function [mols,mols_tot,kernel,d_zeroed]=TPRS(filename,varargin)
%filename is the input file
%varargin:{'library file names'},molecule,fragment(s)
%molecule is the # entered in varargin
%example command entry:
%mols,mols_tot=TPRS('131818-11.csv',{"CO2","H2O","H2CO"},3,[28,29]);
%will show the 28 and 29 mass contributions to the 3rd molecule entered,
%H2CO.'

%Library files are text documents located in a folder called 'Library' in
%the same folder as TPRS.m.
%The structure for a library file is as follows:
%File Name: CO2
%File Text:
%ionization cross-section,
%mass, relative intensity
%mass, relative intensity
%For all masses that contribute to the molecule. The units do not matter,
%as they will be normalized.

warning('off','MATLAB:rankDeficientMatrix');
close all
filterwidth=2; %e^-folding width of gaussian filter put on Temperature.
% Helps smooth things out.

t_threshold = 3; % increase in temperature, before which data is discarded

percentile = .2; % baseline signal to this percentile.
%ie percentile=.2 the baseline is set to the bottom 20th
%percentile of data points, so the noise should be around
%zero, not above it.

t_min = 150; %temperature for peak integration start

t_max = 350; %temperature for peak integration end

%correcting for transmission/detection- interpolation from UTI/Hiden
%manuals
delta_raw = [ % column 1 = m/z; column 2 = transmission*detection coefficient
0 2;
25 1;
35 0.9;
45 0.8;
55 0.7;
65 0.65;
75 0.588;
85 0.495;
95 0.4505;
105 0.357;
115 0.294;
125 0.2115;
145 0.084;
200 0.084;];

%load file, define m (masses), t (temperature), and d (signals)
d = dlmread(filename, ',', 1, 2);
%ignore last column if it contains zeros
if d(1,end)==0
    d=d(:,1:end-1);
end

%convert voltage to temperature and subtract off beginning and end of run
t = -76.883+127.71*d(:,1); %input temperature conversion here
% (particular to instrument, but usually a polynomial)
dummy = t(2:end);
tindex1=find((dummy-t(1:end-1)>t_threshold,1);
tindex2=find(t==max(t));
t = t(tindex1:tindex2);
d = d(tindex1:tindex2,2:end); % d is used for the signals

% file reading
fid = fopen(filename);
str = fgetl(fid);
fclose(fid);
colons=find(str=='
');
quotes=find(str=='');

% defining "M", the array of masses
m = zeros(1,size(d,2));
for nn = 2:length(colons)
m(nn-1)=round(str2double(str(colons(nn)+7:quotes(find(quotes>colons(nn),1))-1)));
end

% defining fragment transmission/detection coefficients: must be row vector
delta = interp1(delta_raw(:,1),delta_raw(:,2),m);

% populate kernel with molecule fragment data from 'library' directory
% this has rows "mass i" and columns "molecule j"
% entries in the kernel are fragments fij*sigma, where sigma is ionization
% cross section
sigma=zeros(1,length(varargin{1}));
sigma=zeros(length(m),length(varargin{1}));
for nn = 1:length(varargin{1})
    if exist(['library',varargin{1}{nn},',file'])==2
        fprintf(2,['Error: the file ',varargin{1}{nn},', does not exist\n']);
        return
    end
    dummy = dlmread(['library',varargin{1}{nn},',',0,0);
sigma(nn)=dummy(1,1); % first line of file is the ionization cross section
delta_dummy = interp1(delta_raw(:,1),delta_raw(:,2),dummy(2:end,1));
dummy(2:end,2)=dummy(2:end,2)./delta_dummy; % account for transmission coefficients

delta(2:end,2)=dummy(2:end,2).*sum(dummy(2:end,2)); % divide by total to convert to fractions, so it doesn't matter how you input the data (ie NIST works too)

% populate kernel with relevant masses (ie the ones measured)
for ii = 2:size(dummy,1)
    kernel(m==round(dummy(ii,1)),nn)=dummy(ii,2)*sigma(nn); % round used to ensure
end

% start baseline subtraction
dummy = sort(d,1,'ascend');
dummy2= repmat(dummy(round(length(t)*percentile),:),length(t),1);
d_zeroed = (d-dummy2)./repmat(delta,size(d,1),1); % zeroed traces, divided by transmission/detection coeffs.

% first allocate space:
d3 = zeros(size(d_zeroed));
mols = zeros(length(sigma),length(t));
mols1 = mols; mols2=mols;
d_est = zeros(size(d_zeroed)); % d_est is estimate of matrix reconstruction
%what will be included in figure?
if length(varargin)==2
    include_mols = varargin{2}; %molecules that you want to include in figure
elseif length(varargin)==3
    include_mols = varargin{2};
    include_frags=zeros(size(varargin{3})); %fragments that you want to include in figure
    for ii = 1:length(varargin{3})
        include_frags(ii)=find(m==varargin{3}(ii),1);
    end
    include_frags = sort(include_frags, 'ascend');
else
    include_mols = 1:size(kernel,2); %if no molecule/fragment is specified, include everything in figure
    include_frags = 1:size(kernel,1);
end

%subtract out fragments if a mass is unique for one molecule
dummy=logical(kernel);
dummy2=false(size(dummy,1),1);
for nn = 1:size(dummy,1)
    if sum(dummy(nn,:))>1
        dummy(nn,:)=false(1,size(dummy,2));
    elseif sum(dummy(nn,:))~=0
        dummy2(nn)=true;
    end
end
dummy3=sum(dummy)<=0;
dummy4=logical(dummy3-1);
kernell1=kernel.*dummy; %kernel of unique fragments
kernel2=kernel.*(1-dummy); %residual kernel

%Ideally we will be able to specify how to weight the importance of different masses
%by making minimizing their error more important, but cannot do so at this time.

%SOLVING for amount of molecule at each temperature point
%mols_dummy=mols;
for nn = 1:length(t)
    filter = repmat(exp(-(t-t(nn))./filterwidth).^2,1,size(d,2)); %gaussian filter
    can do without if just use small number, but makes noise less of an issue.
    d3(nn,:) = trapz(t, d_zeroed.*filter,1) ./ trapz(t, filter,1); %creating the filtered version of d_zeroed
    mols1(dummy4,nn)=sparse(kernell1(dummy2, dummy4))\(d3(nn, dummy2))'; %SOLVES for the molecules with unique fragments
    mols_dummy(:,nn)=lsqnonneg(kernel,d3(nn,:))';
    d3(nn,:)=d3(nn,:)-(kernel*mols1(:,nn))'; %subtracts contribution to signal from molecules with unique fragments
    mols2(dummy3,nn)=lsqnonneg(kernel2(:,dummy3),d3(nn,:))';
    mols2(dummy3,nn)=sparse(kernel2(:,dummy3))\(d3(nn,:))'; %SOLVES for the residual mols (not used)
    mols1(:,nn)=mols1(:,nn)+mols2(:,nn); %total molecules
    d_est(nn,:)=(kernel(:,include_mols)*mols(include_mols,nn))'; %estimate of reconstructed signal
end
figure
plot(t,mols', 'linewidth',1)
legend(varargin{1})

figure
plot(t,d_zeroed(:,include_frags), 'linewidth',1);
hold on
plot(t,d_est(:,include_frags), 'linestyle', '--', 'linewidth',1)
str = cell(1, length(include_frags));
for nn = 1:length(include_frags)
    str(nn) = int2str(m(include_frags(nn)));
end
legend(str)

% setting temperature for peak integration
index3 = find(t > t_min, 1) - 1;
index4 = find(t > t_max, 1);

% find total fragment concentration (same as above, but integrated signals)
d_int = trapz(t(index3:index4), d_zeroed(index3:index4,:), 1); % integrated signals

in T, divided by transmission coefficient
mols2 = zeros(size(sigma')); mols1 = mols2;
mols1(dummy4) = sparse(kernel1(dummy2, dummy4)) \ (d_int(dummy2)');
d_dummy = d_int - (kernel * mols1)';
mols2(dummy3) = lsqnonneg(kernel2(:, dummy3), d_dummy');
mols_tot = mols1 + mols2;

% get residual to compare to signal and find error
res = (d_int' - kernel * mols_tot);
figure
bar(m, d_int)
hold on
scatter(m, res, 'r')
grid on
legend('signal', 'Measured-Estimated')