MATLAB CODE

No-Neighbour Solution

```
clear;
KonT = 10e3;
KoffT = 10e-2;
KonP = 10e-2;
KoffP = 10e-4;
Mfree = 20e-6;
Meff = 50; %Input of various kinetic rate constants and the
concentrations relevant to solving the ODEs

dSdt = @(t, S) [-KonT*Mfree*S(1)+KoffT*S(2);
                KonT*Mfree*S(1)-KoffT*S(2);]; %Input of
differential equations derived using the mass-action
principle. The states defined are as seen in the schematic
diagram of the particular neighbour solution

range = 0:0.0001:2; %Length of time to integrate over
[min:step:max]
S0 = [20e-9; 0;]; %Initial conditions in form [S1; S2; S3; S4]

[t, S] = ode23s(dSdt, range, S0); %Solving the system of ODE
and storing the solution in variable S

plot(t, S(:, 1), t, S(:, 2)); %Plotting the columns of S over
time
legend('N0T0P0', 'N1T1P0');
title('No Neighbour'); %Formatting the graph by labelling each
state, and adding a title
hold on;
KonT = 10e5;
KoffT = 10e-2;
KonP = 10e-3;
KoffP = 10e-2;
Mfree = 0;
Meff = 50; %Inputting rate constants and concentrations for
the dissociation stage, in which the solution concentration of
monomers goes to zero

dSdt = @(t, S) [-KonT*Mfree*S(1)+KoffT*S(2);
                KonT*Mfree*S(1)-KoffT*S(2);];

range = 1:0.001:500; %Length of time to integrate over
[min:step:max]
S0 = [0; 20e-9;]; %Initial conditions in form [S1; S2; S3; S4]

[t, S] = ode23s(dSdt, range, S0); %Solving the system of ODE
```

```
plot(t, S(:, 1), t, S(:, 2));
legend('N0T0P0', 'N1T1P0');
title('No Neighbour');
```

One-Neighbour Solution

```
KonT = 10e3;
KoffT = 10e-3;
KonP = 10e2;
KoffP = 10e-1;
Mfree = 20e-6;
Meff = 20e-4; %Input of various kinetic rate constants and the
concentrations relevant to solving the ODEs

dSdt = @(t, S) [-KonT*Mfree*S(1)+KoffT*S(2)-
KonP*Mfree*S(1)+KoffP*S(3);
                -KoffT*S(2)+KonT*Mfree*S(1)-
KonP*Meff*S(2)+KoffP*S(4);
                -KoffP*S(3)+KonP*Mfree*S(1)-
KonT*Meff*S(3)+KoffT*S(4);
                -KoffT*S(4)+KonT*Meff*S(3)-
KoffP*S(4)+KonP*Meff*S(2);]; %Input of differential equations
derived using the mass-action principle. The states defined
are as seen in the schematic diagram of the particular
neighbour solution


range = [0:0.005:500]; %Length of time to integrate over
[min:step:max]%
S0 = [0; 0; 0; 20e-9]; Initial conditions in form [S1; S2; S3;
S4]


[t, S] = ode23s(dSdt, range, S0); %Solving the system of ODE
and storing the solution in variable S

plot(t, S(:, 1), t, S(:, 2), t, S(:, 3), t, S(:, 4));
legend('N0T0P0', 'N2T1P0', 'N1T0P1', 'N2T1P1');
title('One Neighbour');
hold on;
```

Two-Neighbour Solution

```
KonT = 100;
KoffT = 10;
KonP = 100;
KoffP = 10;
```

```
Mfree = 2;
Meff = 5;
dSdt = @(t, S) [-KonT*Mfree*S(1)+KoffT*S(2)-
2*KonP*Mfree*S(1)+KoffP*S(3);
                -KoffT*S(2)+KonT*Mfree*S(1)-
2*KonP*Meff*S(2)+KoffP*S(4);
                -KoffP*S(3)+2*KonP*Mfree*S(1)-
KonT*Meff*S(3)+KoffT*S(4)-KonP*Meff*S(3)+2*KoffP*S(5);
                -KoffP*S(4)+2*KonP*Meff*S(2)-
KoffT*S(4)+KonT*Meff*S(3)-KonP*Meff*S(4)+2*KoffP*S(6);
                -2*KoffP*S(5)+KonP*Meff*S(3)-
KonT*Meff*S(5)+KoffT*S(6);
                -2*KoffP*S(6)+KonP*Meff*S(4)-
KoffT*S(6)+KonT*Meff*S(5);];

range = [0:0.0001:0.1];
S0 = [20; 0; 0; 0; 0; 0];
[t, S] = ode23s(dSdt, range, S0);

plot(t, S(:, 1), t, S(:, 2), t, S(:, 3), t, S(:, 4), t, S(:,
5), t, S(:, 6));
legend('N2T0P0', 'N3T1P0', 'N2T0P1', 'N3T1P1', 'N2T0P2',
'N3T1P2');
title('Two Neighbour');
```

Pentamer Formation

```
Kon01=10e3;
Koff10=10e-2;
Kon12=15e3;
Koff21=3.5e-2;
Kon23=35e3;
Koff32=2e-2;
M=10e-3; %Inputting rate constants relevant to particular
neighbour solutions. These can be used to construct the
pentamer protein aggregate.
dSdt = @(t, S) [-5*Kon01*M*S(1)+Koff10*S(2);
                -Koff10*S(2)+5*Kon01*M*S(1)-
2*Kon12*M*S(2)+2*Koff21*S(3)-2*Kon01*M*S(2)+2*Koff10*S(4);
                -2*Koff21*S(3)+2*Kon12*M*S(2)-
2*Kon12*M*S(3)+2*Koff21*S(6)-Kon01*M*S(3)+Koff10*S(5);
                -2*Koff10*S(4)+2*Kon01*M*S(2)-
2*Kon12*M*S(4)+2*Koff21*S(5)-Kon23*M*S(4)+Koff32*S(6);
                -Koff10*S(5)+Kon01*M*S(3)-
2*Koff21*S(5)+2*Kon12*M*S(4)-2*Kon23*M*S(5)+2*Koff32*S(7);
                -2*Koff21*S(6)+2*Kon12*M*S(3)-
Koff32*S(6)+Kon23*M*S(4)-2*Kon12*M*S(6)+2*Koff21*S(7);
                -2*Koff32*S(7)+2*Kon23*M*S(5)-
2*Koff21*S(7)+2*Kon12*M*S(6)-Kon23*M*S(7)+5*Koff32*S(8);
```

```
                -5*Koff32*S(8)+Kon23*M*S(7);]; %Input of
differential equations derived in this manuscript

range = [0:0.0001:0.1];
S0 = [10e-6; 0; 0; 0; 0; 0; 0; 0];
[t, S] = ode23s(dSdt, range, S0); %Solving the ODEs using the
initial conditions specified, over the time interval defined

plot(t, S(:, 1), t, S(:, 2), t, S(:, 3), t, S(:, 4), t, S(:,
5), t, S(:, 6), t, S(:, 7), t, S(:, 8));
legend('F00', 'F01', 'F20', 'F02', 'F21', 'F30', 'F40',
'F50');
xlabel('Time (s)');
ylabel('Concentration');
title('Pentamer Formation'); %Plotting the solutions to the
ODE
hold on;
S(:,9)=0*S(:,1)+1*S(:,2)+2*(S(:,3)+S(:,4))+3*(S(:,5)+S(:,6))+4
*S(:,7)+5*S(:,8); %Creation of the 'Total read-out' plot. SPR
will make measurements of ALL states, not just the 'fully
constructed' state. As such, it is important to weight
respective states based on their occupation. For example,
state 5 has 3 proteins bound to it. So, it's magnitude is
multiplied by 3. By comparison, the 'empty' state S(1) will
not be read by an SPR machine, and so is multiplied by zero.
By adding up these weighted states, we can get an overall
curve.
plot(t,S(:,9));
legend('F00', 'F01', 'F20', 'F02', 'F21', 'F30', 'F40', 'F50',
'total curve');
```

Pentamer Formation

```
K01=10e3;
K10=10e-2;
K12=15e3;
K21=3.5e-2;
K23=35e3;
K32=2e-2;
M=20e-3;
dSdt = @(t, S) [-6*K01*M*S(1)+K10*S(2);
                -K10*S(2)+6*K01*M*S(1)-K01*M*S(2)+2*K10*S(3)-
2*K01*M*S(2)+2*K10*S(4)-2*K12*M*S(2)+2*K21*S(5);
                -2*K10*S(3)+K01*M*S(2)-4*K12*M*S(3)+K21*S(7);
                -2*K10*S(4)+2*K01*M*S(2)-
K01*M*S(4)+3*K10*S(6)-2*K12*M*S(4)+K21*S(7)-
K23*M*S(4)+K32*S(8);
                -2*K21*S(5)+2*K12*M*S(2)-
2*K01*M*S(5)+K10*S(7)-2*K12*M*S(5)+2*K21*S(8);
                -3*K10*S(6)+K01*M*S(4)-3*K23*M*S(6)+K32*S(10);
```

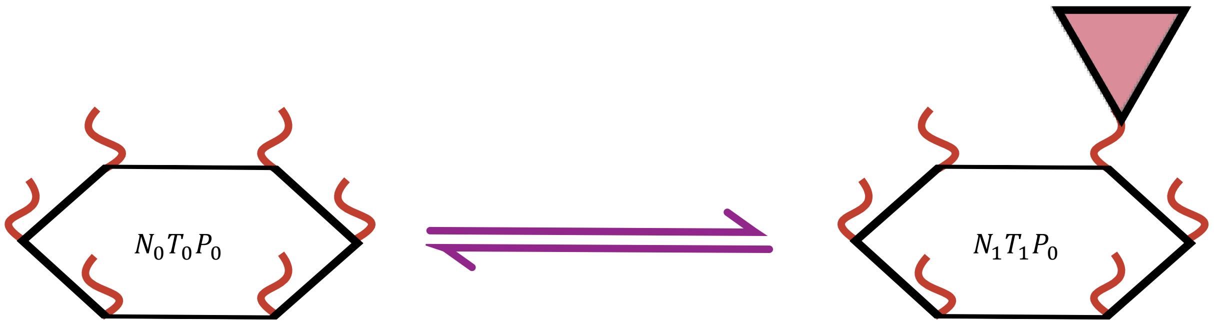
```
                    -K21*S(7)+3*K12*M*S(3)-K21*S(7)+2*K12*M*S(4)-
K10*S(7)+2*K01*M*S(5)-K23*M*S(7)+2*K32*S(11)-
K12*M*S(7)+2*K21*S(10)-K12*M*S(7)+4*K21*S(9);
                    -K32*S(8)+K23*M*S(4)-2*K21*S(8)+2*K12*M*S(5)-
2*K12*M*S(8)+2*K21*S(11)-K01*M*S(8)+K10*S(10);
                    -4*K21*S(9)+K12*M*S(7)-2*K23*M*S(9)+K32*S(12);
                    -K32*S(10)+3*K23*M*S(6)-
2*K21*S(10)+K12*M*S(7)-K10*S(10)+K01*M*S(8)-
2*K23*M*S(10)+2*K32*S(12);
                    -2*K32*S(11)+K23*M*S(7)-
2*K21*S(11)+2*K12*M*S(8)-2*K12*M*S(11)+2*K21*S(12);
                    -K32*S(12)+2*K23*M*S(9)-
2*K32*S(12)+2*K23*M*S(10)-2*K21*S(12)+2*K12*M*S(11)-
K23*M*S(12)+6*K32*S(13);
                    -6*K32*S(13)+K23*M*S(12);];

range = [0:0.0001:0.025];
S0 = [20e-6; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];
[t, S] = ode23s(dSdt, range, S0);

plot(t, S(:, 1), t, S(:, 2), t, S(:, 3), t, S(:, 4), t, S(:,
5), t, S(:, 6), t, S(:, 7), t, S(:, 8), t, S(:,9), t, S(:,10),
t, S(:,11), t, S(:,12), t, S(:,13));
legend('1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
'11', '12', '13');
xlabel('Time (s)');
ylabel('Concentration');
title('Hexamer Formation');
hold on;
S(:,14)=1*(0*S(:,1)+1*S(:,2)+2*S(:,3)+2*S(:,4)+2*S(:,5)+3*S(:,
6)+3*S(:,7)+3*S(:,8)+4*S(:,9)+4*S(:,10)+4*S(:,11)+5*S(:,12)+6*
S(:,13));
plot(t,S(:,14));
legend('1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
'11', '12', '13', 'Total curve');
```

STOCHASTIC INVESTIGATION

We also began investigation of stochastic models, which is a labelled-form of modelling. Each 'particle' is followed as it interacts in predetermined probabilistic ways. Whilst we didn't progress very far on this front, you can find our notes below.

No Neighbour Case

The stochastic modelling of the no-neighbour case of DNA Template/HIV Capsid binding is reasonably simple. We can imagine this as a single DNA Template with multiple, separated, discrete binding sites that cannot interact with each other.



The overall reaction takes the form

$$[Monomer] + [Template] \leftrightharpoons [Monomer - Template]$$

This stochastic model ignores the possibility of Protein dimerization.

The rate of change of the concentration of reactants, ignoring dissociation of product, is

$$\frac{d[M]}{dt} = k_{on}[M][T]Ms^{-1}$$

The rate of dissociation is

$$\frac{d[MT]}{dt} = k_{off}[MT]MS^{-1}$$

The rate of change of both products and reactants is therefore

$$R_{total} = k_{on}[M][T] + k_{off}[MT]\ M\ s^{-1}$$

Forward and reverse reactions

The forward reaction will have probability

$$P(M + T \rightarrow MT) = \frac{k_{on}[M][T]}{R_{total}}$$

The reverse reaction will have probability

$$P(MT \rightarrow M + T) = 1 - \frac{k_{on}[M][T]}{R_{total}}$$

Stochastic modelling

The Gillespie algorithm will randomly generate a number r between 0 and 1. If $r < P(M + T \rightarrow MT)$ then the simulation performs a forward reaction. Alternatively, if $r > P(A + B \rightarrow AB)$, then the simulation performs a reverse reaction. The system will continue until the probability of forward reaction is equal to the probability of backward reaction. However, due to the step-wise nature of the simulation, oscillations are expected

Time

The model must account for the fact that, whilst a single reaction may be more likely to occur at set time intervals, there is some probability that multiple reactions occur in rapid succession. Thus, time intervals are selected in an exponential distribution.

$$\delta t = -\log(r)$$

r is a random number between 0 and 1, and R_{total} is the number of particles converted per second. We may use a sampling_rate to increase the time resolution.

$$\delta t_{scaled} = -\log(r) \times sampling\_rate$$

$$R_{total,scaled} = R_{total} \times sampling\_rate$$

Pseudocode

Setup reaction conditions

```
- Define $k_{on} = 1 \times 10^6 M^{-1} s^{-1}$
- Define $k_{off} = 1 \times 10^{-4} s^{-1}$
- Initial $[M] = 20 \times 10^{-6} M$
- Initial $[T] = 20 \times 10^{-9} M$
- Initial $[MT] = 0\ M$
- Define approximate simulation time max_time = 10 seconds
- Define sampling_rate = 0.01 seconds
- Max number of simulation steps max_steps=max_time/sampling_rate
- In MatLab create an array 'sim_result' of size 4 by max_steps
    o Item 1 = [M]
    o Item 2 = [T]
    o Item 3 = [MT]
    o Item 4 = time
```

```
% Author Lawrence Lee, adapted by Jacob Silove
% My first attempt at writing a kinetic simulation. I am going
to use the
% Gillespie Algorithm
%
%
% The reaction is Template + Monomer <-> Template-Monomer
% forward reaction rate = k_on (/M/sec)
% reverse reaciton rate = k_off (/sec)
% we assume that k_on and k_off are constants


%%%% model parameters and initial conditions %%%%
k_on = 10e5; %/M/sec
k_off = 10e-3; %/sec
```

```
A_conc = 20e-6; %
B_conc = 1E-9; %
AB_conc = 0.0; % no product to begin with

sampling_rate = 0.005; %seconds

%%% Notes on making this in terms of seconds  - none %%%
current_time = 0.0; % seconds -
max_time = 10; %seconds
max_steps = max_time/sampling_rate;

% 1 = [Monomer]
% 2 = [Template]
% 3 = [Monomer-Template]
% 4 = time
sim_result = zeros(4,max_steps);
sim_result(1,1) = A_conc;
sim_result(2,1) = B_conc;
sim_result(3,1) = AB_conc;

current_step=1;

% simluate association phase
while current_step < max_steps
    %%setup loop%%
    bind_rate =
k_on*sim_result(1,current_step)*sim_result(2,current_step);
    diss_rate = k_off*sim_result(3,current_step);

    Rtotal = bind_rate + diss_rate;
    p_fwd = bind_rate/Rtotal;
    r = rand;
    fwd_step = 0;
    if rand < p_fwd
        fwd_step = 1;
    end;

    %determine reaction time
    Rtotal_inc = Rtotal * sampling_rate;
    dT = -log(rand)*sampling_rate;

    sim_result(4,current_step+1) =
sim_result(4,current_step)+dT;
    if fwd_step == 1
        sim_result(1,current_step+1) =
sim_result(1,current_step) - Rtotal_inc;
        sim_result(2,current_step+1) =
sim_result(2,current_step) - Rtotal_inc;
        sim_result(3,current_step+1) =
sim_result(3,current_step) + Rtotal_inc;
    else
```

```
        sim_result(1,current_step+1) =
sim_result(1,current_step) + Rtotal_inc;
        sim_result(2,current_step+1) =
sim_result(2,current_step) + Rtotal_inc;
        sim_result(3,current_step+1) =
sim_result(3,current_step) - Rtotal_inc;
    end

    current_step = current_step + 1;

end;

%%simulate dissociation
current_DisStep=1;
diss_result = zeros(2,max_steps);
diss_result(2,1) = sim_result(3,current_step);
diss_result(1,1) = sim_result(4,current_step);
% 1 = time
% 2 = [Monomor-Template]

while current_DisStep < max_steps
    diss_rate = k_off*diss_result(2,current_DisStep);
    %determine reaction time
    Rtotal_inc = diss_rate * sampling_rate;
    dT = -log(rand)*sampling_rate;
    diss_result(2,current_DisStep+1) =
diss_result(2,current_DisStep)-Rtotal_inc;
    diss_result(1,current_DisStep+1) =
diss_result(1,current_DisStep)+dT;
    current_DisStep = current_DisStep+1;
end

%plot(sim_result(4,:), sim_result(3,:))
plot(sim_result(4,:).', 1e10*sim_result(3,:).')
hold on
plot(diss_result(1,:), 1e10*diss_result(2,:))


%%calculate binding constants
% % % % Fit equation is below
 sing = fitoptions('Method','NonlinearLeastSquares',...
 'Lower',[-Inf, 1e-15 0],...
 'Upper',[Inf,Inf,Inf],...
 'Startpoint',[-1e-10, 0.1, max(sim_result(3,:).')],...
 'TolFun', 10e-7,'TolX', 10e-15);
fsing = fittype('a*exp(-b*x) + c','options',sing);
[curve goodness] =
fit(sim_result(4,:).',1e10*sim_result(3,:).',fsing);
%[curve goodness] =
fit(sim_result(4,:).',sim_result(3,:).','exp1');
goodness
```

```
curve

x = sim_result(4,:);
y = curve.a*exp(-curve.b*x)+curve.c;
%y = curve.a*exp(-1*curve.b*x);
plot(x,y);
hold off

%
% xlabel('Time')
% ylabel('Particle count')
%
% legend('N_A','N_B')
```