APPENDIX 1: MATLAB PROGRAMS

When experiments call for the repeated analysis of large quantities of data, it is essential to have reliable scripts and programs on hand to automatically plot and analyze data. The programs and scripts developed by researchers are usually written to be precisely applicable to the exact experiments they are doing, making them an extraordinary resource for future researchers in the group. Unfortunately, these resources are often lost, either because digital copies are not archived or because the programs lack the appropriate annotations and/or explanations to make them useful to future researchers.

In this appendix, a number of MATLAB functions and/or scripts created and/or expanded upon by the author are presented and explained. These files have also been archived on the Gray Group computer in 327 Noyes.

A.1 Fluorimeter data

The MATLAB script 'import_XLS.m' is a script to import and plot files from the Gray Group fluorimeter in BI 018D. After using the fluorimeter's software to export steady-state luminescence data as Excel files, this script will plot selected files in order in a single plot.

```
import_XLS.m
%Program to import .XLS files from the fluorimeter in BI 018D
%Written by Tania V. Darnton on May 20, 2013, borrowing components from
%Oliver Shafaat's import_DX script and James McKone's import_CV script
%
%This function imports multiple fluorimeter data files as a set of 2 column
%vectors labeled "Wavelength" and "Absorbance" and plots them.

Legend = {}; %This needs to stay here or else it doesn't reset the Legend
variable when you run the program again and again, it just appends the new
file names on
%Call getfileinfo first
[filename,pathname,filterindex]=uigetfile('*.*','MultiSelect','on');


%If a single file was selected, the output of filename is stored as char
```

```
%array.  It needs to be converted to a cell in order to ensure output to
%a single format.
%---------------
if isequal(filename,0)
       disp('User pressed cancel');
       return;
end

if ischar(class(filename));
    filename=cellstr(filename);
end
%----------------



 ttl = input('Give your plot a title!  ','s');

for j=1:length(filename) %function will step through each filename
%'char' command changes the pathname and filename entries from cell to
string class
    file = [char(pathname),char(filename(j))];

% Analyze the file to be opened in order to check its extension
    [pathstr, name, ext] = fileparts(file);



% Use STRCMP to compare EXT to '.xls'. If same, STRCMP returns 1.
    if strcmp(ext, '.xls');
        C = xlsread(file);
        Wavelength = C(2:end,1);
        Absorbance = C(2:end,2);
    else error('Error: File is not .xls');
    end;
leg = input('Give the names of your data sets in order  ', 's');
    Legend = strvcat(Legend, leg);
%This will prompt you for the name of the data you just chose
%This appends names of data onto each other for later use in the legend.
% 'Hold all' command causes each plot to be a different color
hold all
plot(Wavelength,Absorbance,'LineWidth',2,'DisplayName',name);
set(gca,'FontSize',14);
xlabel('Wavelength (nm)','FontSize',14);
ylabel('Intensity','FontSize',14);
title(ttl,'FontSize',14);
legend(Legend,'Location', 'Best');
    hold off

end
disp('Plot completed!')
```

A.2 UV-Vis data plotter

The Cary 50 UV-Vis spectrometer in Noyes 327 exports all the data taken in a session in a single Excel file in which the first set of wavelength and absorbance data taken are in columns A and B, the second set are in columns C and D, etc. Given such a file, the script 'import_CSV.m' prompts the user for a vector denoting the columns containing the wavelength data for the desired sets. For example, if the user wished to plot the first and third spectra recorded by the Cary, he or she would input the vector [1 5], to indicate that the desired data sets were to be found in columns A and B and columns E and F (columns C and D would contain the wavelength and absorption data for the undesired second spectrum recorded). The script then plots the data sets and allows for adjustment of the axes and normalization of the intensities.[i]

```
import_CSV.m
```

```
%Program to import .CSV files from the Cary 50 UV-Vis in Noyes
%Written by Tania V. Darnton on June 13, 2013
%
%This function imports a UV-Vis data file as sets of 2 column
%vectors labelled "Wavelength" and "Absorbance" and plots them.
%import_CSV.m

Legend = {}; %This needs to stay here or else it doesn't reset the Legend
variable when you run the program again and again, it just appends the new
file names on
%Call getfileinfo first
[filename,pathname,filterindex]=uigetfile('*.*','MultiSelect','on');


%If a single file was selected, the output of filename is stored as char
%array.  It needs to be converted to a cell in order to ensure output to
%a single format.
%--------------
if isequal(filename,0)
        disp('User pressed cancel.');
        return;
end

if ischar(class(filename));
```

[i] These portions of the code were originally written by Oliver Shafaat in his script 'DXreader.m'

```
    filename=cellstr(filename);
end
%----------------



 ttl = input('Give your plot a title!   ','s');
 %This is not strictly necessary, but I find it helpful as a reminder to
myself because then I know
 %what the length of the vector "wavelengths" should be
 numplots = input('How many data sets do you want to import today?   ');
 wavelengths = input('What columns do your data sets start in? \n Input your
answer as a vector, e.g. [a b c]   ');
 absorbances = wavelengths + 1;
%'char' command changes the pathname and filename entries from cell to
string class
    file = [char(pathname),char(filename)];

% Analyze the file to be opened in order to check its extension
    [pathstr, name, ext] = fileparts(file);

% Open the file.  If file is not opened, FOPEN returns -1.
% Test to make sure file opened correctly.

fid = fopen(file);
if fid==-1
  error('File not found or permission denied.');
end;
% Use STRCMP to compare EXT to '.csv'. If same, STRCMP returns 1.
    if strcmp(ext, '.csv');
        %Snoop the number of commas in one line of the data to see how many
        %columns of data there are
        headertest = textscan(fid, '%s', 1, 'HeaderLines', 20);
        commaindex = strfind(headertest{1},',');
        numcommas = length(commaindex{1});

        %create a string of "%f" such that the number of "%f"s equals the
        %number of data columns there are in the file
        f = repmat('%f ',1,numcommas);

        %must open file again to reset the position of the cursor
        fid2 = fopen(file);
        if fid2==-1
        error('File not found or permission denied.');
        end;

        %Import the entire data file using a comma as the delimiter and
        %skipping the two header lines
        C = textscan(fid2, f, 'Delimiter', ',', 'HeaderLines', 2,
'EmptyValue', 0, 'EndOfLine', '\r\n');

    else error('File is not .csv');
    end;

Normalize = input('Do you want to normalize your data over the selected
wavelength? (y/n)','s');
```

```
for j=1:numplots %Performs these tasks for each data set that you want to
graph

        Wavelength = C{wavelengths(j)};
        Absorbance_temp = C{absorbances(j)};
        if Normalize == 'y'
            minval = min(Absorbance_temp);
            maxval = max(Absorbance_temp);
            norm_abs = (Absorbance_temp-minval)/(maxval-minval);
            Absorbance = norm_abs;
        else Absorbance = Absorbance_temp;
        end

%Prompt for the names of the data to go in the legend
leg = input('Give the names of your data sets in order  ', 's');

%This appends names of data onto each other for later use in the legend.
    Legend = strvcat(Legend, leg);

% 'Hold all' command causes each plot to be a different color
hold all
plot(Wavelength,Absorbance,'LineWidth',2,'DisplayName',name);
set(gca,'FontSize',14);
xlabel('Wavelength (nm)','FontSize',14);
if Normalize == 'y'
    ylabel('Normalized Absorbance','FontSize',14)
else ylabel('Absorbance','FontSize',14);
end
title(ttl,'FontSize',14);
legend(Legend,'Location', 'Best');
    hold off

end

AxisAdj = input('Do you want to look at a specific wavelength range? (y/n)',
's');

while AxisAdj == 'y'
    minlambda = input('What do you want your minimum wavelength to be (in
nm)?');
    maxlambda = input('What do you want your maximum wavelength to be (in
nm)?');
    set(gca, 'Xlim', [minlambda maxlambda]);
    AxisAdj = input('Do you want to change the wavelength range again?
(y/n)', 's');
end

disp('Plot completed!')
```

A.3 Fluorimeter data importer

Given the .XLS fluorimeter file type described in A.1, the function ‘extractxls’ takes user-defined variable names to create matrix variables in the workspace representing the wavelength and intensity data recorded by the fluorimeter. For example, the MATLAB input ‘`[x,y]=extractxls;`’ will open a window for the user to select a fluorimeter data output file and will then create a matrix ‘`x`’ with the data set’s wavelengths and a matrix ‘`y`’ with the accompanying intensity data. This data will then be plotted and the created matrix variables can be saved or called for use in further analyses.

```
extractxls
```

```
function[t,y]=extractxls()
[filename,pathname,filterindex]=uigetfile('*.*','MultiSelect','off');
M = xlsread(filename);
t=M(:,1);
y=M(:,2);
plot(t,y)
```

A.4 UV-Vis data importer

Given the file type described in A.2, the function ‘extractcsv’ takes user-defined variable names and a user-defined data location within the Excel file to create matrix variables in the workspace representing the wavelength and absorbance intensity recorded by the spectrometer. For example, the MATLAB input ‘`[x,y]=extractcsv(1);`’ will open a window for the user to select a UV-Vis data output file and will then create a matrix ‘`x`’ with the wavelength data from Column A of that file and a matrix ‘`y`’ with the accompanying absorbance data from Column B of that file. The created matrix variables can be saved or called for use in further analyses.

```
extractcsv( c )
function [ x,y ] = extractcsv( c )
%extract_csv Summary of this function goes here
%
[filename,pathname,filterindex]=uigetfile('*.csv');


%If a single file was selected, the output of filename is stored as char
%array.  It needs to be converted to a cell in order to ensure output to
%a single format.
%---------------
if isequal(filename,0)
       disp('User pressed cancel or file not found');
       return;
end

if ischar(class(filename));
    filename=cellstr(filename);
end
%----------------

 wavelengths = c;
 absorbances = wavelengths + 1;
%'char' command changes the pathname and filename entries from cell to
string class
    file = [char(pathname),char(filename)];

% Analyze the file to be opened in order to check its extension
    [pathstr, name, ext] = fileparts(file);

% Open the file.  If file is not opened, FOPEN returns -1.
% Test to make sure file opened correctly.

fid = fopen(file);
if fid==-1
  error('File not found or permission denied.');
end;
% Use STRCMP to compare EXT to '.csv'. If same, STRCMP returns 1.
    if strcmp(ext, '.csv');
        %Snoop the number of commas in one line of the data to see how many
        %columns of data there are
        headertest = textscan(fid, '%s', 1, 'HeaderLines', 20);
        commaindex = strfind(headertest{1},',');
        numcommas = length(commaindex{1});

        %create a string of "%f" such that the number of "%f"s equals the
        %number of data columns there are in the file
        f = repmat('%f ',1,numcommas);

        %must open file again to reset the position of the cursor
        fid2 = fopen(file);
        if fid2==-1
        error('File not found or permission denied.');
        end;

        %Import the entire data file using a comma as the delimiter and
        %skipping the two header lines
        C = textscan(fid2, f, 'Delimiter', ',', 'HeaderLines', 2,
'EmptyValue', 0, 'EndOfLine', '\r\n');

    else error('File not .csv');
    end;

        x = C{wavelengths};
        y = C{absorbances};

end
```

A.5 Quantum yield calculations

A.5.1 Fluorimeter intensity integration

Given a set of fluorimetry data in matrix form (as might be output by the code from section A.3), the function 'fluorointeg' calculates and reports the area under the entire plot. For example, the MATLAB input '`a=fluorointeg[x,y];`' will calculate the area under the wavelength vs intensity plot created by `x` and `y` and store that value to the variable `a`.

```
fluorointeg(x,y)
```

```
function [area] = fluorointeg(x,y)
%fluorointeg.m
%Written by Tania Darnton on March 17, 2014
%This is a function that will take a set of fluorimetry data in matrix form
%and calculate the area under the curve.
xprime=x(2:end);
yprime=y(2:end);
yzero=max(yprime,0);
area = trapz(xprime,yzero);
fprintf('\n The area under the curve is %s. \n',area)
end
```

A.5.2 Fluorimeter intensity integration: singlet vs triplet

In the case of Pt(pop-BF$_2$)$^{4-}$, it is often useful to know the relative areas of the singlet and triplet emissions in a set of fluorimetry data. The function 'fluorointegpartial' assumes that the singlet emission has reached zero intensity by the 77th value in the matrix; for a scan of Pt(pop-BF$_2$)$^{4-}$

excited at 355 nm and observed from 375 to 650 nm, this is true. Inputs for ‘fluorointegpartial’ are analogous to those of ‘fluorointeg’.

```
fluorointegpartial(x,y)
```

```
function [area] = fluorointegpartial(x,y)
%fluorointeg.m
%Written by Tania Darnton on March 17, 2014
%This is a function that will take a set of fluorimetry data in matrix form
%and calculate the area under the curve.
xprime=x(2:77);
yprime=y(2:77);
xdoubleprime=x(78:end);
ydoubleprime=y(78:end);
yzero=max(yprime,0);
ydoublezero=max(ydoubleprime,0);
areafluoro = trapz(yzero);
areaphos=trapz(ydoublezero);
ratio = areafluoro/areaphos;
area = areafluoro;
totalarea = areafluoro + areaphos;
fprintf('\n The area under the fluoroescence curve is %s. \n',areafluoro)
fprintf('\n The area under the phosphorescence curve is %s. \n',areaphos)
fprintf('\n The ratio of fluorescence to phosphorescence is %s. \n',ratio)
fprintf('\n The area under the entire curve is %s. \n',totalarea)

end
```

A.5.3 Singlet quantum yield calculation

The script ‘qyanalog’ takes a named set of data from the user and calculates a quantum yield for the singlet state of Pt(pop-BF$_2$)$^{4-}$. As upwards of sixteen individual variables must be called, the user must utilize a methodical nomenclature system for their variables so that all the data will be called correctly by the script. Users should give their data systematic names ending in ‘x’ and ‘y’ as appropriate; fluorescence and absorbance data for the same sample should vary only in the addition of an ‘f’ at the beginning of the variable name. For example, a sample analyzed at 10 ºC could have absorbance data imported as ‘10degx’ and ‘10degy’ while the corresponding fluorimetry data might be called ‘f10degx’ and ‘f10degy’. By setting the parent experimental variable name to '10deg', the script will concatenate tags for 'f', 'x', and 'y' to the parent name and call the appropriate variables. This script also requires the subfunction 'fluorointegpartial_qyanalog', which is presented in section A.5.3.1.

qyanalog.m

```
%qyanalog.m
%Written by Tania Darnton on March 15, 2014 (approximately)
%Last revision on: June 6, 2014 --> changed QY of anthracene to be 0.27
%This script uses the function fluorointeg to determine the
%quantum yield of an unknown solution.
%The final function 'Qx' calculates the quantum yield of an unknown compound
%based on a standard
%   This function is from J. Phys Chem, Crosby & Demas, vol 75, number 8,
%   p.991, April 1971. On p 999, a function for calculating the quantum
%   yield of an optically dilute solution is put forward.
empty = [];
Qr = input('\n What is the quantum yield of your reference solution at the
parameters of interest? If you do not input a value, 0.27 (anthracene) will
be used. \n');
if isequal(Qr,empty)
        Qr = 0.27;
    end
nx = input('\n What is the average refractive index of your REFERENCE
solution to the luminescence? If you do not input a value, the RI for EtOH
will be used.\n');
if isequal(nx,empty)
        nx = 1.3611;
end
nr = input('\n What is the average refractive index of your SAMPLE solution
to the luminescence? If you do not input a value, the RI for MeCN will be
used.\n');
if isequal(nr,empty)
        nr = 1.3442;
end

locrabs = input('\n What is the parent variable name of your reference
absorbance data?\n','s');
locrlum = input('\n What is the parent variable name of your reference
luminescence data?\n','s');
locsabs = input('\n What is the parent variable name of your sample
absorbance data?\n','s');
locslum = input('\n What is the parent variable name of your sample
luminescence data?\n','s');

strsx = strcat(locsabs,'x'); % Concatenate absorbance parent variable name
with tags 'x' and 'y'
strsy = strcat(locsabs,'y');
strrx = strcat(locrabs,'x');
strry = strcat(locrabs,'y');
locx = evalin('base',strsx); % Evaluate the new variables (for some reason
it doesn't work without this step; it will keep having strx be a string
instead of the var). 'Base' is the basic workspace
locy = evalin('base',strsy);
```

```
locrx = evalin('base',strrx);
locry = evalin('base',strry);
strflsx = strcat(locslum,'x');% Concatenate fluorometry parent variable name
with tags 'f', 'x' and 'y'
strflsy = strcat(locslum,'y');
strflrx = strcat(locrlum,'x');
strflry = strcat(locrlum,'y');
flsx = evalin('base',strflsx); % Evaluate the new variables (for some reason
it doesn't work without this step; it will keep having strx be a string
instead of the var). 'Base' is the basic workspace
flsy = evalin('base',strflsy);
flrx = evalin('base',strflrx);
flry = evalin('base',strflry);

wavelength = input('At what wavelength do you wish to know the absorbance?
If you do not input a value, 355 nm will be used.');
    if isequal(wavelength,empty) %Having default wavelength be 355 nm
        wavelength = 355;
    end
    c = find(locx >= 355 & locx <= 356); %Find the index of the desired
wavelength in the sample
  Ax = locy(c); %Find the absorbance at the desired wavelength in the sample
 Dx = fluorointegpartial_qyanalog(flsx,flsy); %Find the area under the
fluorescence curve of the sample



   d = find(locrx >= 355 & locrx <= 356); %Find the index of the desired
wavelength in the reference
  Ar = locry(d); %Find the absorbance at the desired wavelength in the
reference
 Dr = fluorointeg(flrx,flry); %Find the area under the fluorescence curve of
the reference

Qx = Qr*(Ar/Ax)*((nx)^2/(nr)^2)*(Dx/Dr);
fprintf('\n The quantum yield is %.4f. \n The following variables were used:
\n Reference data = %s \n Sample data = %s \n Wavelength of interest = %.3f
nm \n Qr (quantum yield of reference) = %.4f \n Ar (absorbance of reference)
= %.4f \n Ax (absorbance of sample)=%.4f \n nx (refractive index of sample)
= %.4f \n nr (refractive index of reference) = %.4f \n Dx (area under sample
fluoroescence curve) = %s \n Dr (area under reference fluoroescence curve) =
%s \n',Qx,locrabs,locsabs,wavelength,Qr,Ar,Ax,nx,nr,Dx,Dr)
```

A.5.3.1

```
fluorointegpartial_qyanalog(x,y)
function [area] = fluorointegpartial_qyanalog(x,y)
%Written by Tania Darnton on May 1, 2014
%This is a function that will take a set of fluorometry data in matrix
form
%and calculate the area under the fluorescence part of the curve only.
yprime=y(2:77);
yzero=max(yprime,0);
```

```
area = trapz(yzero);

end
```