```matlab
% Generates a file 'calculatePropensities.m' that will contain a function able to calculate the necessary propensities for that system.
function GeneratePropensityCalculatorFile(SBMLModel, VHolder)

% determine number of reactions and species present in the model
numReactions = length(SBMLModel.reaction);
numSpecies = length(SBMLModel.species);

[cNames, cValues] = GetParameters(SBMLModel);

% matricies to hold propensity values
A = zeros(numReactions,1);

V = VHolder.V;
vNumOfReactant = VHolder.vNumOfReactant;
vReactant = VHolder.vReactant;
vDimerMap = VHolder.vDimerMap;

fid = fopen('calculatePropensities.m','w');
fprintf(fid,'function A = calculatePropensities(X)\n\n');

for i = 1:numReactions

    % set initially to that reaction's parameter
    format long;
    fprintf(fid, 'A(%d) = (%d)', i, cValues(i) );

    % multiply current value by each ractant's value if applicable, and account for dimerisation reactions
    for k = 1:(vNumOfReactant(i));

        curSpeciesIndex = vReactant(i,k);

        fprintf(fid,'*X(%d)', curSpeciesIndex);

        % determine is reactant is part of a dimerisation reaction using dimerisation map, then alter propensity accordingly
        dimer_number = vDimerMap(curSpeciesIndex, i);
        if dimer_number > 1
            count = 1;
            while (count < dimer_number)
                fprintf(fid,'*(X(%d)-%d)', curSpeciesIndex, count);
                count = count + 1;
            end

            fprintf(fid,'/%d', factorial(dimer_number) );
        end
    end

    fprintf(fid, ';\n', i, cValues(i) );
end

fprintf(fid,'\nend',numReactions);

fclose(fid);

end
```

```
function totalsIndecies =  GenerateSpecifiedTotalsCalculatorFile(SBMLModel)

numReactions = length(SBMLModel.reaction);
numSpecies = length(SBMLModel.species);

[speNames, speValues] = GetSpecies(SBMLModel);

totalsIndecies = zeros(numSpecies,1);

fid = fopen('calculateSpecifiedTotals.m','w');
fprintf(fid,'function X = calculateSpecifiedTotals(X)\n\n');

count = 0;
for i = 1:length(SBMLModel.rule)
    curRule = SBMLModel.rule(i);

    for j = 1:length( speNames )
        if strcmp( speNames(j), curRule.variable )

            totalsIndecies(count+1) = j;
            count = count + 1;
            fprintf(fid, 'X(%d) = 0', j);

            % token string array
            ruleToks = strsplit( curRule.formula ,'+');

            for l = 1:length( ruleToks )
                for m = 1:length(speNames)
                    if strcmp( speNames(m), ruleToks(l) )
                        fprintf(fid, ' + X(%d)', m);
                    end
                end
            end

            fprintf(fid, ';\n', m);

            break;
        end
    end
end

fprintf(fid,'\nend',numReactions);

fclose(fid);

totalsIndecies = totalsIndecies(1:count);

end
```

```matlab
function speConstIndecies = GetConstantSpeciesIndecies(SBMLModel)

numSpecies = length(SBMLModel.species);

speConstIndecies = zeros(numSpecies,1);

count = 0;
for i = 1:numSpecies
    if SBMLModel.species(i).constant
        speConstIndecies(count + 1) = i;
        count = count + 1;
    end
end

speConstIndecies = speConstIndecies(1:count,1);

end
```

```matlab
function [cNames, cValues] = GetParameters(SBMLModel)

numReactions = length(SBMLModel.reaction);

% get all parameter names, values
[allCNames, allCValues] = GetAllParameters(SBMLModel);

% create matricies to hold parameter names, values
cValues = zeros(numReactions,1);
cNames = cell(numReactions,1);

% get parameters for each individual reaction as they may not be in order
for i = 1:numReactions

    % attempt to get parameters from local rection context
    [cNamesTemp,cValuesTemp] = GetParameterFromReaction(SBMLModel.reaction(i));

    % if parameter values are NOT embedded in each local reaction context (return values will be NULL)
    if ( length(cNamesTemp) == 0 ) && ( length(cValuesTemp) == 0 )

        % declare parameter found flag, get string from reaction formula field, tokenize it by operator
        done = 0;
        kinLawString = SBMLModel.reaction(i).kineticLaw.formula;
        kinLawStringToks = strsplit( kinLawString ,'*');

        % searches for each token in the list of all parameters in the model, assigns those values to the correct reaction if found
        for j = 1:length(kinLawStringToks)
            curTok = kinLawStringToks(j);
            for k = 1:length(allCNames);
                if strcmp( allCNames{k} , curTok )
                    cNames{i} = allCNames{k};
                    cValues(i) = allCValues(k);
                    done = 1;
                    break;
                end

                if done == 1
                    break;
                end
            end

            if done == 1
                break;
            end
        end
    % if parameters were embedded in the local reaction contexts, assign them
    else
        cNames(i) = cNamesTemp;
        cValues(i) = cValuesTemp;
    end
end

end
```

```
function GraphHist(Y, speciesToGraph, speNames, split_flag, filename, method_name)


num_bins = length(Y)^(1/3);

specific_species_flag = 0;
speIndLength = length(speciesToGraph);

if speIndLength == 0
    end_point = min ( size(Y) );
else
    end_point = speIndLength;
    specific_species_flag = 1;
end

if ~split_flag
    figure
    hold all
end

for i = 1:end_point

    if specific_species_flag
        current_species = speciesToGraph(i);
    else
        current_species = i;
    end


    min_val = min( Y(:,current_species) );
    max_val = max( Y(:,current_species) );

    % determine optimal number of bins (educated guess)
    space = max_val - min_val + 1;
    if space > num_bins
        bins = linspace(min_val, max_val, num_bins);
        h = hist( Y(:,current_species) , num_bins );
    else
        bins = linspace(min_val, max_val, space);
        h = hist( Y(:,current_species) , space );
    end

    if split_flag
        figure
    end

    plot( bins, h/length(Y) , 'o-' );

    if split_flag
        legend( speNames(current_species) );
        xlabel('Number of Species','FontSize',12, 'FontName', 'Helvetica');
        ylabel('Frequency','FontSize',12,'FontName', 'Helvetica');
        title('Histogram of number of species at end of simulation','FontSize',16,'FontName',↙
'Helvetica');
    end
end

if ~split_flag
```

```
    hold off

    if specific_species_flag
        legend( speNames(speciesToGraph) );
    else
        legend( speNames )
    end

    xlabel('Number of Species','FontSize',12, 'FontName', 'Helvetica');
    ylabel('Frequency','FontSize',12,'FontName', 'Helvetica');

    title_string = ['Histogram of number of species at end of simulation from model source ''' filename ''' using ' method_name];
    title(title_string, 'Fontsize', 16,'FontName', 'Helvetica');
end

end
```

```
function GraphResults(Y, time, speciesToGraph, speNames, split_flag, filename, method_name)

specific_species_flag = 0;
speIndLength = length(speciesToGraph);

if speIndLength == 0
    dims = size(Y);
    end_point = dims(1);
else
    end_point = speIndLength;
    specific_species_flag = 1;
end

if ~split_flag
    figure
    hold all
end

for i = 1:end_point

    if specific_species_flag
        current_species = speciesToGraph(i);
    else
        current_species = i;
    end

    if split_flag
        figure
    end

    plot( time , Y(current_species,:) );

    if split_flag
        legend( speNames(current_species) );
        xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
        ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
        title('Species vs time using SSA','FontSize',16,'FontName', 'Helvetica');
    end
end

if ~split_flag
    hold off

    if specific_species_flag
        legend( speNames(speciesToGraph) );
    else
        legend( speNames )
    end

    xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
    ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');

    title_string = ['Species vs time from model source ''' filename ''' using ' method_name];
    title(title_string,'FontSize',16,'FontName', 'Helvetica');
end

end
```

```
function varargout = MARS(filename, varargin)

% Options:
%
% 'Hist'    - generate a histogram of the results of 10,000 trajecories
% 'Verbose' - enable diagnostic outputs
% 'Time'    - max time to run the simulation for
% 'Record'  - step size between recording simulation state information
% 'Tau'     - use tau-leaping
% 'CLE'     - use Chemical Langevin Equation
% 'RRE'     - use raction rate equations
% 'GPU'     - use the system's CUDA-supported GPU for multiple trajectory generation
% 'MLMC'    - generate histogram using Multi-level Monte-Carlo simulation (experimental)
% 'Error'   - error to use for MLMC method
% 'Steps'   - number of data points to generate over the integration interval for all non-GPU↙
parallel methods
% 'Graph'   - plot a graph of the results

% default values for user-provided arguments
hist_flag       = 0;
verbose_flag    = 0;
tau_flag        = 0;
cle_flag        = 0;
rre_flag        = 0;
stiff_flag      = 0;
split_flag      = 0;
keep_flag       = 0;
record_flag     = 0;
gpu_flag        = 0;
graph_flag      = 0;
mlmc_flag       = 0;
m_flag          = 0;
err_flag        = 0;
steps_flag      = 0;
tfinal          = 50;
recordStep      = 20;
numSteps        = 100;
err             = 0;
speciesToGraph  = [];

i = 1;
while (1+i) <= nargin
    switch varargin{i}
        case 'Hist'
            hist_flag = 1;

        case 'Verbose'
            verbose_flag = 1;

        case 'Time'
            if (i+2) > nargin
                disp( sprintf('\nSimulation time argument missing.\n') );
                return;
            else
                i = i + 1;
                time_arg = varargin{i};
                if ~isnumeric(time_arg)
                    disp( sprintf('\nSimulation time argument must be a number.\n') );
                    return;
                else
                    tfinal = time_arg;
                end
            end

        case 'Record' % get record step argument, check for validity (integer)
            if (i+2) > nargin
                disp( sprintf('\nRecord step size argument missing.\n') );
                return;
            else
                i = i + 1;
                record_arg = varargin{i};
                if ~isnumeric(record_arg) || mod(record_arg,1) ~= 0
                    disp( sprintf('\nRecord step size argument must be an integer.\n') );
                    return;
                else
                    recordStep = record_arg;
                    record_flag = 1;
                end
            end

        case 'Tau' % use tau-leaping, get value to use for tau
            if (i+2) > nargin
                tau = 0;
            else
                next_arg = varargin{i+1};
                if isnumeric(next_arg)
                    tau = next_arg;
                    i = i + 1;
                else
                    tau = 0;
                end
            end
            tau_flag = 1;

        case 'CLE' % use Langevin leaping algorithm, get value to use for tau
            if (i+2) > nargin
                tau = 0;
            else
                next_arg = varargin{i+1};
                if isnumeric(next_arg)
                    tau = next_arg;
                    i = i + 1;
                else
                    tau = 0;
                end
            end
            cle_flag = 1;

        case 'MLMC' % use MLMC method, get value to use for M, defaults to 100 steps
            if (i+2) > nargin
                M = 0;
            else
                m_arg = varargin{i+1};
                if isnumeric(m_arg) && mod(m_arg,1) ~= 0
                    disp( sprintf('\nMLMC M-value argument must be an integer.\n') );
                    return;
                elseif isnumeric(m_arg) && mod(m_arg,1) == 0
                    M = m_arg;
                    i = i + 1;
                    m_flag = 1;
                else
                    M = 0;
                end
            end
            mlmc_flag = 1;

        case 'Steps' % get specific numer of steps to generate for parallel methods not on a↙
GPU
            if (i+2) > nargin
                disp( sprintf('\nNumber of steps size argument missing.\n') );
                return;
            else
                step_arg = varargin{i+1};
                if isnumeric(step_arg) && mod(step_arg,1) ~= 0
                    disp( sprintf('\nNumber of steps argument must be an integer.\n') );
                    return;
                elseif isnumeric(step_arg) && mod(step_arg,1) == 0
                    numSteps = step_arg;
                    i = i + 1;
                else
                    disp( sprintf('\nNumber of steps size argument missing or invalid.\n') );
                    return;
                end
            end

        case 'Error' % get error to use for MLMC method, overrides default
            if (i+2) > nargin
                disp( sprintf('\nError argument missing.\n') );
                return;
            else
                i = i + 1;
                err_arg = varargin{i};
                if ~isnumeric(err_arg)
                    disp( sprintf('\nError argument missing.\n') );
                    return;
                elseif isnumeric(err_arg) && err_arg < 0
                    disp( sprintf('\nError must be a positive number.\n') );
                else
                    err = err_arg;
                end
            end

        case 'Graph' % whether or not to grapht the results, and if so for which species↙
(default is all)
            if (i+2) > nargin
                speciesToGraph = [];
            else
                next_arg = varargin{i+1};
                if isvector(next_arg) && isnumeric(next_arg)
                    speciesToGraph = next_arg;
                    i = i + 1;
                else
                    speciesToGraph = [];
                end
            end
            graph_flag = 1;

        case 'RRE'
            rre_flag = 1;

        case 'Stiff'
            stiff_flag = 1;

        case 'Split'
            split_flag = 1;

        case 'Keep'
            keep_flag = 1;

        case 'GPU'
            gpu_flag = 1;

        otherwise
            disp( sprintf('\nInvalid option detected at argument %d.\n',i) );
            return;
    end
    i = i + 1;
end
if (tau_flag + cle_flag + rre_flag + mlmc_flag + gpu_flag) > 1
    disp( sprintf('\nInvalid options: multiple methods selected. You may only pick one of↙
CLE, Tau-Leaping, RRE, MLMC, or GPU\n') );
    return
end

if tfinal == 0
    disp( sprintf('\nInvalid final time argument\n') );
    return
end

if recordStep == 0
    disp( sprintf('\nInvalid record step size argument\n') );
    return
end

if numSteps == 0
    disp( sprintf('\nInvalid number of steps argument\n') );
    return
end

if m_flag && M < 2
    disp( sprintf('\nMLMC M-value must be an integer greater than 1\n') );
    return
end

addpath(genpath('./toolbox/SBMLToolbox'));

platform_str = computer;

% get type of platform so proper libraries can be added to path, currently only PC, Mac↙
supported
switch platform_str
    case 'MACI64'
```

```
        addpath(genpath('./toolbox/libSBML/mac'));
    case 'PCWIN'
        addpath(genpath('./toolbox/libSBML/win32'));
    case 'PCWIN64'
        addpath(genpath('./toolbox/libSBML/win64'));
    case 'GLNXA64'
        addpath(genpath('./toolbox/libSBML/linux'));
    otherwise
        disp('Platform not supported');
        return;
end

% create files to be filled, then close all
file_list = {'calculatePropensities.m';...
            'calculateSpecifiedTotals.m';...
            'RRE_functions.m';...
            'fireGpuTrajectories.m'};

num_files = length(file_list);
for i = 1:num_files
    cur_file = file_list{i};
    fid = fopen(cur_file,'w');
    fclose(fid);
end

% get system (model) inforamtion from SBML file
SysInf = SSA_setup(filename, verbose_flag);
numSpecies        = SysInf.numSpecies;
numReactions      = SysInf.numReactions;
speNames          = SysInf.speNames;
speValues         = SysInf.speValues;
cNames            = SysInf.cNames;
cValues           = SysInf.cValues;
speConstIndecies  = SysInf.speConstIndecies;
totalsIndecies    = SysInf.totalsIndecies;
VHolder           = SysInf.VHolder;

% check for bad species-to-graph entries
if graph_flag && ~mlmc_flag
    numSpeToGraph = length(speciesToGraph);
    if numSpeToGraph ~= 0
        for i = 1:numSpeToGraph

            curIndex = speciesToGraph(i);
            if curIndex > numSpecies || curIndex < 1
                disp( sprintf(['Error: invalid species index provided, exceeds number of↙
species in system or is less than 1.'...
                           ' Graph will not be displayed.\n']) );
                graph_flag = 0;
            end

            for j = 1:(i-1)
                checkIndex = speciesToGraph(j);
                if checkIndex == curIndex
                    disp( sprintf('Error: invalid species index provided, duplicate index.↙
Graph will not be displayed.\n') );
                    graph_flag = 0;
                end
            end

        end
    end
end

method_name = '';
rehash

if gpu_flag
    Y = SSA_gpu(filename, SysInf, tfinal, verbose_flag);
    method_name = 'SSA on GPU';

elseif mlmc_flag
    %open parallel pool based on installed toolbox version
    version_less_flag = verLessThan('distcomp', '6.3');
    if version_less_flag
        matlabpool open;
    else
        parpool;
    end

    [Mean, Step] = MLMCGen(SysInf, tfinal, numSteps, verbose_flag, split_flag,↙
speciesToGraph, graph_flag, M, err);
    Y = Mean;
    varargout{3} = Step;
    time = linspace(0,tfinal,numSteps);
    varargout{2} = time;
    method_name = 'MLMC';

    % close parallel pool
    if version_less_flag
        matlabpool close;
    else
        delete(gcp);
    end

elseif hist_flag
    %open parallel pool based on installed toolbox version
    version_less_flag = verLessThan('distcomp', '6.3');
    if version_less_flag
        matlabpool open;
    else
        parpool;
    end

    % use indicated method to generate trajectories accross multiple CPUs or CPU cores
    if tau_flag
        [Y,Mean,Std] = SSAGen_parfor_tauleap(SysInf, tfinal, recordStep, verbose_flag, tau,↙
speciesToGraph, numSteps, graph_flag, 0);
        method_name = 'SSA with tau-leaping on parallel CPUs';
    elseif cle_flag
        [Y,Mean,Std] = SSAGen_parfor_cle(SysInf, tfinal, recordStep, verbose_flag, tau,↙
speciesToGraph, numSteps, graph_flag);
        method_name = 'CLE on parallel CPUs';
    elseif rre_flag
        disp( sprintf('\n''Hist'' is not a valid option to use with the Reaction Rate↙
Equation method\n') );
```

```
    else
        [Y,Mean,Std] = SSAGen_parfor(SysInf, tfinal, recordStep, verbose_flag,↙
speciesToGraph, numSteps, graph_flag);
        method_name = 'SSA on parallel CPUs';
    end

    varargout{2} = Mean;
    varargout{3} = Std;

    % close parallel pool
    if version_less_flag
        matlabpool close;
    else
        delete(gcp);
    end

else
    % generate single trajectory based on indicated method
    if tau_flag
        [time, Y] = SSAGen_tauleap(SysInf, tfinal, recordStep, verbose_flag, tau);
        method_name = 'SSA with tau-leaping';
    elseif cle_flag
        [time, Y] = SSAGen_cle(SysInf, tfinal, recordStep, verbose_flag, tau);
        method_name = 'CLE';
    elseif rre_flag
        if record_flag
            disp(sprintf('\nWarning: ''Record'' argument will be ignored - not valid with↙
Reaction Rate Equation method\n'));
        end
        [time, Y] = RREGen(SysInf, tfinal, verbose_flag, stiff_flag);
        method_name = 'RRE';
    else
        [time, Y] = SSAGen(SysInf, tfinal, recordStep, verbose_flag);
        method_name = 'SSA';
    end

    varargout{2} = time;
end

varargout{1} = Y;

arg_type = class(filename);
switch arg_type
    case 'struct'
        filename = filename.name;
end

% graph end of simulation histogram is using a parallel method and graph flag has been set
if graph_flag
    if gpu_flag || hist_flag
        GraphHist(Y, speciesToGraph, speNames, split_flag, filename, method_name);
    else
        GraphPlot(Y, time, speciesToGraph, speNames, split_flag, filename, method_name);
    end
end

% keep temporary files if indicated
if ~keep_flag
```

```
    for i = 1:num_files
        cur_file = file_list{i};
        delete(cur_file);
    end
end

end
```

```
function [Y, Step] = MLMCGen(SysInf, tfinal, numSteps, verbose_flag, split_flag, speciesToGraph, graph_flag, M, err)

numSpecies        = SysInf.numSpecies;
numReactions      = SysInf.numReactions;
speNames          = SysInf.speNames;
speValues         = SysInf.speValues;
cNames            = SysInf.cNames;
cValues           = SysInf.cValues;
speConstIndecies  = SysInf.speConstIndecies;
totalsIndecies    = SysInf.totalsIndecies;
VHolder           = SysInf.VHolder;

% initial values
X = speValues;
numSteps = numSteps - 1;

if err == 0
    err = 1/100;
end

% extract V from VHolder and display
V = VHolder.V;

if verbose_flag
    disp( sprintf('Stoichiometric Matrix:\n') ); disp(V);
end

% matricies to hold propensity values, number of species present after each step, and the length of each step
A = zeros(numReactions,1);

% parameters
N = max(X);

% default M
if M == 0
    M = 3;
end

alp = zeros(numSpecies,1);
for i = 1:numSpecies
    val = X(i);
    if val == 0
        alp(i) = 0;
    else
        alp(i) = log(val)/log(N);
    end
end

bet = zeros(numReactions,1);
for i = 1:numReactions
    val = cValues(i);
    if val ~= 0
        bet(i) = log(val)/log(N);
    end
end

```

```matlab
V_pos = -V;
V_pos(V_pos < 0) = 0;

% get gamma value from largest of candidates
gam = -Inf;
for i = 1:numSpecies
    for k = 1:numReactions
        if V(i,k) ~= 0
            gam_can = bet(k) + dot(V_pos(:,k),alp) - alp(i);
            if gam_can > gam
                gam = gam_can;
            end
        end
    end
end

% get rho value from largest of candidates
rho = Inf;
for k = 1:numReactions
    for i = 1:numSpecies
        if V(i,k) ~= 0
            rho_can = alp(i);
            if rho_can < rho
                rho = rho_can;
            end
        end
    end
end

L = ceil(abs(log(err)));

% should have three levels
if L <= 2
    l_0 = 0;
else
    l_0 = L - 2;
end

num_levels = L - l_0 + 1;

% get required number of coarsest trajectories
n_0 = 4 * ceil( (N^-rho * N^-gam * err^-2) / 4 );

% get level step sizes and required number of trajectories
h_l = zeros(num_levels, 1);
n_l = zeros(num_levels, 1);
for i = 1:num_levels
    l = l_0 + i - 1;
    h_l(i) = tfinal/(M^l);
    n_l(i) = ceil( N^-rho * N^gam * (L - l_0) * h_l(i) * err^-2 );
end

% make each n_l divisible by 4
for i = 1:num_levels
    val = n_l(i);
    while mod(val,4) ~= 0
        val = val+1;
    end
    n_l(i) = val;
end

% print information if required
if verbose_flag
    fprintf('N:\t%d\n', N);
    fprintf('Gamma:\t%d\n', gam);
    fprintf('Rho:\t%d\n', rho);
    fprintf('M:\t%d\n', M);
    fprintf('Error:\t%d\n', err);
    fprintf('Granularities:\n\n');
        disp(h_l);
    fprintf('Number of trajectories at each level:\n\n');
        disp(n_0);
        disp(n_l(2:num_levels));
end

num_trajectories = sum(n_l);

interval = tfinal/(numSteps+1);

% level 0
[~, Mean_coarse, ~] = SSAGen_parfor_tauleap(SysInf, tfinal, 0, 0, h_l(1), speciesToGraph,↙
numSteps+1, 0, n_0);

Y = Mean_coarse;
time = linspace(0,tfinal,numSteps+1);

% ----------------------------- % start MLMC

for i = 2:num_levels

    num_runs = n_l(i);

    Y_sub = zeros( num_runs , numSpecies, numSteps+1);

    parfor k = 1:num_runs

        % setup that level trajectory
        hl      = h_l(i);
        hl_1    = M*hl;
        l           = l_0 + i - 1;
        zl          = X;
        zl_1        = X;
        Zl          = zeros(numSpecies, numSteps+1);
        Zl_1        = zeros(numSpecies, numSteps+1);
        Zl(:,1)     = X;
        Zl_1(:,1)   = X;
        t           = 0;
        n           = 2;

        while n <= (numSteps+1)

            lam_bot = calculatePropensities(zl_1)';
            A = zeros(numReactions, 3);

            for j = 1:M

```

```
                lam_top = calculatePropensities(zl)';

                % (a)
                A(:,1) = min(lam_top, lam_bot);
                A(:,2) = lam_top - A(:,1);
                A(:,3) = lam_bot - A(:,1);

                % (b)
                Lam = poissrnd(A*hl);

                % (c)
                delta_top = Lam(:,1) + Lam(:,2);
                delta_bot = Lam(:,1) + Lam(:,3);
                zl = zl + V*delta_top;
                zl_1 = zl_1 + V*delta_bot;

                zl(speConstIndecies) = speValues(speConstIndecies);
                zl_1(speConstIndecies) = speValues(speConstIndecies);
                zl = calculateSpecifiedTotals(zl);
                zl_1 = calculateSpecifiedTotals(zl_1);

            end

            t = t + hl_1;

            % record
            if t > ((n-1)*interval)
                Zl(:,n) = zl;
                Zl_1(:,n) = zl_1;
                n = n + 1;
            end
        end

        data = Zl - Zl_1;
        Y_sub(k,:,:) = data;

    end

    Mean_level = zeros(numSpecies, numSteps+1);

    for i = 1:numSpecies
        for j = 1:(numSteps+1)
            Mean(i,j) = mean( Y_sub(:,i,j) );
        end
    end

    Y = Y + Mean_level;

end

Step = h_l(length(h_l));

end
```

```matlab
function [time, Y] = RREGen(SysInf, tfinal, verbose_flag, stiff_flag)

numSpecies         = SysInf.numSpecies;
numReactions       = SysInf.numReactions;
speNames           = SysInf.speNames;
speValues          = SysInf.speValues;
cNames             = SysInf.cNames;
cValues            = SysInf.cValues;
speConstIndecies   = SysInf.speConstIndecies;
totalsIndecies     = SysInf.totalsIndecies;
VHolder            = SysInf.VHolder;

if verbose_flag
    disp( sprintf('\nNumber of species types:\n') ); disp(numSpecies);
    disp( sprintf('\nNumber of reactions:\n') ); disp(numReactions);
end

if verbose_flag
    disp( sprintf('\nParameter names:\n') ); disp(cNames);
    disp( sprintf('\nParameter values:\n') ); disp(cValues);
end

if verbose_flag
    disp( sprintf('\nSpecies'' names:\n') ); disp(speNames);
    disp( sprintf('\nSpecies'' initial amounts:\n') ); disp(speValues);
end

% holder strings for RHS of RREs
for i = 1:numReactions
    A{i} = '';
end
A = A';

V = VHolder.V;
vNumOfReactant = VHolder.vNumOfReactant;
vReactant = VHolder.vReactant;
vDimerMap = VHolder.vDimerMap;

if verbose_flag
    disp( sprintf('Stoichiometric Matrix:\n') ); disp(V);
end

for i = 1:numReactions

    % set initially to that reaction's parameter
    format long;
    A{i} = strcat( A{i} , num2str( cValues(i) ) );

    % multiply current value by each reactant's value if applicable, and account for dimerisation reactions
    for k = 1:(vNumOfReactant(i));

        curSpeciesIndex = vReactant(i,k);

        A{i} = strcat( A{i} , sprintf('*X(%d)',curSpeciesIndex) );

        % determine if reactant is part of a dimerisation reaction using dimerisation map, then alter propensity accordingly
        dimer_number = vDimerMap(curSpeciesIndex, i);
        if dimer_number > 1
            for j = 1:(dimer_number-1)
                A{i} = strcat( A{i} , sprintf('*(X(%d)-%d)', curSpeciesIndex, j ) );
            end

            A{i} = strcat( A{i} , sprintf('/%d', factorial(dimer_number) ) );
        end
    end
end

fid = fopen('RRE_functions.m','w');
fprintf(fid,'function dXdt = RRE_functions(t,X)');

fprintf(fid,'\n\ndXdt = zeros(%d,1);\n\n', numSpecies);

for i = 1:numSpecies

    fprintf(fid,'dXdt(%d) = 0',i);

    if ~ismember(i,speConstIndecies)
        for j = 1:numReactions
            if V(i,j) ~= 0
                fprintf(fid, ' + %d*%s', V(i,j), A{j});
            end
        end
    end

    fprintf(fid,' ;\n');

end

fprintf(fid,'\nend');
fclose(fid);

%tspan = linspace(0,tfinal,tfinal);
tspan = [0 tfinal];

disp( sprintf('==========Starting Solver==========') )

if stiff_flag
    [time,y] = ode15s(@RRE_functions,tspan,speValues);
else
    [time,y] = ode45(@RRE_functions,tspan,speValues);
end

Y = y';

if length(totalsIndecies) ~= 0
    for i = 1:length(time)
        Y(:,i) = calculateSpecifiedTotals(Y(:,i));
    end
end

if verbose_flag
    disp(sprintf('\nSpecies'' final amounts:\n'));
    Amount = Y(:,length(Y));
    dataTable = table(Amount,'RowNames',speNames);
    disp(dataTable);
end

end
```

```
function [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies,↙
speValues, tfinal, recordStep, verbose_flag)

% set max muber of data points for each chunk of the recorded values matrix
numMaxDataPoints = 10000;

Y = zeros(numSpecies, numMaxDataPoints);
time = zeros(1, numMaxDataPoints);

% assign initial values recorded values
Y( : , 1 ) = X;
time(1) = 0;

% initial values
t = 0;
count = 1;

speConstIndeciesLength = length(speConstIndecies);

if verbose_flag
    disp( sprintf('\n=================STARTING SSA==============\n') );
end

tic

while t < tfinal

    %----------------------------------------------------------------
    % calculate value of each propensity at that step
    A = calculatePropensities(X);
    %----------------------------------------------------------------

    asum = sum(A);

    % break out of simulation if all species are consumed
    if asum == 0
        if verbose_flag
            disp(sprintf('\n=========REACTION HALTED - ALL SPECIES COMSUMED========\n'));
            disp(sprintf('Final time was %f', t) );
        end

        Y( : , ceil(count/recordStep) + 1 ) = X;
        time( ceil(count/recordStep) + 1 ) = t;

        break
    end

    j = min( find( rand < cumsum(A/asum) ) );
    tau = log(1/rand)/asum;

    X = X + V(:,j);

    %----------------------------------------------------------------
    X = calculateSpecifiedTotals(X);
    %----------------------------------------------------------------

    for i = 1:speConstIndeciesLength
        X(speConstIndecies(i)) = speValues(speConstIndecies(i));
    end

    t = t + tau;

    if mod(count, recordStep) == 0
        Y( : , (count/recordStep) + 1) = X;
        time(count/recordStep + 1) = t;
    end

    count = count + 1;
end

run_time = toc;

Y = Y( : ,1:(floor(count/recordStep)) );
time = time( 1:(floor(count/recordStep)) );

if verbose_flag
    disp( sprintf('\n%d steps taken\n', count ) );
    disp( sprintf('%d steps recorded\n', floor(count/recordStep) ) );
end

end
```

```
function [Y, X, time, run_time] = SingleTrajectory_cle(V, X, speConstIndecies, numSpecies,↙
speValues, tfinal, recordStep, verbose_flag, tau)

% set max muber of data points for each chunk of the recorded values matrix
numMaxDataPoints = 10000;

Y = zeros(numSpecies, numMaxDataPoints);
time = zeros(1, numMaxDataPoints);

% assign initial values recorded values
Y( : , 1 ) = X;
time(1) = 0;

% initial values
t = 0;
count = 1;

speConstIndeciesLength = length(speConstIndecies);

if verbose_flag
    disp( sprintf('\n=================STARTING SSA WITH CLE==============\n') );
end

tic

while t < tfinal

    %----------------------------------------------------------------
    % calculate value of each propensity at that step
    A = calculatePropensities(X);
    %----------------------------------------------------------------

    % break out of simulation if all species are consumed
    if cumsum(A) == 0
        if verbose_flag
            disp(sprintf('\n=========REACTION HALTED - ALL SPECIES COMSUMED========\n'));
            disp(sprintf('Final time was %f', t) );
        end

        Y( : , floor(count/recordStep) + 1) = X;
        time( floor(count/recordStep) + 1) = t;

        break
    end

    % get sampling of random variables and sub into CLE formula
    d = tau*A + sqrt( abs(tau*A) ) * rand;

    % update values
    X = X + V * d';

    %----------------------------------------------------------------
    X = calculateSpecifiedTotals(X);
    %----------------------------------------------------------------

    for i = 1:speConstIndeciesLength
        X(speConstIndecies(i)) = speValues(speConstIndecies(i));
    end

    t = t + tau;

    if mod(count, recordStep) == 0
        Y( : , (count/recordStep) + 1) = X;
        time(count/recordStep + 1) = t;
    end

    count = count + 1;
end

run_time = toc;

Y = Y( : ,1:(floor(count/recordStep)) );
time = time( 1:(floor(count/recordStep)) );

if verbose_flag
    disp( sprintf('\n%d steps taken\n', count-1 ) );
    disp( sprintf('%d steps recorded\n', floor(count/recordStep)-1 ) );
end

end
```

```
function [Y, X, time, run_time] = SingleTrajectory_tauleap(V, X, speConstIndecies, numSpecies,↙
speValues, tfinal, recordStep, verbose_flag, tau)

% set max muber of data points for each chunk of the recorded values matrix
numMaxDataPoints = 10000;

Y = zeros(numSpecies, numMaxDataPoints);
time = zeros(1, numMaxDataPoints);

% assign initial values recorded values
Y( : , 1 ) = X;
time(1) = 0;

% initial values
t = 0;
count = 1;

speConstIndeciesLength = length(speConstIndecies);

if verbose_flag
    disp( sprintf('\n=================STARTING SSA WITH TAU_LEAPING==============\n') );
end

tic

while t < tfinal

    %----------------------------------------------------------------
    % calculate value of each propensity at that step
    A = calculatePropensities(X);
    %----------------------------------------------------------------

    % break out of simulation if all species are consumed
    if cumsum(A) == 0
        if verbose_flag
            disp(sprintf('\n=========REACTION HALTED - ALL SPECIES COMSUMED========\n'));
            disp(sprintf('Final time was %f', t) );
        end

        Y( : , floor(count/recordStep) + 1) = X;
        time( floor(count/recordStep) + 1) = t;

        break
    end

    % get sampling of poisson random variables
    pois_rand_vars = poissrnd(A*tau);

    % update values
    X = X + V * pois_rand_vars';

    %----------------------------------------------------------------
    X = calculateSpecifiedTotals(X);
    %----------------------------------------------------------------

    for i = 1:speConstIndeciesLength
        X(speConstIndecies(i)) = speValues(speConstIndecies(i));
    end

    t = t + tau;

    if mod(count, recordStep) == 0
        Y( : , (count/recordStep) + 1) = X;
        time(count/recordStep + 1) = t;
    end

    count = count + 1;
end

run_time = toc;

Y = Y( : ,1:(floor(count/recordStep)) );
time = time( 1:(floor(count/recordStep)) );

if verbose_flag
    disp( sprintf('\n%d steps taken\n', count-1 ) );
    disp( sprintf('%d steps recorded\n', floor(count/recordStep)-1 ) );
end

end
```

```
function Y = SSA_gpu(filename, SysInf, tfinal, verbose_flag)

numSpecies          = SysInf.numSpecies;
numReactions        = SysInf.numReactions;
speNames            = SysInf.speNames;
speValues           = SysInf.speValues;
cNames              = SysInf.cNames;
cValues             = SysInf.cValues;
speConstIndecies    = SysInf.speConstIndecies;
totalsIndecies      = SysInf.totalsIndecies;
VHolder             = SysInf.VHolder;
gpu = gpuDevice();

if verbose_flag
    disp( sprintf('GPU Device detected:\n') );
    disp( gpu );
end

V = VHolder.V;
vNumOfReactant = VHolder.vNumOfReactant;
vReactant = VHolder.vReactant;
vDimerMap = VHolder.vDimerMap;

fid = fopen('fireGpuTrajectories.m','w');
fprintf(fid,'function Y = fireGpuTrajectories(VHolder, verbose_flag)\n\n');

fprintf(fid, 'V = VHolder.V;\n\n');

for i = 1:numSpecies
    fprintf(fid, 'x%d = %d;\n', i, speValues(i));
end

fprintf(fid, '\ntfinal = %d;\n\n', tfinal);

fprintf(fid, '\tfunction [input');
for i = 1:numSpecies
    fprintf(fid, ', x%d', i);
end
fprintf(fid, ']');

fprintf(fid, ' = fire_single_gpu_trajectory(input');
for i = 1:numSpecies
    fprintf(fid, ', x%d', i);
end
fprintf(fid, ')\n\n');

format long

for i = 1:numReactions
    fprintf(fid, '\t\tc%d = %e;\n', i, cValues(i));
end

fprintf(fid, '\n\t\tt = 0;\n');

fprintf(fid, '\n\t\twhile t < tfinal\n\n');

for i = 1:numReactions

```

```matlab
    % set initially to that reaction's parameter
    format long;
    fprintf(fid, '\t\t\ta%d = (%e)', i, cValues(i) );

    % multiply current value by each ractant's value if applicable, and account for dimerisation reactions
    for k = 1:(vNumOfReactant(i));

        curSpeciesIndex = vReactant(i,k);

        fprintf(fid,'*x%d', curSpeciesIndex);

        % determine is reactant is part of a dimerisation reaction using dimerisation map, then alter propensity accordingly
        dimer_number = vDimerMap(curSpeciesIndex, i);
        if dimer_number > 1
            count = 1;
            while (count < dimer_number)
                fprintf(fid,'*(x%d-%d)', curSpeciesIndex, count);
                count = count + 1;
            end

            fprintf(fid,'/%d', factorial(dimer_number) );
        end
    end

    fprintf(fid, ';\n', i, cValues(i) );
end

fprintf(fid, '\n\t\t\tasum =');
for i = 1:numReactions
    fprintf(fid, ' + a%d', i);
end
fprintf(fid, ';\n\n');

fprintf(fid, '\t\t\tif asum == 0\n\t\t\t\treturn\n\t\t\tend\n\n');

for i = 1:numReactions
    fprintf(fid, '\t\t\ta_tot_%d = (', i);
    for j = 1:i
        fprintf(fid,'+a%d', j);
    end
    fprintf(fid, ')/asum;\n');
end

fprintf(fid, '\n\t\t\tj = 1;\n\n');
fprintf(fid, '\t\t\trand_num = rand;\n\n');

fprintf(fid, '\t\t\tif a_tot_1 > rand_num\n');
fprintf(fid, '\t\t\t\tj = 1;\n');

for i = 2:numReactions
    fprintf(fid, '\t\t\telseif a_tot_%d > rand_num\n', i);
    fprintf(fid, '\t\t\t\tj = %d;\n', i);
end

fprintf(fid, '\t\t\tend\n\n');

fprintf(fid, '\t\t\ttau = log(1/rand)/asum;\n\n');

for i = 1:numSpecies
    if ~ismember(i,speConstIndecies)
        fprintf(fid, '\t\t\tx%d = x%d + V(%d,j);\n', i, i, i);
    end
end

fprintf(fid, '\n');
SBMLModel = TranslateSBML(filename);
count = 0;
for i = 1:length(SBMLModel.rule)
    curRule = SBMLModel.rule(i);

    for j = 1:length( speNames )
        if strcmp( speNames(j), curRule.variable )

            totalsIndecies(count+1) = j;
            count = count + 1;
            fprintf(fid, '\t\t\tx%d =', j);

            % token string array
            ruleToks = strsplit( curRule.formula ,'+');

            for l = 1:length( ruleToks )
                for m = 1:length(speNames)
                    if strcmp( speNames(m), ruleToks(l) )
                        fprintf(fid, ' + x%d', m);
                    end
                end
            end

            fprintf(fid, ';\n', m);

            break;
        end
    end
end

fprintf(fid, '\n\t\t\tt = t + tau;\n\n');

fprintf(fid, '\t\tend\n\n');
fprintf(fid, '\tend');

fprintf(fid, '\n\nnum_trajectories = 10000;\n');

fprintf(fid, 'trial_nums = linspace(1,num_trajectories, num_trajectories)'';\n' );
fprintf(fid, 'inputs = gpuArray(trial_nums);\n' );

fprintf(fid, '\n[g_trial');
for i = 1:numSpecies
    fprintf(fid, ', g_x%d', i);
end
fprintf(fid, '] = arrayfun(@fire_single_gpu_trajectory, inputs');
for i = 1:numSpecies
    fprintf(fid, ', x%d', i);
end
fprintf(fid, ');\n\n');

fprintf(fid, 'trials = gather(g_trial);\n');

fprintf(fid, 'Y = [');
for i = 1:numSpecies
    fprintf(fid, ' gather(g_x%d)', i);
end
fprintf(fid, '];\n\n');

fprintf(fid,'\nend');

fclose(fid);

Y = fireGpuTrajectories(VHolder, verbose_flag);
wait(gpu);

Mean = zeros(numSpecies, 1);
Std_dev = zeros(numSpecies, 1);
for i = 1:numSpecies
    data = Y( : , i );
    Mean(i) = mean( data );
    Std_dev(i) = std( data );
end

if verbose_flag
    dataTableMean = table(Mean,'RowNames',speNames);
    disp(dataTableMean);
    dataTableStddev = table(Std_dev,'RowNames',speNames);
    disp(dataTableStddev);
end

end
```

```matlab
function SysInf = SSA_setup(filename, verbose_flag)

if verbose_flag
    disp(' ');
end

arg_type = class(filename);

switch arg_type
    case 'char'
        SBMLModel = TranslateSBML(filename);
    case 'struct'
        SBMLModel = filename;
end


% determine number of reactions and species present in the model
numReactions = length(SBMLModel.reaction);
numSpecies = length(SBMLModel.species);

if verbose_flag
    disp( sprintf('\nNumber of species types:\n') ); disp(numSpecies);
    disp( sprintf('\nNumber of reactions:\n') ); disp(numReactions);
end

[cNames, cValues] = GetParameters(SBMLModel);

if verbose_flag
    [cNames_us, cValues_us] = GetAllParameters(SBMLModel);
    cNames_us = cNames_us';
    cValues_us = cValues_us';

    disp(sprintf('\nParameter Values:\n'))
    Value = cValues_us;
    dataTable = table(Value,'RowNames',cNames_us);
    disp(dataTable);
end

% get species names and values, set X to initial values
[speNames, speValues] = GetSpecies(SBMLModel);
speNames = speNames';
speValues = speValues';

if verbose_flag
    disp(sprintf('\nSpecies'' initial amounts:\n'))
    Amount = speValues;
    dataTable = table(Amount,'RowNames',speNames);
    disp(dataTable);
end

% matricies to hold propensity values, number of species present after each step, and the↙
length of each step
A = zeros(numReactions,1);

VHolder = StoichiometricMatricesHolder(SBMLModel);

% will generate 'calculatePropensities.m' file
GeneratePropensityCalculatorFile(SBMLModel, VHolder);
```

```

% will generate 'calculateSpecifiedTotals.m' file
totalsIndecies = GenerateSpecifiedTotalsCalculatorFile(SBMLModel);

% determine if any species have a boundary condition and get their indecies
speConstIndecies = GetConstantSpeciesIndecies(SBMLModel);

SysInf = SystemInformationHolder;

SysInf.numSpecies        = numSpecies;
SysInf.numReactions      = numReactions;
SysInf.speNames          = speNames;
SysInf.speValues         = speValues;
SysInf.cNames            = cNames;
SysInf.cValues           = cValues;
SysInf.speConstIndecies  = speConstIndecies;
SysInf.totalsIndecies    = totalsIndecies;
SysInf.VHolder           = VHolder;

end
```

```matlab
function [time, Y] = SSAGen(SysInf, tfinal, recordStep, verbose_flag, split_flag)

numSpecies         = SysInf.numSpecies;
numReactions       = SysInf.numReactions;
speNames           = SysInf.speNames;
speValues          = SysInf.speValues;
cNames             = SysInf.cNames;
cValues            = SysInf.cValues;
speConstIndecies   = SysInf.speConstIndecies;
totalsIndecies     = SysInf.totalsIndecies;
VHolder            = SysInf.VHolder;

% initial values
X = speValues;

% extract V from VHolder and display
V = VHolder.V;

if verbose_flag
    disp( sprintf('Stoichiometric Matrix:\n') ); disp(V);
end

% matricies to hold propensity values, number of species present after each step, and the length of each step
A = zeros(numReactions,1);

% Actually do SSA ------------- %
[Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues, tfinal, recordStep, verbose_flag);
% ----------------------------- %

if verbose_flag
    disp(sprintf('\nSpecies'' final amounts:\n'));
    Amount = X;
    dataTable = table(Amount,'RowNames',speNames);
    disp(dataTable);
end

if verbose_flag
    disp(' ');
end

end
```

```matlab
function [time, Y] = SSAGen(SysInf, tfinal, recordStep, verbose_flag, tau)

numSpecies        = SysInf.numSpecies;
numReactions      = SysInf.numReactions;
speNames          = SysInf.speNames;
speValues         = SysInf.speValues;
cNames            = SysInf.cNames;
cValues           = SysInf.cValues;
speConstIndecies  = SysInf.speConstIndecies;
totalsIndecies    = SysInf.totalsIndecies;
VHolder           = SysInf.VHolder;

% initial values
X = speValues;

% extract V from VHolder and display
V = VHolder.V;
if verbose_flag
    disp( sprintf('Stoichiometric Matrix:\n') ); disp(V);
end

% attempt to pick tau if not specified - *extremely* crude estimate
if tau == 0
    % Single SSA trajectory to help determine good tau
    [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues,↙
tfinal, recordStep, verbose_flag);
    tau = ( time(length(time)) / ( length(time)*recordStep ) ) * 3 ;
end

if verbose_flag
    disp( sprintf('Chosen value for tau:\n') ); disp(tau);
end

X = speValues;

tic
[Y, X, time, run_time] = SingleTrajectory_cle(V, X, speConstIndecies, numSpecies, speValues,↙
tfinal, recordStep, verbose_flag, tau);
time_with_leap = toc;

if verbose_flag
    disp(sprintf('\nSpecies'' final amounts:\n'));
    Amount = X;
    dataTable = table(Amount,'RowNames',speNames);
    disp(dataTable);
end

if verbose_flag
    disp(' ');
end

end
```

```
function varargout = SSAGen_parfor(SysInf, tfinal, recordStep, verbose_flag, speciesToGraph,↙
numSteps, graph_flag)

numSpecies          = SysInf.numSpecies;
numReactions        = SysInf.numReactions;
speNames            = SysInf.speNames;
speValues           = SysInf.speValues;
cNames              = SysInf.cNames;
cValues             = SysInf.cValues;
speConstIndecies    = SysInf.speConstIndecies;
totalsIndecies      = SysInf.totalsIndecies;
VHolder             = SysInf.VHolder;

if verbose_flag
    disp(' ');
end

% set max muber of data points for each chunk of the recorded values matrix
numMaxDataPoints = 10008;

% set X to initial values
X = speValues;

% matrix to hold propensity values, number of constant species
speConstIndeciesLength = length(speConstIndecies);
A = zeros(numReactions,1);

% extract V from VHolder and display
V = VHolder.V;
if verbose_flag
    disp( sprintf('\nStoichiometric Matrix:\n') ); disp(V);
end

num_cores = feature('numCores');

if verbose_flag
    disp( sprintf('\nDetected %d CPU cores\n', num_cores) );
end

%Y = zeros(numMaxDataPoints, numSpecies);
Y = zeros(numSpecies, numSteps, numMaxDataPoints);
%time = zeros(1, numMaxDataPoints);

% begin SSA algorithm

if verbose_flag
    disp( sprintf('\n============== STARTING SSA FOR HISTOGRAM ==============='  ) );
    disp( sprintf(  'Remember - this part takes a while. Please be patient.\n') );
end

halt_flag = 0;
time = linspace(0,tfinal,numSteps);
interval  = tfinal/numSteps;

parfor l = 1:numMaxDataPoints

    % initial values
    t = 0;
    n = 2;
    X = speValues;
    A = zeros(numReactions,1);
    Y_sub = zeros(numSpecies, numSteps);
    Y_sub(:,1) = X;

    while n <= numSteps

        next_step = (n-1)*interval;

        %------------------------------------------------------------
        % calculate value of each propensity at that step
        A = calculatePropensities(X);
        %------------------------------------------------------------

        asum = sum(A);

        % break out of simulation if all species are consumed
        if asum == 0
            halt_flag = 1;
            break
        end

        j = find( rand < cumsum(A/asum) , 1 );
        tau = log(1/rand)/asum;

        X = X + V(:,j);

        %------------------------------------------------------------
        X = calculateSpecifiedTotals(X);
        %------------------------------------------------------------

        for i = 1:speConstIndeciesLength
            X(speConstIndecies(i)) = speValues(speConstIndecies(i));
        end

        t = t + tau;

        if t > next_step
            Y_sub(:,n) = X';
            n = n + 1;
        end

    end

    Y(:,:,l) = Y_sub;

end

if halt_flag && verbose_flag
    disp(sprintf('\n=========REACTION HALTED - ALL SPECIES COMSUMED========\n'));
end

if verbose_flag
    disp(' ');
end

Y_last = zeros(numMaxDataPoints, numSpecies);
```

```matlab

% get means, standard deviations, last slice data
Mean    = zeros(numSpecies, numSteps);
Std     = zeros(numSpecies, numSteps);
for i = 1:numSpecies
    for j = 1:numSteps
        data = Y(i,j,:);
        if j == numSteps
            Y_last(:,i) = data;
        end
        Mean(i,j) = mean(data);
        Std(i,j) = std(data);
    end
end

specific_species_flag = 0;

if ~isempty(speciesToGraph)
    stop_point = length(speciesToGraph);
    specific_species_flag  = 1;
else
    stop_point = numSpecies;
end

warning('off','MATLAB:legend:IgnoringExtraEntries');

if graph_flag
    for i = 1:stop_point
        if specific_species_flag
            index = speciesToGraph(i);
        else
            index = i;
        end

        data = zeros(numSteps, numMaxDataPoints);
        data(1:numSteps,1:numMaxDataPoints) = Y(index,:,:);
        data = data';

        figure;

        % graph boxplot for each slice
        subplot(2,1,1);
        boxplot( data , time, 'plotstyle', 'compact');
        legend( findobj(gca,'Tag','Box'),speNames(index) );
        xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
        ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
        title('Box and whisker plot of species vs time at given intervals','FontSize',↙
16,'FontName', 'Helvetica');

        % graph means +- standard deviations for each slice
        subplot(2,1,2);
        hold all
        plot( time, Mean(index,:), 'b-o');
        plot( time, Mean(index,:) - Std(index,:), 'c');
        plot( time, Mean(index,:) + Std(index,:), 'c');
        legend( findobj(gca,'Tag','Box'),speNames(index) );
        xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
        ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
        title('Box and whisker plot of species vs time at given intervals','FontSize',↙
16,'FontName', 'Helvetica');
    end
end

Mean_last = Mean(:,numSteps);
Std_dev_last = Std(:,numSteps);

varargout{1} = Y_last;
varargout{2} = Mean;
varargout{3} = Std;

if verbose_flag
    dataTableMean = table(Mean_last,'RowNames',speNames);
    disp(dataTableMean);
    dataTableStddev = table(Std_dev_last,'RowNames',speNames);
    disp(dataTableStddev);
end

end
```

```
function varargout = SSAGen_parfor_cle(SysInf, tfinal, recordStep, verbose_flag, tau,↙
speciesToGraph, numSteps, graph_flag)

numSpecies        = SysInf.numSpecies;
numReactions      = SysInf.numReactions;
speNames          = SysInf.speNames;
speValues         = SysInf.speValues;
cNames            = SysInf.cNames;
cValues           = SysInf.cValues;
speConstIndecies  = SysInf.speConstIndecies;
totalsIndecies    = SysInf.totalsIndecies;
VHolder           = SysInf.VHolder;

if verbose_flag
    disp(' ');
end

% set max muber of data points for each chunk of the recorded values matrix
numMaxDataPoints = 10008;

% set X to initial values
X = speValues;

% matrix to hold propensity values, number of constant species
speConstIndeciesLength = length(speConstIndecies);
A = zeros(numReactions,1);

% extract V from VHolder and display
V = VHolder.V;
if verbose_flag
    disp( sprintf('\nStoichiometric Matrix:\n') ); disp(V);
end

if tau == 0
    % Single SSA trajectory to help determine good tau
    [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues,↙
tfinal, recordStep, verbose_flag);
    tau = ( time(length(time)) / ( length(time)*recordStep ) ) * 3 ;
end

num_cores = feature('numCores');

if verbose_flag
    disp( sprintf('\nDetected %d CPU cores\n', num_cores) );
end

%Y = zeros(numMaxDataPoints, numSpecies);
Y = zeros(numSpecies, numSteps, numMaxDataPoints);
%time = zeros(1, numMaxDataPoints);

% begin SSA with tau-leaping algorithm

if verbose_flag
    disp( sprintf('\n============== STARTING parallel SSA with tau-leaping ==============' )↙
);
    disp( sprintf(  'Remember - this part takes a while. Please be patient.\n') );
end

```

```
halt_flag = 0;
time = linspace(0,tfinal,numSteps);
interval  = tfinal/numSteps;

parfor l = 1:numMaxDataPoints

    % initial values
    t = 0;
    n = 2;
    X = speValues;
    A = zeros(numReactions,1);
    Y_sub = zeros(numSpecies, numSteps);
    Y_sub(:,1) = X;

    while n <= numSteps

        next_step = (n-1)*interval;

        %----------------------------------------------------------------
        % calculate value of each propensity at that step
        A = calculatePropensities(X);
        %----------------------------------------------------------------

        asum = sum(A);

        % break out of simulation if all species are consumed
        if asum == 0
            halt_flag = 1;
            break
        end

        % get sampling of random variables and sub into CLE formula
        d = tau*A + sqrt( abs(tau*A) ) * randn;

        % update values
        X = X + V * d';

        %----------------------------------------------------------------
        X = calculateSpecifiedTotals(X);
        %----------------------------------------------------------------

        for i = 1:speConstIndeciesLength
            X(speConstIndecies(i)) = speValues(speConstIndecies(i));
        end

        t = t + tau;

        if t > next_step
            Y_sub(:,n) = X';
            n = n + 1;
        end

    end

    Y(:,:,l) = Y_sub;

end

```

```
if halt_flag && verbose_flag
    disp(sprintf('\n=========REACTION HALTED - ALL SPECIES COMSUMED========\n'));
end

if verbose_flag
    disp(' ');
end

Y_last = zeros(numMaxDataPoints, numSpecies);

% get means, standard deviations, last slice data
Mean    = zeros(numSpecies, numSteps);
Std     = zeros(numSpecies, numSteps);
for i = 1:numSpecies
    for j = 1:numSteps
        data = Y(i,j,:);
        if j == numSteps
            Y_last(:,i) = data;
        end
        Mean(i,j) = mean(data);
        Std(i,j) = std(data);
    end
end

specific_species_flag = 0;

if ~isempty(speciesToGraph)
    stop_point = length(speciesToGraph);
    specific_species_flag  = 1;
else
    stop_point = numSpecies;
end

warning('off','MATLAB:legend:IgnoringExtraEntries');

if graph_flag
    for i = 1:stop_point
        if specific_species_flag
            index = speciesToGraph(i);
        else
            index = i;
        end

        data = zeros(numSteps, numMaxDataPoints);
        data(1:numSteps,1:numMaxDataPoints) = Y(index,:,:);
        data = data';

        figure;

        % graph boxplot for each slice
        subplot(2,1,1);
        boxplot( data , time, 'plotstyle', 'compact');
        legend( findobj(gca,'Tag','Box'),speNames(index) );
        xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
        ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
        title('Box and whisker plot of species vs time at given intervals','FontSize',↙
16,'FontName', 'Helvetica');

```

```
        % graph means +- standard deviations for each slice
        subplot(2,1,2);
        hold all
        plot( time, Mean(index,:), 'b-o');
        plot( time, Mean(index,:) - Std(index,:), 'c');
        plot( time, Mean(index,:) + Std(index,:), 'c');
        legend( findobj(gca,'Tag','Box'),speNames(index) );
        xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
        ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
        title('Box and whisker plot of species vs time at given intervals','FontSize',↙
16,'FontName', 'Helvetica');
    end
end

Mean_last = Mean(:,numSteps);
Std_dev_last = Std(:,numSteps);

varargout{1} = Y_last;
varargout{2} = Mean;
varargout{3} = Std;

if verbose_flag
    dataTableMean = table(Mean_last,'RowNames',speNames);
    disp(dataTableMean);
    dataTableStddev = table(Std_dev_last,'RowNames',speNames);
    disp(dataTableStddev);
end

end
```

```
function varargout = SSAGen_parfor_tauleap(SysInf, tfinal, recordStep, verbose_flag, tau, speciesToGraph, numSteps, graph_flag, num_traj)

numSpecies        = SysInf.numSpecies;
numReactions      = SysInf.numReactions;
speNames          = SysInf.speNames;
speValues         = SysInf.speValues;
cNames            = SysInf.cNames;
cValues           = SysInf.cValues;
speConstIndecies  = SysInf.speConstIndecies;
totalsIndecies    = SysInf.totalsIndecies;
VHolder           = SysInf.VHolder;

if verbose_flag
    disp(' ');
end

% set max muber of data points for each chunk of the recorded values matrix
if num_traj == 0
    numMaxDataPoints = 10008;
else
    numMaxDataPoints = num_traj;

% set X to initial values
X = speValues;

% matrix to hold propensity values, number of constant species
speConstIndeciesLength = length(speConstIndecies);
A = zeros(numReactions,1);

% extract V from VHolder and display
V = VHolder.V;
if verbose_flag
    disp( sprintf('\nStoichiometric Matrix:\n') ); disp(V);
end

if tau == 0
    % Single SSA trajectory to help determine good tau
    [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues, tfinal, recordStep, verbose_flag);
    tau = ( time(length(time)) / ( length(time)*recordStep ) ) * 3 ;
end

num_cores = feature('numCores');

if verbose_flag
    disp( sprintf('\nDetected %d CPU cores\n', num_cores) );
end

%Y = zeros(numMaxDataPoints, numSpecies);
Y = zeros(numSpecies, numSteps, numMaxDataPoints);
%time = zeros(1, numMaxDataPoints);

% begin SSA with tau-leaping algorithm

if verbose_flag
    disp( sprintf('\n============== STARTING parallel SSA with tau-leaping =============='  ) );
    disp( sprintf(  'Remember - this part takes a while. Please be patient.\n') );
end

halt_flag = 0;
time = linspace(0,tfinal,numSteps);
interval  = tfinal/numSteps;

parfor l = 1:numMaxDataPoints

    % initial values
    t = 0;
    n = 2;
    X = speValues;
    A = zeros(numReactions,1);
    Y_sub = zeros(numSpecies, numSteps);
    Y_sub(:,1) = X;

    while n <= numSteps

        next_step = (n-1)*interval;

        %---------------------------------------------------------------
        % calculate value of each propensity at that step
        A = calculatePropensities(X);
        %---------------------------------------------------------------

        asum = sum(A);

        % break out of simulation if all species are consumed
        if asum == 0
            halt_flag = 1;
            break
        end

        % get sampling of poisson random variables
        pois_rand_vars = poissrnd(A*tau);

        % update values
        X = X + V * pois_rand_vars';

        %---------------------------------------------------------------
        X = calculateSpecifiedTotals(X);
        %---------------------------------------------------------------

        for i = 1:speConstIndeciesLength
            X(speConstIndecies(i)) = speValues(speConstIndecies(i));
        end

        t = t + tau;

        if t > next_step
            Y_sub(:,n) = X';
            n = n + 1;
        end

    end

    Y(:,:,l) = Y_sub;
```

```matlab

end

if halt_flag && verbose_flag
    disp(sprintf('\n=========REACTION HALTED - ALL SPECIES COMSUMED========\n'));
end

if verbose_flag
    disp(' ');
end

Y_last = zeros(numMaxDataPoints, numSpecies);

% get means, standard deviations, last slice data
Mean    = zeros(numSpecies, numSteps);
Std     = zeros(numSpecies, numSteps);
for i = 1:numSpecies
    for j = 1:numSteps
        data = Y(i,j,:);
        if j == numSteps
            Y_last(:,i) = data;
        end
        Mean(i,j) = mean(data);
        Std(i,j) = std(data);
    end
end

specific_species_flag = 0;

if ~isempty(speciesToGraph)
    stop_point = length(speciesToGraph);
    specific_species_flag  = 1;
else
    stop_point = numSpecies;
end

warning('off','MATLAB:legend:IgnoringExtraEntries');

if graph_flag
    for i = 1:stop_point
        if specific_species_flag
            index = speciesToGraph(i);
        else
            index = i;
        end

        data = zeros(numSteps, numMaxDataPoints);
        data(1:numSteps,1:numMaxDataPoints) = Y(index,:,:);
        data = data';

        figure;

        % graph boxplot for each slice
        subplot(2,1,1);
        boxplot( data , time, 'plotstyle', 'compact');
        legend( findobj(gca,'Tag','Box'),speNames(index) );
        xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
        ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
        title('Box and whisker plot of species vs time at given intervals','FontSize',↙
16,'FontName', 'Helvetica');

        % graph means +- standard deviations for each slice
        subplot(2,1,2);
        hold all
        plot( time, Mean(index,:), 'b-o');
        plot( time, Mean(index,:) - Std(index,:), 'c');
        plot( time, Mean(index,:) + Std(index,:), 'c');
        legend( findobj(gca,'Tag','Box'),speNames(index) );
        xlabel('Time','FontSize',12, 'FontName', 'Helvetica');
        ylabel('Number of Species','FontSize',12,'FontName', 'Helvetica');
        title('Box and whisker plot of species vs time at given intervals','FontSize',↙
16,'FontName', 'Helvetica');
    end
end

Mean_last = Mean(:,numSteps);
Std_dev_last = Std(:,numSteps);

varargout{1} = Y_last;
varargout{2} = Mean;
varargout{3} = Std;

if verbose_flag
    dataTableMean = table(Mean_last,'RowNames',speNames);
    disp(dataTableMean);
    dataTableStddev = table(Std_dev_last,'RowNames',speNames);
    disp(dataTableStddev);
end

end
```

```matlab
function [time, Y] = SSAGen(SysInf, tfinal, recordStep, verbose_flag, tau)

numSpecies        = SysInf.numSpecies;
numReactions      = SysInf.numReactions;
speNames          = SysInf.speNames;
speValues         = SysInf.speValues;
cNames            = SysInf.cNames;
cValues           = SysInf.cValues;
speConstIndecies  = SysInf.speConstIndecies;
totalsIndecies    = SysInf.totalsIndecies;
VHolder           = SysInf.VHolder;

% initial values
X = speValues;

% extract V from VHolder and display
V = VHolder.V;
if verbose_flag
    disp( sprintf('Stoichiometric Matrix:\n') ); disp(V);
end

% attempt to pick tau if not specified - *extremely* crude estimate
if tau == 0
    % Single SSA trajectory to help determine good tau
    [Y, X, time, run_time] = SingleTrajectory(V, X, speConstIndecies, numSpecies, speValues,↙
tfinal, recordStep, verbose_flag);
    tau = ( time(length(time)) / ( length(time)*recordStep ) ) * 3 ;
end

disp( sprintf('Chosen value for tau:\n') ); disp(tau);

X = speValues;

tic
[Y, X, time, run_time] = SingleTrajectory_tauleap(V, X, speConstIndecies, numSpecies,↙
speValues, tfinal, recordStep, verbose_flag, tau);
time_with_leap = toc;

if verbose_flag
    disp(sprintf('\nSpecies'' final amounts:\n'));
    Amount = X;
    dataTable = table(Amount,'RowNames',speNames);
    disp(dataTable);
end

if verbose_flag
    disp(' ');
end

end
```

```matlab
classdef StoichiometricMatricesHolder

    properties
        V;
        vNumOfReactant;
        vReactant;
        vDimerMap;
    end

    methods

        function object = StoichiometricMatricesHolder(SBMLModel)

            numReactions = length(SBMLModel.reaction);
            numSpecies = length(SBMLModel.species);

            % get the stoichiometric matrix representing the species changes for each reaction from the model
            V = zeros(numSpecies, numReactions);

            % matricies to hold the number of reactants in each reaction, the reactants' indecies, and the dimer map
            vNumOfReactant = zeros(numReactions,1);
            vReactant = zeros(numReactions, numSpecies);
            vDimerMap = zeros(numSpecies, numReactions);

            maxCount = 0;

            for j = 1:numReactions

                count = 0;
                for i = 1:numSpecies

                    role = DetermineSpeciesRoleInReaction(SBMLModel.species(i), SBMLModel.reaction(j));
                    if length(role) > 1

                        V(i,j) = role(1) - role(2);

                        if role(2) > 0
                            vReactant( j , (count+1) ) = i;
                            count = count + 1;
                        end

                        if role(2) >= 2
                            vDimerMap(i,j) = role(2);
                        end
                    else
                        V(i,j) = 0;
                    end
                end

                vNumOfReactant(j) = count;

                if count > maxCount
                    maxCount = count;
                end

            end

            object.V = V;
            object.vNumOfReactant = vNumOfReactant;

            % truncate reactant index matrix to discard unnecessary elements
            object.vReactant = vReactant( : , 1:(maxCount) );

            % make dimer reactant matrix sparse to discard unnecessary elements
            object.vDimerMap = vDimerMap;

        end

    end

end
```

```
classdef SystemInformationHolder

    properties
        numSpecies;
        numReactions;
        speNames;
        speValues;
        cNames;
        cValues;
        speConstIndecies;
        totalsIndecies;
        VHolder;
    end

end
```