Supporting Information

Iterative Non-Negative Matrix Factorization Filter for Blind Deconvolution in Photon/Ion Counting

Scott R. Griffin†, John A. Biechele-Speziale†, Casey J. Smith†, XimengYou-Dow‡, Julia K. White†, Si-Wei Zhang‡, Julie Novak‡, Zhen Liu‡, Garth J.Simpson∗†

†Purdue University, West Lafayette, IN, USA 47906
‡Merck & Co., Inc., Kenilworth, NJ, USA 07033

E-mail: gsimpson@purdue.edu

Additional Information on the receiver-operator curve (ROC) Plots:

ROC plots were created using simulations with known ground truth outcomes. Simulated data were generated by first creating a vector of photon positions, created by applying a Poisson distribution with a constant λ value to the entire data trace (length of 500,000 data points), in which λ is the mean number of expected events per time point. This vector of photon positions was then convolved with the simulated IRF, given by an exponentially decaying comb with points every 10 data points. The length of the IRF vector was 100 data points. The simulated data were 500,000 data points in length.

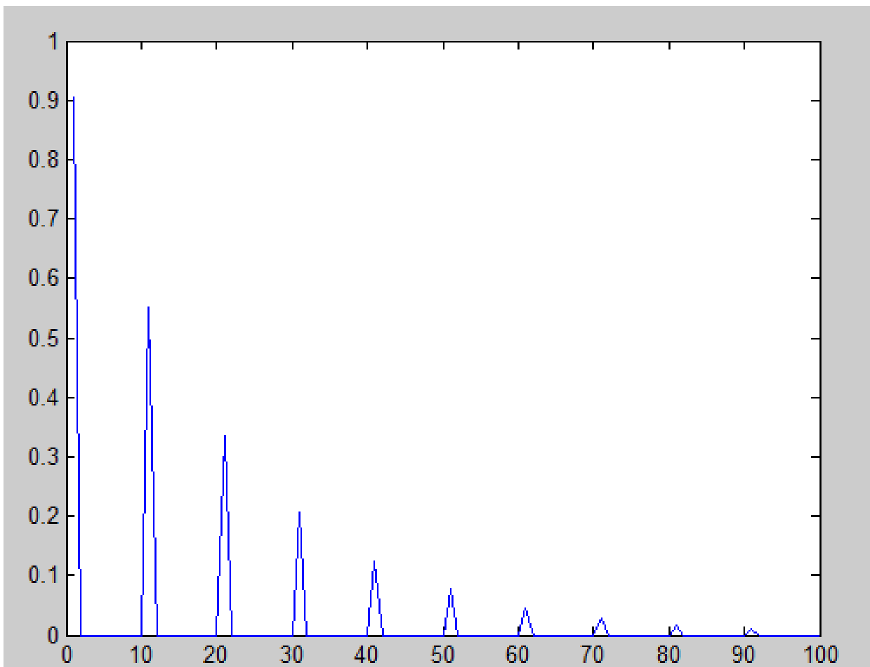

Figure S1: Instrument response function (IRF) used to generate the simulated data.

First simulated data set:

λ = 0.01

True Number of photon events: 4972

Threshold was varied from 0.1 to 1 with a step size of 0.003

A.

B.

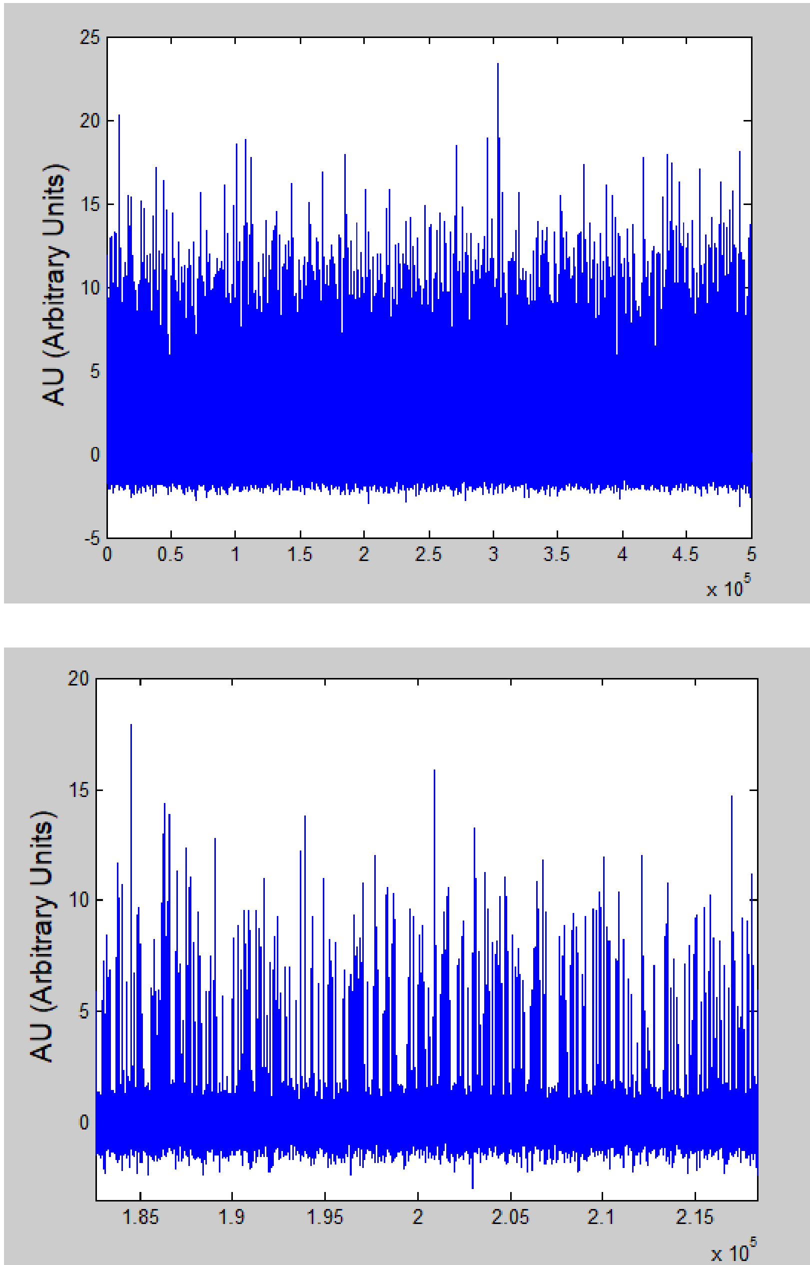

Figure S2: A) Entire simulated data trace with λ = 0.01. B) Zoomed in section to show detail.

Second simulated data set:

$\lambda = 0.1$

True Number of photon events: 47361

Threshold was varied from 0.1 to 1 with a step size of 0.001

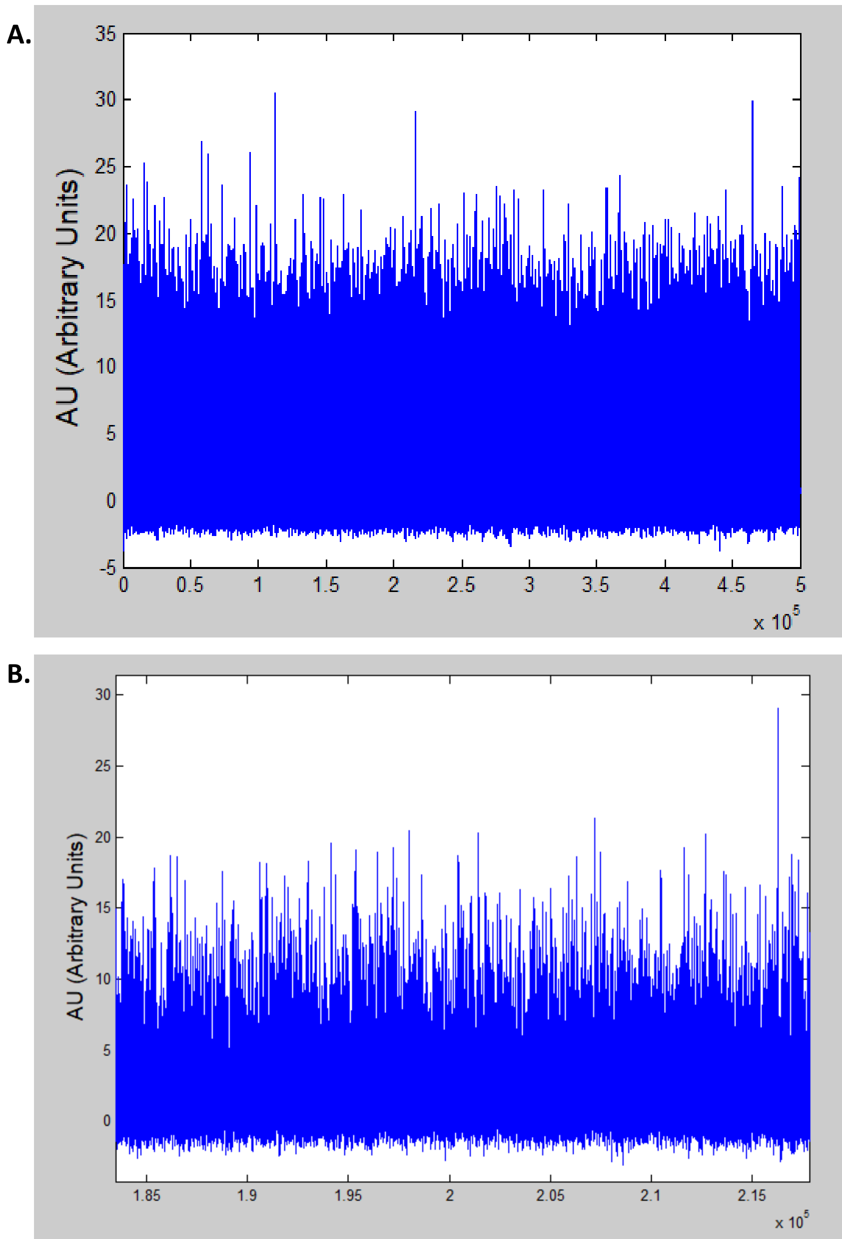

Figure S3: A) Entire simulated data trace with $\lambda = 0.1$. B) Zoomed in section to show detail.

MATLAB code:

```
tic
clear all
% close all

PerformAutocorrelation = 0;
Reducing = 0;
datatype = 1; %0 for .bin and 1 for xls
plot_irf = 0;
plot_events = 0;
%% Load data
if datatype == 0
    Filename = '\\10.164.16.153\Data\Scott
Griffin\deconvolution\DoxycyclineHyclate\08-Aug-
2017_16_12_28Channel_B_Raw_Data.bin';%'\\10.164.16.153\Data\Scott
Griffin\deconvolution\DoxycyclineHyclate\08-Aug-
2017_16_12_28Channel_B_Raw_Data.bin';
    fid = fopen(Filename);
    filter = 100;
    Range = 200; %input range for alazar card
    Buffers = 35; %amount of concatenated hits
    corrected_split_data = zeros(2450000,Buffers);
    reduction = 1;
    Data = fread(fid,'uint16');%dlmread(Filename,'\t');
    %% Convert into voltage and make positive
    Rescaled_Data = ((Data / 2^15) * -Range) + Range;%converts to voltage,
flips to positive for ease of visualization, and shifts baseline towards 0
    split_data = reshape(Rescaled_Data,[],Buffers); %splits up the data into
the individual impact events
    for i = 1:Buffers;
     baseline = mean(split_data(1:40000,i)); %finds the mean of the first
millisecond of data in each trace
     corrected_split_data(:,i) = (split_data(:,i)-baseline); %subtracts the
baseline mean from the corresponding trace and saves them
    end
end

if datatype == 1
    Filename = '\\10.164.16.153\Data\Scott  Griffin\deconvolution\ROC
plot\TimeTraces&GroundTruth\TimeTrace9.csv';
    Rescaled_Data = csvread(Filename,0,0);
    Filename2 = '\\10.164.16.153\Data\Scott  Griffin\deconvolution\ROC
plot\true_irf.csv';
    GroundTruthData = csvread(Filename2,0,0);
    filter = 100;
    reduction = 1;
    Buffers = 1;
    corrected_split_data = Rescaled_Data;
end
IRFs = xlsread('\\10.164.16.153\Data\Scott
Griffin\deconvolution\Gaus_IRF.xlsx');
%% This section cuts out the important part of the data for the smashing
event
for i = 1:Buffers
```

```
    maximums(:,i) = max(corrected_split_data(:,i)); % finds max values for
each hit
end

for i = 1:Buffers  % This loop finds the beginning and end of each event
    temp3 = find(corrected_split_data(:,i) == maximums(i)); %pulls out the
correct max value for the impact event
    if size(temp3) > 1
        value1(i) = 0;
        value2(i) = 0;
    elseif size(temp3) == 1
        value1(i) = find(corrected_split_data(:,i) == maximums(i));
        value2(i) = find(corrected_split_data(:,i) == maximums(i));
        while corrected_split_data(value1(i),i) > 0
            if corrected_split_data(value1(i),i) > 0
                value1(i) = value1(i) - 1; % stored values for the beginning
of events
            end
        end
        while corrected_split_data(value2(i),i) > 0
            if corrected_split_data(value2(i),i) > 0
                value2(i) = value2(i) + 1; % stored values for the end of
events
            end
        end
    end
end
recovered_events = cell(1,Buffers);
recovered_irf = cell(1,Buffers);
for n = 1:Buffers
    if value1(n) == 0
        n = n + 1;
    end
    Reduced_Data = corrected_split_data;
%% Create baseline correcting high-pass filter and apply it
    x_hp = linspace(-3,3);
    pdf = normpdf(x_hp,0,1);
    pdf_normalized = -pdf/sum(pdf);
    impulse = zeros(1,100);
    for i = 1:length(x_hp)
        if i == 50
            impulse(1,i) = abs(sum(pdf_normalized));
        else
            impulse(1,i) = 0;
        end
    end
    final_filter = pdf_normalized + impulse;
    Reduced_Data = conv(final_filter,Reduced_Data); %applies the filter
    Reduced_Data = Reduced_Data(50:length(Reduced_Data)-50); %gets rid of the
extra points added by the convolution
    %% Fit
    filtersize = filter/reduction; %size of data transient
    irf_results = zeros(filtersize,1); %preallocate memory for speed
    delta = 0;
    how_many = 1; %how many different MuGuess values you want to use for the
IRF Guess. Not really needed.
    x = linspace(.1,5,filtersize);
```

```
    data_results = zeros(length(Reduced_Data),how_many);%preallocate memory
for speed
    MuGuess = 1.0; %where the guess starts

    for k = 1:how_many
        MuGuess = MuGuess + delta; %original guess value for the eponential
pdf
        comb =
[1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0
,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0]';
        decay_guess = comb .* exppdf(x,MuGuess)'; %original guess pdf
        keepgoing = 1; %the exit condition for the program
        condition1 = 1; %condition for X2(kinda) to be getting smaller
        condition2 = 1; %condition for the amount of recovered photon events
to be getting smaller
        counter = 0;%for how many times it needs to iterate between
recovering the irf and amplitudes
        counter2 = 0;
        while keepgoing == 1
        %% making the orignial P matrix
            data = Reduced_Data;
            L = filtersize;
            shift = 0;
            P_init = zeros(filtersize,filtersize); %change filtersize to L to
make it like original
            for c = 1:L
                for r = 1:length(decay_guess)
                    if r+shift<L+1
                        P_init(r+shift,c) = decay_guess(r);
                    else
                        P_init(r+shift-L,c) = 0;
                    end
                end
                shift= shift+1;
            end
            P = P_init;
            Threshold = .8; %threshold for "non-negativity" or noise
            data_fit = zeros(length(data),1);
            %% Fit - recovering photon arrival times and amplitudes
            startpoint = 1;
            endpoint = filtersize;
            while startpoint <= length(data)
                if endpoint > length(data)
                    endpoint = length(data);
                end
                data_prime = data(startpoint:endpoint,1); %subset of the data
                P_prime = P(1:length(data_prime),1:length(data_prime));
%subset of the E matrix
                More = 1;
                while More == 1
                    C_prime = P_prime\data_prime; %finding the
'concentrations'
                    Index = find(C_prime>Threshold); %keep above threshold
values
                    NegIndex = find(C_prime<Threshold); %discard below the
threshold
```

```
                    if isempty(NegIndex)
                        More = 0;
                    else
                        P_prime = P_prime(:,Index);
                    end
                end
                conc_prime = P_prime\data_prime;
        %% Puts the photon amplitudes together with the arrival times
                Ampl_prime = zeros(filtersize,1);
                for j = 1:size(P_prime,2)
                    index = find(P_prime(:,j)==decay_guess(1));
                        Ampl_prime(index)= conc_prime(j);
                end
                if isempty(P_prime)
                else
                    data(startpoint:endpoint) = data(startpoint:endpoint) -
(P_prime(:,1)*Ampl_prime(1));
                end
                data_fit(startpoint,1) = Ampl_prime(1); %creates the final
array of recovered photon amplitudes and arrival times
                new_start = find(Ampl_prime ~= 0);
                new_start = new_start(new_start(:) >= 2);
                if isempty(new_start) == 1 || new_start(end) <= 1
                    startpoint = startpoint + filtersize;
                    endpoint = endpoint + filtersize;
                else
                    startpoint = new_start(1) + startpoint - 1 ;
                    endpoint = endpoint + new_start(1) - 1;
                end
            end

            I = eye(filtersize,filtersize); %identity matrix for calculating
the weighted convolution matrix
            M = conv2(data_fit,I); % new 'P' matrix for recovering the IRF
            M = M(1:length(M)-(length(I)-1),:); %needs to be cut down to be
the proper length
            %% calculates the irf and decides if the fit is good enough
            irf = M\Reduced_Data;
            irf = irf/sum(irf);
            if counter >= 1
                Past_Diff = Diff;
                Past_nonzero = nonzero;
            end
            Diff = norm(irf - decay_guess); %Euclidian distance between the
recovered IRF and the guess used for the convergence condition
            nonzero = length(find(data_fit>Threshold)); %count how many
photon events were recovered
            if counter >= 1
                comparison_Diff = Diff - Past_Diff;
                if comparison_Diff < 0
                    condition1 = 1;
                else
                    condition1 = 0;
                end
            end
            if condition1 == 1
                decay_guess = irf;
```

```
            else
                keepgoing = 0;
                final_Diff = Past_Diff;
                final_nonzero = Past_nonzero;
                [irf2,final_nonzero2,data_fit_final] =
NNMF_IRF_testing_part2(irf,Reduced_Data); %Sends the results to a final
iteration of the algorithm
                %% 'annealing' step where a non-converging result has noise
added to help it converge by moving it away from the local minima
%                   if final_nonzero2 < 1500 || sum(isnan(irf2)) > 0
%                       if sum(isnan(irf2)) > 0
%                           decay_guess = IRFs(k,:)'/5 + normrnd(0,1,100,1);
%                           keepgoing = 1;
%                       else
%                           keepgoing = 1;
%                           decay_guess = irf2/5 + normrnd(0,1,100,1);
%                       end
%                   else
%                       keepgoing = 0;
%                   end
            end
            counter = counter + 1
        end
        results(:,k) = vertcat(MuGuess,final_Diff,final_nonzero2);
        data_results(:,k) = data_fit_final;
        irf_results(:,k) = irf2;
    end
    minimums = min(results,[],2);
    location = find(results(2,:) == minimums(2));
    recovered_events{:,n} = data_results(:,location);
    recovered_irf{:,n} = irf_results(:,location);
end

%% autocorrelation or recovered data
if PerformAutocorrelation == 1
    FT1=fft(data_results(:,location)); %FT along vib mirror time axis
    FTFT1=gmultiply(FT1,conj(FT1)); %FTFT* = square mag FT
    clear FT1;
    AC1=real(ifft(FTFT1)); %iFT(FTFT*)=AC
    clear FTFT1;
    AC_deconvolved = AC1(1:floor(length(data)/2));

    figure3 = figure;
    plot(AC_deconvolved,'LineWidth',2);
    xlabel({'Data'},'FontSize',14);
    ylabel({'Autocorrelogram'},'FontSize',14);
    title({'Autocorrelogram of deconvolved data'},'FontSize',14);
end
toc
```