifiedTotals.m';...
243             'RRE_functions.m';...
244             'fireGpuTrajectories.m'};
245
246 num_files = length(file_list);
247 for i = 1:num_files
248     cur_file = file_list{i};
249     fid = fopen(cur_file,'w');
250     fclose(fid);
251 end
252
253 % get system (model) information from SBML file
254 SysInf = SSA_setup(filename, verbose_flag);
255 numSpecies      = SysInf.numSpecies;
256 numReactions    = SysInf.numReactions;
257 speNames       = SysInf.speNames;
258 speValues      = SysInf.speValues;
259 cNames        = SysInf.cNames;
260 cValues       = SysInf.cValues;
261 speConstIndecies = SysInf.speConstIndecies;
262 totalsIndecies  = SysInf.totalsIndecies;
263 VHolder        = SysInf.VHolder;
264
265 % check for bad species-to-graph entries
266 if graph_flag && ~mlmc_flag
267     numSpeToGraph = length(speciesToGraph);
268     if numSpeToGraph ~= 0
269         for i = 1:numSpeToGraph
270
271             curIndex = speciesToGraph(i);
272             if curIndex > numSpecies || curIndex < 1
273                 disp( sprintf(['Error: invalid species index provided, exceeds number of
species in system or is less than 1....
274                             ' Graph will not be displayed.\n']) );
275                 graph_flag = 0;
276             end
277
278             for j = 1:(i-1)
279                 checkIndex = speciesToGraph(j);
280                 if checkIndex == curIndex
281                     disp( sprintf('Error: invalid species index provided, duplicate index.
Graph will not be displayed.\n') );
282                     graph_flag = 0;
283                 end

```

```

284         end
285
286     end
287 end
288 end
289
290 method_name = '';
291 rehash
292
293 if gpu_flag
294     Y = SSA_gpu(filename, SysInf, tfinal, verbose_flag);
295     method_name = 'SSA on GPU';
296
297 elseif mlmc_flag
298     %open parallel pool based on installed toolbox version
299     version_less_flag = verLessThan('distcomp', '6.3');
300     if version_less_flag
301         matlabpool open;
302     else
303         parpool;
304     end
305
306     [Mean, Step] = MLMCGen(SysInf, tfinal, numSteps, verbose_flag, split_flag, \
speciesToGraph, graph_flag, M, err);
307     Y = Mean;
308     varargout{3} = Step;
309     time = linspace(0,tfinal,numSteps);
310     varargout{2} = time;
311     method_name = 'MLMC';
312
313     % close parallel pool
314     if version_less_flag
315         matlabpool close;
316     else
317         delete(gcp);
318     end
319
320 elseif hist_flag
321     %open parallel pool based on installed toolbox version
322     version_less_flag = verLessThan('distcomp', '6.3');
323     if version_less_flag
324         matlabpool open;
325     else
326         parpool;
327     end
328
329     % use indicated method to generate trajectories accross multiple CPUs or CPU cores
330     if tau_flag
331         [Y,Mean,Std] = SSAGen_parfor_tauleap(SysInf, tfinal, recordStep, verbose_flag, tau, \
speciesToGraph, numSteps, graph_flag, 0);
332         method_name = 'SSA with tau-leaping on parallel CPUs';
333     elseif cle_flag
334         [Y,Mean,Std] = SSAGen_parfor_cle(SysInf, tfinal, recordStep, verbose_flag, tau, \
speciesToGraph, numSteps, graph_flag);
335         method_name = 'CLE on parallel CPUs';
336     elseif rre_flag
337         disp( sprintf('\n''Hist'' is not a valid option to use with the Reaction Rate \
Equation method\n') );

```

```

338     else
339         [Y,Mean,Std] = SSAGen_parfor(SysInf, tfinal, recordStep, verbose_flag, \
speciesToGraph, numSteps, graph_flag);
340         method_name = 'SSA on parallel CPUs';
341     end
342
343     varargout{2} = Mean;
344     varargout{3} = Std;
345
346     % close parallel pool
347     if version_less_flag
348         matlabpool close;
349     else
350         delete(gcp);
351     end
352
353 else
354     % generate single trajectory based on indicated method
355     if tau_flag
356         [time, Y] = SSAGen_tauleap(SysInf, tfinal, recordStep, verbose_flag, tau);
357         method_name = 'SSA with tau-leaping';
358     elseif cle_flag
359         [time, Y] = SSAGen_cle(SysInf, tfinal, recordStep, verbose_flag, tau);
360         method_name = 'CLE';
361     elseif rre_flag
362         if record_flag
363             disp(sprintf('\nWarning: ''Record'' argument will be ignored - not valid with \
Reaction Rate Equation method\n'));
364         end
365         [time, Y] = RREGen(SysInf, tfinal, verbose_flag, stiff_flag);
366         method_name = 'RRE';
367     else
368         [time, Y] = SSAGen(SysInf, tfinal, recordStep, verbose_flag);
369         method_name = 'SSA';
370     end
371
372     varargout{2} = time;
373 end
374
375 varargout{1} = Y;
376
377 arg_type = class(filename);
378 switch arg_type
379     case 'struct'
380         filename = filename.name;
381 end
382
383 % graph end of simulation histogram is using a parallel method and graph flag has been set
384 if graph_flag
385     if gpu_flag || hist_flag
386         GraphHist(Y, speciesToGraph, speNames, split_flag, filename, method_name);
387     else
388         GraphPlot(Y, time, speciesToGraph, speNames, split_flag, filename, method_name);
389     end
390 end
391
392 % keep temporary files if indicated
393 if ~keep_flag

```

---

```
394     for i = 1:num_files
395         cur_file = file_list{i};
396         delete(cur_file);
397     end
398 end
399
400 end
```

```
1 function [Y, Step] = MLMCGen(SysInf, tfinal, numSteps, verbose_flag, split_flag, %
speciesToGraph, graph_flag, M, err)
2
3 numSpecies      = SysInf.numSpecies;
4 numReactions    = SysInf.numReactions;
5 speNames       = SysInf.speNames;
6 speValues      = SysInf.speValues;
7 cNames         = SysInf.cNames;
8 cValues        = SysInf.cValues;
9 speConstIndecies = SysInf.speConstIndecies;
10 totalsIndecies  = SysInf.totalsIndecies;
11 VHolder        = SysInf.VHolder;
12
13 % initial values
14 X = speValues;
15 numSteps = numSteps - 1;
16
17 if err == 0
18     err = 1/100;
19 end
20
21 % extract V from VHolder and display
22 V = VHolder.V;
23
24 if verbose_flag
25     disp( sprintf('Stoichiometric Matrix:\n') ); disp(V);
26 end
27
28 % matrices to hold propensity values, number of species present after each step, and the %
length of each step
29 A = zeros(numReactions,1);
30
31 % parameters
32 N = max(X);
33
34 % default M
35 if M == 0
36     M = 3;
37 end
38
39 alp = zeros(numSpecies,1);
40 for i = 1:numSpecies
41     val = X(i);
42     if val == 0
43         alp(i) = 0;
44     else
45         alp(i) = log(val)/log(N);
46     end
47 end
48
49 bet = zeros(numReactions,1);
50 for i = 1:numReactions
51     val = cValues(i);
52     if val ~= 0
53         bet(i) = log(val)/log(N);
54     end
55 end
56
```



```
57 V_pos = -V;
58 V_pos(V_pos < 0) = 0;
59
60 % get gamma value from largest of candidates
61 gam = -Inf;
62 for i = 1:numSpecies
63     for k = 1:numReactions
64         if V(i,k) ~= 0
65             gam_can = bet(k) + dot(V_pos(:,k),alp) - alp(i);
66             if gam_can > gam
67                 gam = gam_can;
68             end
69         end
70     end
71 end
72
73 % get rho value from largest of candidates
74 rho = Inf;
75 for k = 1:numReactions
76     for i = 1:numSpecies
77         if V(i,k) ~= 0
78             rho_can = alp(i);
79             if rho_can < rho
80                 rho = rho_can;
81             end
82         end
83     end
84 end
85
86 L = ceil(abs(log(err)));
87
88 % should have three levels
89 if L <= 2
90     l_0 = 0;
91 else
92     l_0 = L - 2;
93 end
94
95 num_levels = L - l_0 + 1;
96
97 % get required number of coarsest trajectories
98 n_0 = 4 * ceil( (N^-rho * N^-gam * err^-2) / 4 );
99
100 % get level step sizes and required number of trajectories
101 h_l = zeros(num_levels, 1);
102 n_l = zeros(num_levels, 1);
103 for i = 1:num_levels
104     l = l_0 + i - 1;
105     h_l(i) = tfinal/(M^l);
106     n_l(i) = ceil( N^-rho * N^gam * (L - l_0) * h_l(i) * err^-2 );
107 end
108
109 % make each n_l divisible by 4
110 for i = 1:num_levels
111     val = n_l(i);
112     while mod(val,4) ~= 0
113         val = val+1;
114     end
```

```

115     n_l(i) = val;
116 end
117
118 % print information if required
119 if verbose_flag
120     fprintf('N:\t%d\n', N);
121     fprintf('Gamma:\t%d\n', gam);
122     fprintf('Rho:\t%d\n', rho);
123     fprintf('M:\t%d\n', M);
124     fprintf('Error:\t%d\n', err);
125     fprintf('Granularities:\n\n');
126     disp(h_l);
127     fprintf('Number of trajectories at each level:\n\n');
128     disp(n_0);
129     disp(n_l(2:num_levels));
130 end
131
132 num_trajectories = sum(n_l);
133
134 interval = tfinal/(numSteps+1);
135
136 % level 0
137 [~, Mean_coarse, ~] = SSAGen_parfor_tauleap(SysInf, tfinal, 0, 0, h_l(1), speciesToGraph, numSteps+1, 0, n_0);
138
139 Y = Mean_coarse;
140 time = linspace(0,tfinal,numSteps+1);
141
142 % ----- % start MLMC
143
144 for i = 2:num_levels
145     num_runs = n_l(i);
146
147     Y_sub = zeros( num_runs , numSpecies, numSteps+1);
148
149     parfor k = 1:num_runs
150
151         % setup that level trajectory
152         hl      = h_l(i);
153         hl_1    = M*hl;
154         l       = l_0 + i - 1;
155         zl      = X;
156         zl_1    = X;
157         Zl      = zeros(numSpecies, numSteps+1);
158         Zl_1    = zeros(numSpecies, numSteps+1);
159         Zl(:,1) = X;
160         Zl_1(:,1) = X;
161         t       = 0;
162         n       = 2;
163
164         while n <= (numSteps+1)
165
166             lam_bot = calculatePropensities(zl_1)';
167             A = zeros(numReactions, 3);
168
169             for j = 1:M

```

```

172         lam_top = calculatePropensities(zl)';
173
174         % (a)
175         A(:,1) = min(lam_top, lam_bot);
176         A(:,2) = lam_top - A(:,1);
177         A(:,3) = lam_bot - A(:,1);
178
179         % (b)
180         Lam = poissrnd(A*hl);
181
182         % (c)
183         delta_top = Lam(:,1) + Lam(:,2);
184         delta_bot = Lam(:,1) + Lam(:,3);
185         zl = zl + V*delta_top;
186         zl_1 = zl_1 + V*delta_bot;
187
188         zl(speConstIndecies) = speValues(speConstIndecies);
189         zl_1(speConstIndecies) = speValues(speConstIndecies);
190         zl = calculateSpecifiedTotals(zl);
191         zl_1 = calculateSpecifiedTotals(zl_1);
192
193     end
194
195     t = t + hl_1;
196
197     % record
198     if t > ((n-1)*interval)
199         Zl(:,n) = zl;
200         Zl_1(:,n) = zl_1;
201         n = n + 1;
202     end
203 end
204
205 data = Zl - Zl_1;
206 Y_sub(k,:,:) = data;
207
208 end
209
210 Mean_level = zeros(numSpecies, numSteps+1);
211
212 for i = 1:numSpecies
213     for j = 1:(numSteps+1)
214         Mean(i,j) = mean( Y_sub(:,i,j) );
215     end
216 end
217
218 Y = Y + Mean_level;
219
220 end
221
222 Step = h_l(length(h_l));
223
224 end

```

```

1 function [time, Y] = RREGen(SysInf, tfinal, verbose_flag, stiff_flag)
2
3 numSpecies      = SysInf.numSpecies;
4 numReactions    = SysInf.numReactions;
5 speNames       = SysInf.speNames;
6 speValues      = SysInf.speValues;
7 cNames         = SysInf.cNames;
8 cValues        = SysInf.cValues;
9 speConstIndecies = SysInf.speConstIndecies;
10 totalsIndecies  = SysInf.totalsIndecies;
11 VHolder        = SysInf.VHolder;
12
13 if verbose_flag
14     disp( sprintf('\nNumber of species types:\n') ); disp(numSpecies);
15     disp( sprintf('\nNumber of reactions:\n') ); disp(numReactions);
16 end
17
18 if verbose_flag
19     disp( sprintf('\nParameter names:\n') ); disp(cNames);
20     disp( sprintf('\nParameter values:\n') ); disp(cValues);
21 end
22
23 if verbose_flag
24     disp( sprintf('\nSpecies' names:\n') ); disp(speNames);
25     disp( sprintf('\nSpecies' initial amounts:\n') ); disp(speValues);
26 end
27
28 % holder strings for RHS of RREs
29 for i = 1:numReactions
30     A{i} = '';
31 end
32 A = A';
33
34 V = VHolder.V;
35 vNumOfReactant = VHolder.vNumOfReactant;
36 vReactant = VHolder.vReactant;
37 vDimerMap = VHolder.vDimerMap;
38
39 if verbose_flag
40     disp( sprintf('Stoichiometric Matrix:\n') ); disp(V);
41 end
42
43 for i = 1:numReactions
44     % set initially to that reaction's parameter
45     format long;
46     A{i} = strcat( A{i} , num2str( cValues(i) ) );
47
48     % multiply current value by each reactant's value if applicable, and account for
49     dimerisation reactions
50     for k = 1:(vNumOfReactant(i));
51
52         curSpeciesIndex = vReactant(i,k);
53
54         A{i} = strcat( A{i} , sprintf('*X(%d)',curSpeciesIndex) );
55
56         % determine if reactant is part of a dimerisation reaction using dimerisation map,
57         then alter propensity accordingly

```

```

57     dimer_number = vDimerMap(curSpeciesIndex, i);
58     if dimer_number > 1
59         for j = 1:(dimer_number-1)
60             A{i} = strcat( A{i} , sprintf('*(X(%d)-%d)', curSpeciesIndex, j ) );
61         end
62
63         A{i} = strcat( A{i} , sprintf('/%d', factorial(dimer_number) ) );
64     end
65 end
66 end
67
68 fid = fopen('RRE_functions.m','w');
69 fprintf(fid,'function dXdt = RRE_functions(t,X)');
70
71 fprintf(fid,'\n\ndXdt = zeros(%d,1);\n\n', numSpecies);
72
73 for i = 1:numSpecies
74
75     fprintf(fid,'dXdt(%d) = 0',i);
76
77     if ~ismember(i,speConstIndecies)
78         for j = 1:numReactions
79             if V(i,j) ~= 0
80                 fprintf(fid, ' + %d*%s', V(i,j), A{j});
81             end
82         end
83     end
84
85     fprintf(fid,' ;\n');
86
87 end
88
89 fprintf(fid,'\nend');
90 fclose(fid);
91
92 %tspan = linspace(0,tfinal,tfinal);
93 tspan = [0 tfinal];
94
95 disp( sprintf('=====Starting Solver=====') )
96
97 if stiff_flag
98     [time,y] = ode15s(@RRE_functions,tspan,speValues);
99 else
100     [time,y] = ode45(@RRE_functions,tspan,speValues);
101 end
102
103 Y = y';
104
105 if length(totalsIndecies) ~= 0
106     for i = 1:length(time)
107         Y(:,i) = calculateSpecifiedTotals(Y(:,i));
108     end
109 end
110
111 if verbose_flag
112     disp(sprintf('\nSpecies'' final amounts:\n'));
113     Amount = Y(:,length(Y));
114     dataTable = table(Amount,'RowNames',speNames);

```

```
115     disp(dataTable);  
116 end  
117  
118 end
```

```

1 function [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues, tfinal, recordStep, verbose_flag)
2
3 % set max muber of data points for each chunk of the recorded values matrix
4 numMaxDataPoints = 10000;
5
6 Y = zeros(numSpecies, numMaxDataPoints);
7 time = zeros(1, numMaxDataPoints);
8
9 % assign initial values recorded values
10 Y( : , 1 ) = X;
11 time(1) = 0;
12
13 % initial values
14 t = 0;
15 count = 1;
16
17 speConstIndeciesLength = length(speConstIndecies);
18
19 if verbose_flag
20     disp( sprintf('\n=====STARTING SSA=====\\n') );
21 end
22
23 tic
24
25 while t < tfinal
26
27     %-----
28     % calculate value of each propensity at that step
29     A = calculatePropensities(X);
30     %-----
31
32     asum = sum(A);
33
34     % break out of simulation if all species are consumed
35     if asum == 0
36         if verbose_flag
37             disp(sprintf('\n=====REACTION HALTED - ALL SPECIES COMSUMED=====\\n'));
38             disp(sprintf('Final time was %f', t) );
39         end
40
41         Y( : , ceil(count/recordStep) + 1 ) = X;
42         time( ceil(count/recordStep) + 1 ) = t;
43
44         break
45     end
46
47     j = min( find( rand < cumsum(A/asum) ) );
48     tau = log(1/rand)/asum;
49
50     X = X + V(:,j);
51
52     %-----
53     X = calculateSpecifiedTotals(X);
54     %-----
55
56     for i = 1:speConstIndeciesLength
57         X(speConstIndecies(i)) = speValues(speConstIndecies(i));

```

```
58     end
59
60     t = t + tau;
61
62     if mod(count, recordStep) == 0
63         Y( : , (count/recordStep) + 1) = X;
64         time(count/recordStep + 1) = t;
65     end
66
67     count = count + 1;
68 end
69
70 run_time = toc;
71
72 Y = Y( : ,1:(floor(count/recordStep)) );
73 time = time( 1:(floor(count/recordStep)) );
74
75 if verbose_flag
76     disp( sprintf('\n%d steps taken\n', count ) );
77     disp( sprintf('%d steps recorded\n', floor(count/recordStep) ) );
78 end
79
80 end
```



```

1 function [Y, X, time, run_time] = SingleTrajectory_cle(V, X, speConstIndecies, numSpecies, speValues, tfinal, recordStep, verbose_flag, tau)
2
3 % set max muber of data points for each chunk of the recorded values matrix
4 numMaxDataPoints = 10000;
5
6 Y = zeros(numSpecies, numMaxDataPoints);
7 time = zeros(1, numMaxDataPoints);
8
9 % assign initial values recorded values
10 Y( : , 1 ) = X;
11 time(1) = 0;
12
13 % initial values
14 t = 0;
15 count = 1;
16
17 speConstIndeciesLength = length(speConstIndecies);
18
19 if verbose_flag
20     disp( sprintf('\n=====STARTING SSA WITH CLE=====\n') );
21 end
22
23 tic
24
25 while t < tfinal
26
27     %-----
28     % calculate value of each propensity at that step
29     A = calculatePropensities(X);
30     %-----
31
32     % break out of simulation if all species are consumed
33     if cumsum(A) == 0
34         if verbose_flag
35             disp(sprintf('\n=====REACTION HALTED - ALL SPECIES COMSUMED=====\n'));
36             disp(sprintf('Final time was %f', t) );
37         end
38
39         Y( : , floor(count/recordStep) + 1) = X;
40         time( floor(count/recordStep) + 1) = t;
41
42         break
43     end
44
45     % get sampling of random variables and sub into CLE formula
46     d = tau*A + sqrt( abs(tau*A) ) * rand;
47
48     % update values
49     X = X + V * d';
50
51     %-----
52     X = calculateSpecifiedTotals(X);
53     %-----
54
55     for i = 1:speConstIndeciesLength
56         X(speConstIndecies(i)) = speValues(speConstIndecies(i));
57     end

```

```
58
59     t = t + tau;
60
61     if mod(count, recordStep) == 0
62         Y( : , (count/recordStep) + 1) = X;
63         time(count/recordStep + 1) = t;
64     end
65
66     count = count + 1;
67 end
68
69 run_time = toc;
70
71 Y = Y( : ,1:(floor(count/recordStep)) );
72 time = time( 1:(floor(count/recordStep)) );
73
74 if verbose_flag
75     disp( sprintf('\n%d steps taken\n', count-1 ) );
76     disp( sprintf('%d steps recorded\n', floor(count/recordStep)-1 ) );
77 end
78
79 end
```

```

1 function [Y, X, time, run_time] = SingleTrajectory_tauleap(V, X, speConstIndecies, numSpecies, speValues, tfinal, recordStep, verbose_flag, tau)
2
3 % set max muber of data points for each chunk of the recorded values matrix
4 numMaxDataPoints = 10000;
5
6 Y = zeros(numSpecies, numMaxDataPoints);
7 time = zeros(1, numMaxDataPoints);
8
9 % assign initial values recorded values
10 Y( : , 1 ) = X;
11 time(1) = 0;
12
13 % initial values
14 t = 0;
15 count = 1;
16
17 speConstIndeciesLength = length(speConstIndecies);
18
19 if verbose_flag
20     disp( sprintf('\n=====STARTING SSA WITH TAU_LEAPING=====\\n') );
21 end
22
23 tic
24
25 while t < tfinal
26
27     %-----
28     % calculate value of each propensity at that step
29     A = calculatePropensities(X);
30     %-----
31
32     % break out of simulation if all species are consumed
33     if cumsum(A) == 0
34         if verbose_flag
35             disp(sprintf('\\n=====REACTION HALTED - ALL SPECIES COMSUMED=====\\n'));
36             disp(sprintf('Final time was %f', t) );
37         end
38
39         Y( : , floor(count/recordStep) + 1) = X;
40         time( floor(count/recordStep) + 1) = t;
41
42         break
43     end
44
45     % get sampling of poisson random variables
46     pois_rand_vars = poissrnd(A*tau);
47
48     % update values
49     X = X + V * pois_rand_vars';
50
51     %-----
52     X = calculateSpecifiedTotals(X);
53     %-----
54
55     for i = 1:speConstIndeciesLength
56         X(speConstIndecies(i)) = speValues(speConstIndecies(i));
57     end

```

```
58
59     t = t + tau;
60
61     if mod(count, recordStep) == 0
62         Y( : , (count/recordStep) + 1) = X;
63         time(count/recordStep + 1) = t;
64     end
65
66     count = count + 1;
67 end
68
69 run_time = toc;
70
71 Y = Y( : , 1:(floor(count/recordStep)) );
72 time = time( 1:(floor(count/recordStep)) );
73
74 if verbose_flag
75     disp( sprintf('\n%d steps taken\n', count-1 ) );
76     disp( sprintf('%d steps recorded\n', floor(count/recordStep)-1 ) );
77 end
78
79 end
```

```
1 function Y = SSA_gpu(filename, SysInf, tfinal, verbose_flag)
2
3 numSpecies      = SysInf.numSpecies;
4 numReactions    = SysInf.numReactions;
5 speNames        = SysInf.speNames;
6 speValues       = SysInf.speValues;
7 cNames          = SysInf.cNames;
8 cValues         = SysInf.cValues;
9 speConstIndecies = SysInf.speConstIndecies;
10 totalsIndecies  = SysInf.totalsIndecies;
11 VHolder         = SysInf.VHolder;
12 gpu = gpuDevice();
13
14 if verbose_flag
15     disp( sprintf('GPU Device detected:\n') );
16     disp( gpu );
17 end
18
19 V = VHolder.V;
20 vNumOfReactant = VHolder.vNumOfReactant;
21 vReactant = VHolder.vReactant;
22 vDimerMap = VHolder.vDimerMap;
23
24 fid = fopen('fireGpuTrajectories.m','w');
25 fprintf(fid,'function Y = fireGpuTrajectories(VHolder, verbose_flag)\n\n');
26
27 fprintf(fid, 'V = VHolder.V;\n\n');
28
29 for i = 1:numSpecies
30     fprintf(fid, 'x%d = %d;\n', i, speValues(i));
31 end
32
33 fprintf(fid, '\ntfinal = %d;\n\n', tfinal);
34
35 fprintf(fid, '\tfunction [input]');
36 for i = 1:numSpecies
37     fprintf(fid, ', x%d', i);
38 end
39 fprintf(fid, '];\n');
40
41 fprintf(fid, ' = fire_single_gpu_trajectory(input);\n');
42 for i = 1:numSpecies
43     fprintf(fid, ', x%d', i);
44 end
45 fprintf(fid, ')\n\n');
46
47 format long
48
49 for i = 1:numReactions
50     fprintf(fid, '\t\tc%d = %e;\n', i, cValues(i));
51 end
52
53 fprintf(fid, '\n\t\tt = 0;\n');
54
55 fprintf(fid, '\n\t\twhile t < tfinal\n\n');
56
57 for i = 1:numReactions
58
```

```

59 % set initially to that reaction's parameter
60 format long;
61 fprintf(fid, '\t\t\t a%d = (%e)', i, cValues(i) );
62
63 % multiply current value by each reactant's value if applicable, and account for
dimerisation reactions
64 for k = 1:(vNumOfReactant(i));
65     curSpeciesIndex = vReactant(i,k);
66     fprintf(fid, '*x%d', curSpeciesIndex);
67
68     % determine if reactant is part of a dimerisation reaction using dimerisation map,
then alter propensity accordingly
69     dimer_number = vDimerMap(curSpeciesIndex, i);
70     if dimer_number > 1
71         count = 1;
72         while (count < dimer_number)
73             fprintf(fid, '* (x%d-%d)', curSpeciesIndex, count);
74             count = count + 1;
75         end
76         fprintf(fid, '/%d', factorial(dimer_number) );
77     end
78 end
79 fprintf(fid, ';\n', i, cValues(i) );
80 end
81
82 fprintf(fid, '\n\t\t\t a_sum = ');
83 for i = 1:numReactions
84     fprintf(fid, ' + a%d', i);
85 end
86 fprintf(fid, ';\n\n');
87
88 fprintf(fid, '\t\t\t if a_sum == 0\n\t\t\t\t return\n\t\t\t\t end\n\n');
89
90 for i = 1:numReactions
91     fprintf(fid, '\t\t\t a_tot%d = (', i);
92     for j = 1:i
93         fprintf(fid, '+a%d', j);
94     end
95     fprintf(fid, ')/a_sum;\n');
96 end
97
98 fprintf(fid, '\n\t\t\t t_j = 1;\n\n');
99 fprintf(fid, '\t\t\t rand_num = rand;\n\n');
100
101 fprintf(fid, '\t\t\t if a_tot_1 > rand_num\n');
102 fprintf(fid, '\t\t\t\t t_j = 1;\n');
103
104 for i = 2:numReactions
105     fprintf(fid, '\t\t\t\t elseif a_tot%d > rand_num\n', i);
106     fprintf(fid, '\t\t\t\t t_j = %d;\n', i);
107 end
108
109 fprintf(fid, '\t\t\t end\n\n');
110
111
112
113
114

```

```

115 fprintf(fid, '\t\t\ttau = log(1/rand)/asum;\n\n');
116
117 for i = 1:numSpecies
118     if ~ismember(i,speConstIndecies)
119         fprintf(fid, '\t\t\tx%d = x%d + V(%d,j);\n', i, i, i);
120     end
121 end
122
123 fprintf(fid, '\n');
124 SBMLModel = TranslateSBML(filename);
125 count = 0;
126 for i = 1:length(SBMLModel.rule)
127     curRule = SBMLModel.rule(i);
128
129     for j = 1:length( speNames )
130         if strcmp( speNames(j), curRule.variable )
131
132             totalsIndecies(count+1) = j;
133             count = count + 1;
134             fprintf(fid, '\t\t\tx%d =', j);
135
136             % token string array
137             ruleToks = strsplit( curRule.formula , '+');
138
139             for l = 1:length( ruleToks )
140                 for m = 1:length(speNames)
141                     if strcmp( speNames(m), ruleToks(l) )
142                         fprintf(fid, ' + x%d', m);
143                     end
144                 end
145             end
146
147             fprintf(fid, ';\n', m);
148
149             break;
150         end
151     end
152 end
153
154 fprintf(fid, '\n\t\t\tt = t + tau;\n\n');
155
156 fprintf(fid, '\t\t\tend\n\n');
157 fprintf(fid, '\t\t\tend');
158
159 fprintf(fid, '\n\nnum_trajectories = 10000;\n');
160
161 fprintf(fid, 'trial_nums = linspace(1,num_trajectories, num_trajectories)';\n' );
162 fprintf(fid, 'inputs = gpuArray(trial_nums);\n' );
163
164 fprintf(fid, '\n[g_trial');
165 for i = 1:numSpecies
166     fprintf(fid, ', g_x%d', i);
167 end
168 fprintf(fid, '] = arrayfun(@fire_single_gpu_trajectory, inputs');
169 for i = 1:numSpecies
170     fprintf(fid, ', x%d', i);
171 end
172 fprintf(fid, ');;\n\n');

```

```
173
174 fprintf(fid, 'trials = gather(g_trial);\n');
175
176 fprintf(fid, 'Y = [');
177 for i = 1:numSpecies
178     fprintf(fid, ' gather(g_x%d)', i);
179 end
180 fprintf(fid, '];\n\n');
181
182 fprintf(fid, '\nend');
183
184 fclose(fid);
185
186 Y = fireGpuTrajectories(VHolder, verbose_flag);
187 wait(gpu);
188
189 Mean = zeros(numSpecies, 1);
190 Std_dev = zeros(numSpecies, 1);
191 for i = 1:numSpecies
192     data = Y( : , i );
193     Mean(i) = mean( data );
194     Std_dev(i) = std( data );
195 end
196
197 if verbose_flag
198     dataTableMean = table(Mean, 'RowNames', speNames);
199     disp(dataTableMean);
200     dataTableStddev = table(Std_dev, 'RowNames', speNames);
201     disp(dataTableStddev);
202 end
203
204 end
```



```
1 function SysInf = SSA_setup(filename, verbose_flag)
2
3 if verbose_flag
4     disp(' ');
5 end
6
7 arg_type = class(filename);
8
9 switch arg_type
10     case 'char'
11         SBMLModel = TranslateSBML(filename);
12     case 'struct'
13         SBMLModel = filename;
14 end
15
16
17 % determine number of reactions and species present in the model
18 numReactions = length(SBMLModel.reaction);
19 numSpecies = length(SBMLModel.species);
20
21 if verbose_flag
22     disp( sprintf('\nNumber of species types:\n') ); disp(numSpecies);
23     disp( sprintf('\nNumber of reactions:\n') ); disp(numReactions);
24 end
25
26 [cNames, cValues] = GetParameters(SBMLModel);
27
28 if verbose_flag
29     [cNames_us, cValues_us] = GetAllParameters(SBMLModel);
30     cNames_us = cNames_us';
31     cValues_us = cValues_us';
32
33     disp(sprintf('\nParameter Values:\n'))
34     Value = cValues_us;
35     dataTable = table(Value, 'RowNames', cNames_us);
36     disp(dataTable);
37 end
38
39 % get species names and values, set X to initial values
40 [speNames, speValues] = GetSpecies(SBMLModel);
41 speNames = speNames';
42 speValues = speValues';
43
44 if verbose_flag
45     disp(sprintf('\nSpecies' ' initial amounts:\n'))
46     Amount = speValues;
47     dataTable = table(Amount, 'RowNames', speNames);
48     disp(dataTable);
49 end
50
51 % matrices to hold propensity values, number of species present after each step, and the
length of each step
52 A = zeros(numReactions,1);
53
54 VHolder = StoichiometricMatricesHolder(SBMLModel);
55
56 % will generate 'calculatePropensities.m' file
57 GeneratePropensityCalculatorFile(SBMLModel, VHolder);
```

```
58
59 % will generate 'calculateSpecifiedTotals.m' file
60 totalsIndecies = GenerateSpecifiedTotalsCalculatorFile(SBMLModel);
61
62 % determine if any species have a boundary condition and get their indecies
63 speConstIndecies = GetConstantSpeciesIndecies(SBMLModel);
64
65 SysInf = SystemInformationHolder;
66
67 SysInf.numSpecies      = numSpecies;
68 SysInf.numReactions    = numReactions;
69 SysInf.speNames        = speNames;
70 SysInf.speValues       = speValues;
71 SysInf.cNames          = cNames;
72 SysInf.cValues         = cValues;
73 SysInf.speConstIndecies = speConstIndecies;
74 SysInf.totalsIndecies  = totalsIndecies;
75 SysInf.VHolder         = VHolder;
76
77 end
```

```
1 function [time, Y] = SSAGen(SysInf, tfinal, recordStep, verbose_flag, split_flag)
2
3 numSpecies      = SysInf.numSpecies;
4 numReactions    = SysInf.numReactions;
5 speNames        = SysInf.speNames;
6 speValues       = SysInf.speValues;
7 cNames          = SysInf.cNames;
8 cValues         = SysInf.cValues;
9 speConstIndecies = SysInf.speConstIndecies;
10 totalsIndecies  = SysInf.totalsIndecies;
11 VHolder         = SysInf.VHolder;
12
13 % initial values
14 X = speValues;
15
16 % extract V from VHolder and display
17 V = VHolder.V;
18
19 if verbose_flag
20     disp( sprintf('Stoichiometric Matrix:\n') ); disp(V);
21 end
22
23 % matrices to hold propensity values, number of species present after each step, and the
24 length of each step
25 A = zeros(numReactions,1);
26
27 % Actually do SSA ----- %
28 [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues,
29 tfinal, recordStep, verbose_flag);
30 % ----- %
31
32 if verbose_flag
33     disp(sprintf('\nSpecies' final amounts:\n'));
34     Amount = X;
35     dataTable = table(Amount, 'RowNames', speNames);
36     disp(dataTable);
37 end
38
39 if verbose_flag
40     disp(' ');
41 end
```

```
1 function [time, Y] = SSAGen(SysInf, tfinal, recordStep, verbose_flag, tau)
2
3 numSpecies      = SysInf.numSpecies;
4 numReactions    = SysInf.numReactions;
5 speNames        = SysInf.speNames;
6 speValues       = SysInf.speValues;
7 cNames          = SysInf.cNames;
8 cValues         = SysInf.cValues;
9 speConstIndecies = SysInf.speConstIndecies;
10 totalsIndecies  = SysInf.totalsIndecies;
11 VHolder         = SysInf.VHolder;
12
13 % initial values
14 X = speValues;
15
16 % extract V from VHolder and display
17 V = VHolder.V;
18 if verbose_flag
19     disp( sprintf('Stoichiometric Matrix:\n') ); disp(V);
20 end
21
22 % attempt to pick tau if not specified - *extremely* crude estimate
23 if tau == 0
24     % Single SSA trajectory to help determine good tau
25     [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues, tfinal, recordStep, verbose_flag);
26     tau = ( time(length(time)) / ( length(time)*recordStep ) ) * 3 ;
27 end
28
29 if verbose_flag
30     disp( sprintf('Chosen value for tau:\n') ); disp(tau);
31 end
32
33 X = speValues;
34
35 tic
36 [Y, X, time, run_time] = SingleTrajectory_cle(V, X, speConstIndecies, numSpecies, speValues, tfinal, recordStep, verbose_flag, tau);
37 time_with_leap = toc;
38
39 if verbose_flag
40     disp(sprintf('\nSpecies'' final amounts:\n'));
41     Amount = X;
42     dataTable = table(Amount, 'RowNames', speNames);
43     disp(dataTable);
44 end
45
46 if verbose_flag
47     disp(' ');
48 end
49
50 end
```

```

1 function varargout = SSAGen_parfor(SysInf, tfinal, recordStep, verbose_flag, speciesToGraph, numSteps, graph_flag)
2
3 numSpecies      = SysInf.numSpecies;
4 numReactions    = SysInf.numReactions;
5 speNames       = SysInf.speNames;
6 speValues      = SysInf.speValues;
7 cNames         = SysInf.cNames;
8 cValues        = SysInf.cValues;
9 speConstIndecies = SysInf.speConstIndecies;
10 totalsIndecies  = SysInf.totalsIndecies;
11 VHolder        = SysInf.VHolder;
12
13 if verbose_flag
14     disp(' ');
15 end
16
17 % set max muber of data points for each chunk of the recorded values matrix
18 numMaxDataPoints = 10008;
19
20 % set X to initial values
21 X = speValues;
22
23 % matrix to hold propensity values, number of constant species
24 speConstIndeciesLength = length(speConstIndecies);
25 A = zeros(numReactions,1);
26
27 % extract V from VHolder and display
28 V = VHolder.V;
29 if verbose_flag
30     disp( sprintf('\nStoichiometric Matrix:\n') ); disp(V);
31 end
32
33 num_cores = feature('numCores');
34
35 if verbose_flag
36     disp( sprintf('\nDetected %d CPU cores\n', num_cores) );
37 end
38
39 %Y = zeros(numMaxDataPoints, numSpecies);
40 Y = zeros(numSpecies, numSteps, numMaxDataPoints);
41 %time = zeros(1, numMaxDataPoints);
42
43 % begin SSA algorithm
44
45 if verbose_flag
46     disp( sprintf('\n===== STARTING SSA FOR HISTOGRAM =====' ) );
47     disp( sprintf('Remember - this part takes a while. Please be patient.\n') );
48 end
49
50 halt_flag = 0;
51 time = linspace(0,tfinal,numSteps);
52 interval = tfinal/numSteps;
53
54 parfor l = 1:numMaxDataPoints
55
56     % initial values
57     t = 0;

```

```

58     n = 2;
59     X = speValues;
60     A = zeros(numReactions,1);
61     Y_sub = zeros(numSpecies, numSteps);
62     Y_sub(:,1) = X;
63
64     while n <= numSteps
65
66         next_step = (n-1)*interval;
67
68         %-----
69         % calculate value of each propensity at that step
70         A = calculatePropensities(X);
71         %-----
72
73         asum = sum(A);
74
75         % break out of simulation if all species are consumed
76         if asum == 0
77             halt_flag = 1;
78             break
79         end
80
81         j = find( rand < cumsum(A/asum) , 1 );
82         tau = log(1/rand)/asum;
83
84         X = X + V(:,j);
85
86         %-----
87         X = calculateSpecifiedTotals(X);
88         %-----
89
90         for i = 1:speConstIndeciesLength
91             X(speConstIndecies(i)) = speValues(speConstIndecies(i));
92         end
93
94         t = t + tau;
95
96         if t > next_step
97             Y_sub(:,n) = X';
98             n = n + 1;
99         end
100
101     end
102
103     Y(:, :, l) = Y_sub;
104
105 end
106
107 if halt_flag && verbose_flag
108     disp(sprintf('\n=====REACTION HALTED - ALL SPECIES COMSUMED=====\\n'));
109 end
110
111 if verbose_flag
112     disp(' ');
113 end
114
115 Y_last = zeros(numMaxDataPoints, numSpecies);

```

```

116
117 % get means, standard deviations, last slice data
118 Mean = zeros(numSpecies, numSteps);
119 Std = zeros(numSpecies, numSteps);
120 for i = 1:numSpecies
121     for j = 1:numSteps
122         data = Y(i,j,:);
123         if j == numSteps
124             Y_last(:,i) = data;
125         end
126         Mean(i,j) = mean(data);
127         Std(i,j) = std(data);
128     end
129 end
130
131 specific_species_flag = 0;
132
133 if ~isempty(speciesToGraph)
134     stop_point = length(speciesToGraph);
135     specific_species_flag = 1;
136 else
137     stop_point = numSpecies;
138 end
139
140 warning('off','MATLAB:legend:IgnoringExtraEntries');
141
142 if graph_flag
143     for i = 1:stop_point
144         if specific_species_flag
145             index = speciesToGraph(i);
146         else
147             index = i;
148         end
149
150         data = zeros(numSteps, numMaxDataPoints);
151         data(1:numSteps,1:numMaxDataPoints) = Y(index,:,:);
152         data = data';
153
154         figure;
155
156         % graph boxplot for each slice
157         subplot(2,1,1);
158         boxplot( data , time, 'plotstyle', 'compact');
159         legend( findobj(gca,'Tag','Box'),speNames(index) );
160         xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
161         ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
162         title('Box and whisker plot of species vs time at given intervals','FontSize',16,'FontName', 'Helvetica');
163
164         % graph means +- standard deviations for each slice
165         subplot(2,1,2);
166         hold all
167         plot( time, Mean(index,:), 'b-o');
168         plot( time, Mean(index,:) - Std(index,:), 'c');
169         plot( time, Mean(index,:) + Std(index,:), 'c');
170         legend( findobj(gca,'Tag','Box'),speNames(index) );
171         xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
172         ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');

```

```
173         title('Box and whisker plot of species vs time at given intervals','FontSize',16, 'FontName', 'Helvetica');
174     end
175 end
176
177 Mean_last = Mean(:,numSteps);
178 Std_dev_last = Std(:,numSteps);
179
180 varargout{1} = Y_last;
181 varargout{2} = Mean;
182 varargout{3} = Std;
183
184 if verbose_flag
185     dataTableMean = table(Mean_last, 'RowNames', speNames);
186     disp(dataTableMean);
187     dataTableStddev = table(Std_dev_last, 'RowNames', speNames);
188     disp(dataTableStddev);
189 end
190
191 end
```



```

1 function varargout = SSAGen_parfor_cle(SysInf, tfinal, recordStep, verbose_flag, tau,
speciesToGraph, numSteps, graph_flag)
2
3 numSpecies      = SysInf.numSpecies;
4 numReactions    = SysInf.numReactions;
5 speNames       = SysInf.speNames;
6 speValues      = SysInf.speValues;
7 cNames         = SysInf.cNames;
8 cValues        = SysInf.cValues;
9 speConstIndecies = SysInf.speConstIndecies;
10 totalsIndecies  = SysInf.totalsIndecies;
11 VHolder         = SysInf.VHolder;
12
13 if verbose_flag
14     disp(' ');
15 end
16
17 % set max muber of data points for each chunk of the recorded values matrix
18 numMaxDataPoints = 10008;
19
20 % set X to initial values
21 X = speValues;
22
23 % matrix to hold propensity values, number of constant species
24 speConstIndeciesLength = length(speConstIndecies);
25 A = zeros(numReactions,1);
26
27 % extract V from VHolder and display
28 V = VHolder.V;
29 if verbose_flag
30     disp( sprintf('\nStoichiometric Matrix:\n') ); disp(V);
31 end
32
33 if tau == 0
34     % Single SSA trajectory to help determine good tau
35     [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues,
tfinal, recordStep, verbose_flag);
36     tau = ( time(length(time)) / ( length(time)*recordStep ) ) * 3 ;
37 end
38
39 num_cores = feature('numCores');
40
41 if verbose_flag
42     disp( sprintf('\nDetected %d CPU cores\n', num_cores) );
43 end
44
45 %Y = zeros(numMaxDataPoints, numSpecies);
46 Y = zeros(numSpecies, numSteps, numMaxDataPoints);
47 %time = zeros(1, numMaxDataPoints);
48
49 % begin SSA with tau-leaping algorithm
50
51 if verbose_flag
52     disp( sprintf('\n===== STARTING parallel SSA with tau-leaping =====' )
);
53     disp( sprintf('Remember - this part takes a while. Please be patient.\n') );
54 end
55

```

```

56 halt_flag = 0;
57 time = linspace(0,tfinal,numSteps);
58 interval = tfinal/numSteps;
59
60 parfor l = 1:numMaxDataPoints
61     % initial values
62     t = 0;
63     n = 2;
64     X = speValues;
65     A = zeros(numReactions,1);
66     Y_sub = zeros(numSpecies, numSteps);
67     Y_sub(:,1) = X;
68
69     while n <= numSteps
70         next_step = (n-1)*interval;
71
72         %-----
73         % calculate value of each propensity at that step
74         A = calculatePropensities(X);
75         %-----
76
77         asum = sum(A);
78
79         % break out of simulation if all species are consumed
80         if asum == 0
81             halt_flag = 1;
82             break
83         end
84
85         % get sampling of random variables and sub into CLE formula
86         d = tau*A + sqrt( abs(tau*A) ) * randn;
87
88         % update values
89         X = X + V * d';
90
91         %-----
92         X = calculateSpecifiedTotals(X);
93         %-----
94
95         for i = 1:speConstIndeciesLength
96             X(speConstIndecies(i)) = speValues(speConstIndecies(i));
97         end
98
99         t = t + tau;
100
101         if t > next_step
102             Y_sub(:,n) = X';
103             n = n + 1;
104         end
105
106     end
107
108     Y(:, :, l) = Y_sub;
109
110 end
111
112
113

```

```

114 if halt_flag && verbose_flag
115     disp(sprintf('\n=====REACTION HALTED - ALL SPECIES CONSUMED=====\\n'));
116 end
117
118 if verbose_flag
119     disp(' ');
120 end
121
122 Y_last = zeros(numMaxDataPoints, numSpecies);
123
124 % get means, standard deviations, last slice data
125 Mean = zeros(numSpecies, numSteps);
126 Std = zeros(numSpecies, numSteps);
127 for i = 1:numSpecies
128     for j = 1:numSteps
129         data = Y(i,j,:);
130         if j == numSteps
131             Y_last(:,i) = data;
132         end
133         Mean(i,j) = mean(data);
134         Std(i,j) = std(data);
135     end
136 end
137
138 specific_species_flag = 0;
139
140 if ~isempty(speciesToGraph)
141     stop_point = length(speciesToGraph);
142     specific_species_flag = 1;
143 else
144     stop_point = numSpecies;
145 end
146
147 warning('off','MATLAB:legend:IgnoringExtraEntries');
148
149 if graph_flag
150     for i = 1:stop_point
151         if specific_species_flag
152             index = speciesToGraph(i);
153         else
154             index = i;
155         end
156
157         data = zeros(numSteps, numMaxDataPoints);
158         data(1:numSteps,1:numMaxDataPoints) = Y(index, :, :);
159         data = data';
160
161         figure;
162
163         % graph boxplot for each slice
164         subplot(2,1,1);
165         boxplot( data , time, 'plotstyle', 'compact');
166         legend( findobj(gca,'Tag','Box'),speNames(index) );
167         xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
168         ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
169         title('Box and whisker plot of species vs time at given intervals','FontSize',16,'FontName', 'Helvetica');
170

```

```
171     % graph means +- standard deviations for each slice
172     subplot(2,1,2);
173     hold all
174     plot( time, Mean(index,:), 'b-o');
175     plot( time, Mean(index,:) - Std(index,:), 'c');
176     plot( time, Mean(index,:) + Std(index,:), 'c');
177     legend( findobj(gca,'Tag','Box'),speNames(index) );
178     xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
179     ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
180     title('Box and whisker plot of species vs time at given intervals','FontSize',12,
16, 'FontName', 'Helvetica');
181     end
182 end
183
184 Mean_last = Mean(:,numSteps);
185 Std_dev_last = Std(:,numSteps);
186
187 varargout{1} = Y_last;
188 varargout{2} = Mean;
189 varargout{3} = Std;
190
191 if verbose_flag
192     dataTableMean = table(Mean_last, 'RowNames', speNames);
193     disp(dataTableMean);
194     dataTableStddev = table(Std_dev_last, 'RowNames', speNames);
195     disp(dataTableStddev);
196 end
197
198 end
```

```

1 function varargout = SSAGen_parfor_tauleap(SysInf, tfinal, recordStep, verbose_flag, tau,
speciesToGraph, numSteps, graph_flag, num_traj)
2
3 numSpecies      = SysInf.numSpecies;
4 numReactions    = SysInf.numReactions;
5 speNames       = SysInf.speNames;
6 speValues      = SysInf.speValues;
7 cNames         = SysInf.cNames;
8 cValues        = SysInf.cValues;
9 speConstIndecies = SysInf.speConstIndecies;
10 totalsIndecies  = SysInf.totalsIndecies;
11 VHolder        = SysInf.VHolder;
12
13 if verbose_flag
14     disp(' ');
15 end
16
17 % set max muber of data points for each chunk of the recorded values matrix
18 if num_traj == 0
19     numMaxDataPoints = 10008;
20 else
21     numMaxDataPoints = num_traj;
22
23 % set X to initial values
24 X = speValues;
25
26 % matrix to hold propensity values, number of constant species
27 speConstIndeciesLength = length(speConstIndecies);
28 A = zeros(numReactions,1);
29
30 % extract V from VHolder and display
31 V = VHolder.V;
32 if verbose_flag
33     disp( sprintf('\nStoichiometric Matrix:\n') ); disp(V);
34 end
35
36 if tau == 0
37     % Single SSA trajectory to help determine good tau
38     [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues,
tfinal, recordStep, verbose_flag);
39     tau = ( time(length(time)) / ( length(time)*recordStep ) ) * 3 ;
40 end
41
42 num_cores = feature('numCores');
43
44 if verbose_flag
45     disp( sprintf('\nDetected %d CPU cores\n', num_cores) );
46 end
47
48 %Y = zeros(numMaxDataPoints, numSpecies);
49 Y = zeros(numSpecies, numSteps, numMaxDataPoints);
50 %time = zeros(1, numMaxDataPoints);
51
52 % begin SSA with tau-leaping algorithm
53
54 if verbose_flag
55     disp( sprintf('\n===== STARTING parallel SSA with tau-leaping =====' )
);

```

```
56     disp( sprintf( 'Remember - this part takes a while. Please be patient.\n') );
57 end
58
59 halt_flag = 0;
60 time = linspace(0,tfinal,numSteps);
61 interval = tfinal/numSteps;
62
63 parfor l = 1:numMaxDataPoints
64
65     % initial values
66     t = 0;
67     n = 2;
68     X = speValues;
69     A = zeros(numReactions,1);
70     Y_sub = zeros(numSpecies, numSteps);
71     Y_sub(:,1) = X;
72
73     while n <= numSteps
74
75         next_step = (n-1)*interval;
76
77         %-----
78         % calculate value of each propensity at that step
79         A = calculatePropensities(X);
80         %-----
81
82         asum = sum(A);
83
84         % break out of simulation if all species are consumed
85         if asum == 0
86             halt_flag = 1;
87             break
88         end
89
90         % get sampling of poisson random variables
91         pois_rand_vars = poissrnd(A*tau);
92
93         % update values
94         X = X + V * pois_rand_vars';
95
96         %-----
97         X = calculateSpecifiedTotals(X);
98         %-----
99
100        for i = 1:speConstIndeciesLength
101            X(speConstIndecies(i)) = speValues(speConstIndecies(i));
102        end
103
104        t = t + tau;
105
106        if t > next_step
107            Y_sub(:,n) = X';
108            n = n + 1;
109        end
110
111    end
112
113    Y(:, :, l) = Y_sub;
```

```
114
115 end
116
117 if halt_flag && verbose_flag
118     disp(sprintf('\n=====REACTION HALTED - ALL SPECIES CONSUMED=====\\n'));
119 end
120
121 if verbose_flag
122     disp(' ');
123 end
124
125 Y_last = zeros(numMaxDataPoints, numSpecies);
126
127 % get means, standard deviations, last slice data
128 Mean = zeros(numSpecies, numSteps);
129 Std = zeros(numSpecies, numSteps);
130 for i = 1:numSpecies
131     for j = 1:numSteps
132         data = Y(i,j,:);
133         if j == numSteps
134             Y_last(:,i) = data;
135         end
136         Mean(i,j) = mean(data);
137         Std(i,j) = std(data);
138     end
139 end
140
141 specific_species_flag = 0;
142
143 if ~isempty(speciesToGraph)
144     stop_point = length(speciesToGraph);
145     specific_species_flag = 1;
146 else
147     stop_point = numSpecies;
148 end
149
150 warning('off','MATLAB:legend:IgnoringExtraEntries');
151
152 if graph_flag
153     for i = 1:stop_point
154         if specific_species_flag
155             index = speciesToGraph(i);
156         else
157             index = i;
158         end
159
160         data = zeros(numSteps, numMaxDataPoints);
161         data(1:numSteps,1:numMaxDataPoints) = Y(index,:,:);
162         data = data';
163
164         figure;
165
166         % graph boxplot for each slice
167         subplot(2,1,1);
168         boxplot( data , time, 'plotstyle', 'compact');
169         legend( findobj(gca,'Tag','Box'),speNames(index) );
170         xlabel('Time','FontSize',12,'FontName','Helvetica');
171         ylabel('Number of Species','FontSize',12,'FontName','Helvetica');
```

```
172         title('Box and whisker plot of species vs time at given intervals','FontSize',↵
16,'FontName', 'Helvetica');
173
174         % graph means +- standard deviations for each slice
175         subplot(2,1,2);
176         hold all
177         plot( time, Mean(index,:), 'b-o');
178         plot( time, Mean(index,:) - Std(index,:), 'c');
179         plot( time, Mean(index,:) + Std(index,:), 'c');
180         legend( findobj(gca,'Tag','Box'),speNames(index) );
181         xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
182         ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
183         title('Box and whisker plot of species vs time at given intervals','FontSize',↵
16,'FontName', 'Helvetica');
184     end
185 end
186
187 Mean_last = Mean(:,numSteps);
188 Std_dev_last = Std(:,numSteps);
189
190 varargout{1} = Y_last;
191 varargout{2} = Mean;
192 varargout{3} = Std;
193
194 if verbose_flag
195     dataTableMean = table(Mean_last,'RowNames',speNames);
196     disp(dataTableMean);
197     dataTableStddev = table(Std_dev_last,'RowNames',speNames);
198     disp(dataTableStddev);
199 end
200
201 end
```



```
1 function [time, Y] = SSAGen(SysInf, tfinal, recordStep, verbose_flag, tau)
2
3 numSpecies      = SysInf.numSpecies;
4 numReactions    = SysInf.numReactions;
5 speNames        = SysInf.speNames;
6 speValues       = SysInf.speValues;
7 cNames          = SysInf.cNames;
8 cValues         = SysInf.cValues;
9 speConstIndecies = SysInf.speConstIndecies;
10 totalsIndecies  = SysInf.totalsIndecies;
11 VHolder         = SysInf.VHolder;
12
13 % initial values
14 X = speValues;
15
16 % extract V from VHolder and display
17 V = VHolder.V;
18 if verbose_flag
19     disp( sprintf('Stoichiometric Matrix:\n') ); disp(V);
20 end
21
22 % attempt to pick tau if not specified - *extremely* crude estimate
23 if tau == 0
24     % Single SSA trajectory to help determine good tau
25     [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues,
26 tfinal, recordStep, verbose_flag);
27     tau = ( time(length(time)) / ( length(time)*recordStep ) ) * 3 ;
28 end
29 disp( sprintf('Chosen value for tau:\n') ); disp(tau);
30
31 X = speValues;
32
33 tic
34 [Y, X, time, run_time] = SingleTrajectory_tauleap(V, X, speConstIndecies, numSpecies,
35 speValues, tfinal, recordStep, verbose_flag, tau);
36 time_with_leap = toc;
37
38 if verbose_flag
39     disp(sprintf('\nSpecies'' final amounts:\n'));
40     Amount = X;
41     dataTable = table(Amount, 'RowNames', speNames);
42     disp(dataTable);
43 end
44
45 if verbose_flag
46     disp(' ');
47 end
48 end
```

```

1 classdef StoichiometricMatricesHolder
2
3     properties
4         V;
5         vNumOfReactant;
6         vReactant;
7         vDimerMap;
8     end
9
10    methods
11
12        function object = StoichiometricMatricesHolder(SBMLModel)
13
14            numReactions = length(SBMLModel.reaction);
15            numSpecies = length(SBMLModel.species);
16
17            % get the stoichiometric matrix representing the species changes for each reaction
18            V = zeros(numSpecies, numReactions);
19
20            % matrices to hold the number of reactants in each reaction, the reactants'
21            % indices, and the dimer map
22            vNumOfReactant = zeros(numReactions,1);
23            vReactant = zeros(numReactions, numSpecies);
24            vDimerMap = zeros(numSpecies, numReactions);
25
26            maxCount = 0;
27
28            for j = 1:numReactions
29
30                count = 0;
31                for i = 1:numSpecies
32
33                    role = DetermineSpeciesRoleInReaction(SBMLModel.species(i), SBMLModel.
34                    reaction(j));
35
36                    if length(role) > 1
37
38                        V(i,j) = role(1) - role(2);
39
40                        if role(2) > 0
41                            vReactant( j , (count+1) ) = i;
42                            count = count + 1;
43                        end
44
45                        if role(2) >= 2
46                            vDimerMap(i,j) = role(2);
47                        end
48                    else
49                        V(i,j) = 0;
50                    end
51                end
52
53                vNumOfReactant(j) = count;
54
55                if count > maxCount
56                    maxCount = count;
57                end
58            end
59        end
60    end
61 end

```

```
56         end
57
58         object.V = V;
59         object.vNumOfReactant = vNumOfReactant;
60
61         % truncate reactant index matrix to discard unnecessary elements
62         object.vReactant = vReactant( : , 1:(maxCount) );
63
64         % make dimer reactant matrix sparse to discard unnecessary elements
65         object.vDimerMap = vDimerMap;
66
67     end
68
69 end
70
71 end
```

```
1 classdef SystemInformationHolder
2
3     properties
4         numSpecies;
5         numReactions;
6         speNames;
7         speValues;
8         cNames;
9         cValues;
10        speConstIndecies;
11        totalsIndecies;
12        VHolder;
13    end
14
15 end
```