S2-1: The code for simulation and making plot

%% Homework 1: question 2

% set global parameter

global k

k = 1.5;

for n = 1:20

N = 1:n; % the N-th event

t\_exp(n) = sum(1./(k .^N)); % time for the n events to happend

end

% expected time for population to reach infinity

t\_exp\_infty = 1/k \* (1/ (1- (1./(k))));

% plot expected waiting time

figure

myplot(1:20, t\_exp, 'B');

xlabel('state (n)'); ylabel('expect waiting time')

title({'The "explosive dynamic": $$\lambda\_n = k^n $$'}, ...

'Interpreter','latex',...

'fontsize', 16);

text(3, 1, ...

{'Expected time for populaton to go infinity',...

[' $$\sum\_{n=1}^{\infty} \frac{1}{\lambda\_n} = $$' , num2str(t\_exp\_infty)]}, ...

'Interpreter','latex',...

'fontsize', 14);

%% simulation

tlim = 1.5;

X0 =2;

figure

mysubplot(2,1,1)

rng(1); % set seed

vline(tlim, 7,':');hold on

for i =1:20

[t, x] = explodGillespie(X0, tlim);

stairs(t,x, 'LineWidth',1, 'color', mycolor(i));hold on

end

vline(tlim, 7,':');% replot so it's on top

legend('tlim', 'simulation 1', 'simulation 2', 'simulation 3', '...')

ylabel('state(n)');

title('"Explosive" pure birth process')

axis([ 0 2 0 2000])

mysubplot(2,1,2)

rng(1); % set seed; so this will be identical simulations

for i =1:20

[t, x] = explodGillespie(X0, tlim);

stairs(t,x, 'LineWidth',1, 'color', mycolor(i));hold on

end

axis([ 0 2 0 25])

vline(tlim, 7,':')

ylabel('state(n)');

xlabel('time');

title('close-up look')

S2-2: The Gillespie function: explodGillespie.m

function [t, x] = explodGillespie(X0, tlim)

%% Use global parameters

global k

%% reactiona and vectors

% \* reaction 1: X --> X

t = zeros(1, 1e4);

x = zeros(1, 1e4);

x(1) = X0;

point = 1; %This variable is to keep track of the amount of points.

%%

while (t(point) < tlim) && (~isinf(k^x(point)))

% 1. calculate rate of each event

a0 = k^x(point);

% 2. inter event time

t(point+1) = t(point) + log(1/rand)/a0; % Calculating the interevent time.

% 3. update state

x(point+1) = x(point) +1; % Updating the state.

% 5. update the point counter.

point = point + 1;

end

% The last thing we do is get rid of the memory that was not use during the

% simulation and we are done.

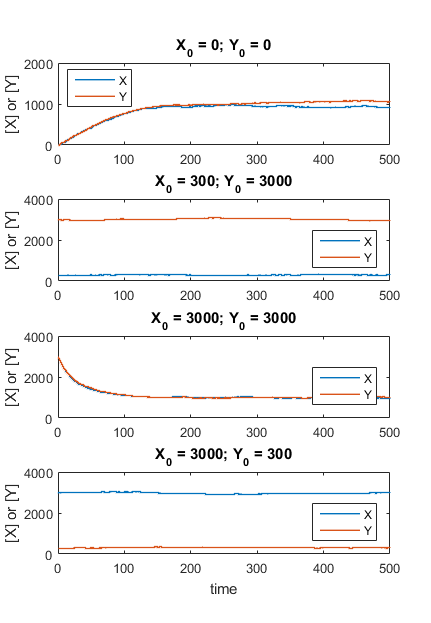
t = t(1:point);

x = x(1:point);

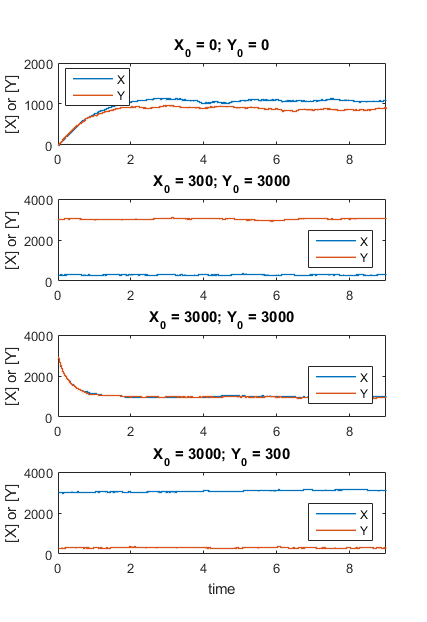
S3-1: The code and document for exploring steady state and initial values

The (quasi-) steady state depend on the initial value. The theoretical fix point (X ~ 3162, Y ~ 316) only realized when started at values near the fix point (the second panels in the graphs).

With parameter set 1, population reaches the (quasi-) steady state at around 200 time units.

****

With parameter set 2, population reaches the (quasi-) steady state at around 2 time units. Beside the differences in time scales, the behavior in the two system looks similar (everything 100 times faster with the parameter set 2).

****

% set global parameters

global k a1 a2 ka

% Set seed

rng(123)

% Parameter set 1

k = 10; a1 = 10^(-6); a2 = 10^(-5); ka =10^(-5) ;

% simulation

tlim = 500;

X0{1} = [0, 0];

X0{2} = [300 3000];

X0{3} = [3000, 3000];

X0{4} = [3000, 300 ];

%%

figure

for j = 1:4

% simulation

[t, x] = chemGillespie(X0{j}, tlim);

% plot

subplot(4,1,j)

stairs(t, x(:,1), 'LineWidth',1);hold on

stairs(t,x(:,2),'LineWidth',1);

legend('X', 'Y', 'Location','southeast')

ylabel('[X] or [Y]')

xlim([0 500])

title(['X\_0 = ' num2str( X0{j}(1) ) '; Y\_0 = ' num2str( X0{j}(2) ) ])

end

xlabel('time');

% Parameter set 2

k = 10^3; a1 = 10^(-4); a2 = 10^(-3); ka =10^(-3) ;

% simulation

tlim = 9;

%%

figure

for j = 1:4

% simulation

[t, x] = chemGillespie(X0{j}, tlim);

% plot

subplot(4,1,j)

stairs(t, x(:,1), 'LineWidth',1);hold on

stairs(t,x(:,2),'LineWidth',1);

legend('X', 'Y', 'Location','southeast')

ylabel('[X] or [Y]')

xlim([0 tlim])

title(['X\_0 = ' num2str( X0{j}(1) ) '; Y\_0 = ' num2str( X0{j}(2) ) ])

end

xlabel('time');

S3-2: The code for simulation and making plot

% set global parameters

global k a1 a2 ka

X0 = [0, 0];

it = 100;

% Parameter set 1

k = 10; a1 = 10^(-6); a2 = 10^(-5); ka =10^(-5) ;

% simulation

tlim = 300;

it = 100;

xend = zeros(1, it);

yend = zeros(1, it);

%%

figure

mysubplot(1,5,1:3, [], [], 0.1,0.1)

tic

rng(123)

for j = 1:it

%%%%%%%%%%% simulation %%%%%%%%%%%%%%%

[t, x] = chemGillespie(X0, tlim);

xend(j) = x(end, 1); % record the end point of x

yend(j) = x(end, 2); %

% plot the first 15 trajectories

if j <= 15

stairs(t, x(:,1), 'LineWidth',1, 'color', mycolor(11));hold on

stairs(t,x(:,2),'LineWidth',1, 'color', mycolor(13)); hold on

end

%%%%%%%%%%% simulation %%%%%%%%%%%%%%%

end

toc

xlabel('time'); ylabel('[X](t) or [Y](t)')

xlim([0 tlim])

title('the first 15 trajectories')

legend('X', 'Y', 'Location','southeast')

vaxis = ylim;

% plot the stationary distribution of X

mysubplot(1,5,4, [], [], 0.1,0.1)

histogram(xend,'Orientation','horizontal','BinLimits', vaxis, 'Normalization','pdf'); hold on

set(gca, 'ytick', []); ylim(vaxis)

xlabel('pdf'); title('[X] dist.')

% plot the stationary distribution of Y

mysubplot(1,5,5, [], [], 0.1,0.1)

histogram(yend,'Orientation','horizontal','BinLimits', vaxis, 'Normalization','pdf'); hold on

set(gca, 'ytick', [])

xlabel('pdf'); title('[Y] dist.')

% Parameter set 2

k = 10^3; a1 = 10^(-4); a2 = 10^(-3); ka =10^(-3) ;

% simulation

tlim = 300;

it = 100;

xend = zeros(1, it);

yend = zeros(1, it);

%% simulation

figure

mysubplot(1,5,1:3, [], [], 0.1,0.1)

tic

rng(123)

for j = 1:it

% simulation

[t, x] = chemGillespie(X0, tlim);

% plot

if j <= 15

stairs(t, x(:,1), 'LineWidth',1, 'color', mycolor(11));hold on

stairs(t,x(:,2),'LineWidth',1, 'color', mycolor(13)); hold on

end

xend(j) = x(end, 1); % record the end point of x

yend(j) = x(end, 2); % y

end

toc

xlabel('time'); ylabel('[X](t) or [Y](t)')

xlim([0 tlim])

title('the first 15 trajectories')

legend('X', 'Y', 'Location','southeast')

vaxis = ylim;

mysubplot(1,5,4, [], [], 0.1,0.1)

histogram(xend,'Orientation','horizontal','BinLimits', vaxis, 'Normalization','pdf'); hold on

set(gca, 'ytick', [])

xlabel('pdf'); title('[X] dist.')

mysubplot(1,5,5, [], [], 0.1,0.1)

histogram(yend,'Orientation','horizontal','BinLimits', vaxis, 'Normalization','pdf'); hold on

set(gca, 'ytick', [])

xlabel('pdf'); title('[Y] dist.')

S3-3: The Gillespie function: chemGillespie.m

function [t, X, E] = chemGillespie(X0, tlim)

%% Use global parameters

global k a1 a2 ka

%% reactiona and vectors

% \* reaction 1: A --> X

% \* reaction 2: B --> Y

% \* reaction 3: X -->

% \* reaction 4: Y -->

% \* reaction 5: X + Y --> C

% There are five reactions

v = [];

v{1} = [1 0]; v{2} = [0 1]; v{3} = [-1 0]; v{4} = [0 -1]; v{5} = [-1 -1];

%%

S = length(X0); % number of species

t = zeros(1e6,1); %

X = zeros(1e6,S);

E = zeros(1e6,1); % record events

X(1, :) = X0;

point = 1; % keep track of points: a point can be any change in state

while t(point) < tlim

% 1. calculate rate of each event

rates = [k, ...

k, ...

a1\*X(point, 1), ... %

a2\*X(point, 2), ...

ka\*X(point, 1)\*X(point, 2)];

a0 = sum(rates, 2); % the rate that "any" event happens

% 2. inter event time

t(point+1) = t(point) + log(1/rand)/a0; % Calculating the inter event time.

% 3. witch event

eventID = datasample(1:5, 1, 'weight', rates);

E(point+1) = eventID; %record event

% 4. update state

X(point+1, :) = X(point, :) + v{eventID}; % Updating the state.

% 5. update the point counter.

point = point + 1;

end

% The last thing we do is get rid of the memory that was not use during the

% simulation and we are done.

t = t(1:point);

X = X(1:point,:);

S4-1: The code for simulation and making plot

%% The Yule process

% Model parameter

lambda = 1; % per capita birth rate

% Simulation parameters

n0 = 10 % initial value

tau= 0.2; % "tau"

T = 10; % total time

points =T/tau; % number of time points in the simulation

% Variable

N =nan(1, points); % state, population size

it = 1000; % iteration

NendPois = nan(1, it); % end points of the Poisson simulation

NendNorm = nan(1, it); % end points of the Normal

%% Poisson algorism

rng(123)

figure

tic

mysubplot(1,3,1:2, [], [], 0.1,0.1)

for i = 1: it

N( 1 ) = n0;

%%%% The tau-leaping algorithm %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for p = 1: ( points - 1)

n = N( p ); % N at the beginning of interval

events = random( 'Poisson', tau \* lambda \* n ) ; % number of events

N( p + 1 ) = n + events ;

end

if i <= 20 % plot the first 20 trajectories

stairs( ( 1 : points ) \* tau , N ) ; hold on

end

NendPois( i ) = N( end ) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

second = toc;

ylim( [ 0 200000 ] );

vaxis = ylim; % record axis so the can set the next plot

ylabel( 'n (population size)' ) ;

xlabel( 'time' );

title( 'the first 20 trajectories' )

% Plot distribution of states N(T)

mysubplot( 1, 3, 3, [], [], 0.1, 0.1 )

histogram( NendPois, ...

'Orientation', 'horizontal', ...

'BinLimits', vaxis, ...

'Normalization', 'pdf' )

haxis = xlim; % record axis so the can set the next plot

xlabel( 'pdf' );

title( 'Distribution of n(T)' )

set( gca, 'ytick', [] )

% make major title

mysubplot(1,3, 0, {'\tau-leapping simulation (Poisson)', ...

[num2str(it) ' simulations'], ...

['simulation time = ' num2str(second) 's']});

%% Normal algorism

rng( 123 ) % set seed

figure

tic

mysubplot(1,3,1:2, [], [], 0.1,0.1)

for i = 1: it

flag = 1; % include this in records

N(1) = n0;

%%%% The tau-leaping algorithm %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for p = 1: ( points-1)

n = N( p ); % N at the beginning of interval

rate = tau \* lambda \* n;

events = randn \* sqrt( rate ) + rate; % number of events

% std, mean

N(p+1) = n + events;

if N(p+1) < 0 % normal random variable could give unreasonable negative numbers

% exclude those with negative populations

N(p+1) =0; %

flag = 0; % this simulation will be excluded in the recorded sample

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if i <= 20 % plot the first 20 trajectories

stairs((1:points) \* tau,N); hold on

end

if flag == 1 % include in the sample

NendNorm(i) = N(end);

end

end

second = toc;

ylim( vaxis );% so it is the same as in the first figure

ylabel('n (population size)');

xlabel('time');

title('the first 20 trajectories');

% Plot distribution of states N(T)

mysubplot( 1, 3, 3, [], [], 0.1, 0.1 )

histogram( NendNorm, 'Orientation', 'horizontal', 'BinLimits', vaxis, 'Normalization', 'pdf' )

xlabel( 'pdf' );

title( 'Distribution of N(T)' )

xlim(haxis); % so it is the same as in the first figure

set( gca, 'ytick', [] )

% make major title

mysubplot( 1, 3, 0, { '\tau-leapping simulation (Normal approximation)', ...

[num2str(it) ' simulations'] , ...

['simulation time = ' num2str(second) 's']} );

S5-1: ODE function for the Birth-death-immigration model

% ODE for logistic equation

function dndt = BDIODE(t, n)

global rB rD Im

dndt= (rB - rD) \* n + Im;

S5-2: Gillespie function for the Birth-death-immigration model

function [t, X, E] = BDIGillespie(X0, tlim)

%% Use global parameters

global rB rD Im

%% reactiona and vectors

% \* reaction 1: X --> 2X

% \* reaction 2: X -->

% \* reaction 3: --> X

% There are 3 reactions

v = [];

v{1} = [1]; v{2} = [-1]; v{3} = [1];

%%

t = zeros(1e6,1); %

X = zeros(1e6,1);

E = zeros(1e6,1); % record events

X(1) = X0;

point = 1; % keep track of points: a point can be any change in state

while t(point) < tlim

% 1. calculate rate of each event

rates = [rB \* X(point), ...

rD \* X(point), ...

Im];

a0 = sum(rates, 2); % the rate that "any" event happens

% 2. inter event time

t(point+1) = t(point) + log(1/rand)/a0; % Calculating the interevent time.

% 3. witch event

eventID = min(find(rand < cumsum(rates/a0))); % like the above but zero-prove

E(point+1) = eventID;

% 4. update state

X(point+1) = X(point) + v{eventID}; % Updating the state.

% 5. update the point counter.

point = point + 1;

end

% The last thing we do is get rid of the memory that was not use during the

% simulation and we are done.

t = t(1:point);

X = X(1:point);

E = E(1:point);

S5-3: ODE function for the logistic model

% ODE for logistic equation

function dndt = logisticODE(t, n)

global r K

dndt= r\*n\*(1 - n./K);

S5-4: Gillespie function for the logistic model

function [t, X] = logisticGillespie(X0, tlim)

%% Use global parameters

global b1 b2 d1 d2

% process

v=[];

v{1} = 1; % birth

v{2} = -1; % death

v{3} = 0; % a pseudo-event: maintain extinct

%%

S = length(X0); % number of species

t = zeros(1e6,1); %

X = zeros(1e6,S);

X(1, :) = X0;

point = 1; % keep track of points: a point can be any change in state

% X0=n0

while t(point) < tlim

n = X(point);

% 1. calculate rate of each event

rates = [b1\*n + b2\*n^2 , ...

d1\*n + d2\*n^2 ];

a0 = sum(rates, 2); % the rate that "any" event happens

% 2. inter event time

t(point+1) = t(point) + log(1/rand)/a0; % Calculating the interevent time.

% 3. witch event

if n>0

eventID = min(find(rand < cumsum(rates/a0))); % like the above but zero-prove

else

eventID = 3;

end

% 4. update state

X(point+1, :) = X(point, :) + v{eventID}; % Updating the state.

% 5. update the point counter.

point = point + 1;

end

% The last thing we do is get rid of the memory that was not use during the

% simulation and we are done.

t = t(1:point);

X = X(1:point,:);

S6: Utility functions

I some plotting function in the code. Here is a list and generally what they do. Full code and document can be found on (https://github.com/weitingwlin/matlabutility)

* **fixsample** : re-sample the original time-population data to create sample at user-assigned time points.
* **hline.m** : make horizontal line
* **mycolor.m** : select color
* **myplot.m** : make pretty plots
* **mystyle.m** : select style
* **mysubplot.m** : make subplot with major title
* **vline.m** : make vertical line