Appendix D

MATLAB SCRIPTS

D.1. Plotting x,y data: MEplotter

The script **MEplotter** plots x and y data, including steady-state absorption spectra and single-wavelength transient absorption kinetics.

MEplotter

```
% TO PLOT WORKED-UP TRACES

i=1;
while i ~= 0
% READ DATAFILE
    [file,dirpath]=uigetfile('*.*');
    if isequal(file,0) || isequal(dirpath,0)
        disp('no more')
        i=0;
        return
    end

    filet=[dirpath file];
    data=dlmread(filet);
% EXTRACT X AND Y
    xx=data(:,1);
    yy=data(:,2);
% PLOT
    a=rand(3);
    plot(xx,yy,'color',a(1:3));
    %semilogx(xx,yy,'color',a(1:3));

end
disp('voila!')
```

D.2. Time-resolved single-wavelength data workup

Raw data collected on the ns1 (nanosecond pulsed, single-wavelength) system are saved into a " *.ns1" file. This must first be converted to intensity vs. time (luminescence) or ΔOD vs. time (transient absorption) using the ns1_read.m script provided by Dr. Jay Winkler (available on the bilrc.caltech.edu website.) The ns1_read.m script saves the new, two column data as a .txt file, with time in the first column and intensity/ΔOD in the second.

D.2.1. Time-zero adjustment

The NSI instrumentation designates a "time-zero". However, this designated time point is not always accurate, and can fluctuate by as much as 30 ns over the course of several months. All of the kinetics data in this Thesis are time-adjusted using **xadjuster** so that time-zero corresponds to the beginning of the rise in signal (within a few data points).

xadjuster

```
% xadjuster
% To adjust x data to time=0

i=1;
while i ~= 0

% READ DATAFILE
    [file,dirpath]=uigetfile('*.*');
    if isequal(file,0) || isequal(dirpath,0)
        disp('no more')
        i=0;
        return
    end
    filet=[dirpath file];
    data=dlmread(filet);

%DETERMINE FILE PROPERTIES
    data=dlmread(filet);                %read raw file to var data
    len=length(filet);                  %measure length of file name
    file1=filet(1:len-4);               %remove the .dat
    look=find(file1=='/');              %find all the / in the file
```

```
name
    maxf=max(look)+1;                    %find the last / and index 1
value past it (file name)
    file2=file1(maxf:end);               %this was the file name
    file3=double(file2);                 %file name converted to ascii

% EXTRACT X AND Y
    xx=data(:,1);
    yy=data(:,2);
% PLOT
    a=rand(3);
    plot(xx,yy,'b');
    hold

% ASK FOR X-ADJUST
    adjfactor=input('input the x offset (in seconds): ');
    disp(adjfactor)
    xnew=xx+adjfactor;
    plot(xnew,yy,'r')
    hold

%WRITE DATA TO FILE
        corcmp=[xnew yy];
        filecmp1=[file3, 100 46   116   120   116];   %Adds 'd.txt' to
the end of the file name
        filecmp2=char(filecmp1);
        dlmwrite(filecmp2,corcmp,'precision',14);       %write a file
of the adjusted xy data
        disp(filecmp2);                                  %tell you the
name of the new file

end
```

D.2.2. Log-compression

Data that would be subjected to (multi)exponential fitting protocols were log-compressed, to avoid weighting of longer timepoints. The **compress** script was written to streamline this process. The function **logtimej** was modified from a script provided by Dr. Jay Winkler. In this case, **compress** sets the number of points per decade (ppd) to 600; the TA data in Chapters 2, 3, and most of Chapter 5 used this compression. Picosecond TA data (ps1), and picosecond-to-millisecond data overlays (Chapter 5 CO rebinding studies) were compressed to 200 ppd.

compress

```
%COMPRESSES WORKED UP DATA, AND WRITES A NEW FILE
clear

ans=input('how many traces do you want to compress? -->');
xi=[];
yi=[];

for i=1:ans
    %READ DATAFILE
        [file,dirpath]=uigetfile('/Users/Maraia/MATLAB/0Ahora/.dat');
% -> CHANGE THIS TO YOUR DIRECTORY
        if isequal(file,0) || isequal(dirpath,0)
% So it doesn't give an error if you press cancel
            disp('cancelled!')
            i=0;
            return
        end
        %file=input('enter file path','s');
        filet=[dirpath file];

    %DETERMINE FILE PROPERTIES
        data=dlmread(filet);                %read raw file to var data
        len=length(filet);                  %measure length of file
name
        file1=filet(1:len-4);               %remove the .dat
        look=find(file1=='/');              %find all the / in the
file name
        maxf=max(look)+1;                   %find the last / and index
1 value past it (file name)
        file2=file1(maxf:end);              %this was the file name
        file3=double(file2);                %file name converted to
ascii
```

```
    %EXTRACT X AND Y
        t=data(:,1);                            %read the time data out to
var xx
        y=data(:,2);                            %read intensity out to yy
        ppd=600;                                %points per decade

        [tout,yout,wt] = logtimej(t,y,ppd);
        tout=tout';
        yout=yout';

    %WRITE DATA TO FILE
        corcmp=[tout yout];
        filecmp1=[file3, 99    46   116   120   116];   %Adds 'c.txt'
to the end of the file name
        filecmp2=char(filecmp1);
        dlmwrite(filecmp2,corcmp,'precision',14);       %write a file
of the adjusted xy data
        disp(filecmp2);                                 %tell you the
name of the new file
end
disp('compressed!');
```

logtimej

```
function [tout,yout,wt]=logtimej(t,y,ppd)
%
%     function to convert y=f(t) data from linear in time
%     to logarithmic in time
%
%     SYNTAX:
%        [tout,yout,wt]=logtime(t,y,ppd)
%
%          t is the input time vector, linearly spaced
%          y is the input y vector
%          ppd is the number of points per decade for the output
vectors
%          tout is the new time vector space logarithmically
%          yout is the new time vector space logarithmically
%          wt is vector of weights giving the number of points averaged
%                   to get the new yout value

%Cut out time<0 data, which can't be plotted on a log scale
ncount=1;
while t(ncount)<=0
    ncount=ncount+1;
end
t=t(ncount:end);
y=y(ncount:end);


zt=length(t);
zy=length(y);
%
%
if zt ~= zy
   fprintf('ERROR: y and t vectors are different lengths')
else

   tmin=min(t);
   tmax=max(t);
%
   ltmin=log10(tmin);
   ltmax=log10(tmax);
%
   tt=ltmin:(1./(2.*ppd)):ltmax;
   tt=10.^tt;
   ztt=length(tt);
%
%
   icount=1;
   jcount=1;
   for i=2:2:ztt
      tout(jcount)=0;
      yout(jcount)=0;
      wt(jcount)=0;
      while (icount < zt) & t(icount) < tt(i)
        tout(jcount)=tout(jcount)+t(icount);
        yout(jcount)=yout(jcount)+y(icount);
```

```
        wt(jcount)=wt(jcount)+1;
        icount=icount+1;
      end
      if wt(jcount) ~= 0
         tout(jcount)=tout(jcount)./wt(jcount);
         yout(jcount)=yout(jcount)./wt(jcount);
         jcount=jcount+1;
      end
   end
end
%
%
%
```

D.3. Data-splicing

In order to cover the (sub)nanosecond-to-second time range, log-compressed data taken at multiple timescales were spliced together to generate a complete trace. The script **overlayer** was used to import data for three timescales and multiplicatively scale at the users discretion. The function **combine** was used to designate splicing points between timescales, and generate the final combined data.

overlayer

```
%OVERLAYER
%plots fastamp, slowamp, and no-stirring traces, and asks
%for multiplyers to adjust the fast and no-stirring traces
%so that they overlay better

tscale=input('To splice 2us, 100us, and 10ms data, enter 1. \n To
splice 100us, 10ms, 500ms data, enter 2. \n To splice 5ns, 50ns, and
2us-500ms data, enter 3 ->');

% READ DATAFILES
disp('select the fastest trace');

    [file,dirpath]=uigetfile('*.txt');  %opens a window to pick my file
     if isequal(file,0) || isequal(dirpath,0)
            disp('cancelled!')
            hold
            i=0;
            return
     end
    filet=[dirpath file];
    data1=dlmread(filet);

disp('select the corresponding middle trace-->');

    [file,dirpath]=uigetfile('*.txt');  %opens a window to pick my file
    if isequal(file,0) || isequal(dirpath,0)
            disp('cancelled!')
            hold
            i=0;
            return
    end
    filet=[dirpath file];
    data2=dlmread(filet);

disp('select the corresponding longest trace-->');
```

```
    [file,dirpath]=uigetfile('*.txt');  %opens a window to pick my file
    if isequal(file,0) || isequal(dirpath,0)
            disp('cancelled!')
            hold
            i=0;
            return
    end
    filet=[dirpath file];
    data3=dlmread(filet);

% EXTRACT DATA X and Y DATA

x1=data1(:,1);
y1=data1(:,2);
x2=data2(:,1);
y2=data2(:,2);
x3=data3(:,1);
y3=data3(:,2);


% FIRST OVERLAY OF DATA
semilogx(x1,y1,'r');
hold
semilogx(x3,y3,'b');
semilogx(x2,y2,'g');
hold

mult2=1;
mult3=1;

seguir=input('does it need adjustment? (y/n) ->', 's');
if seguir == 'n'
    y1adj=y1;
    y2adj=y2;
    y3adj=y3;
end

while seguir == 'y';
    mult1=input('select a new first trace multiplier (default 1)->');
    if mult1 <= 0
        mult1=1;
        disp('defaulting multf=1');
    end
    mult2=input('select a new second trace multiplier (default 1)->');
    if mult2 <= 0
        mult2=1;
        disp('defaulting mults=1');
    end
    mult3=input('select a new third trace multiplier (default 1)->');
    if mult3 <= 0
        mult3=1;
        disp('defaulting multL=1');
    end

    y1adj=mult1.*y1;
    y2adj=mult2.*y2;
    y3adj=mult3.*y3;
```

```
    semilogx(x1,y1adj,'r');
    hold
    semilogx(x3,y3adj,'b');
    semilogx(x2,y2adj,'g');
    hold

    seguir=input('does it need adjustment? (y/n)', 's');
end
disp(mult1);
disp(mult2);
disp(mult3);

datac=combine(tscale,x1,y1adj,x2,y2adj,x3,y3adj);

T=datac(:,1);
Y=datac(:,2);
TY=[T,Y];

keep=input('do you want to keep this combined trace? (y/n)->', 's');
if keep == 'y';
    [Savefile,Savedirpath,Savefilter]=uiputfile([dirpath,'*.txt'],'Save
overlayed traces',[dirpath,file]);
    eval(['save ''',Savedirpath,Savefile, ''' TY -ASCII -DOUBLE']);
end
```

combine

```
function datac=combine(tscale,x1,y1adj,x2,y2adj,x3,y3adj)
%function datac=combine(xf,yfadj,xs,ys)
%TO CREATE A SINGLE DATA SET COMBINING DATA FOR THREE TIMESCALES
% INPUTS
% vectors containing time (x) and delta OD (y) data
% key:f,s & L are fast amp, slow amp, and no stirring.
%   yf and yL are adjusted (hence yfadj, yLadj) by a multiplier to
%   remove dicontinuities between the data from the three time ranges
%
% OUTPUTS
% datac is a matrix with the combined time values in the first
% column, and delta OD values in the second column

%Timescales for different purposes
if tscale == 1

%For combining the 2us, 100us, 10ms data
    time1l = 1e-9;
    time1u = 1.5e-6;

    time2l=1.5e-6;
    time2u=0.5e-4;      %used to be 0.7e-4/0.4e-4

    time3l=0.5e-4;

elseif tscale == 2

%For combining the 400us, 10ms, 500ms data
    time1l = 1e-9;
    time1u = 5e-5;

    time2l=5e-5;
    time2u=1e-3;

    time3l=1e-3;  %USED to be 2.5e-3

elseif tscale == 3

%For combining the 5ns, 50ns, 2us-500ms data
    time1l = 1e-12;
    time1u = 3.5e-9;

    time2l=3.5e-9;
    time2u=35e-9;

    time3l=35e-9;


end
```

```
%ADJUSTING FIRST TRACE

    %cycles to find indexes of the lower and upper bound times

    xli=1;
    while x1(xli) <= time1l
        xli=xli+1;
    end

    xui=xli;
    while x1(xui) <=time1u
        xui=xui+1;
    end

    newx1=x1(1:xui);
    newy1=y1adj(1:xui);
    newx1=newx1';
    newy1=newy1';

% ADJUSTING SECOND TRACE

    %cycles to find indexes of the lower and upper bound times
    xli=1;
    while x2(xli) <= time2l
        xli=xli+1;
    end

    xui=xli;
    while x2(xui) <=time2u
        xui=xui+1;
    end

    newx2=x2(xli:xui);
    newy2=y2adj(xli:xui);
    newx2=newx2';
    newy2=newy2';

% ADJUSTING THIRD TRACE

    %cycles to find indexes of the lower bound times
    xli=1;
    while x3(xli) <= time3l
        xli=xli+1;
    end

    newx3=x3(xli:(end));
    newy3=y3adj(xli:(end));
    newx3=newx3';
    newy3=newy3';

%creates the combined trace
combx=[newx1 newx2 newx3];
comby=[newy1 newy2 newy3];

combx=combx';
comby=comby';
```

```
hold
semilogx(combx, comby, 'y');
hold

datac=[];
datac(:,1)=combx;
datac(:,2)=comby;
end
```

D.4. Singular Value Decomposition

These scripts use truncated, generalized singular value decomposition (from Regularization Tools, Per Christian Hansen) to determine the number of kinetic components in single-wavelength transient absorption kinetics traces. This can be done for a single kinetics trace using **svder1**, or accomplished and overlayed for six wavelengths using **svderMulti.**

svder1

```
logkspace=0:log10(2):10;
kspace=10.^logkspace;
t=input('define the time vector  ');
logAspace=-t*kspace;
Aspace=exp(logAspace);

Adata=input('define the absorbance data vector  ');

L=get_l(34,1);
[UU sm XX]=cgsvd(Aspace,L);

figure
trunc=input('enter the truncation value (<34) ');
X_L=tgsvd(UU,sm,XX,Adata,trunc);
bar(log10(kspace),X_L);

Bcalc=Aspace*X_L;
figure
semilogx(t,Adata,'b',t,Bcalc,'r')
```

svderMulti

```
%svderMulti
%Does SVD for 6 wavelengths

logkspace=0:log10(2):10;
kspace=10.^logkspace;
t=input('define the time vector  ');
logAspace=-t*kspace;
Aspace=exp(logAspace);

Adata=input('define the absorbance data MATRIX  ');

L=get_l(34,1);
[UU sm XX]=cgsvd(Aspace,L);

len=length(kspace);
X_LMat=zeros(len,6);
trunc=input('enter the truncation value (<34) ');

for icount = 1:6
    X_L=tgsvd(UU,sm,XX,Adata(:,icount),trunc);
    X_LMat(:,icount)=X_L;           %put it back in a big matrix
end

%multiplotting
col(1)='k';
col(2)='b';
col(3)='c';
col(4)='g';
col(5)='y';
col(6)='r';
col=col(:);

figure
hold

for ocount=1:30;
    barx(:,1)=X_LMat(ocount,:);
    barx(:,2)=[1; 2; 3; 4; 5; 6;];
    barx(:,3)=abs(barx(:,1));
    barz=sortrows(barx,-3);
    for incount=1:6;
        ind=barz(incount,2);
        bar(log10(kspace(ocount)),barz(incount,1),0.25,col(ind));
        %pause(1)
    end
end
```

D.5. Multiexponential fitting

A number of scripts are used to fit TA data to a sum of multiexponentials. **nonlinear_fitter4** fits data to a sum of three and four exponentials, to compare fits. It uses the function **autoresider** to plot residuals and autocorrelation of residuals; these are used to determine whether deviations between data and fit are random or systematic (an additional exponent is indicated). **MExpG_Fitter** is used to globally fit data at multiple wavelengths to a sum of exponentials (defined by **MExpG**), using defined upper and lower bounds and initial guesses (**MExpGvalues**).

nonlinear_fitter4

```
%fitting kinetics data to a sum of exponentials

%getting data
xdata=input('define the x data:\n');
ydata=input('define the y data:\n');
figure
semilogx(xdata,ydata,'c');
hold
%defining fitting & parameters
disp('we will be fitting to a sum of 3, 4 exponentials');

ft=fittype('a.*exp(-x.*2.2e7)+b.*exp(-x.*k2)+c.*exp(-x.*k3)+f');
%     a      b      c      f     k2     k3
pu=[ 0.1,   0.3,    1,     1,  1e7,  5e6];
pl=[-0.1,  -0.4,   -1,    -1,  1e5,  5e2];
st=[-0.02, -0.15, 0.1,     0,  1e6,  5e3];

opts=fitoptions(ft);
opts=fitoptions(opts,'lower',pl,'upper',pu,'startpoint',st);
[f,gof]=fit(xdata,ydata,ft,opts);
disp(f);
xcalc=xdata;
ycalc=feval(f,xcalc);
plot(xcalc,ycalc,'r');


%used for all wavelengths
ft2=fittype('a.*exp(-x.*2.2e7)+b.*exp(-x.*k2)+c.*exp(-x.*k3)+d.*exp(-
x.*k4)+f');
%     a      b      c     d     f     k2   k3    k4
pu2=[ 0.1,     0.3,    1,   1,   1,   5e6,  5e5   1e4];
pl2=[-0.1,  -0.4,   -1,  -1,  -1,   5e5,  5e4   1e1];
```

```
st2=[-0.02, -0.15, 0.1,   0,   0,   1e6,  1e5   1e3];
%pu=input('define upper bounds for a-e,k2-k5: 10 parameters\n');
%pl=input('define lower bounds for a-e,k2-k5: 10 parameters\n');
%st=input('define starting values for a-e,k2-k5: 10 parameters\n');

opts=fitoptions(ft2);
opts=fitoptions(opts,'lower',pl2,'upper',pu2,'startpoint',st2);
[f,gof]=fit(xdata,ydata,ft2,opts);
disp(f);

%numpts=length(xdata);
%numpts=1000;
%xmin=log10(min(xdata));
%xmax=log10(max(xdata));
%xcalc=logspace(xmin,xmax,numpts);
ycalc2=feval(f,xcalc);
plot(xcalc,ycalc2,'b');
hold
autoresider(ydata,ycalc,ycalc2);
```

autoresider

```
function [AC_resid,AC_resid2]=autoresider(y,ycalc,ycalc2)
%
%
%   Calculate and plot the autocorrelation of the residuals between
ycalc and y
%
%   Syntax: [AC_resid]=autoresid(y,ycalc);
%
%
y=y(:);
ycalc=ycalc(:);
ycalc2=ycalc2(:);
%
len=length(y);
jlen=floor(len./1.1);
%
resid=y-ycalc;
resid2=y-ycalc2;
%
acorrin=zeros(len,jlen);
acorrin2=zeros(len,jlen);
%
for aij=1:jlen
    acorrin(1:len-aij,aij)=resid(aij+1:len); %
    acorrin2(1:len-aij,aij)=resid2(aij+1:len); %
end
mvect=len-1:-1:len-jlen;
autoc=(resid'*acorrin)./mvect;
autoc2=(resid2'*acorrin2)./mvect;
AC_resid=autoc./((resid'*resid)./len);
AC_resid2=autoc2./((resid2'*resid2)./len);
%
figure
subplot(2,1,1)
plot(y-ycalc,'r')
hold
plot(y-ycalc2,'b')
subplot(2,1,2)
plot(AC_resid,'r')
hold
plot(AC_resid2,'b')

return
```

MExpGFitter

```
%MultiExponential Global Fitter (fits to 6 wavelengths)
%asking for data
tvector=input('define the time vector -> ');
dODdata=input('define the matrix of absorbance data -> ');

x=tvector;
%defining initial values and bounds
[values,ubound,lbound] = MExpGvalues;

%defining fit parameters
fxn=@(values,x) MExpG(values,x);
optm=optimset('lsqcurvefit');
optm=optimset(optm,'Display','iter');
optm=optimset(optm,'TolFun',1e-14);
optm=optimset(optm,'TolX',1e-14);

bfit=lsqcurvefit(fxn,values,tvector,dODdata,lbound,ubound,optm);
bfite=bfit';
ks=bfite(25:end);
display(bfite)
%display(ks)

A=MExpG(bfit,tvector);
for icount = 1:6
subplot(2,3,icount)
semilogx(tvector,dODdata(:,icount),'b',tvector,A(:,icount),'r');
end
```

MExpG

```
function [A]=MExpG(values,tvector)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
a=values(1:6);
b=values(7:12);
c=values(13:18);
f=values(19:24);
k=values(25:27);
x=tvector;

%lumdec=2.7e7;

A1=(a(1).*exp(-x.*k(1))+b(1).*exp(-x.*k(2))+c(1).*exp(-x.*k(3))+f(1));
A2=(a(2).*exp(-x.*k(1))+b(2).*exp(-x.*k(2))+c(2).*exp(-x.*k(3))+f(2));
A3=(a(3).*exp(-x.*k(1))+b(3).*exp(-x.*k(2))+c(3).*exp(-x.*k(3))+f(3));
A4=(a(4).*exp(-x.*k(1))+b(4).*exp(-x.*k(2))+c(4).*exp(-x.*k(3))+f(4));
A5=(a(5).*exp(-x.*k(1))+b(5).*exp(-x.*k(2))+c(5).*exp(-x.*k(3))+f(5));
A6=(a(6).*exp(-x.*k(1))+b(6).*exp(-x.*k(2))+c(6).*exp(-x.*k(3))+f(6));

A=[A1, A2, A3, A4, A5, A6];
end
```

MExpGvalues

```
function [values,ubound,lbound] = MExpGvalues
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

%defining initial values and bounds
%'a' defines six pre-exponential terms corresponding to the first rate
%constant, 'b' defines those pre-exponentials for the second rate
constant,
%and the same for 'c'
a=[-0.2, -0.2, -0.2, -0.2, -0.2, -0.2];
b=[-0.15, -0.15, -0.15, -0.15, -0.15, -0.15];
c=[0.1, 0.1, 0.1, 0.1, 0.1, 0.1,];
f=[0.01, 0.01, 0.01, 0.01, 0.01, 0.01];
k=[2e7, 6e4, 1e3];              %these are the three exponential terms

values=[a b c f k];

%upper bounds
ua=[1, 1, 1, 1, 1, 1];
ub=[1, 1, 1, 1, 1, 1];
uc=[1, 1, 1, 1, 1, 1];
uf=[1, 1, 1, 1, 1, 1];
uk=[3e8,  5e6,  1e4];

ubound=[ua ub uc uf uk];

%lower bounds
la=[-1, -1, -1, -1, -1, -1];
lb=[-1, -1, -1, -1, -1, -1];
lc=[-1, -1, -1, -1, -1, -1];
lf=[-1, -1, -1, -1, -1, -1];
lk=[1e6, 5e3, 5e0];

lbound=[la lb lc lf lk];

end
```

D.6. Nonnegative least squares analysis

Multiexponential kinetics data (with all positive magnitudes), such as fluorescence decays or CO rebinding in substrate-free P450 CYP119, can be examined using nonnegative least squares (nnls) fitting. This procedure requires that the data be log-compressed, and that the first time point be exactly zero. The **nnls_prep** script sets the initial time to zero and can be used to normalize data; the script **nnls_grad_reg_r2_KT** (written by Jay Winkler and modified by Kana Takematsu) does the fitting analysis. To further analyze the nnls outputs, the script **Panalyzer** and function **Pmoments** allow the user to define populations within the nnls histogram of rate constants and amplitude, and determine statistics (integrated amplitudes, first moment, and second and third centered moments) for each population.

nnls_prep

```
% nnls_prep
% This script selects the maximum absorbance data point from x,y TA
data, sets that time point to t=0, and
% normalizes Y (absorbance) data

i=1;
while i ~= 0

% READ DATAFILE
    [file,dirpath]=uigetfile('*.*');
    if isequal(file,0) || isequal(dirpath,0)
        disp('no more')
        i=0;
        return
    end
    filet=[dirpath file];

%DETERMINE FILE PROPERTIES
    data=dlmread(filet);                %read raw file to var data
    len=length(filet);                  %measure length of file name
    file1=filet(1:len-4);               %remove the .dat
    look=find(file1=='/');              %find all the / in the file
name
    maxf=max(look)+1;                   %find the last / and index 1
value past it (file name)
    file2=file1(maxf:end);              %this was the file name
```

```
    file3=double(file2);                    %file name converted to ascii

% EXTRACT X AND Y
    xx=data(:,1);
    yy=data(:,2);

% DATA ADJUSTMENTS
%    [myy,indmyy]=max(yy);                  %find value and index of max
Y-data point
    indmyy=21;
    xadj=xx(indmyy);                        %find time value of max Y-data
point
    xtrunc=xx(indmyy:end);                  %truncating rise from X-data
    ytrunc=yy(indmyy:end);                  %truncating rise from Y-data
    xnew=xtrunc-xadj;                       %setting first data point as
time zero
%    ynorm=ytrunc/myy;                      %normalizing Y-data to max Y-
data point

% PLOT
    plot(xx,yy,'r')
    plot(xnew,ytrunc,'b');


%WRITE DATA TO FILE
        corcmp=[xnew ytrunc];
        filecmp1=[file3, 95  116  48  110  46   116   120   116];
%Adds '_t0n.txt' to the end of the file name
        filecmp2=char(filecmp1);
        dlmwrite(filecmp2,corcmp,'precision',14);       %write a file
of the adjusted xy data
        disp(filecmp2);                                 %tell you the
name of the new file

end
```

nnls_grad_reg_r2_KT

```
%   Script to read [t,y] data from a file
%   Fit using chisq and first derivative constraint
%   with lsqnonneg and scan lambda to maximize breadth.
%
%
clear all
close all
%
%lambda_min_def=0.001;
%lambda_max_def=10;
lambda_min_def=0.01;
lambda_max_def=50;
lambda_inc_def=1.5;
k_inc_def=1.5;
%
H0=figure;
%
[FileName,PathName] = uigetfile('*.txt') ;
filet=[PathName FileName];
xy=dlmread(filet);
%[a,b]=size(xy);
%if (a~=2)&(b~=2)
 %   return
%elseif (a==2)
 %   xy=xy';
%end
%
x=xy(:,1);
y=xy(:,2);
warning off
wt=1./sqrt(y);
wttest=isfinite(wt);
wttest=isinf(1./wttest);
sqrt(y+wttest);
wt=1./sqrt(abs(y+wttest));
wt=[wt(1:214); 3*wt(215:length(y))];
%kana: add abs and define new wt
%MUST REMEMBER TO change this value for each data set!!!
warning on
%
%
chk3=-1;
while chk3 < 0
chk=-1;
prompt={'Enter the minimum value for \lambda',
        'Enter the maximum value for \lambda',
        'Enter the ratio of adjacent \lambda values',
        'Enter the ratio of adjacent rate constants'};
dlg_title='Fitting Parameters (enter ? for help)';
numlines=1;
options.Resize='on';
options.WindowStyle='normal';
options.Interpreter='tex';
helpstr1=['The program minimizes the sum of the squared deviations
```

```
between calculated and experimental intensities (chi-squared) AND the
sum of the squared gradient of the P(k) function. The weighting factor
between these two functions is lambda. Small \lambda values give more
weight to chi-squared minimization. Large lambda values give more
weight to P(k) gradient minimization. The program will scan from a
minimum to a maximum value of lambda. The scanned lambda values are
logarithmically spaced. This query asks for the maximum and minimum
values of lambda as well as the ratio of adjacent lambda values.'];
helpstr2=['The k-space vector is logarithmically spaced. This query
asks for the raio of adjacent k values.]'];
help1=' ';
help2=' ';
while chk < 0

defanswer={num2str(lambda_min_def),num2str(lambda_max_def),num2str(lamb
da_inc_def),num2str(k_inc_def),};
    answer=inputdlg(prompt,dlg_title,numlines,defanswer,options);
    ans1=cell2mat(answer(1));
    ans2=cell2mat(answer(2));
    ans3=cell2mat(answer(3));
    ans4=cell2mat(answer(4));
    %
    chk=1;
    hep1=0;
    hep2=0;
    if ishandle(help1)==1
        close(help1);
    end
    if ishandle(help2)==1
        close(help2);
    end
    %
    if (isempty(ans1))
        chk=-1;
    elseif ans1(1) == '?'
        hep1=1;
        chk=-1;
    else
        lambda_min=str2num(ans1);
        if isempty(lambda_min)
            chk=-1
        else
            lambda_min_def=lambda_min;
        end
    end
    %
    if (isempty(ans2))
        chk=-1;
    elseif ans2(1) == '?'
        hep1=1;
        chk=-1;
    else
        lambda_max=str2num(ans2);
        if isempty(lambda_max)
            chk=-1
        else
            lambda_max_def=lambda_max;
        end
```

```
    end
    %
    if (isempty(ans3))
        chk=-1;
    elseif ans3(1) == '?'
        hep1=1;
        chk=-1;
    else
        lambda_inc=str2num(ans3);
        if isempty(lambda_inc)
            chk=-1
        else
            lambda_inc_def=lambda_inc;
        end
    end
    %
    if (isempty(ans4))
        chk=-1;
    elseif ans4(1) == '?'
        hep2=1;
        chk=-1;
    else
        k_inc=str2num(ans4);
        if isempty(k_inc)
            chk=-1
        else
            k_inc_def=k_inc;
        end
    end
    %
    if (hep1 == 1)
        help1=helpdlg(helpstr1,'Lambda Values')
        hpos1=get(help1,'Position');
        set(help1,'Position',[hpos1(1)+0.5.*hpos1(3),
hpos1(2)+0.5.*hpos1(4), hpos1(3), hpos1(4)]);
    end
    %
    if (hep2 == 1)
        help2=helpdlg(helpstr2,'k-ratio')
        hpos2=get(help2,'Position');
        set(help2,'Position',[hpos2(1)+0.5.*hpos2(3),
hpos2(2)+0.5.*hpos2(4), hpos2(3), hpos2(4)]);
    end
    %
end
%
%lambda_max
%lambda_min
%lambda_inc
%k_inc
%
%
loglambda=log10(lambda_min):log10(lambda_inc):log10(lambda_max);
lambda=10.^loglambda;
lenlam=length(lambda);
%
%
Off_set=0;
```

```
chk=-1;
menprompt='Offset Rate Constant';
menopt1='YES';
menopt2='NO';
menopt3='Help';
helpstr3='If the data do not decay to zero, a slow rate constant can be
added to the k-space vector to produce an offset in the data. This
query asks if you want to include that offset rate constant.';
help3=' ';
while chk < 0
    %
    choice=menu(menprompt,menopt1,menopt2,menopt3);
    %
    if ishandle(help3)==1
        close(help3);
    end
    if choice == 1
        Off_set=1;
        chk=1;
    elseif choice == 2
        Off_set=0;
        chk=1;
    elseif choice ==3
        help3=helpdlg(helpstr3,'Weighting');
    end
end
%
%
iwt=0;
chk=-1;
menprompt='Data Weighting';
menopt1='No weighting';
menopt2='Weight y(i) values by 1/sqrt(yi)';
menopt3='Weight y(i) values by 1/yi';
menopt4='Help';
helpstr4='No weighting minimizes sum of {yi(exp)-yi(calc)}^2; higher
intensity values carry more weight. The uncertainty in each yi(exp)
value is approximately sqrt(yi(exp)). The sum of {[yi(exp)-
yi(calc)]/sqrt(yi(exp))}^2 is used to generate reduced chi-squared
values. The optimum result is reduced chi-squared = 1. The sum of
{[yi(exp)-yi(calc)]/yi(exp)}^2 minimizes the fractional deviations of
experiment from calculated values.';
help4=' ';
while chk < 0
    %
    choice=menu(menprompt,menopt1,menopt2,menopt3,menopt4);
    %
    if ishandle(help4)==1
        close(help4);
    end
    %
    if choice == 1
        WT=eye(length(y));
        chk=1;
    elseif choice == 2
        WT=diag(wt);
        chk=1;
    elseif choice == 3
```

```
        WT=diag(wt);
        WT=WT.^2;
        chk=1;
    elseif choice == 4
        help4=helpdlg(helpstr4,'Weighting');
    end
end
%
chk=-1;
menprompt='Gradient Method';
menopt1='Two-point';
menopt2='Three-point';
menopt3='Help';
helpstr5='The two-point gradient for P(ki) is [P(k(i+1))-
P(k(i))]/delta. The three-point gradient for P(ki) is calculated by
fitting the three points, [P(k(i-1)), P(k(i)), P(k(i+1))] to a second
order polynomial and calculaing the gradient from the polynomial
coefficients.';
help5=' ';

while chk < 0
    %
    choice=menu(menprompt,menopt1,menopt2,menopt3);
    %
    if ishandle(help5)==1
        close(help5);
    end
    %
    if choice == 1
        grad=1;
        chk=1;
    elseif choice == 2
        grad=2;
        chk=1;
    elseif choice == 3
        help5=helpdlg(helpstr5,'Weighting');
    end
    %
end
%
%kana: what is the appropriate kmax?
len=length(y);
kmax=5./(x(2)-x(1));
kmax=min(kmax,1e12);
kmin=0.2./x(len);
kmin=1.0./x(len);
%
%
lkmin=log10(kmin);
lkmax=log10(kmax);
incr=log10(k_inc);
%
lkmin=floor(lkmin);
lkmax=ceil(lkmax);
logk=lkmin:incr:lkmax;
%
k=10.^logk;
if (Off_set == 1)
```

```
    k=[k(1)./100,k];
end
%
%
lenk=length(k);
jlen=floor(len./2);


%
%
A=x*k;
A=exp(-A);
%
if (grad == 1)
    P_reg=2.*ones(lenk,1);
    P_reg(1,1)=1;
    P_reg(lenk,1)=1;
    P_reg_3=zeros(lenk,1);
    AA=diag(P_reg);
    P_reg_2=-1.*ones(lenk-1,1);
    AA1=diag(P_reg_2,1);
    AA2=diag(P_reg_2,-1);
else
    P_reg=2.*ones(lenk,1);
    P_reg(1,1)=1;
    P_reg(lenk,1)=1;
    P_reg_3=zeros(lenk,1);
    AA=diag(P_reg);
    P_reg_2=-1.*ones(lenk-2,1);
    AA1=diag(P_reg_2,2);
    AA2=diag(P_reg_2,-2);
end
%
AA=AA+AA1+AA2;
%
%
for ijk=1:lenlam
    %
    AA_lam=lambda(ijk).*AA;
    %
    Aw=WT*A;
    AAA=[Aw; AA_lam];
    yw=WT*y;
    yy=[yw; P_reg_3];
    %
    Pr=lsqnonneg(AAA,yy);
    gradsq(ijk)=(AA*Pr)'*(AA*Pr);
    %
    chisq(ijk)=(yw-Aw*Pr)'*(yw-Aw*Pr);
    chisq(ijk)=chisq(ijk)./(length(y)-length(Pr)-1);
    %kana: deleted as necessary?
    ycalc(:,ijk)=A*Pr;
    resid(:,ijk)=y-ycalc(:,ijk);
    resid(:,ijk)=resid(:,ijk).*wt;
    %kana: wt is okay sometimes? the autocorrelation makes more sense
if it
    %is weighted?
    %
```

```
    acorrin=zeros(len,jlen);
    for aij=1:jlen
        acorrin(1:len-aij,aij)=resid(aij+1:len,ijk); %",ijk" added
4/27/2010 jrw
    end
    mvect=len-1:-1:len-jlen;
    autoc(ijk,:)=(resid(:,ijk)'*acorrin)./mvect;
    autoc(ijk,:)=autoc(ijk,:)./((resid(:,ijk)'*resid(:,ijk))./len);
    %
    Pok(:,ijk)=Pr;
    %
    subplot(3,2,1)
    semilogx(x,y-ycalc(:,ijk),'r',[min(x) max(x)],[0 0],'k')
    %semilogx(x,resid(:,ijk),'r',[min(x) max(x)],[0 0],'k')
    %kana replace
    A331=axis;
    axis([min(x) max(x) A331(3) A331(4)]);
    xlabel('time');
    ylabel('y_{obsd}-y_{calc}');
    title(['Residuals: {\chi}^2 = ',num2str(chisq(ijk),'%4g'),';
(grad(P))^2 = ',num2str(gradsq(ijk),'%4g')]);
    %
    subplot(3,2,3)
    [aa,bb]=size(autoc(ijk,:));
    plot([1:bb],autoc(ijk,:),'r',[1,bb],[0 0],'k')
    xlabel('Correlation Channel');
    ylabel('Cr(j)');
    title('Autocorrelation of Residuals');
    %
    subplot(3,2,5)
    semilogx([min(x) max(x)],[0 0],'k',x,y,'r',x,ycalc(:,ijk),'k')
    xlabel('time');
    ylabel('Intensity');
    title(['{\lambda} = ',num2str(lambda(ijk),'%4g')]);
    %
    subplot(3,2,4)
    HH=loglog(lambda(1:ijk),chisq(1:ijk),'ro-');
    xlabel('\lambda');
    ylabel('{\chi}^2');
    title('{\chi}^2 vs. \lambda');
    %
    subplot(3,2,2)
    bar(log10(k),Pr,'r')
    axis([min(log10(k)), max(log10(k)), 0, 1.025.*max(Pr)])
    xlabel('log(k)');
    ylabel('P(k)');
    title(['{\lambda} = ',num2str(lambda(ijk),'%4g')]);
    %
    subplot(3,2,6)
    loglog(chisq(1:ijk),gradsq(1:ijk),'ro-')
    xlabel('{\chi}^2');
    ylabel('(grad(P))^2');
    title(['L-curve: {\lambda} = ',num2str(lambda(ijk),'%4g')]);
    %
    drawnow
end
%
subplot(3,2,4)
```

```
title('LEFT-CLICK on a point to see its fit, residual, and
distribution.','Fontsize',12,'Color','b','FontWeight','Bold');
drawnow
%
chk2=-1;
menprompt='Select an Option';
menopt1='Chose a different point';
menopt2='Refit the data';
menopt3='EXIT THE PROGRAM';
while chk2 < 0
    [xtest,ytest, button]=ginput(1);
    %
    while (button == 1)
    %
        [lambdatest,ijktest]=min(abs(lambda-xtest));
    %
        subplot(3,2,1)
        semilogx(x,y-ycalc(:,ijktest),'r',[min(x) max(x)],[0 0],'k')
        %semilogx(x,resid(:,ijktest),'r',[min(x) max(x)],[0 0],'k')
        %kana
        A331=axis;
        %axis([min(x) max(x) A331(3) A331(4)]);
        axis([x(450) x(length(y)) -0.005 0.005])
        %kana: manually adjust0.1 to 0.005.
        xlabel('time');
        ylabel('y_{obsd}-y_{calc}');
        title(['Residuals: {\chi}^2 =
',num2str(chisq(ijktest),'%4g'),';  (grad(P))^2 =
',num2str(gradsq(ijktest),'%4g')]);
        %
        subplot(3,2,3)
        plot([1:bb],autoc(ijktest,:),'r',[1,bb],[0 0],'k')
        %axis ([1 bb -0.1 0.1])
        %kana
        xlabel('Correlation Channel');
        ylabel('Cr(j)');
        title('Autocorrelation of Residuals');
    %
        subplot(3,2,5)
        semilogx([min(x) max(x)],[0
0],'k',x,y,'r',x,ycalc(:,ijktest),'k')
        xlabel('time');
        ylabel('Intensity');
        title(['{\lambda} = ',num2str(lambda(ijktest),'%4g')]);
        %
        subplot(3,2,4)
        loglog(lambda(1:ijk),chisq(1:ijk),'ro-')
        hold on
        h1=loglog(lambda(ijktest),chisq(ijktest),'bo');
        set(h1,'MarkerFaceColor','b');
        hold off
        xlabel('\lambda');
        ylabel('{\chi}^2');
        title('LEFT-CLICK on a point to see its fit, residual, and
distribution; RIGHT-CLICK to
proceed.','Fontsize',12,'Color','b','FontWeight','Bold');
        drawnow
        %
```

```
        subplot(3,2,2)
        bar(log10(k),Pok(:,ijktest),'r')
        axis([min(log10(k)), max(log10(k)), 0,
1.025.*max(Pok(:,ijktest))])
        xlabel('log(k)');
        ylabel('P(k)');
        title(['{\lambda} = ',num2str(lambda(ijktest),'%4g')]);
        %
        subplot(3,2,6)
        loglog(chisq(1:ijk),gradsq(1:ijk),'ro-')
        hold on
        h2=loglog(chisq(ijktest),gradsq(ijktest),'bo');
        set(h2,'MarkerFaceColor','b');
        hold off
        xlabel('{\chi}^2');
        ylabel('(grad(P))^2');
        title(['L-curve: {\lambda} = ',num2str(lambda(ijk),'%4g')]);
        %
        drawnow
        %
        [xtest,ytest, button]=ginput(1);
        %
        %
    end
%
    choice=menu(menprompt,menopt1,menopt2,menopt3);
    %
    if choice == 1
        chk2=-1;
    elseif choice == 2
        chk2=1;
        chk3=-1;
    elseif choice == 3
        chk2=1;
        chk3=1;
    end
%     close(H1);
%     close(H2);
%     close(H3);
    %kana
end
%
end
%kana: insert saving
pause(3)
[filename2,PathName2] = uiputfile('*.txt','choose directory','bozo');
[pathstr1,name1,ext1] = fileparts(filet);
filet2=[PathName2,name1];
dlmwrite(strcat(strcat(filet2,'_kPok'),'.dat'),[log10(k)',Pok])
dlmwrite(strcat(strcat(filet2,'_residual'),'.dat'),[x,repmat(y,1,size(y
calc,2))-ycalc])
dlmwrite(strcat(strcat(filet2,'_lambda'),'.dat'),[lambda'])
dlmwrite(strcat(strcat(filet2,'_autoc'),'.dat'),[autoc'])
close all hidden
%
%
```

Panalyzer

Written to determine population statistics, given a histogram (such as the population distributions from fitting of CO rebinding data, Chapter 5).

```
%Panalyzer


% READ DATAFILE
    [file,dirpath]=uigetfile('*.*');
    if isequal(file,0) || isequal(dirpath,0)
        disp('no more')
        i=0;
        return
    end
    filet=[dirpath file];
    data=dlmread(filet);

% EXTRACT X AND Y
    ks=data(:,1);
    Poks=data(:,2:end);

% GET INDEX
    %ind=17;
    ind=input('what is the index for the desired P values? ');
    desPoks=Poks(:,ind);

% PLOT
    %a=rand(3);
    figure
    bar(ks,desPoks,'r');
    hold
    %semilogx(xx,yy,'color',a(1:3));

% SELECT POPULATIONS
    klen=length(ks)
    n=0;

    seguir=input('would you like to analyze a population? (y/n):
','s');

    col(1)='y';
    col(2)='g';
    col(3)='c';
    col(4)='b';
    col(5)='m';
    col(6)='k';

    while seguir=='y'
        redo='y';
        n=n+1;
        while redo=='y'
```

```
            hold
            bar(ks,desPoks,'r');
            hold
            Plow(n)=input('enter the lower k(index) for the population:
');
            Pup(n)=input('enter the upper k(index) for the population:
');
            bar(ks(Plow(n):Pup(n)),desPoks(Plow(n):Pup(n)),col(n))
            redo=input('would you like to select a different region for
this population? (y/n): ','s');
        end
        seguir=input('would you like to select another population?
(y/n): ','s');
    end

    %DETERMINE STATISTICS (MOMENTS)
    for m=1:n

[tempint,tempcentro,tempcmoms]=Pmoments(desPoks(Plow(m):Pup(m)),ks(Plow
(m):Pup(m)));
    Pint(m)=tempint;
    Pcentro(m)=tempcentro;
    Pcmoms(:,m)=tempcmoms;
    end

   Pint
   Pcentro
   Pcmoms
```

Pmoments

```
function [P1int,centro,cmoms] = Pmoments(Ps,ks)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

%Ps are the populations
%ks are the log10 of the rate constants

centro=sum(Ps.*ks)./sum(Ps);
centks=ks-centro;

P1int=trapz(Ps);            %Integrating the population distribution

%uncentered moments
% for n=1:4
%     moms(n)=sum(Ps.*(ks.^n))./sum(Ps);
%     moms=moms';
% end

%centered moments
for n=1:4
    cmoms(n)=sum(Ps.*(centks.^n))./sum(Ps);
    cmoms=cmoms';
end

end
```

D.7. Hopping Maps

The script **MapMaker** generates Hopping Maps using **MapPlotter**, **MapValues**, **tauM**, and **tauETM**.

MapMaker

```
%MapMaker - A GUI

%MapMaker takes user input for various electron transfer parameters,
%and a desired map range, and runs the hopping mapper programs
(MapValues,MapPlotter)

numlines=1;
halt1=0;
halt2=0;

%Acquiring Electron Transfer Parameters
prompt1={'temperature (K)','A-I distance (A)','\beta step 1
(1/A)','\lambda step 1 (eV)','I-B distance (A)','\beta step 2
(1/A)','\lambda step 2 (eV)','A-B distance (A)','\beta single step
(1/A)','\lambda single step (eV)'};
defaults1={'298','8.1','1.1','0.8','12.8','1.1','0.8','19.4','1.1','0.8
'};
options.Interpreter='tex';

ETparams=inputdlg(prompt1,'ET Parameters',numlines,defaults1,options);

%Acquiring the Hopping Map Range
prompt2={'\Delta G(total) min  (eV)','\Delta G(total) max
(eV)','\Delta G(1^s^t step) min  (eV)','\Delta G(1^s^t step) max
(eV)','resolution  (eV)','contour interval (-log(sec))'};
defaults2={'-1.5','0','-0.4','0.3','0.005','0.2'};
options.Interpreter='tex';
mapRange=inputdlg(prompt2,'Map Parameters',numlines,defaults2,options);

%THE LOOP - repeats until the user presses 'cancel'
while halt1~= 1 && halt2~=1

%converting user input values to useable numbers
conv1=char(ETparams);
convETparams=str2num(conv1);
conv2=char(mapRange);
convmapRange=str2num(conv2);

%Making the HoppingMap
[dGT,dG1,taoslog]=MapValues(convETparams,convmapRange);
%generates map values
MapPlotter(dGT,dG1,taoslog,convETparams,convmapRange);
%plots the map
```

```
%Repeating the cycle
ETparams=inputdlg(prompt1,'ET Parameters',numlines,ETparams,options);
halt1=isempty(ETparams);

mapRange=inputdlg(prompt2,'Map Parameters',numlines,mapRange,options);
halt2=isempty(mapRange);

end
```

MapPlotter

```
function MapPlotter(dGT,dG1,taoslog,convETparams,convmapRange)
%Mapper3 Plots the hopping map
%   Detailed explanation goes here

etValues={'','\bfET Parameters','',['\rmT =
',num2str(convETparams(1)),' K'],'',['r (A-I) =
',num2str(convETparams(2)),' A'],['\beta (A-I) =
',num2str(convETparams(3)),' A^-^1'],['\lambda (A-I) =
',num2str(convETparams(4)),' eV'],'',['r (I-B) =
',num2str(convETparams(5)),' A'],['\beta (I-B) =
',num2str(convETparams(6)),' A^-^1'],['\lambda (I-B) =
',num2str(convETparams(7)),' eV'],'',['r (A-B) =
',num2str(convETparams(8)),' A'],['\beta (A-B) =
',num2str(convETparams(9)),' A^-^1'],['\lambda (A-B) =
',num2str(convETparams(10)),' eV']};
contourSpacing=convmapRange(6);

loglines=(1:contourSpacing:15); %defines the position and spacing of
contour lines

scrsz = get(0,'ScreenSize');
figure1=figure('Position',[scrsz(3)/1.8 scrsz(4)/4 scrsz(3)/2.3
scrsz(4)/2]);
annotation(figure1,'textbox',[0.81 0.3 0.187
0.53],'FitBoxToText','on','Interpreter','tex','String',etValues,'Backgr
oundColor',[1,1,1])
axes1=axes('Parent',figure1,'XDir','reverse','Position',[0.13
0.122249388753056 0.68
0.802750611246944]','Layer','top','FontSize',12);
xlim(axes1,[-1.5 0]);
ylim(axes1,[-0.4 0.3]);
box(axes1,'on')
hold(axes1,'all');

contour(dGT,dG1,taoslog,loglines,'LineColor',[0 0
0],'Fill','on','Parent',axes1);
xlabel('\Delta G^0_T_o_t_a_l (eV)','FontSize',14);
ylabel('\Delta G^0_1_s_t _s_t_e_p (eV)','FontSize',14);
colorbar('peer',axes1);
annotation(figure1,'textbox',[0.72 0 0.187
0.1],'FitBoxToText','on','Interpreter','tex','String','\bf-
log(\tau)','Edgecolor','none','Fontsize',11)

end
```

Map Values

```
function [dGT,dG1,taosNEWlog]=MapValues(ETparams,mapRange)
%MapValues - CALCULATES HOPPING MAP VALUES

%General parameters
temp=ETparams(1);

%First step (A->I) parameters
r1=ETparams(2);
beta1=ETparams(3);
lamda1=ETparams(4);

%Second step (I->B) parameters
r3=ETparams(5);
beta2=ETparams(6);
lamda2=ETparams(7);

%Single step (A->B) parameters
rT=ETparams(8);
betaT=ETparams(9);
lamdaT=ETparams(10);

%The range for the x axis (dGT)
dGTmin=mapRange(1);
dGTmax=mapRange(2);

%The range and step for the y axis (dG1)
dG1min=mapRange(3);
dG1max=mapRange(4);

%The map resolution
dGstep=mapRange(5);

%creates vectors for x (dGT) and y (dG1) axes of the Hopping Map
dGT=(dGTmin:dGstep:dGTmax);
dG1=(dG1min:dGstep:dG1max);
lenx=length(dGT);
leny=length(dG1);

%GENERATING THE VALUES OF TAO: THE "Z" AXIS OF THE PLOT

%Creates the matrix taos of dimensions lenx & leny which holds the "z"
values for the Hopping Map, at
%each (dG1,dGT) point
taos=zeros(leny,lenx);      %defines the empty matrix taos of
appropriate size
ycount=1;
while ycount<=leny
    xcount=1;
    dG1now=dG1(ycount);
    while xcount<=lenx
        dGTnow=dGT(xcount);
        %this if statement defines the "sink" region
```

```
        if dG1now < dGTnow
            taoVal=1e100;
        else

taoVal=tauM(dGTnow,dG1now,r1,r3,beta1,beta2,lamda1,lamda2,temp);
        end

        taos(ycount,xcount)=taoVal;
        xcount=xcount+1;
    end
ycount=ycount+1;
end

%Creates a vector taosET with length lenx holding values of taoET at
%each dGT and dG1
taosET=zeros(lenx);     %defines the empty matrix taosET of appropriate
size
tcount=1;
while tcount<=lenx
    dGTnow=dGT(tcount);
    taoETVal=tauETM(dGTnow,rT,betaT,lamdaT,temp);
    taosET(tcount)=taoETVal;
    tcount=tcount+1;
end

%HOPPING ADVANTAGE

%Where taoET/tao > 1, two step hopping is faster than single-step
tunneling.
%The values of a new matrix taoNEW are set to 1 if there is
%no advantage - otherwise the values are transfered, untouched.

taosNEW=zeros(leny,lenx);
rcount=1;
while rcount<=leny
    ccount=1;
    while ccount<=lenx
        taoETC=taosET(ccount);
        taoC=taos(rcount,ccount);
        advantage=taoETC/taoC;
        if advantage<1
            taosNEW(rcount,ccount)=1;      %Only displays hopping areas
        else
            taosNEW(rcount,ccount)=taoC;
        end
        ccount=ccount+1;
    end
    rcount=rcount+1;
end

%Puts tao values on a log scale to make things easier
taosNEWlog=-log10(taosNEW);
```

tauM

```
function [tauVal] = tauM(dGT,dG1,r1,r3,B1,B3,lam1,lam3,T)
%this function

%Values for the constants
h=4.13574*10^-15;   %planck's constant (eV*s)
R=8.61733*10^-5;    %gas constant (eV/K)
Habr02=0.0005323;   %Hab(r0)^2 (eV^2)
r0=3;               %r0 (A)

dG3=dGT-dG1;

eB1=exp(-B1.*(r1-r0));
eB3=exp(-B3.*(r3-r0));

eG1l=exp((-(dG1+lam1)^2)/(4.*lam1.*R.*T));
eG3l=exp((-(dG3+lam3)^2)/(4.*lam3.*R.*T));

eG1=exp(dG1/(R.*T));
eG3=exp(dG3/(R.*T));
eGT=exp(dGT/(R.*T));

c0a=(((4*pi()^3)/((h^2).*lam1.*R.*T))^0.5)*Habr02;
c0b=(((4*pi()^3)/((h^2).*lam3.*R.*T))^0.5)*Habr02;

taunum=c0a*eB1*eG1l*(1+eG1)+c0b*eB3*eG3l*(1+eG3);
tauden=c0a*c0b*(eB1*eG1l*eB3*eG3l*(1+eG3+eGT));

tauVal=taunum/tauden;
```

tauETM

```
function [tauETVal] = tauETM(dGT,rT,BT,lamT,T)
%Returns taoVal - the value of tao, for a given dG1 and dGT

%Values for the constants
h=4.13574*10^-15;   %planck's constant (eV*s)
R=8.61733*10^-5;    %gas constant (eV/K)
Habr02=0.0005323;   %Hab(r0)^2 (eV^2)
r0=3;               %r0 (A)

%values for recurring sets of constants
c0=sqrt((4*pi^3)/((h^2)*lamT*R*T))*Habr02;    %the constants in front
of the exponents in tao

%piece by piece putting together the function taoET(dGT)

%the denominator...
t1=exp(-BT*(rT-r0));
t2=exp(-((dGT+lamT)^2)/(4*lamT*R*T));
t3=exp(-((dGT-lamT)^2)/(4*lamT*R*T));
tauETden=c0*t1*(t2+t3);

%taoET!
tauETVal=1/tauETden;
```

D.8. Ferric/ferrous deconvolution

The ***SpectralDeconvoluter*** script is used to deconvolute a two-component absorption spectrum and determine the percent of each parent component (e.g., ferric and ferrous components during potentiometric titration). Each spectrum must use the same wavelength data (and have the same number of points).

SpectralDeconvoluter

```
%SpectralDeconvoluter

%Ask for data
xInt=input('Enter the wavelength data (must be the same for all
spectra) ');
yInt=input('Enter the intermediate spectrum to deconvolute ');
yFe3=input('Enter the parent FeIII spectrum data ');
yFe2=input('Enter the parent FeII spectrum data ');


%Compare FeIII to data
error1=yInt-yFe3;
error1=error1.^2;
error1_tot=sum(error1);

%Make the midpoint n=0.5, compare to data
n=0.4;
yMid=n.*yFe2+(n-1).*yFe3;

%compare to data
error2=yInt-yMid;
error2=error2.^2;
error2_tot=sum(error2);

nInc=n;
nNew=n;
lastError_tot=error1_tot;
errorTest_tot=error2_tot;
rebound=0;
turn=1;
counter=0;
vals=[];

%iterate to find deconvolution
while nInc > 0.001 && rebound < 20

if errorTest_tot > lastError_tot
    turn=-1*turn;
    nInc=nInc/2;
end
```

```
lastError_tot=errorTest_tot;

nNew=nNew+turn.*nInc;
if nNew < 0
    nNew=0;
    disp('lbounded!')
    rebound=rebound+1;
    turn=-1*turn;
    nInc=nInc/2;
end

if nNew > 1
    nNew=1;
    disp('ubounded!')
    rebound=rebound+1;
    turn=-1*turn;
end

[errorTest1,errorTest_tot,yTestNew]=errorTester(nNew,yFe2,yFe3,yInt);


%optimization visualization
counter=counter+1;
vals(1,counter)=lastError_tot;
vals(2,counter)=nNew;
vals(3,counter)=errorTest_tot;

end

%display optimization
xvals=1:1:counter;


%Output answers
disp(nNew)
figure
hold
%plot final residuals
plot(xInt,errorTest1,'k')
plot(xInt,yInt,'b')
plot(xInt,yTestNew,'r')
hold
figure
plot(xvals,vals(1,:),'r')
figure
plot(xvals,vals(2,:),'b')
figure
plot(xvals,vals(3,:),'g')
```