Appendix I: Computer Code

Crossovers.cc

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
main()
{
const int libraryN=1000;    //number of libraries
const int crossN=5;     // number of crossovers
const int MINRES=24;  //minimum residue
const int MAXRES=290; //maximum residue
 int count;
int Crossovers[crossN];
 count=0;
int tog=0; //a toggle for loop generating random crossovers.
const int MINEND=15; //minimum distance to the ends of the protein
const int MINDIST=15; //minimum distance between crossovers.

 int temp; // temporary variable for sorting algorithm.
 ofstream crossoverfile ("5Crossovers.txt", ios::out);
 for (int i=0; i<libraryN; i++){
            tog=0;
            while (tog<=0){
              for (int j=0;j<crossN;j++){
                Crossovers[j]=(rand()%(MAXRES-MINRES))+MINRES;
              }
              for (int pass=0; pass<crossN-1; pass++){
                for (int k=0; k < crossN-1; k++){
                  if (Crossovers[k]> Crossovers[k+1]){
                        temp=Crossovers[k];
                        Crossovers[k]=Crossovers[k+1];
                        Crossovers[k+1]=temp;
                  }}}
                        //generates the list of random crossovers in array Crossovers
                        //the crossovers appear in chronological order
              count=count+1;
              int togl=1;
              if (Crossovers[0]<=MINRES+MINEND)
                togl=0;

              for (int k=0; k<crossN-1; k++){
                if (Crossovers[k+1]<((Crossovers[k])+MINDIST))
                  togl=0;
              }
              if (Crossovers[crossN-1]>MAXRES-MINEND)
                togl=0;
                        tog=togl;
            }
            for (int j=0; j<crossN; j++){
              crossoverfile<<Crossovers[j]<<"\t";
            }
            crossoverfile<<endl;
 }
 cout<<count<<endl;
return 0; }
```

Testlibraries.cc

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

int minimum(int array[], int number){
  int M=1000;
  for (int i=1; i<=number; i++){
    if (array[i]<=M){
      M=array[i];
    }}
  return M;
}

float average(int array[], int number){
  float sum=0;
  float A=0;
  if (number!=0){
  for (int i=0;i<number; i++){
      sum=sum+array[i];
    }
  A=sum/number;
  }
return A;
}

float standarddev (int array[], int number, float  average){
float deviation=0, sum=0;
float S=0;
 if (number !=0){
  for (int i=0; i<number; i++){
    deviation=array[i]-average;
    deviation=deviation*deviation;
    sum=sum+deviation;
  }
    S=sqrt(sum/number);
 }
return S;
}

main()
{
  //contants that we need througout the whole program
    const int MAXN = 1506;
  //This number is dependent upon the pdbout file.
  const float DC = 4.5;
  // temporary variables for input before putting into structure
  char TATOM[5], TAATYPE[5];
  int  TATOMN, TRESN;
  float TX_CORR, TY_CORR, TZ_CORR;
  // define loop variable
  int i;
 //define a structure PDBin that contains all the info we want
      static struct PDBin{
        int   ATOMN;   //the atom number, unique identifier
```

```
    int   RESN;    // the residue number which the atom belongs to
    float X_CORR;       // x coordinate
    float Y_CORR;       // y coordinate
    float Z_CORR;       // z coordinate
  }protein[MAXN];    //the name of the structure type protein with maxn atoms
  // designed to take the data
  ifstream pdb("atomfile.txt");
  //opens file test.txt and calls it pdb
  //if the file cannot be opened return an error message
  if (pdb.bad()){
   cerr<<"error couldn not open test.txt";
   exit (8);
  }
  for (i=1; i<MAXN; ++i){
   pdb >> TATOMN>>TRESN>>TX_CORR>>TY_CORR>>TZ_CORR;
   protein[i].ATOMN = TATOMN;
   protein[i].RESN =TRESN;
   protein[i].X_CORR = TX_CORR;
   protein[i].Y_CORR = TY_CORR;
   protein[i].Z_CORR = TZ_CORR;
  }
   //This ends the section of code that is necessary for intaking the PDB file
  //The data is stored in structure protein
 // this next section is for determining a contact matrix between all the residues in the //protein.  It does not
//include any identity characteristics
  //First we are going to create a matrix in which to put the data
  //Then we are going determine the distance measurements and fill in the matrix.
    //These variables are required for the contact matrix formation.
     int j, k;
  int MAXRES, MINRES;
  float XX, YY, ZZ, Distance;
  MAXRES=protein[MAXN-1].RESN;
  MINRES=protein[1].RESN;
  int CMatrix[MAXRES+1][MAXRES+1];
  for (i=1;i<=MAXRES;i++){
   for (j=1;j<=MAXRES;j++)
         CMatrix[i][j]=0;
  }
   //We also require constants MAXN and DC for this code segment.
  int count=0;
  for (i=26;i<=MAXRES;i++){
   //this loop goes through each residue in the structure protein
   for (j=1; j<=MAXN; j++){
         // This loop goes through all atoms in the structure protein
         if (protein[j].RESN==i){
           //If atom (j) is in residue (i) then:
           for (k=j; k<MAXN; k++){
             //loop through all atoms again
             if(protein[k].RESN!=i){
               //check and make sure that the residue is not the one being examined
               XX=(protein[k].X_CORR-protein[j].X_CORR)*(protein[k].X_CORR-protein[j].X_CORR);
               YY=(protein[k].Y_CORR-protein[j].Y_CORR)*(protein[k].Y_CORR-protein[j].Y_CORR);
               ZZ=(protein[k].Z_CORR-protein[j].Z_CORR)*(protein[k].Z_CORR-protein[j].Z_CORR);
               Distance=sqrt(XX+YY+ZZ);
               //Determine distance between atoms j and k
               if (Distance<DC){
```

```
                    CMatrix[i][protein[k].RESN]=1;
                    CMatrix[protein[k].RESN][i]=1;
                    //makes both halves of the contact matrix;
                }}}}}}
    count=0;
    for (i=1;i<MAXRES;i++){
      for (j=1;j<MAXRES;j++)
            if (CMatrix[i][j]==1)
              count=count+1;
    }
    // This is the end of the code for generating the contact matrix.
    // This contact matrix is used for the remainder of the program.
// the next job is to correct for identitiy in the contact matrix.
            ifstream align1and2("TEM1PSE4.txt");
            //opens alignment between parent 1 and parent 2
            ifstream align1and3("PSE4SED1.txt");
            //opens alignment between parent 1 and parent 3
            ifstream align2and3("TEM1SED1.txt");
            //opens alignment between parent 2 and parent 3
            ifstream align1and4("PSE4AST.txt");
            //opens alignment between parent 1 and parent 4
            ifstream align2and4("TEM1AST.txt");
            //opens alignment between parent 2 and parent 4
            ifstream align3and4("AST1SED1.txt");
            //opens alignment between parent 3 and 4
            if (align1and2.bad()){
      cerr<<"error couldn not open TEM1PSE4.txt";
      exit (8);
    }
            if (align1and3.bad()){
      cerr<<"error couldn not open TEM1SHV1.txt";
      exit (8);
    }
            if (align2and3.bad()){
      cerr<<"error couldn not open PSE4SHV1.txt";
      exit (8);
    }
            if (align1and4.bad()){
              cerr<<"error could not open 1to4";
              exit (8);
            }
            if (align2and4.bad()){
              cerr<<"error could not open 2to4";
              exit(8);
            }
            if (align3and4.bad()){
              cerr<<"error could not open 3to4";
              exit (8);
                }
            //error messages for bad file inputs
            int index, value, l;
            static int alignmaster[5][5][350];
            static int mastercontact[5][5][350][350];
            int parentN=4;
            for (i=1; i<=parentN; i++){
                    for (j=1; j<=parentN; j++){
```

```
                        for (k=1; k<350; k++){
                                alignmaster[i][j][k]=1;
                                for (l=1;l< 350; l++){
                                mastercontact[i][j][k][l]=0;
                                }}}}
            //initialize matrices with correct ones or zeros
            //declare input variables and the matrix into which they are put
            for (i=MINRES; i<=MAXRES;i++){
              align1and2>>index>>value;
              alignmaster[1][2][index]=value;
              alignmaster[2][1][index]=value;
              align2and3>>index>>value;
              alignmaster[2][3][index]=value;
              alignmaster[3][2][index]=value;
              align1and3>>index>>value;
              alignmaster[1][3][index]=value;
              alignmaster[3][1][index]=value;
              align1and4>>index>>value;
              alignmaster[1][4][index]=value;
              alignmaster[4][1][index]=value;
              align2and4>>index>>value;
              alignmaster[2][4][index]=value;
              alignmaster[4][2][index]=value;
              align3and4>>index>>value;
              alignmaster[3][4][index]=value;
              alignmaster[4][3][index]=value;
                        }
            for (i=1; i<=parentN; i++){
              for (j=i;j<=parentN; j++){
                for (k=MINRES; k<=MAXRES;k++){
                  for (l=MINRES; l<=MAXRES; l++){
            mastercontact[i][j][k][l]=alignmaster[i][j][k]*alignmaster[i][j][l]*CMatrix[k][l];
            mastercontact[j][i][k][l]=alignmaster[i][j][k]*alignmaster[i][j][l]*CMatrix[k][l];
                 }}}}
            //this finished the contact array entering and identity correction
            //next we need to open files for outputting data and inputing crossover points.
            ifstream cross("crossovers.txt");
            ofstream output("9Clibsnew2.txt", ios::out);
            //we also start doing each library one at a time now
            //first read in the crossovers, then generate the chimeras
            //and finally evaluate each chimera in the library
            //compile the data and write to the output file.
            int Maxdis=55;
            int counter=0;
            int libraryN=29; //number of libraries to analyze
            int crossN=9;    //number of crossovers
            int total=262144; // possible number of chimeras in each library
            static int Chimeras[262144][300];
            for (i=0;i<total; i++){
              for (k=1;k<=MAXRES; k++){
                Chimeras[i][k]=0;
              }}
// this part must be modified for a greater number of crossovers.
int C1, C2, C3, C4, C5, C6, C7, C8, C9;  // these are the fragments created by 8 //crossovers.
int Crossovers[crossN];
 int sum,c, count1;
```

```
 float  AvD, AvM, AvDG, AvMG, AvDD, AvMD;
 float StD, StM, StDG, StMG;
int mut[parentN+1];
for(j=1;j<=parentN; j++)
  mut[j]=0;
static int gdisruption[262144];
static int gmutation [262144];
static int disruption[262144];
static int mutation[262144];
double  fractionH, fractionG, probabilityH, probabilityG, MutationP;
for (j=0;j<total; j++){
  disruption[j]=-1;
  mutation[j]=0;
  gmutation[j]=0;
  gdisruption[j]=-1;
 }
  for (j=0; j<libraryN; j++){ //this goes through each library
   counter=0;
   c=0;
   count1=0;
   for (k=0; k<crossN; k++){
     cross>>Crossovers[k];
   }
  for (C1=1; C1<=parentN; C1++){
     for (C2=1;C2<=parentN;C2++){
       for (C3=1;C3<=parentN;C3++){
              for (C4=1;C4<=parentN;C4++){
                for (C5=1;C5<=parentN;C5++){
                  for(C6=1;C6<=parentN;C6++){
                    for(C7=1;C7<=parentN; C7++){
                          for(C8=1;C8<=parentN; C8++){
                            for(C9=1;C9<=parentN; C9++){
                              for (i=1; i<=Crossovers[0]; i++)
                                Chimeras [counter][i]=C1;
                              for (i=(Crossovers[0]+1); i<=Crossovers[1]; i++)
                          Chimeras [counter][i]=C2;
                    for (i=(Crossovers[1]+1); i<=Crossovers[2]; i++)
                          Chimeras [counter][i]=C3;
                    for (i=(Crossovers[2]+1); i<=Crossovers[3]; i++)
                          Chimeras [counter][i]=C4;
                    for (i=(Crossovers[3]+1); i<=Crossovers[4]; i++)
                          Chimeras [counter][i]=C5;
                    for (i=(Crossovers[4]+1); i<=Crossovers[5]; i++)
                          Chimeras [counter][i]=C6;
                    for(i=(Crossovers[5]+1); i<=Crossovers[6]; i++)
                          Chimeras[counter][i]=C7;
                    for(i=(Crossovers[6]+1); i<=Crossovers[7]; i++)
                          Chimeras[counter][i]=C8;
                    for(i=(Crossovers[7]+1); i<=Crossovers[8]; i++)
                          Chimeras[counter][i]=C9;
                    for(i=(Crossovers[8]+1); i<=MAXRES; i++)
                          Chimeras[counter][i]=C1;
                          counter=counter+1;
              } } } } }}}}}
  fractionH=0;
  fractionG=0;
```

```
  MutationP=0;
  for (i=0; i<total; i++){
   sum=0;
   for (k=MINRES; k<=MAXRES; k++){
    for (l=k+1; l<=MAXRES; l++){
          if (Chimeras[i][k]!=Chimeras[i][l])
           sum=sum+mastercontact[(Chimeras[i][k])][(Chimeras[i][l])][k][l];
    }
     for (l=1; l<=parentN; l++){
          if (Chimeras[i][k]!=l){
          if ((alignmaster[(Chimeras[i][k])][l][k])==1)
           mut[l]=mut[l]+1;
          }}}
   disruption[i]=sum;
   probabilityH=pow((1-((.1*sum)/322)),322);
   probabilityG=pow((1-((.04*sum)/322)),322);
   fractionG=fractionG +probabilityG;
   fractionH=fractionH+probabilityH;
   mutation[i]=minimum (mut, parentN);
   MutationP=MutationP+(probabilityG*mutation[i]);
   for (l=1; l<=parentN; l++){
    mut[l]=0;
   }
   if (disruption[i]<Maxdis){
    gdisruption[c]=disruption[i];
    gmutation[c]=mutation[i];
    c=c+1;
   } }
  AvD=average(disruption, total);
  AvM=average(mutation, total);
  AvDG=average(gdisruption, c);
  AvMG=average(gmutation, c);
  StM=standarddev(mutation, total, AvM);
  StD=standarddev(disruption, total, AvD);
  StDG=standarddev(gdisruption, c, AvDG);
  StMG=standarddev(gmutation, c, AvMG);
                output<<AvD<<"\t"<<AvM<<"\t"<<StD<<"\t"<<StM<<"\t"
                      <<c<<"\t"<<AvDG<<"\t" <<AvMG<<"\t"<<StDG<<"\t"
                      <<StMG<<"\t"<<fractionG<<"\t"<<fractionH<<"\t"<<MutationP<<"\t";
  for (i=0; i<crossN; i++){
   output<<Crossovers[i]<<"\t";
  }
  output<<endl;
 }
return 0;
}
```

Librarytest.cc

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
int minimum(int array[], int number){
  int M=1000;
  for (int i=1; i<=number; i++){
    if (array[i]<=M){
      M=array[i];
    } }
  return M;
}
float average(int array[], int number){
  float sum=0;
  float A=0;
  if (number!=0){
  for (int i=0;i<number; i++){
      sum=sum+array[i];
    }
  A=sum/number;
  }
return A;
}
float standarddev (int array[], int number, float  average){
float deviation=0, sum=0;
float S=0;
 if (number !=0){
  for (int i=0; i<number; i++){
    deviation=array[i]-average;
    deviation=deviation*deviation;
    sum=sum+deviation;
  }
    S=sqrt(sum/number);
 }
return S;
}

main()
{
  //contants that we need througout the whole program
  const int MAXN = 2057;
  const float DC = 4.5;
    // temporary variables for input before putting into structure
  char junk[4], junk2[2], junk5[2];
  float junk3, junk4;
  char TATOM[5], TAATYPE[5];
  int  TATOMN, TRESN;
  float TX_CORR, TY_CORR, TZ_CORR;
  // define loop variable
  int i;
  //define a structure PDBin that contains all the info we want
   static struct PDBin{
        int   ATOMN;   //the atom number, unique identifier
        int   RESN;    // the residue number which the atom belongs to
        float X_CORR;       // x coordinate
```

```
    float Y_CORR;       // y coordinate
    float Z_CORR;       // z coordinate
  }protein[MAXN];    //the name of the structure type protein with maxn atoms
  // designed to take the data
  ifstream pdb("Atomfile.txt");
  //opens file test.txt and calls it pdb
  //if the file cannot be opened return an error message
  if (pdb.bad()){
   cerr<<"error couldn not open test.txt";
   exit (8);
  }
  for (i=1; i<MAXN; ++i){
   pdb >>TATOMN>>TRESN>>TX_CORR>>TY_CORR>>TZ_CORR;
   protein[i].ATOMN = TATOMN;
   protein[i].RESN =TRESN;
   protein[i].X_CORR = TX_CORR;
   protein[i].Y_CORR = TY_CORR;
   protein[i].Z_CORR = TZ_CORR;
  }
   //This ends the section of code that is necessary for intaking the PDB file
  //The data is stored in structure protein
  // this next section is for determining a contact matrix between all the residues in the protein.  It does not
include any identity characteristics
  //First we are going to create a matrix in which to put the data
  //Then we are going determine the distance measurements and fill in the matrix.
     //These variables are required for the contact matrix formation.
    int j, k;
  int MAXRES, MINRES;
  float XX, YY, ZZ, Distance;
  MAXRES=protein[MAXN-1].RESN;
  MINRES=protein[1].RESN;
 int CMatrix[MAXRES+1][MAXRES+1];
  for (i=1;i<=MAXRES;i++){
   for (j=1;j<=MAXRES;j++)
        CMatrix[i][j]=0;
  }
     //We also require constants MAXN and DC for this code segment.
  int count=0;
  for (i=26;i<=MAXRES;i++){
   //this loop goes through each residue in the structure protein
 for (j=1; j<=MAXN; j++){
        // This loop goes through all atoms in the structure protein
        if (protein[j].RESN==i){
          //If atom (j) is in residue (i) then:
          for (k=j; k<MAXN; k++){
           //loop through all atoms again
                  if(protein[k].RESN!=i){
            //check and make sure that the residue is not the one being examined
            XX=(protein[k].X_CORR-protein[j].X_CORR)*(protein[k].X_CORR-protein[j].X_CORR);
           YY=(protein[k].Y_CORR-protein[j].Y_CORR)*(protein[k].Y_CORR-protein[j].Y_CORR);
            ZZ=(protein[k].Z_CORR-protein[j].Z_CORR)*(protein[k].Z_CORR-protein[j].Z_CORR);
             Distance=sqrt(XX+YY+ZZ);
            //Determine distance between atoms j and k
            if (Distance<DC){
                  CMatrix[i][protein[k].RESN]=1;
                  CMatrix[protein[k].RESN][i]=1;
```

```
                    //makes both halves of the contact matrix;
                }} }}}}
    count=0;
    for (i=1;i<MAXRES;i++){
      for (j=1;j<MAXRES;j++)
            if (CMatrix[i][j]==1)
              count=count+1;
    }
       // This is the end of the code for generating the contact matrix.
    // This contact matrix is used for the remainder of the program.
// the next job is to correct for identitiy in the contact matrix.
            ifstream align1and2("TEM1PSE4.txt");
            //opens alignment between parent 1 and parent 2
            ifstream align1and3("PSE4SED1.txt");
            //opens alignment between parent 1 and parent 3
            ifstream align2and3("TEM1SED1.txt");
            //opens alignment between parent 2 and parent 3
            ifstream align1and4("PSE4AST.txt");
            //opens alignment between parent 1 and parent 4
            ifstream align2and4("TEM1AST.txt");
            //opens alignment between parent 2 and parent 4
            ifstream align3and4("AST1SED1.txt");
            //opens alignment between parent 3 and 4
            if (align1and2.bad()){
      cerr<<"error couldn not open TEM1PSE4.txt";
      exit (8);
    }
            if (align1and3.bad()){
      cerr<<"error couldn not open TEM1SHV1.txt";
      exit (8);
    }
            if (align2and3.bad()){
      cerr<<"error couldn not open PSE4SHV1.txt";
      exit (8);
    }
            if (align1and4.bad()){
              cerr<<"error could not open 1to4";
              exit (8);
            }
            if (align2and4.bad()){
              cerr<<"error could not open 2to4";
              exit(8);
            }
            if (align3and4.bad()){
              cerr<<"error could not open 3to4";
              exit (8);
                }
            //error messages for bad file inputs
            int index, value, l;
            static int alignmaster[5][5][350];
            static int mastercontact[5][5][350][350];
            int parentN=4;

            for (i=1; i<=parentN; i++){
                        for (j=1; j<=parentN; j++){
                                    for (k=1; k<350; k++){
```

```
                                        alignmaster[i][j][k]=1;
                                        for (l=1;l< 350; l++){
                                        mastercontact[i][j][k][l]=0;
                                        }}}}
                    //initialize matrices with correct ones or zeros
                    //declare input variables and the matrix into which they are put
                    for (i=MINRES; i<=MAXRES;i++){
                      align1and2>>index>>value;
                      alignmaster[1][2][index]=value;
                      alignmaster[2][1][index]=value;
                      align2and3>>index>>value;
                      alignmaster[2][3][index]=value;
                      alignmaster[3][2][index]=value;
                      align1and3>>index>>value;
                      alignmaster[1][3][index]=value;
                      alignmaster[3][1][index]=value;
                      align1and4>>index>>value;
                      alignmaster[1][4][index]=value;
                      alignmaster[4][1][index]=value;
                      align2and4>>index>>value;
                      alignmaster[2][4][index]=value;
                      alignmaster[4][2][index]=value;
                      align3and4>>index>>value;
                      alignmaster[3][4][index]=value;
                      alignmaster[4][3][index]=value;
                                }
                    for (i=1; i<=parentN; i++){
                      for (j=i;j<=parentN; j++){
                        for (k=MINRES; k<=MAXRES;k++){
                          for (l=MINRES; l<=MAXRES; l++){
                    mastercontact[i][j][k][l]=alignmaster[i][j][k]*alignmaster[i][j][l]*CMatrix[k][l];
                    mastercontact[j][i][k][l]=alignmaster[i][j][k]*alignmaster[i][j][l]*CMatrix[k][l];
                         }}}}
                    //this finished the contact array entering and identity correction
                    //next we need to open files for outputting data and inputing crossover points.
                    ifstream cross("libraryX.txt");
                    ofstream output("Libraryxs.txt", ios::out);
                    //we also start doing each library one at a time now
                    //first read in the crossovers, then generate the chimeras
                    //and finally evaluate each chimera in the library
                    //compile the data and write to the output file.
                    int Maxdis=55;
                    int counter=0;
                    int crossN=9;    //number of crossovers
                    int total=262144; // possible number of chimeras in each library
                    static int Chimeras[262144][300];
                    for (i=0;i<total; i++){
                      for (k=1;k<=MAXRES; k++){
                        Chimeras[i][k]=0;
                      }}
// this part must be modified for a greater number of crossovers.
int C1, C2, C3, C4, C5, C6, C7, C8, C9;  // these are the fragments created by 9 crossovers.
int Crossovers[crossN];
 int sum,c, count1;
int mut[parentN+1];
for(j=1;j<=parentN; j++)
```

```
  mut[j]=0;
static int disruption[262144];
static int mutation[262144];
 for (j=0;j<total; j++){
  disruption[j]=-1;
  mutation[j]=0;
 }
cout<<"made it here"<<endl;
   counter=0;
   c=0;
   count1=0;
   for (k=0; k<crossN; k++){
     cross>>Crossovers[k];
   }

   for (C1=1; C1<=parentN; C1++){
     for (C2=1;C2<=parentN;C2++){
       for (C3=1;C3<=parentN;C3++){
              for (C4=1;C4<=parentN;C4++){
                for (C5=1;C5<=parentN;C5++){
                  for(C6=1;C6<=parentN;C6++){
                    for(C7=1;C7<=parentN; C7++){
                          for(C8=1;C8<=parentN; C8++){
                            for(C9=1; C9<=parentN; C9++){
                    for (i=1; i<=Crossovers[0]; i++)
                          Chimeras [counter][i]=C1;
                    for (i=(Crossovers[0]+1); i<=Crossovers[1]; i++)
                          Chimeras [counter][i]=C2;
                    for (i=(Crossovers[1]+1); i<=Crossovers[2]; i++)
                          Chimeras [counter][i]=C3;
                    for (i=(Crossovers[2]+1); i<=Crossovers[3]; i++)
                          Chimeras [counter][i]=C4;
                    for (i=(Crossovers[3]+1); i<=Crossovers[4]; i++)
                          Chimeras [counter][i]=C5;
                    for (i=(Crossovers[4]+1); i<=Crossovers[5]; i++)
                          Chimeras [counter][i]=C6;
                    for(i=(Crossovers[5]+1); i<=Crossovers[6]; i++)
                          Chimeras[counter][i]=C7;
                    for(i=(Crossovers[6]+1); i<=Crossovers[7]; i++)
                          Chimeras[counter][i]=C8;
                    for(i=(Crossovers[7]+1); i<=Crossovers [8]; i++)
                          Chimeras[counter][i]=C9;
                    for(i=(Crossovers[8]+1); i<=MAXRES; i++)
                          Chimeras[counter][i]=C1;
                    counter=counter+1;
                          }}}}}}}}}
   for (i=0; i<total; i++){
     sum=0;
     for (k=MINRES; k<=MAXRES; k++){
       for (l=k+1; l<=MAXRES; l++){
              if (Chimeras[i][k]!=Chimeras[i][l])
                sum=sum+mastercontact[(Chimeras[i][k])][(Chimeras[i][l])][k][l];
       }
             for (l=1; l<=parentN; l++){
              if (Chimeras[i][k]!=l){
              if ((alignmaster[(Chimeras[i][k])][l][k])==1)
```

```
            mut[l]=mut[l]+1;
           }
      }
    }
    disruption[i]=sum;
    mutation[i]=minimum (mut, parentN);
    for (l=1; l<=parentN; l++){
      mut[l]=0;
    }
    output<<disruption[i]<<"\t"<<mutation[i]<<"\t";
    for (l=0; l<crossN;l++)
      output<<Chimeras[i][Crossovers[l]]<<"\t";
    output<<Chimeras[i][291]<<endl;

  }

return 0;
}
```

Libraryparse.cc

```
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <cstdlib>
#include <cmath>

float average(float array[], int number){
  float sum=0;
  float A=0;
  if (number!=0){
  for (int i=0;i<number; i++){
     sum=sum+array[i];
    }
  A=sum/number;
  }
return A;
}

float Pfunction(float E){
  float Pf=0;
  Pf=1/(1+exp((0.138*E)-3.44));
  return Pf;
 }

float Pfunctionold(float E){
  float Pf=0;
  Pf=pow(1-0.0734*(E/322), 322);
  return Pf;
}

main ()
{
 ifstream datafile("RASPPdataX7.txt");
   if (datafile.bad()){
      cerr<<"error could not open file.txt";
      exit (8);
    }
  ofstream output("RASPPalllibrarydataME.txt", ios::out);
  //for each library loop through and calculate all the Pf and Pfold
  float libEmean, libMmean, libMPfinal, libMPoldfinal, Ffold, Ffoldold, libPf, libPfold, libmf, libmfold,
AvgME;
  int value1, value2, value3, i, j;
  float chimM[6561];
  float chimPf[6561], chimPfold[6561], chimE[6561], ME[6561];
  for (j=1; j<=1450; j++){
    libEmean=0;
    libMmean=0;
    libMPfinal=0;
    libMPoldfinal=0;
    Ffold=0;
    Ffoldold=0;
    libPf=0;
    libPfold=0;
    libmf=0;
    libmfold=0;
```

```
   for (i=0; i<6561; i++){
     chimPfold[i]=0;
     chimPf[i]=0;
     chimE[i]=0;
     chimM[i]=0;
   }
   for (i=0;i<6561; i++){
     datafile>>value1>>value2>>value3;
     if(i==0){
          cout<<j<<"\t"<<value1<<"\t"<<value2<<"\t"<<value3<<endl;
     }
     chimPf[i]=Pfunction(value2);
     chimPfold[i]=Pfunctionold(value2);
     chimE[i]=value2;
     chimM[i]=value3;
     libPf=libPf+chimPf[i];
     libPfold=libPfold+chimPfold[i];
     libmf=libmf+(chimPf[i]*chimM[i]);
     libmfold=libmfold+(chimPfold[i]*chimM[i]);
          if(value2==0)
                    ME[i]=0;
          else
                    ME[i]=value3/value2;
   }
   libEmean=average(chimE, 6561);
   libMmean=average(chimM, 6561);
   libMPfinal=libmf/libPf;
   libMPoldfinal=libmfold/libPfold;
   Ffold=libPf/6561;
   Ffoldold=libPfold/6561;
   AvgME=average(ME, 6561);

output<<j<<"\t"<<libEmean<<"\t"<<libMmean<<"\t"<<AvgME<<"\t"<<Ffold<<"\t"<<libMPfinal<<"\t"<
<Ffoldold<<"\t"<<libMPoldfinal<<endl;
  }

 return 0;
}
```

Librarysimulates.py
(needs other python tools from http://www.che.caltech.edu/~groups/fha)

```
#! /usr/bin/env python

import sys, os, math, string, random
import pdb, schema

def main ():
	#Read the parents
	parent_list = schema.readMultipleSequenceAlignmentFile (file('lac-msa.txt','r'))
	parents = [p for (key,p) in parent_list]
	pdb_alignment_list = schema.readMultipleSequenceAlignmentFile(file('PSE4-1G68.txt','r'))
	pdb_alignment= [p for (key, p) in pdb_alignment_list]

	# Read in the contact map
	pdb_residues = pdb.File().read(file('1G68.pdb','r'))
	residues = schema.alignPDBResidues(pdb_residues, pdb_alignment[1], pdb_alignment[0],
parents[0], ['A',' '])
	pdb_contacts = schema.getPDBContacts (residues, 4.5)
	contacts =schema.getSCHEMAContacts(pdb_contacts, parents)


	filename=file('7XRASPPdata.txt', 'w')
	filename.write("# E	m\n")

	lines = file('7Xraspplib.txt', 'r').readlines()

	for line in lines:
		if line[0] == '#':
			continue
		flds = line.split()
		crossovers = [int(x) for x in flds]
		filtered_contacts = schema.getSCHEMAContactsWithCrossovers(contacts, parents,
crossovers)
		fragments = schema.getFragments(crossovers, parents[0])
		p=len(parents)
		n=len(fragments)
		for i in range (p**n):
			#make chimeras into block pattersn
			n2c = schema.base(i,p)
			chimera_blocks = ''.join(['1']*(n-len(n2c))+['%d'%(int(x)+1,) for x in n2c])
			E = schema.getChimeraDisruption(chimera_blocks, filtered_contacts,
fragments, parents)
			m = schema.getChimeraShortestDistance(chimera_blocks, fragments, parents)
			filename.write("%d\t%d\t%d\n" % (i , E, m))



main ()
```

GAMS file for LRA

```
Sets

i chimera number /1*163/
j1 block /1*8/
p1 parent1   /1*3/
;

alias (j1, j2);
alias (j1,j3);
alias(j1,j4);
alias (p1, p2);

Parameters

chimeras(i,j1,p1)
/
 1.1.1 = 0,  1.1.2 = 0,  1.1.3 = 1,  1.2.1 = 1,  1.2.2 = 0,  1.2.3 = 0,  1.3.1 = 0,  1.3.2 = 0,  1.3.3 = 1,
1.4.1 = 1,  1.4.2 = 0,  1.4.3 = 0,  1.5.1 = 0,  1.5.2 = 1,  1.5.3 = 0,  1.6.1 = 0,  1.6.2 = 0,  1.6.3 = 1,  1.7.1
= 0,  1.7.2 = 0,  1.7.3 = 1,  1.8.1 = 0,  1.8.2 = 0,  1.8.3 = 1,
```

All chimeras represented in this particular format where 1 (31312333) and 163 (13313332) are each a chimera

```
 163.1.1 = 0,  163.1.2 = 1,  163.1.3 = 0,  163.2.1 = 0,  163.2.2 = 0,  163.2.3 = 1,  163.3.1 = 0,  163.3.2
= 0,  163.3.3 = 1,  163.4.1 = 1,  163.4.2 = 0,  163.4.3 = 0,  163.5.1 = 0,  163.5.2 = 0,  163.5.3 = 1,
163.6.1 = 0,  163.6.2 = 0,  163.6.3 = 1,  163.7.1 = 0,  163.7.2 = 0,  163.7.3 = 1,  163.8.1 = 0,  163.8.2 =
1,  163.8.3 = 0

/
*0 = unfolded , 1 = functional
fold(i)
/
1          0
163        0
/;

Variables
E(i)
Eo
Es(j1,p1)
Ep(j1,p1,j2,p2)
D;
Equations
energy_defn(i)
deviance
dof_single(j1)
dof_pair_row(j1,j2,p1)
dof_pair_col(j1,j2,p2);
energy_defn(i) .. E(i) =e= Eo + sum(j1,sum(p1$chimeras(i,j1,p1),Es(j1,p1))) +
sum(j1,sum(p1$chimeras(i,j1,p1),sum(j2$(ord(j2) > ord(j1)),sum(p2$chimeras(i,j2,p2),Ep(j1,p1,j2,p2)))));
dof_single(j1) .. 0 =e= sum(p1,Es(j1,p1));
dof_pair_row(j1,j2,p1) .. 0 =e= sum(p2,Ep(j1,p1,j2,p2));
dof_pair_col(j1,j2,p2) .. 0 =e= sum(p1,Ep(j1,p1,j2,p2));
deviance .. D =e= -2*sum(i$(fold(i)=0),E(i)) + 2*sum(i,log(1+exp(E(i))));
```

```
Model
logistic /all/;
logistic.optfile = 1;
Scalar enrg_limit /40/;
*Solve reference model the one bodies and that's it.
Es.up(j1,p1) = enrg_limit;
Es.lo(j1,p1) = -enrg_limit;
Ep.fx(j1, p1, j2, p2) = 0;
Solve logistic using nlp minimizing D;
*these are paramters to store the differences in logistic function D
*dbase is the base value from the last model (assigned D.1 to last model)
*delta pair is for pair of fragments added
*delta single of for removing a single fragment
Parameter
        Dbase,
        deltaD_singlefragment(j3),
        deltaD_pairoffragments(j3,j4);

        Dbase = D.l;
*This loop goes through all single fragments and removes/adds them to calculate the change in model from
removing these fragments
        loop(j3$(ord(j3) > 0),
                Es.fx(j3,p1) = 0;
        Solve logistic using nlp minimizing D;
                deltaD_singlefragment(j3) = Dbase - D.l;
*reset parameter bounds on energies
                Es.up(j3,p1) = enrg_limit;
                Es.lo(j3,p1) = -enrg_limit;
        );
*now we go through each fragment and try adding each pair of fragments to calculate the change in the
model from adding this pair
        loop(j4$(ord(j4)>0),
                Ep.up('1',p1,j4,p2) = enrg_limit;
                Ep.lo('1',p1,j4,p2) = -enrg_limit;
        Solve logistic using nlp minimizing D;
                deltaD_pairoffragments('1',j4) = Dbase - D.l;
                Ep.fx('1', p1,j4,p2) = 0
        );
        loop(j4$(ord(j4)>0),
                Ep.up('2',p1,j4,p2) = enrg_limit;
                Ep.lo('2',p1,j4,p2) = -enrg_limit;
        Solve logistic using nlp minimizing D;
                deltaD_pairoffragments('2',j4) = Dbase - D.l;
                Ep.fx('2', p1,j4,p2) = 0
        );
        loop(j4$(ord(j4)>0),
                Ep.up('3',p1,j4,p2) = enrg_limit;
                Ep.lo('3',p1,j4,p2) = -enrg_limit;

        Solve logistic using nlp minimizing D;
                deltaD_pairoffragments('3',j4) = Dbase - D.l;
                Ep.fx('3', p1,j4,p2) = 0
        );

        loop(j4$(ord(j4)>0),
                Ep.up('4',p1,j4,p2) = enrg_limit;
```

```
            Ep.lo('4',p1,j4,p2) = -enrg_limit;

        Solve logistic using nlp minimizing D;
            deltaD_pairoffragments('4',j4) = Dbase - D.l;
            Ep.fx('4', p1,j4,p2) = 0
        );


        loop(j4$(ord(j4)>0),
            Ep.up('5',p1,j4,p2) = enrg_limit;
            Ep.lo('5',p1,j4,p2) = -enrg_limit;

        Solve logistic using nlp minimizing D;
            deltaD_pairoffragments('5',j4) = Dbase - D.l;
            Ep.fx('5', p1,j4,p2) = 0
        );

        loop(j4$(ord(j4)>0),
            Ep.up('6',p1,j4,p2) = enrg_limit;
            Ep.lo('6',p1,j4,p2) = -enrg_limit;

        Solve logistic using nlp minimizing D;
            deltaD_pairoffragments('6',j4) = Dbase - D.l;
            Ep.fx('6', p1,j4,p2) = 0
        );


        loop(j4$(ord(j4)>0),
            Ep.up('7',p1,j4,p2) = enrg_limit;
            Ep.lo('7',p1,j4,p2) = -enrg_limit;

        Solve logistic using nlp minimizing D;
            deltaD_pairoffragments('7',j4) = Dbase - D.l;
            Ep.fx('7', p1,j4,p2) = 0
        );

        loop(j4$(ord(j4)>0),
            Ep.up('8',p1,j4,p2) = enrg_limit;
            Ep.lo('8',p1,j4,p2) = -enrg_limit;

        Solve logistic using nlp minimizing D;
            deltaD_pairoffragments('8',j4) = Dbase - D.l;
            Ep.fx('8', p1,j4,p2) = 0
        );


Display Dbase;
Display deltaD_singlefragment;
Display deltaD_pairoffragments;
```

FAM.cc

```
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <cstdlib>
#include <cmath>
#include <ctime>
main()
{
 int i,j; //loop variables
 //open the alignment file
 ifstream alignment("lactamasePfam.txt");
 if(alignment.bad()){
  cerr<<"error could not open alignment";
  exit(8);
 }

 ofstream distributions("distributionlac.txt", ios::out);
 ofstream scores ("scoreslac.txt", ios::out);

 const int size=703;// (j)
 const int numbersequences=133; //(i)
 static int sequencestore[numbersequences][size];
 char aasequence[size];
 char name;

 //extract each line(i) and assign each letter(j) the appropriate info

 for (i=0; i<numbersequences; i++){
  alignment>>aasequence;
  //cout<<aasequence<<endl<<endl;
  for(j=0; j<size; j++){
   if(aasequence[j]=='-'|| aasequence[j]=='*'|| aasequence[j]=='.'){
         sequencestore[i][j]=0;     }
   else if(aasequence[j]=='A'|| aasequence[j]=='a'){
         sequencestore[i][j]=1;     }
   else if (aasequence[j]=='C'|| aasequence[j]=='c'){
         sequencestore[i][j]=2;     }
   else if (aasequence[j]=='D'|| aasequence[j]=='d'){
         sequencestore[i][j]=3;     }
   else if (aasequence[j]=='E'|| aasequence[j]=='e'){
         sequencestore[i][j]=4;     }
   else if (aasequence[j]=='F'|| aasequence[j]=='f'){
         sequencestore[i][j]=5;      }
   else if (aasequence[j]=='G'|| aasequence[j]=='g'){
         sequencestore[i][j]=6;     }
   else if (aasequence[j]=='H'|| aasequence[j]=='h'){
         sequencestore[i][j]=7;     }
   else if (aasequence[j]=='I'|| aasequence[j]=='i'){
         sequencestore[i][j]=8;     }
   else if (aasequence[j]=='K'|| aasequence[j]=='k'){
         sequencestore[i][j]=9;     }
   else if (aasequence[j]=='L'|| aasequence[j]=='l'){
         sequencestore[i][j]=10;     }
   else if (aasequence[j]=='M'|| aasequence[j]=='m'){
         sequencestore[i][j]=11;     }
```

```
    else if (aasequence[j]=='N'|| aasequence[j]=='n'){
            sequencestore[i][j]=12;      }
    else if (aasequence[j]=='P'|| aasequence[j]=='p'){
            sequencestore[i][j]=13;      }
     else if (aasequence[j]=='Q'|| aasequence[j]=='q'){
            sequencestore[i][j]=14;       }
    else if (aasequence[j]=='R'|| aasequence[j]=='r'){
            sequencestore[i][j]=15;       }
     else if (aasequence[j]=='S'|| aasequence[j]=='s'){
             sequencestore[i][j]=16;       }
     else if (aasequence[j]=='T'|| aasequence[j]=='t'){
             sequencestore[i][j]=17;      }
     else if (aasequence[j]=='V'|| aasequence[j]=='v'){
             sequencestore[i][j]=18;      }
     else if (aasequence[j]=='W'|| aasequence[j]=='w'){
             sequencestore[i][j]=19;       }
            else if  (aasequence[j]=='Y'|| aasequence[j]=='y'){
              sequencestore[i][j]=20;    }
    else {
            sequencestore[i][j]=21;
                }    } }

  int bins[size][22];
  int a;
  for (j=0; j<size; j++){
    for (a=0; a<22; a++){
      bins[j][a]=0;
    }
    for (i=0; i<numbersequences; i++){
      bins[j][sequencestore[i][j]]=bins[j][sequencestore[i][j]]+1;
    }
    if(sequencestore[0][j]!=0){
      distributions<<j<<"\t"<<sequencestore[0][j]<<"\t";
      for (a=0; a<22; a++){
            distributions<<bins[j][a]<<"\t";
      }
      distributions<<endl;
    } }

  float weight[3][j];
  float  sum[j];
  for (j=0; j<size; j++){
    sum[j]=0;
    if(sequencestore[0][j]!=0 && sequencestore[0][j]!=21){

      for (a=1; a<21; a++){
            sum[j]=sum[j]+bins[j][a];
      }
      for (a=0; a<3; a++){
            weight[a][j]=bins[j][sequencestore[a][j]]/sum[j];
      }
scores<<j<<"\t"<<sum[j]<<"\t"<<sequencestore[0][j]<<"\t"<<weight[0][j]<<"\t"<<weight[1][j]<<"\t"<<w
eight[2][j]<<endl;
    }    }}
```