Fitting Program Code and Instructions

a. General Description

The data from the Job plot experiments can be fit parametrically to a mathematical model for a specific ensemble. The programs for performing these fits were written for use in MatLab version 7.1. Fitting Job plot data can be achieved without any prior knowledge of programming or use of MatLab by following the directions for the end user (vide infer). Annotated program code is also included. In the program code sections, the primary functions are boxed in by a double line, and the other code is boxed in by a single line so as to be easily distinguished from the annotations. Note that although relative integrations and relative populations are different, here they are used synonymously.

One of three programs fits the experimental data to the model ensemble.

- **Single Aggregate** allows for fits to ensembles composed of a single aggregation state--ensembles of dimers, tetramers, and hexamers. Single aggregate allows constraint of the fit to a symmetric plot for ensembles of dimers or of enantiomers.
- **Monomer_Dimer** allows for fits to ensembles of monomers and dimers.
- **Dimer_Tetramer** allows for fits to ensembles of dimers and tetramers.

The Single_Aggregate functions were extended to allow for fitting data of mixtures of monomers and dimers or mixtures of dimers and tetramers. Note that although the functions share the same or similar names as those used for the Single_Aggregate case, their input and behavior differs. Each program consists of a data file, the experimental result input file, in addition to several other files described below.

- **Try_fit.** The script compares the experimentally measured data against current model parameters by plotting the curves for the model and the experimental data provided in the data file. The script also computes and reports the error between the model and the measurements. The inputs are: 1) XA or Xa 2) phi, phi_monomer, phi_dimer and/or phi_tetramer 3) peak_assignment, peak_assignment_monomer, peak_assignment_dimer, and/or peak_assignment_tetramer 4) Expt_Populations 5) Expt_Errors.

  The outputs are: 1) plot of experimental relative integrations versus XA 2) a plot of the theoretical model according to the phi's provided on the same axis as the experimental data 3) the root mean square difference between theory and experiment. If no errors are given, then all points are weighted equally. If errors are given, then the most precise points are weighted more heavily in the error calculations. For each $[A_nB_m]$, the script computes both the mean error and root mean square error. If the mean error is positive, then the model over-estimates the abundance of $[A_nB_m]$ on average. The root mean square error measures the goodness of the fit.

- **Refine_fit**. The concentration of each species in an ensemble, $[A_nB_m]$, depends on the mole fraction of the subunit **A,** XA(j), and on the relative

stability, phi(n+1). This function refines the initial estimate of phi to more closely match the predicted relative integrations to the experimental relative integrations. When the script reaches completion, it reports the new values of phi and the root mean square difference between the predicted and observed populations. The inputs are: 1) XA or Xa 2) phi, phi_monomer, phi_dimer and/or phi_tetramer 3) peak_assignment, peak_assignment_monomer, peak_assignment_dimer, and/or peak_assignment_tetramer 4) Expt_Populations 5) Expt_Errors. The outputs are the refined values of phi, phi_new, phidnew and/or phitnew, and the root mean square error of the new estimate, error. The script Refine_fit_s is used for fitting data from ensembles of a single aggregation state which requires a symmetric plot. Symmetry is required if A and B are enantiomers or if the ensemble is of dimers although the reason is different in each case. The symmetry is achieved in this script by requiring that A_nB_m and A_mB_n have same value of phi. For ensembles which do not have inherent symmetry in the plot, the script refine_fit should be used. The inputs and outputs are the same as in refine_fit.

- **Multimers**. This script computes the concentrations of each species in the ensemble according to the predicted XA(j) and phi's. In this parameterization, the aggregate concentrations are adjusted until the closest XA(j) to the experimental result is obtained. These adjustments in turn determine all the aggregate concentrations. Multimers can be fed multiple values XA(j) and will determine the aggregate concentrations for each value. Parameterizations specific to each fitting program are detailed before the code.

- **Populations**. The concentrations of A_nB_m computed by Multimers are converted by the function in Populations into calculated values for the relative integrations. For each value of XA(j), a set of populations (PP(j,n+1) or Model_populations) is predicted.

- **Error_of_Model**. This script evaluates the fit of the model by comparing each model population to the experimentally measured population, weighted by the accuracy of the measurement.

b. Instructions for the end user

1. Open the program folder and then open the data file.
   Single Aggregate: Data1
   Monomer_Dimer: Data_Monomer_Dimer
   Dimer_Tetramer: Data_Dimer_Tetramer
2. Enter the experimental data as described in the data file. Data may be copied and pasted from Excel or other programs. Save.
3. Open MatLab.
4. Choose the directory where the program is stored from the drop down menu at the top of the workspace.
5. Load the data by entering the name of the data file in the command line.

6. Enter the appropriate Try_fit function for plotting the experimental data and the initial curves removing the Expt_Errors input from the function if no Expt_Errors were entered:
   Single Aggregate: try_fit(XA, phi, peak_assignment, Expt_Populations, Expt_Errors)
   Monomer_Dimer: try_fit(Xa, Ctotal, phi_monomer, peak_assignment_monomer, phi_dimer, peak_assignment_dimer, Expt_Populations, Expt_Errors)
   Dimer_Tetramer: try_fit(Xa, Ctotal, phi_dimer, peak_assignment_dimer, phi_tetramer, peak_assignment_tetramer, Expt_Populations, Expt_Errors)
7. Look at the resulting figure and check that the data points are plotted correctly and that the curves' colors coordinate with the data points' colors. If not, an error in the peak assignment has been made. If a curve is missing, check that the phi's have been input correctly. Entering *whos* in the command line will return information about which parameters the computer has recognized as well as the size of the parameter arrays. If need be, return to the data file and make corrections. Be sure to save, reload the data file, and check the plot again after making the corrections. Enter *clf* in the command line clear the figure.
8. Adjust the initial phi values such that the curves have a reasonable starting point by returning to the data file and entering new values. Be sure to save, reload the data file, and check the plot again. Alternatively, enter *phi = [# # # etc.]* in the command line. Use the Try_fit function to check how close the curves are to the data.
9. Enter the function for performing the fit found in the Refine_fit or Refine_fit_s script. If no Expt_Errors were entered, remove the Expt_Errors term from the refine_fit input.
   Single Aggregate: [phi_new, error] = refine_fit(XA, phi, peak_assignment, Expt_Populations, Expt_Errors)
   Single Aggregate symmetric: [phi_new, error] = refine_fit_s(XA, phi, peak_assignment, Expt_Populations, Expt_Errors)
   Monomer_Dimer: try_fit(Xa, Ctotal, phi_monomer, peak_assignment_monomer, phi_dimer, peak_assignment_dimer, Expt_Populations, Expt_Errors)
   Dimer_Tetramer: try_fit(Xa, Ctotal, phi_dimer, peak_assignment_dimer, phi_tetramer, peak_assignment_tetramer, Expt_Populations, Expt_Errors)
10. If the fit stalls, press ctrl+c to abort the fit. A list of where the error occurred will be returned. For fitting ensembles of monomers and dimers or dimers and tetramers, a stalled fit likely indicates the need to set and fix one of the phi's for missing aggregates to 0.00001. The output provides the optimized phi's and the error (root mean square of the weighted sum of squares of the residuals).
11. Replace the initial phi values with the optimized values by entering in the command line: *phi = phi_new* for the single aggregate fit, *phi_monomer = phi_monomer_new* and *phi_dimer = phi_dimer_new* for the monomer_dimer fit, and *phi_dimer = phi_dimer_new* and *phi_tetramer = phi_tetramer_new* for the dimer_tetramer fit.

12. Observe the optimized fit using the try_fit function.
13. Better fits may be obtained using different start values if a local minimum was obtained.

c. Definitions

1. **XA(j)** is the mole fraction of **A** for a set of relative integrations.
2. **Ctotal** is the absolute subunit concentration and is only used in the Monomer_Dimer and Dimer_Tetramer fits.
3. **Expt_Populations** are the relative integrations of the observed aggregates with a range of (j,k) -> k-th NMR peak.
4. **Expt_Errors** is the experimental error in Expt_Populations(j,k) and is an optional input. In the Try_fit script, Expt_Errors are the size of the error bars.
5. **peak_assignment**, **peak_assignment_monomer**, **peak_assignment_dimer**, and **peak_assignment_tetramer** assign each column of Expt_Populations to a specific aggregate. The program expects aggregates to be ordered by increasing number of A subunits.
6. **phi**, **phi_monomer**, **phi_dimer**, and **phi_tetramer** are a measure of the relative stability of the aggregate and is related to the free energy of the species by *kT*ln phi. Phi's must be positive integers and have N+1 values for an ensemble consisting of species with N total subunits. The program expects phi's to be listed in the order of increasing **A** subunit as it is in peak_assignment. Phi_monomer, phi_dimer, and phi_tetramer are used in the monomer_dimer and dimer_tetramer fits.
7. **phi_constant** allows some values of phi to be fixed in the monomer_dimer and dimer_tetramer programs and has the same length as the total number of phi's.
8. **Expt_weights(j,k)** are the amount each experimental data point contributes to the error in the fit. It is calculated by 1. /(Expt_Errors + mean(mean(Expt_Errors))). If no Expt_Errors are entered, all points are weighted equally.
9. **Concentrations**, is the set of concentrations for each aggregate in the ensemble, $[A_nB_m]$. **Concentration_monomer(j,n+1)**, **Concentration_dimer(j, n+1)**, and **Concentration_tetramer (j, n+1)** are the concentrations of the monomer, dimer and tetramer aggregates in the Monomer_Dimer and Dimer_Tetramer fits.
10. **PP** and **Model_Populations** are the calculated relative populations/integrations for the model of the ensemble.
11. **mean_error** is the root mean square difference in the weighted sum of squares of the residuals over the entire fit.
12. **pop_error(1,j)** is the mean of population j - mean of model value for population j. For example, pop_error(1,1) is the mean amount the prediction of $[A_0B_N]$ misses the measured amount. The value could be negative, zero, or positive.
13. **pop_error(2,j)** is the root mean square error for population j. For example, pop_error(2,1) is the root square amount the prediction of $[A_0B_N]$ missed by. Its value is always positive.
14. **phi_new**, **phidnew**, and **phitnew** are the refined values of phi.

15. **error** is the root mean square error of new estimate.

d. Single Aggregate Fitting Code

d.1. Single Aggregate--Data1

XA(j). The mole fractions are listed in terms of **A** such that the first mole fraction corresponds to the first row of Expt_Populations.

```
XA = [0.000   0.084   0.145   etc.];
```

Expt_Populations are input as a column of data for each aggregate. If an aggregate has two resonances, they should be summed and entered as a single relative integration. Recall that each row must end with a semi-colon.

```
Expt_Populations =[0.000   0.000   1.000   ;
0.000   0.168   0.832   ;
0.000   0.290   0.710   ;
etc.;];
```

Expt_Errors are input as an array the same size as Expt_Populations. Each column and row should correspond to the row and column of Expt_Populations. Expt_Errors is optional and should be left as "Expt_Errors = [;];" if none are provided.

```
Expt_Errors = [ ;];
```

peak_assignment. The program expects the peaks to be listed in the order of increasing **A** subunit as shown below:

Hexamers: peak_assignment = [$[A_0B_6]$ $[A_1B_5]$ $[A_2B_4]$ $[A_3B_3]$ $[A_4B_2]$ $[A_5B_1]$ $[A_6B_0]$]
Tetramers: peak_assignment = [$[A_0B_4]$ $[A_1B_3]$ $[A_2B_2]$ $[A_3B_1]$ $[A_4B_0]$]
Dimers: peak_assignment = [$[A_0B_2]$ $[A_1B_1]$ $[A_2B_0]$]

The column number of the corresponding relative integration is input in the order designated above. If an aggregate was not observed, it should be assigned to column 1. Its absence will be accounted for in phi. For example, if the populations are listed in the order: $[A_2B_2]$ $[A_3B_1]$ $[A_1B_3]$ $[A_0B_4]$ $[A_4B_0]$, then peak_assignment = [4 3 1 2 5]. If a relative integration is a sum of two unresolved resonances, enter the same column for each aggregate contributing to the resonance.

```
peak_assignment = [ 3 2 1 ];
```

phi is listed in the order of increasing **A** subunit as it is in peak_assignment. The phi's provided here will be the start value for the fit. If an aggregate was not observed, it should be assigned a phi value of 0.

```
phi = [ 1  1  1];
```

d.2. Single Aggregate—Try Fit

```
function try_fit(XA, phi, peak_assignment, Expt_Populations, Expt_Errors
```

Code for weighting the Expt_Populations. If no Expt_Errors are entered, all points are weighted equally.

```
   if (nargin<5)
          Expt_weights=ones(size(Expt_Populations));
   else
          Expt_weights = 1./( Expt_Errors + mean(mean(Expt_Errors)));
   end
```

Code for plotting the experimental data.

```
    hold on ; cscheme='bgrmkcybgrmkcy'; axis([0 1 0 1]); xlabel('X_A');
ylabel('Mole Fractions');
    for j=1:size(Expt_Populations,2)
        if (nargin<5)
           plot(XA, Expt_Populations(:,j),sprintf('%so',cscheme(j)));
        else
           errorbar(XA, Expt_Populations(:,j),
Expt_Errors(:,j),sprintf('%so',cscheme(j)));
        end
    end
```

Code for plotting the model with the provided phi's.

```
    XAc = [0:0.01:1]; TP=Populations(multimers(XAc,phi), peak_assignment);
    for j=1:size(TP,2)
       plot(XAc,TP(:,j),sprintf('%c',cscheme(j)) );
    end
```

Code for computing and reporting the error.

```
   [mean_error, pop_error] = Error_of_Model(XA, phi, peak_assignment,
Expt_Populations, Expt_weights);
   N = length(phi)-1;
        fprintf(1,'\nThe Mean mismatch is %f percent.\n', mean_error*100);
        for j=1:size(pop_error,2)
        fprintf(1,'Predicted value of species A%dB%d +A%dB%d exceeds
measurement by %f percent and mean square error of %f percent.\n ',j-1,N-
j+1,N-j+1,j-1,pop_error(1,j)*100,pop_error(2,j)*100);
        end
```

d.3. Single Aggregate—Refine_fit

```
function [phi_new, error] = refine_fit(XA, phi, peak_assignment,
Expt_Populations, Expt_Errors)
```

If no Expt_Errors are entered, all points are weighted equally.

```
if (nargin<5)
    Expt_weights = ones(size(Expt_Populations));
else
            Expt_weights = 1./( Expt_Errors + mean(mean(Expt_Errors)));
 end
```

Set-up the parameters.

```
N = length(phi)-1;
param = [ 2:(N+1)];
```

The initial step size for improving the model is 10%.

```
step_size = 0.1*phi(param),
```

Initialize the search.

```
N_no_progress = 0;
N_max_trials = 30;
```

Compute and report the initial quality of the fit.

```
[error_best, temp] = Error_of_Model(XA,phi, peak_assignment,
Expt_Populations) ;
fprintf(1,'\n Initial Error of Fit = %f percent.\n', error_best * 100);
```

Iteratively try to improve fit by changing each parameter in turn.

```
while (N_no_progress < N_max_trials)
    flag = 0;
   for k=1:length(param)
```

Step to the right and to the left.

```
          phi_testr = phi;
          phi_testr(param(k))=abs(phi(param(k)) + step_size(k));
```

```
        [error_testr, temp] = Error_of_Model(XA,phi_testr, peak_assignment,
Expt_Populations, Expt_weights);
        phi_testl = phi;
        phi_testl(param(k))=abs(phi(param(k)) - step_size(k));
        [error_testl, temp] = Error_of_Model(XA,phi_testl,
peak_assignment,Expt_Populations, Expt_weights);
```

Decide if a step should be taken. If the step is positive, continue that direction. If the negative step is better, continue that way. If no improvement occurs, flag that step and begin reducing the step size. N_no_progress is the number of steps since the error last improved. N_max_trials is the number of iterations to perform without the error getting better before ending the script. It is set to 30.

```
      if (error_testr<error_best)
          error_best=error_testr; phi=phi_testr; step_size(k) = step_size(k) * 1.5;
          N_no_progress=0;
    elseif     (error_testl <error_best)
          error_best=error_testl; phi=phi_testl; step_size(k) = step_size(k) * 1.5;
          N_no_progress=0;
    else
       flag = flag + 1;
    end
  end
  if (flag>2)
    step_size = step_size * (0.75 + 0.25*rand);
    N_no_progress=N_no_progress+1;
  end
```

After adjusting each element of rel_weight, report the new fit.

```
  fprintf(1,'\nError - %f , Last Good Step - %d , Mean Step Size - %f \n
',error_best, N_no_progress, 100*mean(step_size./phi(param)));
  fprintf(1,' Phi - %f',phi);
  end
  error=error_best;
  phi_new = phi;
```

d.4. Single Aggregate—Refine_fit_s

```
function [phi_new, error] = refine_fit_s(XA,phi, peak_assignment,
Expt_Populations, Expt_Errors)
```

If no Expt_Errors are entered, all points are weighted equally.

```
if (nargin<5)
   Expt_weights = ones(size(Expt_Populations));
```

```
else
    Expt_weights = 1./( Expt_Errors + mean(mean(Expt_Errors)));
 end
```

Because some phi's are equivalent in a symmetric plot and the model is unchanged by multiplication, only some of the parameters need to be adjusted. Here the program is told which parameters to adjust.

```
N = length(phi)-1;
param = [ 1:ceil(N/2)];
```

The initial step size for improving the model is 10%.

```
step_size = 0.1*phi(param),
```

Initialize the search.

```
N_no_progress = 0;
N_max_trials = 30;
```

Compute and report the initial quality of the fit.

```
[error_best, temp] = Error_of_Model(XA,phi, peak_assignment,
Expt_Populations, Expt_weights) ;
fprintf(1,'\n Initial Error of Fit = %f percent.\n', error_best * 100);
```

Iteratively try to improve fit by changing each parameter in turn.

```
while (N_no_progress < N_max_trials)
   flag = 0;
   for k=1:length(param) % Try tweaking each parameter in turn.
```

Step to the right and to the left.

```
          phi_testr = phi;
          phi_testr(param(k))=abs(phi(param(k)) + step_size(k));
          phi_testr(N+2-param(k))=phi_testr(param(k));
          [error_testr, temp] = Error_of_Model(XA,phi_testr, peak_assignment,
Expt_Populations, Expt_weights);
          phi_testl = phi;
          phi_testl(param(k))=abs(phi(param(k)) - step_size(k));
          phi_testl(N+2-param(k))=phi_testl(param(k));
          [error_testl, temp] = Error_of_Model(XA,phi_testl,
peak_assignment,Expt_Populations, Expt_weights);
```

Decide if a step should be taken. If the step is positive, continue that direction. If the negative step is better, continue that way. If no improvement occurs, flag that step and begin reducing the step size.

```
      if (error_testr<error_best)
           error_best=error_testr; phi=phi_testr; step_size(k) = step_size(k) * 1.5;
           N_no_progress=0;
     elseif     (error_testl <error_best)
           error_best=error_testl; phi=phi_testl; step_size(k) = step_size(k) * 1.5;
           N_no_progress=0;
     else
        flag = flag + 1; % Failure.  Add it to the list.
      end
  end
  if (flag>=length(param)) % Failed to improve by stepping in any direction
     step_size = step_size * (0.75 + 0.25*rand);  % Reduce step size
     N_no_progress=N_no_progress+1;
  end
```

After adjusting each element of rel_weight, report the new fit.

```
   fprintf(1,'\nError - %f , Last Good Step - %d , Mean Step Size - %f \n
',error_best, N_no_progress, 100*mean(step_size./phi(param)));
   fprintf(1,' Phi - %f',phi);
   end
   error=error_best;
   phi_new = phi;
```

d.5. Multimers

Two subunits A and B are mixed to form an ensemble of N-mers in solution. A given N-mer is described by it's composition, A_nB_m where n+m = N and N = length(phi)-1. Thus, N+1 different species are possible in solution. The concentration of each species is determined by XA(j) and phi. This ensemble is parameterized with the set,

Concentration(j,n+1) = phi(n+1) * Mn(n+1) * C(A_0:B_N)^(1 - n/N) * C(A_N:B_0)^(n/N)

where C(j,n+1) is the concentration of species A_nB_m for mole fraction XA(j) and Mn is the multiplicity of state j (N!/(j!*(N-j)!). The mole fraction is of course,

XA(j) = Sum(n*Concentration(j,n+1)) / Sum(Concentration(j,n+1))

The concentrations are normalized to be mole fractions of aggregates such that

(Sum over n of (Concentration(j,n+1)) = 1

In this parameterization, C(j,1) and C(j,N+1) are adjusted until the closest XA(j) to the experimental result is obtained because XA(j) is the quantity the experimenter adjusts. These adjustments in turn determine all the C(j,n+1)'s. Multimers can be fed multiple values XA(j) and will determine the aggregate concentrations for each value. The returned Concentration(j,n+1) is the concentration of species $[A_nB_m]$ for the j-th value of XA(j).

The returned XA and Concentrations(n+1) are a function of phi and a parameter, A.

C(n+1) = N * Constant * M(n) * phi(n+1) * A^n * (1-A)^(N-n)

where N, Constant, Mn, and A are numbers so that,

sum(C(n+1)) = 1

XA = sum(C(n+1)*n)/N; 0 <= A <= 1.

The bisection method is used to get the value of A for which XA is the one supplied to bisect. The concentration of each species is then returned for that value of a.

```
function Concentration = multimer(XA, phi)
```

For each XA(j), Concentration(j,:) is determined.

```
for j=1:length(XA)
```

Cconc is returned by the function bisect.

```
Concentration(j,:) = bisect(XA(j),phi);
end
function Conc = bisect(XA, phi)
```

Initialize values of A. XA may differ by 1e-6 at the end of bisection and while not close enough, continue to bisect by the difference of Amin and Amax.

```
   tolerance = 1e-6;
   Amax = 1; Amin = 0;
   [Xmin, Conc]=Cparametric(Amin, phi);
   [Xmax, Conc]=Cparametric(Amax, phi);
        while ((Xmax-XA)>tolerance)
        Atest = (Amin+Amax)/2;
     [Xtest, Conc]=Cparametric(Atest,phi);
     if (Xtest>XA)
       Amax = Atest; Xmax=Xtest;
```

```
    else
       Amin = Atest; Xmin=Xtest;
    end
  end
  function [XA, Concs] = Cparametric( A, phi)
```

Parametric form where mole fraction and Concentrations are determined for A, phi and N.

```
N = length(phi)-1;
B = 1-A;
Concs(1)= B^N*phi(1);
Concs(N+1)=A^N*phi(N+1);
Mn=1;
```

Calculate each unscaled concentration.

```
for k=2:N
   idx=k-1;
   Mn = Mn * (N+1-idx)/idx;
   Concs(k) = phi(k)*A^idx*B^(N-idx)*Mn;
end
```

Normalize the concentration and calculate the mole fraction.

```
Concs = Concs / sum(Concs);
XA = sum(Concs.*[0:N]/N);
```

d.6. Single Aggregate—Populations

```
function result = Populations(Concentrations, peak_assignment)
```

For each type of aggregate, add to correct NMR resonance.

```
   result = zeros(size(Concentrations,1),max(peak_assignment));
   N = size(Concentrations,2);
   for j=1:N
      idx = peak_assignment(j);
      result(:,idx) = result(:,idx) + Concentrations(:,j);
   end
```

d.7. Single Aggregate—Error of Model

Function: [mean_error, pop_error] = Error_of_Model(XA, phi, peak_assignment, Expt_Populations, Expt_Errors)

```
function [mean_error, pop_error] = Error_of_Model(XA, phi, peak_assignment, Expt_Populations, Expt_Errors)
```

If no Expt_weights are given, weight all points equally.

```
   if (nargin<5)
          Expt_weights=ones(size(Expt_Populations));
else
          Expt_weights = 1./( Expt_Errors + mean(mean(Expt_Errors)));
   end
```

Compute values from the model.

```
   Concentrations = multimers(XA,phi);
   PP = Populations(Concentrations, peak_assignment);
```

Compute the mean error.

```
   diff = PP - Expt_Populations;
   mean_error = sqrt(sum(sum(diff.*diff.*Expt_weights)) /
sum(sum(Expt_weights)));
```

Compute the error for each population independently.

```
   pop_error = sum(diff.*Expt_weights,1) ./ sum(Expt_weights,1);
   pop_error(2,:) = sqrt(sum(diff.*diff.*Expt_weights,1) ./ sum(Expt_weights,1));
```

e. Monomer_Dimer Fitting Code

e.1. Monomer_Dimer -- Data_Monomer_Dimer

XA(j). The mole fractions are listed in terms of **A** such that the first mole fraction corresponds to the first row of Expt_Populations.

```
Xa = [ 0.0000  0.1000  1.0000 etc ];
```

Ctotal must have the same value through out and have the same length as Xa.

```
Ctotal = [ 0.100  0.100  0.100  etc ];
```

Expt_Populations are input as a column of data for each aggregate. If an aggregate has two resonances, they should be summed and entered as a single relative integration. Recall that each row must end with a semi-colon.

```
Expt_Populations = [1.000  0.0000   0.0000;
0.8170 0.1830 0.0000;
0.6274 0.3294 0.0432;
etc; ];
```

Expt_Errors are input as an array the same size as Expt_Populations. Each column and row should correspond to the row and column of Expt_Populations. Expt_Errors is optional and should be left as "Expt_Errors = [;];" if none are provided.

```
Expt_Errors = [ ;];
```

peak_assignment_monomer and **peak_assignment_dimer**. The program expects the peaks to be listed in the order of increasing **A** subunit as shown below:

peak_assignment_monomer = [$[A_0B_1]$ $[A_1B_0]$]
peak_assignment_dimer = [$[A_0B_2]$ $[A_1B_1]$ $[A_2B_0]$]

The column number of the corresponding relative integration is input in the order designated above. If an aggregate was not observed, it should be assigned to column 1. For example, if the populations are listed in the order: $[A_1B_1]$ $[A_0B_1]$ $[A_2B_0]$, then peak_assignment_monomer = [2 1] and peak_assignment_dimer = [1 1 3]. If a relative integration is a sum of two unresolved resonances, enter the same column number for each aggregate contributing to the resonance.

```
peak_assignment_monomer = [ 1 3];
peak_assignment_dimer = [1  2  3];
```

phi_monomer and **phi_dimer** are listed in the order of increasing **A** subunit as it is in peak_assignment. The phi's provided here will be the start value for the fit. If an aggregate was not observed, it should be assigned a phi value of 0. The fit may require that a missing aggregate be assigned a phi of 0.00001 rather than 0 to avoid dividing by zero problem in the fit.

```
phi_monomer = [ 0 1];
phi_dimer = [1 1 0.00001];
```

phi_constant indicates the phi's which should not be changed based on all five phi values in the order $[A_0B_1]$, $[A_1B_0]$, $[A_0B_2]$, $[A_1B_1]$, and $[A_2B_0]$. Values of 0 disallow optimization of the corresponding phi. A value of 1 allows for optimization. For example, to fix the phi for $[A_0B_2]$, phi_constant = [1 1 0 1 1].

```
phi_constant = [1 1 1 1 1];
```

e.2. Monomer_Dimer—Try_fit

```
function try_fit(Xa, Ctotal, phi_monomer, peak_assignment_monomer,
phi_dimer, peak_assignment_dimer, Expt_Populations, Expt_Errors)
```

Code for weighting the Expt_Populations. If no Expt_Errors are entered, all points are weighted equally.

```
  if (nargin<8)
        Expt_weights=ones(size(Expt_Populations));
  else
        Expt_weights = 1./( Expt_Errors + mean(mean(Expt_Errors)));
  end
```

Code for plotting the experimental data.

```
    hold on ; cscheme='bgrmkcybgrmkcy'; axis([0 1 0 1]); xlabel('X_A');
ylabel('Relative Integration');
    for j=1:size(Expt_Populations,2)
        if (nargin<8)
           plot(Xa, Expt_Populations(:,j),sprintf('%so',cscheme(j)));
        else
           errorbar(Xa, Expt_Populations(:,j),
Expt_Errors(:,j),sprintf('%so',cscheme(j)));
        end
    end
```

Code for plotting the model with the provided phi's.

```
    Xac = [0:0.01:1];
    Ctotalac = Ctotal(1)*ones(size(Xac));
    [Conc_Monomers, Conc_Dimers] = multimers(Xac, Ctotalac, phi_monomer,
phi_dimer);
    TP=Populations(Conc_Monomers, peak_assignment_monomer,
Conc_Dimers, peak_assignment_dimer);
    for j=1:size(TP,2)
       plot(Xac,TP(:,j),sprintf('%c',cscheme(j)) );
    end
```

Code for computing and reporting the error.

```
  [mean_error, pop_error] = Error_of_Model(Xa, Ctotal, phi_monomer,
peak_assignment_monomer, phi_dimer, peak_assignment_dimer,
Expt_Populations, Expt_weights);
```

```
        fprintf(1,'\nThe Mean mismatch is %f percent.\n', mean_error*100);
        for j=1:size(pop_error,2)
        fprintf(1,'Predicted value of population %d exceeds measurement by %f
percent and mean square error of %f percent.\n
',j,pop_error(1,j)*100,pop_error(2,j)*100);
        end
```

e.3. Monomer_Dimer—Refine_fit

```
function [phi_monomer_new, phi_dimer_new, error] = refine_fit(Xa,
Ctotal,phi_monomer, peak_assignment_monomer, phi_dimer,
peak_assignment_dimer, Expt_Populations, phi_constant, Expt_Errors)
```

If no Expt_Errors are entered, all points are weighted equally.

```
if (nargin<9)
   Expt_weights = ones(size(Expt_Populations));
else
        Expt_weights = 1./( Expt_Errors + mean(mean(Expt_Errors)));
end
```

Set-up the parameters to merge the monomer and dimer inputs to a single input.

```
phimerge = [phi_monomer, phi_dimer];
idx_monomer = [1 2]; idx_dimer = [ 3 4 5];
param = [1:length(phimerge)];
```

The initial step size for improving the model is 10%.

```
step_size = 0.1*phi_constant.*phimerge(param);
```

Initialize the search. N_no_progress is the number of steps since the error last improved. N_max_trials is the number of iterations to perform without the error getting better before ending the script. It is set to 30.

```
N_no_progress = 0;
N_max_trials = 30;
```

Compute and report the initial quality of the fit.

```
[error_best, temp] = Error_of_Model(Xa,Ctotal,phimerge(idx_monomer),
peak_assignment_monomer, phimerge(idx_dimer), peak_assignment_dimer,
Expt_Populations, Expt_weights);
fprintf(1,'\n Initial Error of Fit = %f percent.\n', error_best * 100);
```

Iteratively try to improve fit by changing each parameter in turn.

```
while (N_no_progress < N_max_trials)
   flag = 0;
   for k=1:length(param)
```

Step to the right and to the left.

```
          phi_testr = phimerge;
          phi_testr(param(k))=abs(phimerge(param(k)) + step_size(k));
          [error_testr, temp] = Error_of_Model(Xa,Ctotal,phi_testr(idx_monomer),
peak_assignment_monomer, phi_testr(idx_dimer), peak_assignment_dimer,
Expt_Populations, Expt_weights);
          phi_testl = phimerge;
          phi_testl(param(k))=abs(phimerge(param(k)) - step_size(k));
          [error_testl, temp] = Error_of_Model(Xa,Ctotal,phi_testl(idx_monomer),
peak_assignment_monomer, phi_testl(idx_dimer), peak_assignment_dimer,
Expt_Populations, Expt_weights);
```

Decide if a step should be taken. If the step is positive, continue that direction. If the negative step is better, continue that way. If no improvement occurs, flag that step and begin reducing the step size.

```
       if (error_testr<error_best)
             error_best=error_testr; phimerge=phi_testr; step_size(k) = step_size(k) *
1.5;
             N_no_progress=0;
      elseif     (error_testl <error_best)
   error_best=error_testl; phimerge=phi_testl; step_size(k) = step_size(k) * 1.5;
             N_no_progress=0;
      else
         flag = flag + 1;
      end
   end
   if (flag>=length(param))
   step_size = step_size * (0.75 + 0.25*rand);
   N_no_progress=N_no_progress+1;
   end
```

After adjusting each element of rel_weight, report the new fit.

```
   fprintf(1,'\nError - %f , Last Good Step - %d , Mean Step Size - %f \n
',error_best, N_no_progress, 100*mean(step_size) );
   fprintf('\n Phi Dimer - ');   fprintf(1,'%f ', phimerge(idx_monomer));
   fprintf(1,'\n Phi Tetramer - '); fprintf(1,'%f ',phimerge(idx_dimer));
   end
   error=error_best;
   phi_monomer_new = phimerge(idx_monomer);
```

```
phi_dimer_new = phimerge(idx_dimer);
```

e.4. Monomer_Dimer—Mulitmers

Two subunits A and B are mixed to form an ensemble of monomers and dimers in solution. A given N-mer is described by it's composition, A_nB_m where n+m = N and N = 1 and 2. Thus, five different species are possible in solution. The concentration of each species is determined by Xa(j), Chi, and phi_monomer or phi_dimer. The outputs are Concentration_monomer(j,n+1) and Concentration_dimer(j, 1).

In this parameterization, the "relative chemical potential", a, is adjusted until the closest XA(j) to the experimental result is obtained. These adjustments in turn determine Chi, the monomer/dimer scaler, and all the aggregate concentrations. Multimers can be fed multiple values XA(j) and will determine the aggregate concentrations for each value. The returned Concentration(j,n+1) is the concentration of species $[A_nB_m]$ for the j-th value of XA(j).

The returned XA and Concentrations(n+1) are a function of phi and a such that,

$[B_1]$ = Chi/Ctotal * phi_monomer(1) * b
$[A_1]$ = Chi/Ctotal * phi_monomer(2) * a

$[B_2]$ = 2 * Chi^2/Ctotal * phi_dimer(1) * b * b
[AB] = 2 * 2 * Chi^2/Ctotal * phi_dimer(2) * a * b
$[A_2]$ = 2* Chi^2/Ctotal * phi_dimer(3) * a * a

```
Xa = sum( Conc_monomer.*[0 1]) + sum(Conc_dimer.* [0 0.5 1]);
```

The bisection method is used to get the value of a for which Xa is the one supplied to bisect. The concentration of each species is then returned for that value of a.

```
function [Concentration_monomer, Concentration_dimer] = multimers(Xa,
Ctotal, phi_monomer, phi_dimer)
```

For each Xa(j), Concentration_monomer(j,:) and Concentration_dimer(j,:) are determined.

```
for j=1:length(Xa)
```

Conc_monomer and Conc_dimer are the total concentrations of monomers and dimers and are returned by the function bisect.

```
  [Concentration_monomer(j,:), Concentration_dimer(j,:)] = bisect(Xa(j),
Ctotal(j),phi_monomer, phi_dimer);
end
```

```
function [Conc_monomer, Conc_dimer ] = bisect(Xa, Ctotal,phi_monomer,
phi_dimer);
```

Initialize values of a. Xa may differ by 1e-6 at the end of bisection and while not close enough, continue to bisect by the difference of amin and amax.

```
    tolerance = 1e-6;
    amax = 1; amin = 0;
    [Xmin, Conc_monomer, Conc_dimer]=Cparametric(amin, phi_monomer,
phi_dimer, Ctotal);
    [Xmax, Conc_monomer, Conc_dimer]=Cparametric(amax, phi_monomer,
phi_dimer, Ctotal);
    while ((Xmax-Xa)>tolerance)
        atest = (amin+amax)/2;
        [Xtest, Conc_monomer, Conc_dimer]=Cparametric(atest, phi_monomer,
phi_dimer, Ctotal);
        if (Xtest>Xa)
            amax = atest; Xmax=Xtest;
        else
            amin = atest; Xmin=Xtest;
        end
    end
function [Xa, Conc_monomer, Conc_dimer]=Cparametric(a, phi_monomer,
phi_dimer, Ctotal)
```

Parametric form where Xa, Mtotal, Dtotal, and Chi are determined for a, phi and N. Note that Chi is actually Chi = Mtotal / (4*Dtotal) * (sqrt(1 + 8 * Dtotal*Ctotal/(Mtotal*Mtotal)) - 1), but the equation has been modified to address of case of Mtotal = 0 or Dtotal = 0. The script will still have trouble if both Mtotal and Dtotal =0 but that indicates a mistake in the choice of phi_monomer and phi_dimer has been made.

```
b=1-a;
Mtotal = phi_monomer(1)*b + phi_monomer(2) * a;
Dtotal = phi_dimer(1)*b*b + 2*phi_dimer(2)*b*a + phi_dimer(3)*a*a;
Chi =( sqrt( Mtotal*Mtotal + 8 * Dtotal*Ctotal ) - Mtotal +
(Dtotal==0)*(2*Mtotal*Ctotal) ) / (4* Dtotal + (Dtotal==0));
```

Normalize the concentrations of monomers and dimers and calculate the mole fraction.

```
Conc_monomer = Chi/Ctotal * [ phi_monomer(1)*b, phi_monomer(2) * a] ;
Conc_dimer = 2 * Chi^2/Ctotal * [ phi_dimer(1)*b*b , 2*phi_dimer(2)*b*a ,
phi_dimer(3)*a*a ];
Xa = sum( Conc_monomer.*[0 1]) + sum(Conc_dimer.* [0 0.5 1]);
```

e.5. Monomer_Dimer—Populations

```
function result = Populations(Concentration_monomer,
peak_assignment_monomer, Concentration_dimer, peak_assignment_dimer)
```

For each type of aggregate, add to correct NMR resonance.

```
   result = zeros(size(Concentration_monomer,1), max(
max(peak_assignment_monomer) , max(peak_assignment_dimer) ) );
   N = size(Concentration_monomer,2);
   for j=1:N
      idx = peak_assignment_monomer(j);
      result(:,idx) = result(:,idx) + Concentration_monomer(:,j);
   end
   N = size(Concentration_dimer,2);
   for j=1:N
      idx = peak_assignment_dimer(j);
      result(:,idx) = result(:,idx) + Concentration_dimer(:,j);
   end
```

e.6. Monomer_Dimer—Error of Model

```
function [mean_error, pop_error] = Error_of_Model(Xa, Ctotal, phi_monomer,
peak_assignment_monomer, phi_dimer, peak_assignment_dimer,
Expt_Populations, Expt_Errors)
```

If no Expt_weights are given, weight all points equally.

```
   if (nargin<8)
         Expt_weights=ones(size(Expt_Populations));
   else
         Expt_weights = 1./( Expt_Errors + mean(mean(Expt_Errors)));
   end
```

Compute values from the model.

```
   [Conc_Monomers, Conc_Dimers] = multimers(Xa, Ctotal, phi_monomer,
phi_dimer);
Model_Populations = Populations(Conc_Monomers,
peak_assignment_monomer, Conc_Dimers, peak_assignment_dimer);
```

Compute the mean error.

```
   diff = Model_Populations - Expt_Populations;
```

```
   mean_error = sqrt(sum(sum(diff.*diff.*Expt_weights)) /
sum(sum(Expt_weights)));
```

Compute the error for each population independently.

```
   pop_error = sum(diff.*Expt_weights,1) ./ sum(Expt_weights,1);
   pop_error(2,:) = sqrt(sum(diff.*diff.*Expt_weights,1) ./ sum(Expt_weights,1));
```

f. Dimer_Tetramer Fitting Code

f.1. Dimer_Tetramer -- Data_Dimer_Tetramer

XA(j). The mole fractions are listed in terms of **A** such that the first mole fraction corresponds to the first row of Expt_Populations.

```
Xa = [ 0.0000  0.1000  1.0000 etc ];
```

Ctotal must have the same value through out and have the same length as Xa.

```
Ctotal = [ 0.100  0.100  0.100  etc ];
```

Expt_Populations are input as a column of data for each aggregate. If an aggregate has two resonances, they should be summed and entered as a single relative integration. Recall that each row must end with a semi-colon.

```
Expt_Populations = [1.000  0.0000   0.0000;
0.8170 0.1830 0.0000;
0.6274 0.3294 0.0432;
etc; ];
```

Expt_Errors are input as an array the same size as Expt_Populations. Each column and row should correspond to the row and column of Expt_Populations. Expt_Errors is optional and should be left as "Expt_Errors = [;];" if none are provided.

```
Expt_Errors = [ ;];
```

peak_assignment_dimer and **peak_assignment_tetramer**. The program expects the peaks to be listed in the order of increasing **A** subunit as shown below:

peak_assignment_dimer = [$[A_0B_2]$ $[A_1B_1]$ $[A_2B_0]$]
peak_assignment_tetramer = [$[A_0B_4]$ $[A_1B_3]$ $[A_2B_2]$ $[A_3B_1]$ $[A_4B_0]$]

The column number of the corresponding relative integration is input in the order designated above. If an aggregate was not observed, it should be assigned to column 1. For example, if the populations are listed in the order: $[A_1B_1]$ $[A_0B_4]$ $[A_2B_0]$, then peak_assignment_dimer = [1 1 3] and peak_assignment_tetramer = [2 1 1 1 1]. If a relative integration is a sum of two unresolved resonances, enter the same column number for each aggregate contributing to the resonance.

```
peak_assignment_dimer = [ 1 2  3 ];
peak_assignment_tetramer = [1  2  3  4  5 ];
```

phi_dimer and **phi_tetramer** are listed in the order of increasing **A** subunit as it is in peak_assignment. The phi's provided here will be the start value for the fit. If an aggregate was not observed, it should be assigned a phi value of 0. The fit may require that a missing aggregate be assigned a phi of 0.00001 rather than 0 to avoid dividing by zero problem in the fit.

```
phi_dimer = [ 0.00001 1  1 ];
phi_tetramer = [1 1 0.00001  1  1  ];
```

phi_constant indicates the phi's which should not be changed based on all five phi values in the order $[A_0B_2]$, $[A_1B_1]$, $[A_2B_0]$, $[A_0B_4]$, $[A_1B_3]$, $[A_2B_2]$, $[A_3B_1]$, $[A_4B_0]$. Values of 0 disallow optimization of the corresponding phi. A value of 1 allows for optimization. For example, to fix the phi for $[A_1B_3]$, phi_constant = [1 1 1 1 0 1 1 1].

```
phi_constant = [1 1 1 1 1 1 1 1];
```

f.2. Dimer_Tetramer—Try_fit

```
function try_fit(Xa, Ctotal, phi_dimer, peak_assignment_dimer, phi_tetramer,
peak_assignment_tetramer, Expt_Populations, Expt_Errors)
```

Code for weighting the Expt_Populations. If no Expt_Errors are entered, all points are weighted equally.

```
if (nargin<8)
        Expt_weights=ones(size(Expt_Populations));
else
        Expt_weights = 1./( Expt_Errors + mean(mean(Expt_Errors)));
end
```

Code for plotting the experimental data.

```
    hold on ; cscheme='bgrmkcybgrmkcy'; axis([0 1 0 1]); xlabel('X_R');
ylabel('Mole Fractions');
    for j=1:size(Expt_Populations,2)
        if (nargin<8)
           plot(Xa, Expt_Populations(:,j),sprintf('%so',cscheme(j)));
        else
           errorbar(Xa, Expt_Populations(:,j),
Expt_Errors(:,j),sprintf('%so',cscheme(j)));
        end
    end
```

Code for plotting the model with the provided phi's.

```
    Xac = [0:0.01:1];
    Ctotalac = Ctotal(1)*ones(size(Xac));
    [Conc_Dimers, Conc_Tetramers] = multimers(Xac, Ctotalac, phi_dimer,
phi_tetramer);
    TP=Populations(Conc_Dimers, peak_assignment_dimer, Conc_Tetramers,
peak_assignment_tetramer);
    for j=1:size(TP,2)
       plot(Xac,TP(:,j),sprintf('%c',cscheme(j)) );
    end
```

Code for computing and reporting the error.

```
   [mean_error, pop_error] = Error_of_Model(Xa, Ctotal, phi_dimer,
peak_assignment_dimer, phi_tetramer, peak_assignment_tetramer,
Expt_Populations, Expt_weights);
        fprintf(1,'\nThe Mean mismatch is %f percent.\n', mean_error*100);
        for j=1:size(pop_error,2)
        fprintf(1,'Predicted value of population %d exceeds measurement by %f
percent and mean square error of %f percent.\n
',j,pop_error(1,j)*100,pop_error(2,j)*100);
        end
```

f.3. Dimer_Tetramer—Refine_fit

```
function [phi_dimer_new, phi_tetramer_new, error] = refine_fit(Xa,
Ctotal,phi_dimer, peak_assignment_dimer, phi_tetramer,
peak_assignment_tetramer, Expt_Populations, phi_constant, Expt_Errors)
```

If no Expt_Errors are entered, all points are weighted equally.

```
if (nargin<9)
   Expt_weights = ones(size(Expt_Populations));
else
```

```
            Expt_weights = 1./( Expt_Errors + mean(mean(Expt_Errors)));
end
```

Set-up the parameters to merge the dimer and tetramer inputs to a single input.

```
phimerge = [phi_dimer, phi_tetramer];
idx_dimer = [1 2 3]; idx_tetramer = [ 4 5 6 7 8];
param = [1:length(phimerge)];
```

The initial step size for improving the model is 10%.

```
step_size = 0.1*phi_constant.*phimerge(param);
```

Initialize the search.

```
N_no_progress = 0;
N_max_trials = 30;
```

Compute and report the initial quality of the fit.

```
[error_best, temp] = Error_of_Model(Xa,Ctotal,phimerge(idx_dimer),
peak_assignment_dimer, phimerge(idx_tetramer), peak_assignment_tetramer,
Expt_Populations, Expt_weights);
fprintf(1,'\n Initial Error of Fit = %f percent.\n', error_best * 100);
```

Iteratively try to improve fit by changing each parameter in turn.

```
while (N_no_progress < N_max_trials)
   flag = 0;
   for k=1:length(param)
```

Step to the right and to the left.

```
        phi_testr = phimerge;
        phi_testr(param(k))=abs(phimerge(param(k)) + step_size(k));
        [error_testr, temp] = Error_of_Model(Xa,Ctotal,phi_testr(idx_dimer),
peak_assignment_dimer, phi_testr(idx_tetramer), peak_assignment_tetramer,
Expt_Populations, Expt_weights);
        phi_testl = phimerge;
        phi_testl(param(k))=abs(phimerge(param(k)) - step_size(k));
        [error_testl, temp] = Error_of_Model(Xa,Ctotal,phi_testl(idx_dimer),
peak_assignment_dimer, phi_testl(idx_tetramer), peak_assignment_tetramer,
Expt_Populations, Expt_weights);
```

Decide if a step should be taken. If the step is positive, continue that direction. If the negative step is better, continue that way. If no improvement occurs, flag that step and begin reducing the step size.

```
      if (error_testr<error_best)
           error_best=error_testr; phimerge=phi_testr; step_size(k) = step_size(k) *
1.5;
           N_no_progress=0;
      elseif     (error_testl <error_best)
           error_best=error_testl; phimerge=phi_testl; step_size(k) = step_size(k) *
1.5;
           N_no_progress=0;
      else
         flag = flag + 1;
      end
   end
   if (flag>=length(param))
      step_size = step_size * (0.75 + 0.25*rand);
   N_no_progress=N_no_progress+1;
   end
```

After adjusting each element of rel_weight, report the new fit.

```
   fprintf(1,'\nError - %f , Last Good Step - %d , Mean Step Size - %f \n
',error_best, N_no_progress, 100*mean(step_size) );
   fprintf('\n Phi Dimer - ');   fprintf(1,'%f ', phimerge(idx_dimer));
   fprintf(1,'\n Phi Tetramer - '); fprintf(1,'%f ',phimerge(idx_tetramer));
   end
   error=error_best;
   phi_dimer_new = phimerge(idx_dimer);
   phi_tetramer_new = phimerge(idx_tetramer);
```

f.4. Dimer_Tetramer—Multimers

Two subunits A and B are mixed to form an ensemble of monomers and dimers in solution. A given N-mer is described by it's composition, A_nB_m where n+m = N and N = 1 and 2. Thus, five different species are possible in solution. The concentration of each species is determined by Xa(j), Chi, and phi_dimer or phi_tetramer. The outputs are Concentration_dimer(j,n+1) and Concentration_tetramer(j, 1).

In this parameterization, the "relative chemical potential", a, is adjusted until the closest XA(j) to the experimental result is obtained. These adjustments in turn determine Chi_squared, the dimer/tetramer scaler, and all the aggregate concentrations. Multimers can be fed multiple values Xa(j) and will determine the aggregate concentrations for each value. The returned Concentration(j,n+1) is the concentration of species $[A_nB_m]$ for the j-th value of Xa(j).

The returned Xa and Concentrations(n+1) are a function of phi and a such that,

$[B_2]$ = 2 * Chi_squared/Ctotal * phi_dimer(1) * b * b

[AB] = 2 * 2 * Chi_squared /Ctotal * phi_dimer(2) * a * b

$[A_2]$ = 2* Chi_squared /Ctotal * phi_dimer(3) * a * a

$[B_4]$ = 4 * Chi_squared^2/Ctotal * phi_tetramer(1) * b^4
$[A_1B_3]$ = 4 * 4 * Chi_squared^2 /Ctotal * phi_tetramer(2) * a * b^3
$[A_2B_2]$ = 4 * 6 * Chi_squared^2 /Ctotal * phi_tetramer(3) * a^2 * b^2
$[A_1B_3]$ = 4 * 4 * Chi_squared^2 /Ctotal * phi_tetramer(4) * a^3 * b
$[A_4]$ = 4* Chi_squared^2 /Ctotal * phi_tetramer(5) * a^4

Xa = sum(Conc_dimer.* [0 0.5 1]) + sum(Conc_tetramer.*[0 0.25 0.5 0.75 1])

The bisection method is used to get the value of a for which Xa is the one supplied to bisect. The concentration of each species is then returned for that value of a.

```
function [Concentration_dimer, Concentration_tetramer] = multimers(Xa, Ctotal,
phi_dimer, phi_tetramer)
```

For each Xa(j), Concentration_dimer(j,:) and Concentration_tetramer(j,:) are determined.

```
for j=1:length(Xa)
```

Conc_dimer and Conc_tetramer are the total concentrations of dimers and tetramers and are returned by the function bisect.

```
  [Concentration_dimer(j,:), Concentration_tetramer(j,:)] = bisect(Xa(j),
Ctotal(j),phi_dimer, phi_tetramer);
end
function [Conc_dimer, Conc_tetramer ] = bisect(Xa, Ctotal,phi_dimer,
phi_tetramer);
```

Initialize values of a. Xa may differ by 1e-6 at the end of bisection and while not close enough, continue to bisect by the difference of amin and amax.

```
   tolerance = 1e-6; % Amount Xr may differ by an end of bisection..
   amax = 1; amin = 0;
   [Xmin, Conc_dimer, Conc_tetramer]=Cparametric(amin, phi_dimer,
phi_tetramer, Ctotal);
   [Xmax, Conc_dimer, Conc_tetramer]=Cparametric(amax, phi_dimer,
phi_tetramer, Ctotal);
   while ((Xmax-Xa)>tolerance) % While not close enough, continue to bisect
difference of rmin and rmax.
      atest = (amin+amax)/2;
      [Xtest, Conc_dimer, Conc_tetramer]=Cparametric(atest, phi_dimer,
phi_tetramer, Ctotal);
      if (Xtest>Xa)
        amax = atest; Xmax=Xtest;
```

```
      else
         amin = atest; Xmin=Xtest;
      end
   end
function [Xa, Conc_dimer, Conc_tetramer]=Cparametric(a, phi_dimer,
phi_tetramer, Ctotal)
```

Parametric form where Xa, Dtotal, Ttotal, and Chi_squared are determined for a, phi and N. Note that Chi is actually Chi_squared = Dtotal / (4*Ttotal) * (sqrt(1 + 4 * Ttotal*Ctotal/(Dtotal*Dtotal)) - 1), but the equation has been modified to address of case of Dtotal = 0 or Ttotal = 0. The script will still have trouble if both Dtotal and Ttotal =0 but that indicates a mistake in the choice of phi_dimer and phi_tetramer has been made.

```
b=1-a;
Dtotal = phi_dimer(1)*b*b + 2 * phi_dimer(2) * a * b + phi_dimer(3) * a*a;
Ttotal = phi_tetramer(1)*b^4 + 4*phi_tetramer(2)*b^3*a +
6*phi_tetramer(3)*b*b*a*a + 4*phi_tetramer(4)*b*a^3 + phi_tetramer(5)*a^4;
Chi_squared = ( sqrt( Dtotal*Dtotal + 4 * Ttotal*Ctotal ) - Dtotal +
(Ttotal==0)*(2*Dtotal*Ctotal) ) / (4* Ttotal + (Ttotal==0));
```

Normalize the concentrations of monomers and dimers and calculate the mole fraction.

```
Conc_dimer = 2 * Chi_squared/Ctotal * [ phi_dimer(1)*b*b , 2*phi_dimer(2)*b*a
, phi_dimer(3)*a*a] ;
Conc_tetramer = 4 * Chi_squared^2/Ctotal * [ phi_tetramer(1)*b*b*b*b,
4*phi_tetramer(2)*b*b*b*a, 6*phi_tetramer(3)*b*b*a*a, 4*phi_tetramer(4)*b*a*a*a ,
phi_tetramer(5)*a*a*a*a ];
Xa = sum( Conc_dimer .*[0 0.5 1]) + sum(Conc_tetramer.* [0 0.25 0.5 0.75 1]);
```

f.5. Dimer_Tetramer—Populations

```
function result = Populations(Concentration_dimer, peak_assignment_dimer,
Concentration_tetramer, peak_assignment_tetramer)
```

For each type of aggregate, add to correct NMR resonance.

```
   result = zeros(size(Concentration_dimer,1), max(
max(peak_assignment_dimer) , max(peak_assignment_tetramer) ) );
   N = size(Concentration_dimer,2);
   for j=1:N % Go through each type of aggregate and add to correct NMR peak.
      idx = peak_assignment_dimer(j);
      result(:,idx) = result(:,idx) + Concentration_dimer(:,j);
   end
   N = size(Concentration_tetramer,2);
   for j=1:N % Go through each type of aggregate and add to correct NMR peak.
```

```
    idx = peak_assignment_tetramer(j);
    result(:,idx) = result(:,idx) + Concentration_tetramer(:,j);
  end
```

f.6. Dimer_Tetramer—Error of Model

```
function [mean_error, pop_error] = Error_of_Model(Xa, Ctotal, phi_dimer,
peak_assignment_dimer, phi_tetramer, peak_assignment_tetramer,
Expt_Populations, Expt_Errors)
```

If no Expt_weights are given, weight all points equally.

```
  if (nargin<8)
      Expt_weights=ones(size(Expt_Populations));
else
      Expt_weights = 1./( Expt_Errors + mean(mean(Expt_Errors)));
  end
```

Compute values from the model.

```
  [Conc_Dimers, Conc_Tetramers] = multimers(Xa, Ctotal, phi_dimer,
phi_tetramer);
  Model_Populations = Populations(Conc_Dimers, peak_assignment_dimer,
Conc_Tetramers, peak_assignment_tetramer);
```

Compute the mean error.

```
  diff = Model_Populations - Expt_Populations;
  mean_error = sqrt(sum(sum(diff.*diff.*Expt_weights)) /
sum(sum(Expt_weights)));
```

Compute the error for each population independently.

```
  pop_error = sum(diff.*Expt_weights,1) ./ sum(Expt_weights,1);
  pop_error(2,:) = sqrt(sum(diff.*diff.*Expt_weights,1) ./ sum(Expt_weights,1));
```