Supplementary Information

The SARS-CoV-2 Envelope Protein Forms Clustered Pentamers in Lipid Bilayers

Noah H. Somberg [1], Westley W. Wu [1], João Medeiros-Silva [1], Aurelio J. Dregni [1], Hyunil Jo [2], William F. DeGrado [2] and Mei Hong [1]*

[1] Department of Chemistry, Massachusetts Institute of Technology, 170 Albany Street, Cambridge, MA 02139

[2] Department of Pharmaceutical Chemistry, 555 Mission Bay Blvd. South, University of California, San Francisco, San Francisco, CA 94158

* Corresponding author: Professor Mei Hong, meihong@mit.edu

Calculation of the overlap integral F(0).

The rate constant, k_{ij} for ^{1}H-driven spin diffusion is given by:

$$k_{ij} = \frac{1}{2}\pi\omega_{ij}^2 F_{ij}(0) \tag{1}$$

where ω_{ij} is the homonuclear dipolar coupling:

$$\omega_{ij} = \frac{\mu_0\hbar}{4\pi}\gamma^2 \frac{1}{r_{ij}^3}\frac{(1-3\cos^2\theta_{ij})}{2} \tag{2}$$

The homonuclear dipolar coupling depends on the internuclear distance r_{ij} and the angle θ_{ij} between the internuclear vector and the external magnetic field. $F_{ij}(0)$ is the overlap integral describing the probability that single-quantum transitions occur at the same frequency for spins *i* and *j*.

Spin diffusion among *n* spins can be calculated from the time-evolution of of the *n*-by-1 vector of *z*-magnetization M and the *n*-by-*n* exchange matrix $\widehat{\boldsymbol{K}}$. The time evolution is given by:

$$\frac{d\vec{M}(t)}{dt} = -\widehat{\boldsymbol{K}}\vec{M}(t) \tag{3}$$

For an *n* spin system, the exchange matrix is given by:

$$\widehat{\boldsymbol{K}} = \begin{bmatrix} k_{ab}+k_{ac}+\cdots+k_{an} & -k_{ba} & -k_{ca} & \cdots & -k_{na} \\ -k_{ab} & k_{ba}+k_{bc}+\cdots+k_{bn} & -k_{cb} & \cdots & -k_{nb} \\ -k_{ac} & -k_{bc} & k_{ca}+k_{cb}+\cdots+k_{cn} & \cdots & -k_{nc} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -k_{an} & -k_{bn} & -k_{cn} & \cdots & k_{na}+k_{nb}+k_{nc}+\cdots \end{bmatrix} \tag{4}$$

Detailed balance requires that rate constants satisfy $k_{ab} = k_{ba}$, giving a symmetric matrix. Conservation of magnetization then requires that each column sums to zero. The case for a 2-spin system is highly tractable, and worth solving directly. Taking the two-spin system, we have $k_{ab} = k_{ba} = k$, giving the 2 x 2 exchange matrix:

$$\widehat{\boldsymbol{K}} = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \tag{5}$$

Let A and B be the *z* magnetization on spins A and B at time *t*. Our differential equation is:

$$\frac{d\vec{M}(t)}{dt} = \begin{bmatrix} \dot{A} \\ \dot{B} \end{bmatrix} = \begin{bmatrix} -k & k \\ k & -k \end{bmatrix}\begin{bmatrix} A \\ B \end{bmatrix} \tag{6}$$

The matrix $-\widehat{\boldsymbol{K}}$ has Trace *-2k* and Determinant 0. The eigenvalues then are $\lambda_1 = 0$ and $\lambda_2 = -2k$, corresponding to eigenvectors $[1, 1]$ and $[1, -1]$ respectively. The general solution is then:

$$\begin{bmatrix} A \\ B \end{bmatrix} = c_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + c_2 e^{-2kt} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad (7)$$

Imposing the initial condition [1, 0] gives the solution

$$\begin{bmatrix} A \\ B \end{bmatrix} = 0.5 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 0.5 e^{-2kt} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad (8)$$

The model compound 5-^{19}F-Tryptophan has two orientationally inequivalent spins in the crystal unit cell, and is thus one such system. Fitting experimental data (See **Fig. S3**) to the curve 0.5+0.5e^{-rt} gives a best fit r of 183 s^{-1} with $R^2 = 0.9935$. Note this "apparent rate" is twice the magnitude of the ^{1}H-driven spin diffusion rate constant k_{ij}. L-Tryptophan crystalizes in $P2_1$ space group with two molecules in the unit cell (CSD: 1275812). In the crystal lattice, the nearest neighbor ^{19}F-^{19}F distance between orientationally inequivalent fluorine atoms is 4.62 Å. A second-moment sum over a 10 Å radius across multiple unit cells gives the effective dipolar coupling, which corresponds to an effective distance of 4.16 Å.

With the fit and the equations above, the overlap integral can be calculated directly.

$$F_{ij}(0) = \frac{2k_{ij}}{\pi < \omega_{ij}^2 >} \quad (9)$$

where $<\omega_{ij}^2>$ is the powder-averaged ω_{ij}^2:

$$\omega_{ij}^2 = \left(\frac{\mu_0 \hbar}{4\pi} \gamma^2 \frac{1}{r_{ij}^3} \frac{(1 - 3\cos^2\theta_{ij})}{2} \right)^2 \quad (10)$$

$$< \omega_{ij}^2 > = \left(\frac{\mu_0 \hbar}{4\pi} \gamma^2 \frac{1}{r_{ij}^3} \right)^2 \frac{1}{4\pi} \int_0^{2\pi} \int_0^{\pi} \left(\frac{(1 - 3\cos^2\theta_{ij})}{2} \right)^2 \sin\theta \, d\theta d\phi$$
$$= \left(\frac{\mu_0}{4\pi} \gamma^2 \frac{1}{r_{ij}^3} \right)^2 \times \frac{1}{5} \quad (11)$$

The dipolar coupling constant can be calculated directly, or from the ^{1}H-^{1}H dipolar coupling of ω = 120,120 × 2π/s for a 1 Å distance:

$$< \omega_{ij}^2 > = \left(\frac{\mu_0 \hbar}{4\pi} \gamma^2 \frac{1}{r_{ij}^3} \right)^2 \times \frac{1}{5} = \left(120120 \times (0.941)^2 \times \left(\frac{1\text{Å}}{4.16\text{ Å}} \right)^3 \times 2\pi/s \right)^2 \times \frac{1}{5}$$
$$= 1.726 \times 10^7 \, rad^2/s^2$$

$$F_{ij}(0) = \frac{2k_{ij}}{\pi < \omega_{ij}^2 >} = \frac{183\ s^{-1}}{\pi\ 1.726 \times 10^7\ rad^2/s^2} = 3.38 \times 10^{-6} \frac{s}{rad^2} = 3.4\ \mu s \quad (12)$$

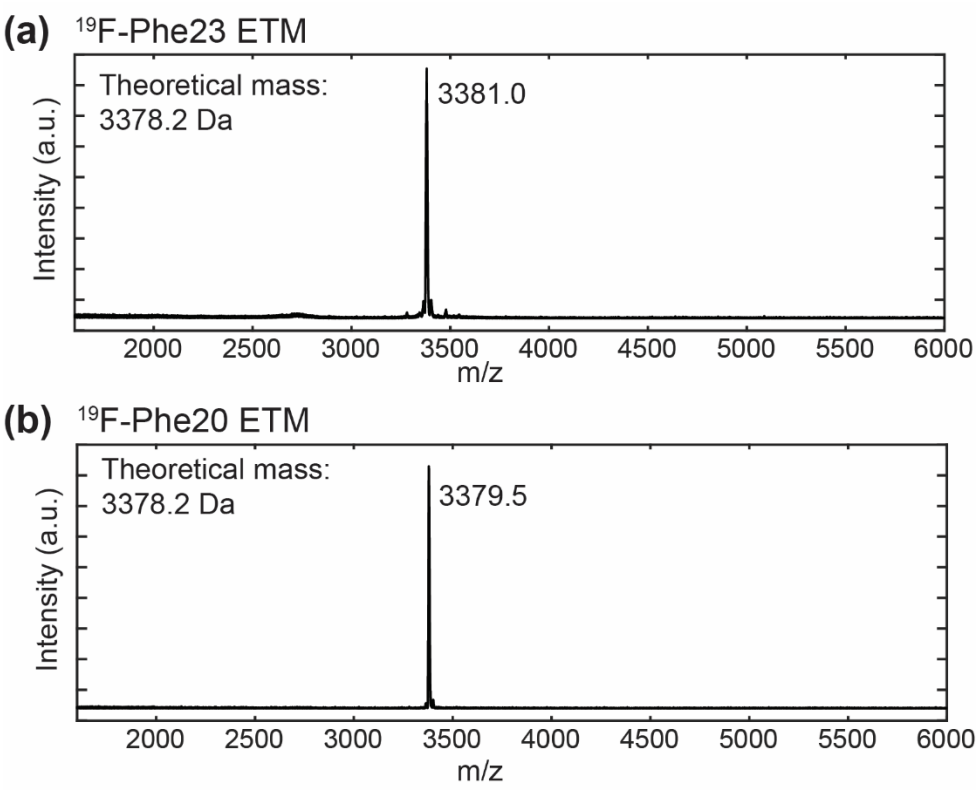

Figure S1 MALDI-TOF mass spectra of synthetic peptides. **(a)** MALDI mass spectrum of synthetic 4^{19}F-Phe23 ETM. **(b)** MALDI mass spectrum of synthetic 4^{19}F-Phe20 ETM.

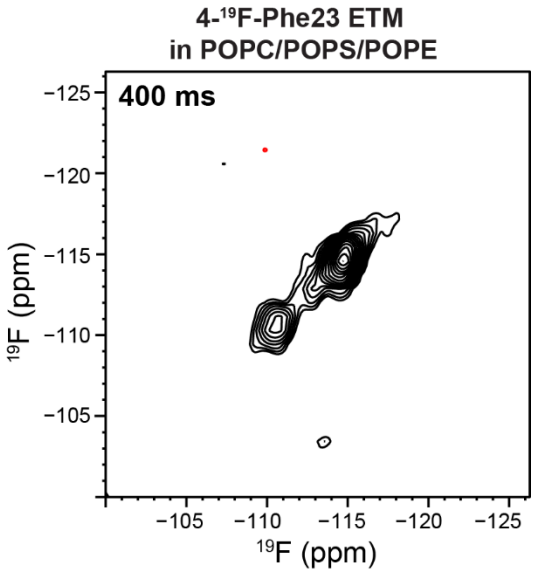

Figure S2. 2D ^{19}F-^{19}F correlation spectra of 4-^{19}F-Phe23 labeled ETM. The spectrum was measured with 400 ms CORD irradiation under 14 kHz MAS at 260 K.

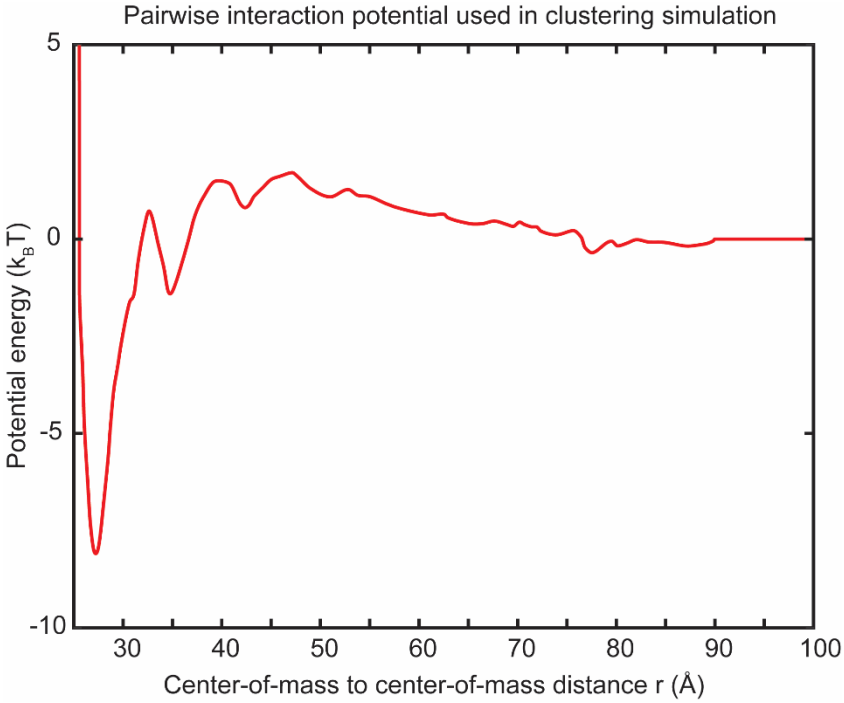

Figure S3. Plot of the pairwise interaction potential used to generate clustered oligomers, reproduced from Morozova et al [1].

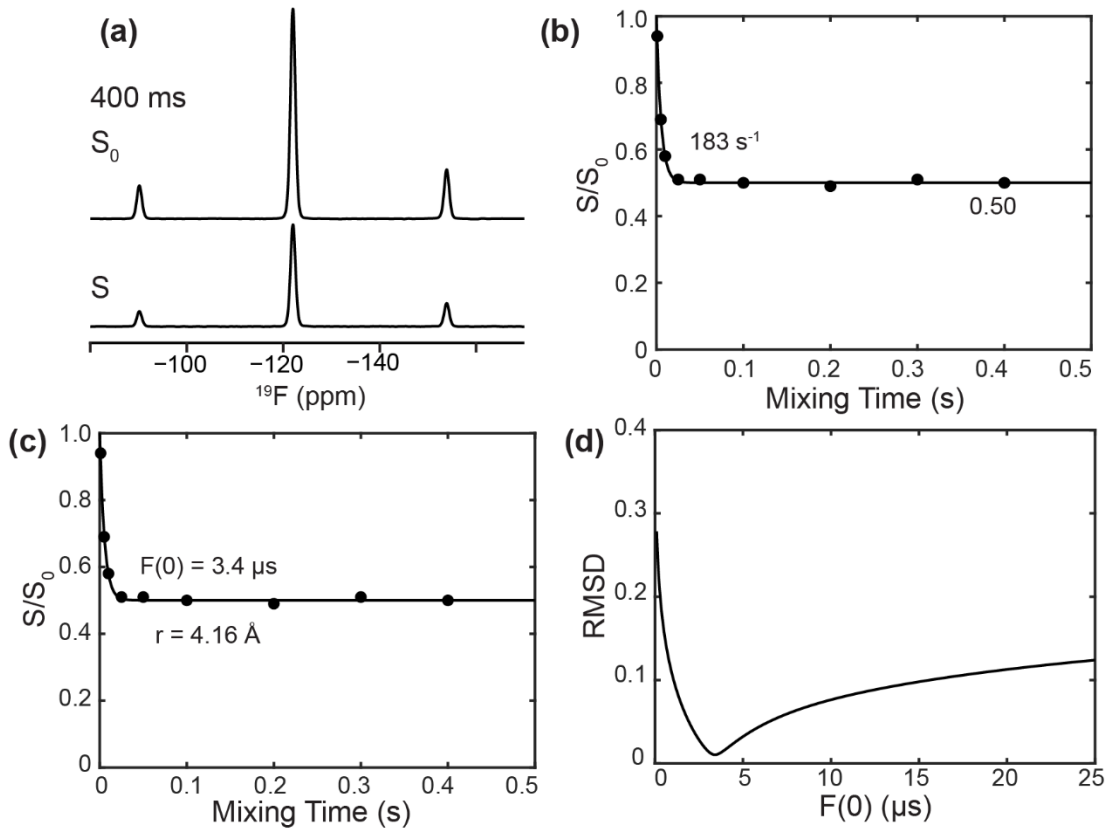

Figure S4. ^{19}F CODEX data of 5-^{19}F-tryptophan, measured under 18 kHz MAS on a 600 MHz NMR, with a ^{19}F Larmor frequency of 564 MHz. (**a**) Representative ^{19}F CODEX S_0 and S spectra, measured at a mixing time of 400 ms. (**b**) Experimental CODEX S/S_0 values (filled circles) superimposed with the best-fit exponential decay (solid line) to 0.5 with the equation $0.5 + 0.5e^{-ct}$, where c = 183 s^{-1}. The equilibrium value of 0.50 is consistent with the $P2_1$ space group of L-tryptophan (CSD: 1275812). (**c**) Best-fit matrix simulation of the experimental CODEX intensities. Second moment analysis summing over additional unit cells gives the effective distance of r = 4.16 Å. This results in best-fit overlap integral F(0) of 3.4 µs. (**d**) RMSD between matrix-simulated CODEX decays and the experimental data as a function of F(0) from 0.01 to 25 µs. The best-fit simulation with the lowest RMSD was obtained at F(0) = 3.4 µs.

Reference

1. Morozova, D.; Weiss, M.; Guigas, G., Shape as a determinant of membrane protein cluster formation. *Soft Matter* **2012,** *8*, 11905-11910.

MATLAB code for matrix calculation of CODEX decay curves

```
%-------------------------------------------------------------------------%
% CODEX_calc.m
% CODEX matrix calculation for a regular polygon of n spins
% Noah H. Somberg, Westley W. Wu, Mei Hong
% Written in MATLAB R2021b
% July 2022
% This script takes a distance matrix (which here is generated based on a
% regular polygon) and calculates a time-dependent CODEX decay using
% 1H-driven spin diffusion theory, and plots the results
%-------------------------------------------------------------------------%

%--------SIMULATION PARAMETERS--------%
gamma = 251.185e6; % Gyromagnetic ratio of fluorine
mu_0 = 1.25663706212e-6; % Vacuum permeability
hbar = 1.054571817e-34; % Planck constant
gammaProt = 267.52218744e6; % Proton gyro ratio
ang = 1e-10; % One angstrom
powd = 0.2; % Powder average of angular dependence

uplimit = 5000; % Upper bound of CODEX plot in ms (x axis)
step = 1; % Time increment in ms (smaller equals smoother curves)

n = 5; % Oligomer number
%s = 8; % Nearest neighbor distance

time_ax = 0:step:uplimit;
F0 = 3.4; % Overlap integra
s = 8.8; % NN distance in A

% Create a polygon with n sides of length n at [0,0] with initial rotation
% 0 degrees
poly = createPoly(n,s,[0;0],0);
dismatrix = zeros(n,n); % Initialize a matrix for all distances


for p1 = 1:n
    for p2 = 1:n % For each set of coordinates
        xdist = abs(poly(1,p2) - poly(1,p1)); % Get x dist
        ydist = abs(poly(2,p2) - poly(2,p1)); % Get y dist
        dismatrix(p1,p2) = sqrt(xdist^2 + ydist^2); % Get total dist
    end
end

% Calculate known couplings to double check parameter values are correct
prot_1a = (mu_0 * hbar * gammaProt^2)/(4*pi*ang^3); % 1A 1H dipolar coup
prot_1a_hz = prot_1a/(2*pi); % Convert to Hz, should be 120120 Hz
```

```
f_1a = (mu_0 * hbar * gamma^2)/(4*pi*ang^3); % 1 A F-F dip coup (in rads!)

dipcoup = f_1a;

M0matrix = eye(n); % Initial state is identity matrix

np=uplimit/step+1; % Number of points

W=dismatrix.^(-3)*dipcoup; % Homonuclear dipolar coupling strength
Wsqu=W.^2; % Coupling squared

for i=1:n % Detailed balance
    Wsqu(i,i)=0; % Zero the diagonal of the coupling matrix
    Wsqu_sums = sum(Wsqu,1); % Sum each column
    Wsqu(i,i) = -Wsqu_sums(i); % Diag set to neg sum
end

% Calculate the exchange matrix K
K=0.5*pi*Wsqu*powd*F0/1000000;
% Mt is a 3d matrix:
% first axis is the ending spin,
% second axis is starting spin,
% 3rd axis is time
Mt = zeros(n,n,np);
prop = expm(step/1000*K); % Calculate the propagator

for currSpin = 1:n
    % For each spin,
    % calc the dip coup matrix W and the dip coup square

    M0 = M0matrix(:,currSpin); % Extract the vector for init mag on spin
    currMat = expm(0/1000*K); % Calculate the initial SD matrix

    t_idx = 1; % Initialize a time index

    for t = 0:step:uplimit
        % For each time step, calculate exchange process
        Mt(currSpin,:,t_idx) = currMat*M0; % Calculate Mt
        currMat = prop*currMat; % Increment exchange matrix

        t_idx = t_idx + 1; % Increment time index
    end
end

Mt_avg = zeros(2,np); % Initialize a matrix for avg magnetization
Mt_avg(1,:) = 0:step:uplimit; % First row is time, second row is M(t) avg

% Calculate average over all spins
for t_idx = 1:np
    % Calc avg mag at each time pt
```

```
    Mt_avg(2,t_idx) = trace(Mt(:,:,t_idx))/n;
end

sim = Mt_avg(2,:);

% Plot the result
figure;
set(gca, 'FontName', 'Arial')
hold on
plot(time_ax./1000,sim,'k','LineWidth',2)
xlim([0,5]);
ylim([0, 1.1]);
xticks([0:1:5]);
yticks([0:0.2:1]);
box on
set(gca, 'FontName', 'Arial')
set(gca,'FontSize',16)
set(gca,'linewidth',2)

function points = createPoly(n,s,origin,rot)
    % Outputs a set of points forming a regular polygon with n sides of
    % length s centered at origin, with an initial rotation with respect to
    % the x axis of rot

    theta = 360/n; % Angle for drawing polygon radial vectors
    points = zeros(2,n); % Initialize output matrix
    Rotmat = [cosd(theta), -sind(theta); sind(theta), cosd(theta)];
    % Length of first vector based on side, using law of cosines
    veclen = sqrt(s^2/(2-2*cosd(theta)));

    xtran = origin(1);
    ytran = origin(2);

    % Get first vertex from length of radial vector and initial inclination
    % angle
    points(:,1) = [veclen*cosd(rot); veclen*sind(rot)];

    % Calculate remaining points by applying rotation matrix
    for i = 2:n
        points(:,i) = Rotmat*points(:,i-1);
    end

    % Translate the shape to the specified origin
    points(1,:) = points(1,:)+xtran;
    points(2,:) = points(2,:)+ytran;
end

%-------------------------------------------------------------------------%
% End of CODEX_calc.m
%-------------------------------------------------------------------------%
```

MATLAB code for simulating a Random Sequential Adsorption of oligomers to a square

```
%-------------------------------------------------------------------------%
% RSAsimulation.m
% Random sequential adsorption of pores
% Westley W. Wu, Noah H. Somberg, Mei Hong
% Written in MATLAB R2021b
% July 2022
% This script takes a specified phospholipid to protein ratio, oligomer
% number, and lipid head group area to place an correspondingly-computed
% number of pores randomly on a 1000 A x 1000 A area, checking for overlap
% with already-placed pores before placing new ones. Outputs the final
% coordinates of the center of each oligomer in
% 'centers_[phospholipid:protein ratio]_[oligomer number].csv'
%-------------------------------------------------------------------------%

% RSAsimulation() places pore centers by randomly adding centers in sequence.
% finalCenters is an output of this function so that other CODEX simulation
% scripts can use the output of this function directly.
function finalCenters = RSAsimulation()
    w = 1000.0; % Width of membrane patch in angstroms
    counter = 0; % Checks how many pores have already been added

    LPRatio = double(input("Lipid:protein ratio: ")); % 10-40
    oligoNum = double(input("Oligomer number: ")); % 4, 5, 6 etc.
    lipidArea = double(input("Area per lipid: ")); % ~60 ang^2
    radius_set = 10; % Radius of the circular footprint of the channel
    % Computes the number of pores
    numPores =
floor((w^2)/(((LPRatio/2.0)*oligoNum*lipidArea)+pi*(double(radius_set)^2)));

    % Initializes array in which the ordered pairs for the final center
    % locations are to be stored
    center_tuples = zeros(numPores, 2);

    while counter < numPores %places the pores sequentially
        x_random = w*rand;
        y_random = w*rand;
        % Checks to see if there is already a pore that could overlap with
        % where we want to place the new pore
        if checkCircle(center_tuples(1:(counter+1),:), x_random, y_random,
radius_set)
            % If there is no overlap, the pore is placed
            center_tuples(counter+1,1) = x_random;
            center_tuples(counter+1,2) = y_random;
            counter = counter + 1;
        end
    end

    finalCenters = center_tuples;
    % Saves the file of pore centers to use later if needed
    writematrix(finalCenters,
strcat('centers_',string(LPRatio),'_',string(oligoNum),'.csv'));
end
```

```
% yesNo returns true if we can place an oligomer at (x,y) given a list of
% previous pore locations and false if we cannot. oldCenters contains the
% list of ordered pairs of already-placed pores, and r is the radius around
% each oligomer center where there cannot be another oligomer.
function yesNo = checkCircle(oldCenters, x, y, r)
    yesNo = true;
    %vectorized computation of distance from (x, y) to every existing point
    hitPoints = repmat([x y], size(oldCenters,1), 1);
    distances = sqrt(sum(((hitPoints-oldCenters).^2),2));

    for i = 1:size(distances,1) % Cycles through already-placed pores
        if (distances(i,1) <= 2*r) % Verifies center-center distance is OK
            % If even a single overlap occurs, yesNo is set to false and we
            % break out of the function because there is no need to cycle
            % through remaining entries in oldCenters
            yesNo = false;
            break
        end
    end
end

%-------------------------------------------------------------------------%
% End of RSAsimulation.m
%-------------------------------------------------------------------------%
```

MATLAB code for Monte Carlo simulation of pore clustering

```
%-------------------------------------------------------------------------%
% ClusteringMonteCarlo.m
% Simulated clustering of oligomers given a pairwise interaction potential
% Westley W. Wu, Noah H. Somberg, Mei Hong
% Written in MATLAB R2021b
% July 2022
% This script takes a specified number of pores and places them randomly on
% a 1000 A x 1000 A area. It then applies the interaction potential and
% simulates with a Metropolis Monte Carlo method. The interaction potential
% is read from the file 'interaction_potential.txt'. This simulation uses
% periodic boundary conditions and outputs the final coordinates of the
% center of each oligomer in 'finalCenters.csv'
%-------------------------------------------------------------------------%

%%%%%%%% Begin main program %%%%%%%%

%%% Initialize a random list of ordered pairs to serve as an initial random
%%% distribution of pores

% How many pores we place – for 1:17 P:L ratio in a 1000 A x 1000 A box,
% 349 pores are appropriate
npores = 349;
% Generates a membrane patch with 1:17 P:L ratio given the number of pores
sl = sqrt(1000000*(npores/349));
% Initial pore center locations are created. MATLAB randomly picks these
% center coordinates based on the number of pores specified by npores. Note
% that oligomer overlap is allowed
coords = sl*rand(npores,2);
% Saves a record of these initial centers just for future reference
writematrix(coords, 'originalCenters.csv');

% Number of iterations of the Metropolis Monte Carlo algorithm we want to
% simulate. In other words, how many individual oligomer positional changes
% we consider making
ntimes = 50000;

%%% For later plotting of average nearest neighbor (NN) distances

% Average NN distances table: first column will be the iteration the
% simulation is on. The second column will be the NN distance averaged over
% all npores oligomers.
nnVtime = zeros(ntimes+1, 2);
% Computes an average NN distance for the initial pore setup prior to any
% Monte Carlo simulations
nnVtime(1, :) = [0 ANNDistance(coords, sl)];

% For each cycle, we attempt to move a randomly-chosen oligomer to a randomly-
% chosen position. We then use the change in energy between the current and
% proposed configurations to decide if we should accept or reject the proposed
% configuration. Periodic boundary conditions are used.
for i = 1:ntimes
    currentRow = randi(npores); % Chooses a random oligomer to try and move
    % coordsFinal always refers to a proposed state of the system. Here,
```

```
    % we reset the proposed state of the system as the most recently-accepted
    % state at the start of each loop
    coordsFinal = coords;

    % Gets positional information about randomly-chosen point
    current_x = coords(currentRow,1);
    current_y = coords(currentRow,2);

    %computes the energy of initial arrangement
    nearOriginal = nearMe(current_x, current_y, coords, sl); %finds nearest points to
include in an energy calculation of the initial state
    E_i = computeEnergy(coords(currentRow,:), nearOriginal); %actually computes the
energy

    %picks up the point we randomly chose before and moves it to a random
    %location
    finalCoordsForPoint = sl*rand(1,2); %chooses random destination
    coordsFinal(currentRow,:) = finalCoordsForPoint; %changes the position of the
point to the proposed new state

    %gathers positional information about the point's new location
    current_x_f = coordsFinal(currentRow,1);
    current_y_f = coordsFinal(currentRow,2);

    %computes the energy of propsed final state
    nearFinal = nearMe(current_x_f, current_y_f, coordsFinal, sl); %finds nearest
points to include in an energy calculation of the final state
    E_f = computeEnergy(coordsFinal(currentRow,:), nearFinal); %actually computes the
energy

    %decision time: do we accept the new state?
    if E_i-E_f < 0 %if the energy of the proposed state is less favorable than the
energy of the starting state
        if rand <= exp(E_i-E_f) %acceptance criterion
            coords = coordsFinal; %if accept, change system
        end
    else %if the energy of the proposed state is lower, we always accept the proposed
state and work from there next iteration
        coords = coordsFinal;
    end
    nnVtime(i+1, :) = [i ANNDistance(coords, sl)]; %updates NN distance datatable
    disp(i); %displays which iteration we are on, just for reference and can delete
this line if so desired
end

%writes centers of a 3x3 extended grid (to display periodicity)
q1 = [coords(:,1)-sl coords(:,2)+sl];
q2 = [coords(:,1) coords(:,2)+sl];
q3 = [coords(:,1)+sl coords(:,2)+sl];
q4 = [coords(:,1)-sl coords(:,2)];
q5 = coords;
q6 = [coords(:,1)+sl coords(:,2)];
q7 = [coords(:,1)-sl coords(:,2)-sl];
q8 = [coords(:,1) coords(:,2)-sl];
q9 = [coords(:,1)+sl coords(:,2)-sl];
```

```
expandedGrid = [q1;q2;q3;q4;q5;q6;q7;q8;q9];

%saves results
writematrix(expandedGrid, 'expandedFinalCenters.csv'); %saves final pore center
positions with periodic boundary conditions
writematrix(coords, 'finalCenters.csv'); %saves final pore centers w/o periodicity
writematrix(nnVtime, 'avg_NN_dist_v_time.csv'); %saves the time vs average nearest
neighbor data

%figures, for descriptions of figures see their titles

figure(1);
plot(nnVtime(:,1),nnVtime(:,2));
title('Average nearest neighbor distance over time');
xlabel('Timestep');
ylabel('Mean NN distance (angstrom)');
xlim([0 ntimes]);

figure(2);
polygonPoints(5, 8, 0);
xlim([-sl 2*sl]);
ylim([-sl 2*sl]);
title('Final centers');

figure(3);
polygonPoints(5, 8, 1);
xlim([-sl 2*sl]);
ylim([-sl 2*sl]);
title('Original centers');

figure(4);
polygonPoints(5, 8, 2);
xlim([-sl 2*sl]);
ylim([-sl 2*sl]);
title('Final centers w/ periodic boundary conditions');

figure(5);
x = movmean(nnVtime(:,1),500);
y = movmean(nnVtime(:,2),500);
plot(x,y)
title('Average nearest neighbor distance over time');
xlabel('Timestep');
ylabel('Moving average of mean NN distance (angstrom)');


%%%%%%%% functions used in main simulation listed below %%%%%%%%

function energy = computeEnergy(pt, nearby) %computes the energy of the system
provided a given center location (pt) and a list of its nearest neighbors (nearby)
    %vectorized approach to end up with a list of all distances from the
    %specified point to all the points in the matrix of nearby points
    pointMatrix = repmat(pt, size(nearby,1), 1);
    dist = sqrt(sum(((pointMatrix-nearby).^2),2));

    %computes the energies from distance information
```

```
    if size(dist,1)>=1 %if there are any distances to compute energies from
        dist = double(dist); %converts to double just to be safe
        energies = potential(dist); %runs the distance list through the potential
energy function
        energy = sum(energies,'all'); %adds together all the individual pairwise
energies computed in the line above
    else %if a pore center has no near neighbors, we don't bother computing energy
        energy = 0;
    end
end

function energy = potential(r) %potential function, r is center-center distance in
angstroms
    values = readtable('interaction_potential.txt'); %reads from file
    d = table2array(values(:,1)); %first column contains distances in angstroms
    y = table2array(values(:,2)); %second column contains energies in units of kB*T
    pp = griddedInterpolant(d,y,'makima'); %datafile is a discrete table, uses the
modified Akima interpolation method between discrete points
    energy = pp(double(r)); %computes a numerical value of the energy
end

function expandedGrid = nearMe(x,y,coords,sl) %finds the center coordinates of all
pores at distances close enough to (x,y) such that the pairwise interaction potential
with (x,y) is possibly nonzero
    %initiates 9 copies of the grid in 3x3 arrangement for periodic boundary
conditions
    q1 = [coords(:,1)-sl coords(:,2)+sl];
    q2 = [coords(:,1) coords(:,2)+sl];
    q3 = [coords(:,1)+sl coords(:,2)+sl];
    q4 = [coords(:,1)-sl coords(:,2)];
    q5 = coords;
    q6 = [coords(:,1)+sl coords(:,2)];
    q7 = [coords(:,1)-sl coords(:,2)-sl];
    q8 = [coords(:,1) coords(:,2)-sl];
    q9 = [coords(:,1)+sl coords(:,2)-sl];
    expandedGrid = [q1;q2;q3;q4;q5;q6;q7;q8;q9];
    rowsToRemove = true(size(expandedGrid,1), 1); %by default, remove the entry
unless told otherwise
    same_counter = 0; %keeps track of how many points in expandedGrid have exact
coorindates (x,y); this is only applicable to the extremely unlikely edge case of
different pores having exactly-overlapping centers
    %removes all entries from the expanded periodic set of points outside a
    %200x200 angstrom square centered on (x,y)
    for row = 1:size(expandedGrid,1)
        if (expandedGrid(row,1) == x) && (expandedGrid(row,2) == y) %if the
coordinate is the same as the query point...
            same_counter = same_counter+1; %keep track of it
        end
        if (expandedGrid(row,1) >= x-100) && (expandedGrid(row,1) <= x+100) &&
(expandedGrid(row,2) >= y-100) && (expandedGrid(row,2) <= y+100) &&
(expandedGrid(row,1) ~= x) && (expandedGrid(row,2) ~= y)
            rowsToRemove(row,1) = false; %within the box of interest, so do not
remove entry
        end
    end
```

```
    expandedGrid(rowsToRemove, :) = [];
    temp = repmat([x y], same_counter-1, 1);
    expandedGrid = [expandedGrid; temp];
    %we end up with a list of all pore centers within a 200x200 box of the
    %query point, EXCEPT for the coordinates of the query point itself,
    %since we don't want a nonexistent self-self pairwise interaction
    %contributing to our energy computations.
end

function polygonPoints(n,s,p) %n = oligo number; s = side length; p = centers file to
plot
%plots randomly-angled symmetric pentamers from the centers file generated
%from the Monte Carlo simulation. The code here is ONLY for purposes of
%visualization. the actual pentagon orientations, vertices, and distance
%matrix used in CODEX are generated in our CODEX pentagon-plotting script,
%not here. The output to the entire Monte Carlo simulation is just a file
%of pore centers.
    if  p == 0
        centers = readmatrix("finalCenters.csv"); %filename
    elseif p == 1
        centers = readmatrix("originalCenters.csv"); %filename
    else
        centers = readmatrix("expandedFinalCenters.csv"); %filename
    end
    randi = zeros(1,n); %to be filled with coordinates of the vertices of a regular
n-gon assuming center is (0,0)
    circumradius = (double(s)/2)/(sin(pi/n));
    for j = 1:size(centers,1) %cycles through pore centers, generating a randomly-
angled regular pentagon at each center
        randStart = rand*2*pi/n;
        for i = 1:n
            randi(i) = randStart+2*pi*i/n;
        end
        x_coord = centers(j,1)+circumradius*cos(randi);
        y_coord = centers(j,2)+circumradius*sin(randi);
        patch(x_coord, y_coord, 'blue'); %plots the entire grid of pentagons
    end
end

function dist = ANNDistance(coords, sl) %computes an average nearest neighbor (NN)
distance for given center coordinates (coords) and side length of non-periodic
membrane patch (sl)
    %initiates 9 copies of the grid in 3x3 arrangement for periodic boundary
conditions
    q1 = [coords(:,1)-sl coords(:,2)+sl];
    q2 = [coords(:,1) coords(:,2)+sl];
    q3 = [coords(:,1)+sl coords(:,2)+sl];
    q4 = [coords(:,1)-sl coords(:,2)];
    q5 = coords;
    q6 = [coords(:,1)+sl coords(:,2)];
    q7 = [coords(:,1)-sl coords(:,2)-sl];
    q8 = [coords(:,1) coords(:,2)-sl];
    q9 = [coords(:,1)+sl coords(:,2)-sl];
    expandedGrid = [q1;q2;q3;q4;q5;q6;q7;q8;q9];
    dist = 0; %running sum of NN distances
```

```
    idx = knnsearch(expandedGrid,coords,"K",2);
    for i = 1:size(coords,1) %cycles through each pore in ONLY the non-expanded
grid...
        dist = dist + distance(coords(i,:), expandedGrid(idx(i,2),:)); %...but NN
computations take into account centers that appear as a result of periodic boundary
conditions
    end
    dist = dist/size(coords,1); %computes the actual average by dividing by number of
pores
end

function dist = distance(a, b) %used by many other functions but not directly called
by main program – computes the distance between two ordered pairs a: (a(1), a(2)) and
b: (b(1), b(2))
    dist = sqrt((a(1)-b(1))^2+(a(2)-b(2))^2);
end

%-------------------------------------------------------------------------%
% End of ClusteringMonteCarlo.m
%-------------------------------------------------------------------------%
```

Interaction potential used in the clustering simulation ('interaction_potential.txt')

First column is *x*-coordinate in angstrom, second column is *y*-coordinate (energy) in units k_BT

```
-50    10000000.0
0      10000000.0
10     10000000.0
20     10000000.0
25.56605531  -1.351146862
25.72576996  -2.443453329
25.91423324  -3.521822236
26.03881066  -4.623287671
26.41254292  -6.26791972
26.69204355  -7.361819051
27.09771874  -8.060289901
27.50179679  -7.920516088
28.1246839   -6.531936923
28.52876195  -5.469098439
28.68368515  -4.902437082
29.08936034  -3.808537751
29.43114968  -3.305590953
29.77293902  -2.730168844
30.61303804  -1.659764256
31.11134773  -1.403711373
31.51542577  -0.588961453
32.05845556  0.20269194
32.68134267  0.712806626
33.61727048  -0.195524052
34.10440014  -0.704842306
34.59632124  -1.365084422
35.03234222  -1.326059255
36.08965315  -0.50374323
36.68059733  0.031459063
37.2092528   0.551529149
38.08129475  1.055670596
39.35741475  1.490124243
40.10328214  1.490124243
40.78845796  1.428002549
41.94000053  0.884835935
42.34407858  0.807582032
42.74815663  0.869305511
43.24646631  1.089518955
43.96198791  1.272698312
45.14387627  1.552245938
45.8913408   1.614367633
47.13551787  1.707550175
47.69611627  1.606403313
48.94029334  1.311723479
50.99422365  1.086731443
52.86128783  1.272698312
53.79561849  1.125358394
55.07014135  1.086731443
56.75033939  0.900366359
57.68467006  0.815148136
61.01312322  0.621216948
```

```
62.53839806  0.636747372
62.91213033  0.551529149
64.40546224  0.434851864
65.41805308  0.388260593
66.42744962  0.396224912
67.62850374  0.462328767
68.91740091  0.373128385
69.50834509  0.326537114
70.22386669  0.434851864
70.69023345  0.373128385
71.34506349  0.318572794
71.93600767  0.31100669
72.37202864  0.202293724
73.89570634  0.109111182
75.57590438  0.217824148
76.44794634  0.046989487
76.8216786   -0.2166295
77.6298347   -0.348438993
79.54002183  -0.056944887
80.15332606  -0.176011469
81.0525195   -0.109111182
82.04913887  -0.015132208
83.32366172  -0.076855687
84.84893657  -0.084820006
87.0274443   -0.178002549
88.86256555  -0.131411277
89.79689621  -0.053759159
90    0.0
100   0.0
200   0.0
600   0.0
1000  0.0
1500  0.0
```