

A tutorial and pseudo-code on how to process SFDI data

Last update: 24.06.19

I. Introduction

This tutorial is designed to guide you through the basic stages of processing SFDI data, providing some pseudo-code examples in Matlab as to how these stages may be executed. SFDI may be implemented in many different ways, each having their own specific contingencies, caveats and/or additional data processing requirements. This tutorial will attempt to cover the most general case for SFDI systems: Images collected sequentially over multiple wavelengths at multiple spatial frequencies. Here, we only consider data sets where each spatial frequency is represented by a sinusoidal intensity pattern along one spatial dimension and projected at three evenly spaced phases (0, 120 and 240 degrees).

The data processing steps outlined here largely follow what is described in Cuccia, et.al JBO 2009 ("Quantitation and mapping of tissue optical properties using modulated imaging"). The steps and functions used in the pseudo code sections are not optimized in any way, but rather written in a way to illustrate the process in a simple, explicit way. While these functions will run correctly, it is strongly encouraged that you modify these codes to run it more efficiently.

This tutorial will continue to evolve and add additional topics and methods in the near future. For now, only the diffusion approximation model for light transport will be introduced, however a tutorial on how to implement a white Monte Carlo model for solving forward and inverse problem in the spatial frequency domain is in the works and hope to be added here in September.

If there are any questions regarding this tutorial or issues with attempting to implement the pseudo code functions, please feel free to contact me: rolf.saager@liu.se

II. Demodulation

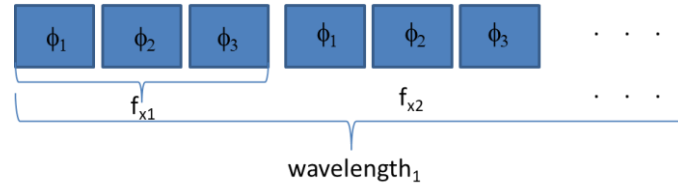
a. Description

SFDI is all about measuring how patterns of light decay and attenuate after interacting with turbid media like tissue. As the spatial frequency content of a pattern will interact differently between absorption and scattering, the shape of this decay as the spatial frequency increases will be unique for absorption and scattering values typically encountered in tissue. When patterns of sinusoidal light intensity are considered, this decay is represented by the AC amplitude of the sinusoid measured from tissue at multiple spatial frequencies.

There have been many approaches in SFDI systems to extract this AC using single snapshot, multi-frequency synthesis and others (most of these can be found in the reference literature list under the SFDI 101 section), but this tutorial will focus on the "classical" 3-phase demodulation approach.

b. Matlab pseudo code:

To start, let's assume that your data is stored as a series of images in the following hierarchy 1) Wavelength 2) Spatial frequency and 3) phase, namely what is shown below:



The approach we have taken is to set up a 3 level loop (loop over wavelengths, frequencies and phases), while this is not computationally efficient; it is simple and a bit easier to visualize:

```
for k=1:length(w) %loop over wavelengths (user defined or read in from meas. param file)
    for i=1:length(f) %loop over spatial frequencies (user defined or read in)
        for j=1:3 %loop over phases (user defined)
            % In this loop, we load the data, 3 images at a time (namely
            % for a given wavelength and frequency, we load the 0, 120, 240
            % phase images). we use a temp variable as these 3 phases will
            % be demodulated to extract the AC images one at a time. Key to
            % running this automatically is to have a consistent naming
            % scheme to identify wavelength, frequency and phase. For our
            % group we had a simple numbering scheme and used the loop
            % index to id the wv, freq, and phase (for example the file:
            % image_2_3_1.png would be the image from the 2nd wavelength at
            % the 3 spatial frequency with a phase of 0 degrees). PathName
            % is the folder location where the data is stored.
            % [FileName,PathName] = uigetfile is a popup window function
            % that can allow you to click your way to the correct folder,
            % rather than writing it all out manually.
            temp(:, :, j) = ...
                imread([PathName 'image_' num2str(k) '_' num2str(i) '_' num2str(j) '.png']);

            end
            %3-phase demodulation scheme: this implements eqn 20 from cuccia.
            %Note that this is now a 4-D array where the first 2 dimensions are
            %the x and y pixels, the 3rd is the wavelength and the 4th spatial
            %frequency. We also divide each demodulated image by a variable
            %int_time(k). This is the exposure time used at each wavelength.
            %So in essence, the resulting demodulated image is an intensity per
            %sec and hence normalized the detected signals across all
            %wavelengths even though the integration times might vary
            AC(:, :, k, i) = sqrt(2)/3*sqrt( (temp(:, :, 1)-temp(:, :, 2)).^2 + ...
                (temp(:, :, 2)-temp(:, :, 3)).^2 + ...
                (temp(:, :, 3)-temp(:, :, 1)).^2 ) ./ int_time(k);

        end
    end
end
```

c. Debugging and Diagnostics

Although this is the first and most basic of processing steps for SFDI, there are a number of potential measurement issues that can be identified at this early stage. Given how computationally intensive the full data processing procedure is, it is often very useful to be able to identify and data integrity issues and also be able to determine where these issues may come from.

Successful demodulation should result in raw images of the tissue without any indication that a series of sinusoidal patterns were used to illuminate it. Residual modulation artifacts can remain in any of the images for a number of reasons. Three of the most common are

1. Changes in the ambient (room) light during the acquisition sequence
 - i. While one distinct advantage of using the 3-phase demodulation approach is that it allows SFDI to be acquired under normal lighting conditions, so long as those room lights are constant over the 3 phases (acting as a constant DC offset). If the lighting

condition changes (additional light turned on or off, someone or something moves over the measurement area, casting a shadow)

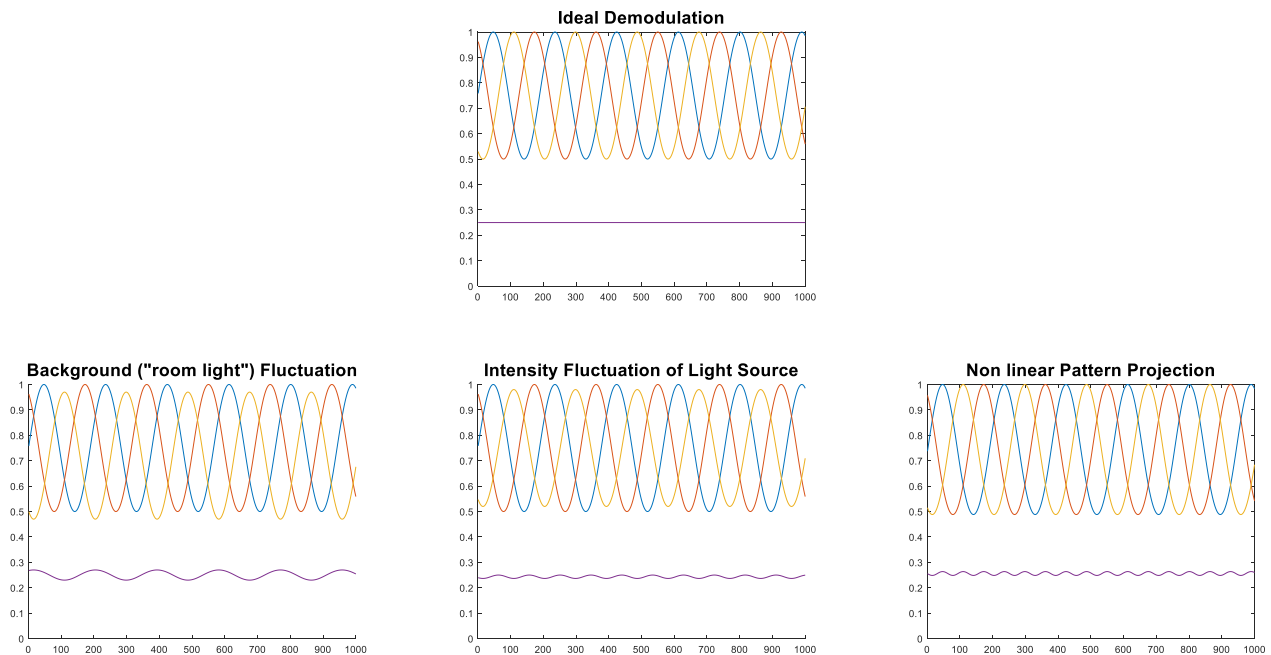
2. Fluctuations in the light source intensity

- i. As the 3 phases are imaged sequentially, it is critical for the output of that source be stable over that time. This typically can arise with pulsed sources where shot to shot variations of the source could occur or the synchronization between illumination and camera become unstable.

3. Nonlinear response to the projector output intensity

- i. This is a common issue with repurposed commercial projectors or running an SFDI system in video mode. Commercial projectors and video rendering are often optimized for the non-linear response of our vision and hence tend to apply a Gamma correction to images/videos input to it. This results in a distorted intensity pattern being projected. In this case, the patterns are no longer sinusoid and hence do not demodulate properly.

While there are many potential pitfalls to acquiring a clean SFDI dataset, each one of the issues identified above will not only show residual modulation, each one is distinct in the frequency of the modulation relative to the spatial frequency of the source (see the simple rendering below)



III. Calibration

a. Description

The motivation behind the calibration step is to convert the raw data collected by the instrument into "absolute" units, namely isolate the spectral characteristics of the tissue from any spectral characteristics of the instrument itself. Since SFDI deals with turbid media and images multiple patterns on to tissue, it is best to calibrate the data at every frequency independently and use a reference standard that exhibits both known absorption and scattering.

The calibration calculation is simple enough:

$$\text{calibrated_reflectance} = (\text{tissue measurement} / \text{reference_measurement}) * \text{"true reference reflectance_model"}$$

the rationale behind this is that every measurement you make is assumed to be the product of your sample reflectance and your instrument response (i.e. $\text{tissue_measurement} = \text{true tissue reflectance} * \text{instrument_response}$ and $\text{reference_measurement} = \text{true reference reflectance} * \text{instrument_response}$) since both tissue and reference measurements used the same spatial frequencies and measured near the same time relative to each other, it is assumed that the instrument response is the same and hence cancel each other out leaving you with:

$\text{calibrated reflectance} = (\text{true tissue reflectance} / \text{true reference reflectance}) * \text{true reference reflectance_model}$

Here, we can model what the specific spatial frequency dependent reflectance, $R(f_x)$, of the reference standard if we know the absorption (μ_a) and scattering (μ_{sp}) properties, index of refraction (n) (typically 1.33 for intralipid phantoms, 1.43 for Silicone (PDMS) phantoms) and the spatial frequencies (f_x) used in the measurement. This can be executed through different models and methods, but initially we will start with using the diffusion approximation model for light transport in tissue. While this is an approximation, it is reasonable to use in SFDI if you are 1) in the NIR, 2) Use spatial frequencies 0-0.25/mm and 3) measuring homogenous tissue.

So long as we know the absorption, scattering and index of refraction of the reference standard in advance, we can plug these values into the model will then generate the appropriate reflectance values for the reference standard at each spatial frequency. The resulting output is the “true reference model” which in the calibration formula then cancels out the “true reference reflectance” leaving you with just the “true tissue reflectance”.

b. Matlab pseudo code:

...for convenience, I would recommend defining the diffusion model as a function (since it will be used in the optical properties fitting routine as well):

```
function reflectance_fx= diff_model_for_SFDI(op,n, fx)
%diffusion model for light transport in tissue (taken from Cuccia 2009)
% op -- is a 2-element array containing [ $\mu_a$   $\mu_{sp}$ ] (in units of  $\text{mm}^{-1}$ )
%       $\mu_a$  -- the input absorption values at every wavelength (array)
%       $\mu_{sp}$  -- the input reduced scattering values at every wavelength
% n -- the index of refraction of the sample measured
% fx -- the spatial frequencies used in the measurement This function will
%      then generate a matrix reflectance_fx(wv,fx) that represents the "true"
%      reflectances at each wavelength and spatial frequency

 $\mu_a = \text{op}(1)$ ;
 $\mu_{sp} = \text{op}(2)$ ;
 $\mu_{tr} = \mu_a + \mu_{sp}$ ; % reduced attenuation coefficient (a.k.a. "mu transport")
 $\text{Reff} = -1.440 ./ n.^2 + 0.710 ./ n + 0.668 + 0.0636 .* n$ ; %surface loss (assumes air interface) eqn(8)
 $A = (1 - \text{Reff}) ./ (2 .* (1 + \text{Reff}))$ ; %eqn(8)

for w = 1:length( $\mu_a$ ) %loop over wavelengths
    for f= 1:length(fx) %loop over spatial frequencies
         $\mu_{eff\_prime} = (3 * \mu_a(w) * \mu_{tr}(w) + (2 * \pi * f_x(f))^2)^{0.5}$ ; % reduced "mu effective"
                                                % eqn(5)

        reflectance_fx(w,f) = ...
             $3 * A(w) .* (\mu_{sp}(w) / \mu_{tr}(w)) / (\mu_{eff\_prime} / \mu_{tr}(w) + 1) / (\mu_{eff\_prime} / \mu_{tr}(w) + 3 * A(w))$ ;
            % derived from eqn(10)
    end
end
```

then for the calibration, all you need as inputs are the demodulated tissue (AC_tissue) and reference data (AC_ref), spatial frequencies used (fx), and the absorption (mua), scattering (musp) index of refraction (n) for the reference

with all those in place:

```
%calculate reflectance values of reference from known absorption and scattering values
reference_model= diff_model_for_SFDI(mua, musp,n, fx);

%correct for instrument response and reference reflectance to isolate tissue reflectance
for i=1:length(mua) %loop over wavelengths
    for k=1:length(fx) %loop over spatial frequencies

calibrated_reflectance(:,:,i,k)=AC_tissue(:,:,i,k)./AC_ref(:,:,i,k).*reference_model(i,k);
    end
end
```

c. Debugging and Diagnostics

Coming Soon!

IV. Inverse Solver

a. Description

In this stage, we are not looking to essentially "match" the measured, calibrated reflectance (as a function of spatial frequency) curve to a model of light transport in tissue. In your case, this is the same diffusion approximation model we used in the calibration step. The key differences here is that (1) this model will now be used to mimic the tissue response, where the model was used to mimic the calibration phantom in the previous step and (2) this model will be used in an "inverse solver" configuration rather than the "forward" model case used in the calibration.

A note on terminology: forward model is deterministic-->which known optical properties (mua, musp, n, etc) the reflectance can be determined, in the inverse case, one tries to use measured reflectance values to infer the optical properties. This is something that in many cases cannot be solved directly, so instead a minimization strategy is deployed. Here

1. Optical properties are guessed and inserted into the forward model
2. Reflectance curves are generated and compared to measured data
3. The error (residual) between these two curves are calculated
 - a. if this error is above some previously specified threshold, new guesses are made and steps 1 and 2 are repeated
 - b. if this error is below some threshold, the guess values are declared close enough and the minimization loop ends, outputting the final set of guesses as the inferred optical properties

This minimization occurs at every pixel and at every wavelength resulting in images of mua and musp for each LED.

b. Matlab pseudo code:

First we need to define another function that sets up the minimization metric for the inverse solver. There are many approaches to doing this, but our group has historically used this as a general function:

```
function y = function_minus_exp(input_parameters,function_to_evaluate,values,varargin)
%This function establishes the minimization metric for comparing model and measured values
%why this particular function is unknown, it has been used for historical reasons
%original author unknown
%input_parameters    -- [mua musp] (a.k.a "op" used in the model function)
%function_to_evaluate -- model function (e.g. diff_model_for_SFDI)
%values              -- measured calibrated data to compare to model
%varargin             -- additional parameters needed for model --> n, fx

temp = (function_to_evaluate(input_parameters,varargin{:})-values).^2;
y = sum(temp(:));
```

Now back to the inverse solver:

```
%simple inverse solver for diffusion model
%code is not optimized for speed, but to step through data one wavelength/pixel at a time
% wv          -- the wavelength values of the data
% n           -- the index of refraction of the sample measured
% fx          -- the spatial frequencies used in the measurement
% The function outputs a 4-D array that contains images of absorption and scattering
% at each individual wavelength: op_fit_maps(y_pixels,x_pixels,wavelengths,[abs scatt])

%initial guesses to seed the minimization process.
%currently blind guesses, but more intelligent guesses can speed up processing time
mua_guess(w) = 0.02; %units of mm^-1
musp_guess(w) = 1.0; %units of mm^-1

for w = 1:length(wv) %loop over wavelengths
for i=1:length(calibrated_reflectance(:,1,1,1)) %loop over x dimension
for j=1:length(calibrated_reflectance(1,:,1,1)) %loop over y dimension
op_fit(i,j,w,:) = fminsearch(...
    @function_minus_exp, ... %minimization metric
    [mua_guess(w) musp_guess(w)], ... % initial guess for input into model
    optimset('TolFun',0.00001,'TolX',0.00001),... % options for minimization
    @diff_model_for_SFDI, ... % model to use to solve inverse
    squeeze(calibrated_reflectance(i,j,w,:)),... % measured values
    n, fx); % remaining parameters for the model
end
end
end
```

c. Debugging and Diagnostics

Coming soon!

V. **Alternative solution methods (e.g. Look Up table methods)**

Coming in September!