Biophotonics

Project: Reconstruction of the Absorption Coefficient Profile

Wanying Li

May 12, 2017

Contents

Abstract

Diffuse Optical Tomography (DOT) is an non-invasive optical imaging technique that measures the optical properties of physiological tissue using light in the near infrared spectrum. Optical properties are extracted from the measurement using reconstruction algorithm. This project examines the steepest descent method for reconstruction of absorption coefficient profile from a set of synthetic data.

1 Introduction

DOT is an important tool in in vivo optical imaging because it can provide quantitative information about light absorption/scattering, hemoglobin concentration and blood oxygenation. The technique is based on diffuse light that penetrates tissue at multiple source projections to create spatial maps of optical properties. Light fibers are used as laser source and detectors which are arranged in arrays around the object.

The technique requires the object of interest being partially or fully light-transmitting or translucent. Therefore, DOT works the best on soft tissues such as breast and brain tissue. Some of its applications include breast cancer imaging and monitoring the effect of cancer treatments, as well as measuring changes in blood oxygenation caused by neural activity.

Based on the detector measurements, tomographic images and spatial maps of tissue's optical properties such as light absorption/scattering and diffusion coefficient can be obtained using model-based reconstruction algorithms.

This project synthesizes measurement data by solving the 2-dimensional finite volume forward model with a set of known optical properties. This projects examines one of the reconstruction algorithms – the steepest descent method in combination with inexact line search – in the task of reconstructing the absorption profile.

2 Methods

2.1 Setup

The object/medium of interest is a 6cm-by-6cm 2D square. Inside, there is a small square centered at position (4cm, 4cm) of size 1cm-by-1cm. The optical properties are $\mu_a = 0.05cm^{-1}$ and $\mu_s = 100cm^{-1}$ for the background medium and $\mu_a = 0.2cm^{-1}$ and $\mu_s = 100cm^{-1}$ for the object. The anisotropic factor is constant over the entire medium, given by $g = 0.95$, and the refractive indices are $n_{medium} = 1.414$ and $n_{air} = 1$.

Following the setup, the orignial μ_a profile is shown in Figure 1. In this project, the following assumptions were made so there was only 1 unknown optical property – absorption coefficient μ_a – needed be extracted

1. the diffusion coefficient D is 1/15 cm and it is constant over the medium;
2. the scattering coefficient μ_s is constant over the medium.

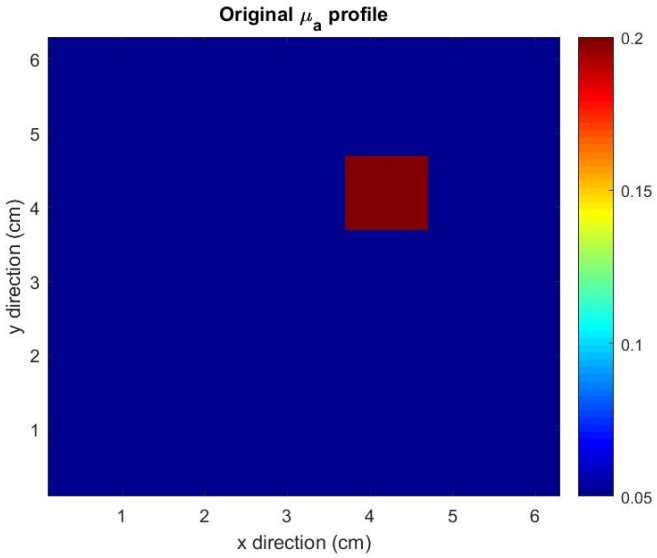

Figure 1. The original μ_a profile.

2.2 Generating the Synthetic Measurements

The first step in generating the measurement data was to obtain the finite volume solutions of the 2D time-independent diffusion equation. This step is referred to as solving the forward model. The 2D time-independent diffusion equation is

$$D\frac{\partial^2 U}{\partial x^2} + D\frac{\partial^2 U}{\partial y^2} - \mu_a U + S = 0 \tag{1}$$

where D is the diffusion coefficient, U is the fluence and S represents the source. Below shows the discretized equations of the 2D diffusion equation

$$\begin{cases} \text{internal: } a_{P_I}U_P - \alpha U_W - \alpha U_E - \alpha U_S - \alpha U_N = 0 \\ \text{west (source): } a_{P_{W_0}}U_P - \alpha U_E - \beta U_S - \beta U_N = b \\ \text{west: } a_{P_W}U_P - \alpha U_E - \beta U_S - \beta U_N = 0 \\ \text{east: } a_{P_E}U_P - \alpha U_W - \beta U_S - \beta U_N = 0 \\ \text{north: } a_{P_N}U_P - \beta U_W - \beta U_E - \alpha U_S = 0 \\ \text{south: } a_{P_S}U_P - \beta U_W - \beta U_E - \alpha U_N = 0 \\ \text{NW: } a_{P_{NW}}U_P - \beta U_E - \beta U_S = 0 \\ \text{NE: } a_{P_{NE}}U_P - \beta U_W - \beta U_S = 0 \\ \text{SW: } a_{P_{SW}}U_P - \beta U_E - \beta U_N = 0 \\ \text{SE: } a_{P_{SE}}U_P - \beta U_W - \beta U_N = 0 \end{cases}$$

where

$$b = \frac{\Delta y}{2A}$$

$$\alpha = D$$
$$\beta = \frac{D}{2}$$
$$a_{P_I} = 4D + \mu_a \Delta x \Delta y$$
$$a_{P_W} = a_{P_{W_0}} = a_{P_E} = a_{P_N} = a_{P_S} = 2D + b + \mu_a \frac{\Delta x \Delta y}{2}$$
$$a_{P_{NW}} = a_{P_{NE}} = a_{P_{SW}} = a_{P_{SE}} = D + \frac{\Delta x + \Delta y}{4A} + \mu_a \frac{\Delta x \Delta y}{4}$$

The discretization equations can be written into a matrix vector equation

$$\boldsymbol{AU} = \boldsymbol{b}. \tag{2}$$

Let N be the number of nodes in the x or y direction, then $\boldsymbol{M}$ is a size N^2-by-N^2 matrix

$$\begin{bmatrix}
a_{P_{NW}} & -\beta & 0 & \cdots & 0 & -\beta & 0 & \cdots & & & & & \cdots & 0 \\
-\beta & a_{P_N} & -\beta & 0 & \cdots & 0 & -\alpha & 0 & \cdots & & & & \cdots & 0 \\
0 & \ddots & \ddots & \ddots & 0 & \cdots & 0 & \ddots & 0 & \cdots & & & \cdots & 0 \\
0 & 0 & -\beta & a_{P_N} & -\beta & 0 & \cdots & 0 & -\alpha & 0 & \cdots & & \cdots & 0 \\
0 & \cdots & 0 & -\beta & a_{P_{NE}} & 0 & 0 & \cdots & 0 & -\beta & 0 & \cdots & \cdots & 0 \\
-\beta & 0 & \cdots & 0 & 0 & a_{P_W} & -\alpha & 0 & \cdots & 0 & -\beta & 0 & \cdots & 0 \\
0 & -\alpha & 0 & \cdots & 0 & -\alpha & a_{P_I} & -\alpha & 0 & \cdots & 0 & -\alpha & 0 & \cdots & 0 \\
0 & 0 & \ddots & 0 & \cdots & 0 & \ddots & \ddots & \ddots & 0 & \cdots & 0 & \ddots & 0 & 0 \\
0 & \cdots & 0 & -\alpha & 0 & \cdots & 0 & -\alpha & a_{P_I} & -\alpha & 0 & \cdots & 0 & -\alpha & 0 \\
0 & \cdots & \cdots & 0 & -\beta & 0 & \cdots & 0 & -\alpha & a_{P_E} & 0 & 0 & \cdots & 0 & -\beta \\
0 & \cdots & & \cdots & 0 & -\beta & 0 & \cdots & 0 & 0 & a_{P_{SW}} & -\beta & 0 & \cdots & 0 \\
0 & \cdots & & & \cdots & 0 & -\alpha & 0 & \cdots & 0 & -\beta & a_{P_S} & -\beta & 0 & 0 \\
0 & \cdots & & & & \cdots & 0 & \ddots & 0 & \cdots & 0 & \ddots & \ddots & \ddots & 0 \\
0 & \cdots & & & & & \cdots & 0 & -\alpha & 0 & \cdots & 0 & -\beta & a_{P_S} & -\beta \\
0 & \cdots & & & & & & \cdots & 0 & -\beta & 0 & \cdots & 0 & -\beta & a_{P_{SE}}
\end{bmatrix} \tag{3}$$

where the middle 5 rows are repeated $m-2$ times. $\boldsymbol{b}$ is a N^2-by-1 matrix with the entry at the center row (ie. $\frac{N^2}{2}$th row)

$$\boldsymbol{b} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ b \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4}$$

Therefore, the fluence $\boldsymbol{U}$ is simply $\boldsymbol{A}^{-1}\boldsymbol{b}$. The measurements of the partial currents at the 16 detector locations is given by $\boldsymbol{QU}$, where $\boldsymbol{Q}$ is the measurement operator hat encodes the location of the detectors. $\boldsymbol{Q}$ takes on the value $\frac{1}{2}\left(\frac{1-R_{eff}}{1+R_{eff}}\right)$ at detector location and zero otherwise. See Figure 2 for the location of detectors. Given that there were 4 sources and 16 detectors located

at the boundaries, 64 measurements were generated in total. See Table 2 in Appendix A for the synthetic measurements.

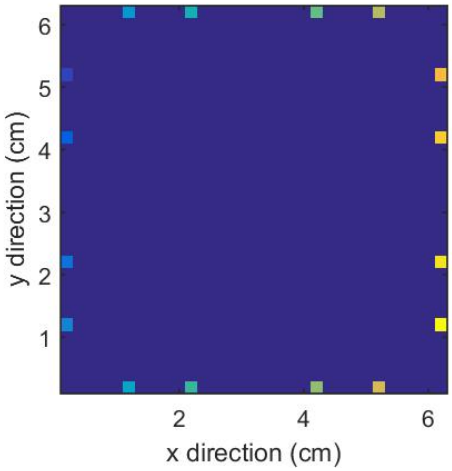

Figure 2. The detectors are shown as the colored dots on the boundaries.

2.3 Solving the Inverse Model

The synthetic measurements were used to derived the μ_a profile. This step is referred to as solving the inverse model. In this project, the steepest descent method along with evolution strategy and inexact line search were used. The goal of solving the inverse model is to minimize the objective function f. This entails

1. solving forward problem for $\boldsymbol{U}$ with an estimated x (where $x = \mu_a$ in this case);
2. evaluation of f using $\boldsymbol{U}$;
3. repeat the steps 1-2 with the new x until minimum of f is found.

2.4 Steepest Descent Method with Inexact Line Search

The basic idea behind the iteration process is to generate a sequence of updates $x(k)$ that can decrease $f(x)$ towards the minimum $f(x^*)$ as

$$x^{k+1} = x^k + \Delta x^k \tag{5}$$

such that

$$f(x^k + \Delta x^k) < f(x^k). \tag{6}$$

The steepest descent method is a gradient-type method that finds Δx using the first derivative of $f(x)$. The negative gradient of the objective function (i.e. the descent direction) is Δx

$$\Delta x^k = -\nabla f(x^k). \tag{7}$$

The algorithm would stop once it satisfies the stopping criterion, which usually takes the form

$$||\nabla f(x^k)||_2 < \epsilon. \tag{8}$$

Sometimes, there is no need to take the full Δx step; and inexact line search finds the step size $t_{new} = 0.5t$ that minimize the objective function using the following condition

$$f(x^k + t^k \Delta x^k) < f(x^k) + \alpha t_{new} \nabla f(x)^T \Delta x \quad \text{with } \alpha = 10^{-4}. \tag{9}$$

In this project, a initial step size of 20 ($t = 20$) was used to speed up the algorithm.

In summary, the reconstruction algorithm contains the following steps

```
given a starting point x,
repeat
1. Determine a descent direction Δx = -∇f(x).
2. Line search. Choose a step size t > 0.
3. Update x := x + tΔx.
4. Evaluate f(x)
until stopping criterion is satisfied.
```

See Listing 1 for the evolution strategy MATLAB code, Listing for the solving forward model function, Listing for the solving inverse model function and Listing for the code that runs the forward and inverse model.

2.5 Evolution Strategy

Evolution strategy finds the optimal initial guess of μ_a by sampling across a range of μ_a values. The steps are as follow

1. Initialization: randomly choosing N_p number of parents μ_a values $(p^1, p^2, p^3, \cdots p^{N_p})$ in the range between 0.01 cm^{-1} and 2 cm^{-1}.

2. Generation of offspring: for each pair of parents, one child is created in a two-step process: recombination and mutation. This project following a simple recombination rule by averaging optical properties of each parent

   $$p^{com(i,j)} = \frac{1}{2}(p^i + p^j) \quad \text{with pair}(i, j). \tag{10}$$

   In the mutation step, the averaged optical properties are further modified as follows

   $$c^{com(i,j)} = p^{com(i,j)} + \sigma\omega \tag{11}$$

   where ω is the Gaussian distributed random number, and σ is the standard deviation that is set to be $\sigma = \frac{1}{3}p_{old}$.

3. Selection: once the offspring are generated, the objective function of each children are evaluated and N_p members with the lowest objective function values are selected as the new parent generation.

4. Repeat steps 1-3 until the objective function converges.

2.6 Objective Function and Gradient

The objective function is the summation of the different between the measurements $\boldsymbol{P}$ and the predictions $\boldsymbol{M}$ given x (i.e. μ_a profile)

$$f(s) = \frac{1}{2}\sum_{s=1}^{N_s}\sum_{d=1}^{N_d}(\boldsymbol{P}_{s,d}(\boldsymbol{U}) - \boldsymbol{M}_{s,d})^T(\boldsymbol{P}_{s,d}(\boldsymbol{U}) - \boldsymbol{M}_{s,d}). \tag{12}$$

To speed up the code, a slightly different objective function was used

$$f(s) = \frac{1}{2}\sum_{s=1}^{N_s}\sum_{d=1}^{N_d}(P_{s,d}(\boldsymbol{U})/M_{s,d} - 1)^T(P_{s,d}(\boldsymbol{U})/M_{s,d} - 1). \tag{13}$$

The gradient is calculated using the adjoint equation

$$\nabla f = -\sum_{s=1}^{N_s}\frac{\partial(\boldsymbol{A}\boldsymbol{U}_s)^T}{\partial x}\eta_s \tag{14}$$

where

$$\eta = \boldsymbol{A}^{-T}\boldsymbol{Q}^T(\boldsymbol{P} - \boldsymbol{M}). \tag{15}$$

Given that the objective function was modifies, η would need to change as well

$$\eta = \boldsymbol{A}^{-T}\boldsymbol{Q}^T(\boldsymbol{P}/\boldsymbol{M} - 1). \tag{16}$$

See Listing 1 for the evolution strategy MATLAB code, Listing for the solving forward model function, Listing for the solving inverse model function and Listing for the code that runs the forward and inverse model.

3 Results

3.1 Evolution Strategy

One hundred parents ($N_p = 100$) were used during the implementation of evolution strategy. This decreased the computational time of each iteration, but it would converge to the global minimum much faster than a N_p. In other words, a larger N_p would take less iteration to converge and therefore less time. Table 1 listed the outcome of the first three iterations. This project used the μ_a returned from the third iteration as the initial guess of the inverse model.

Iteration	Absorption Coeff.	Objective Function
1	0.05525337	0.3264277
2	0.05492386	0.3231841
3	0.05491657	0.3231852

Table 1. The μ_a and objective function in the first three iteration of evolution strategy.

3.2 Steepest Descent with Inexact Line Search

The resolution was set to $0.2cm$ for the sake of speed, so there were 31 grid points along both direction ($N_x = N_y = N = 31$). The objective function plotted against the number of iterations is shown in Figure 3.

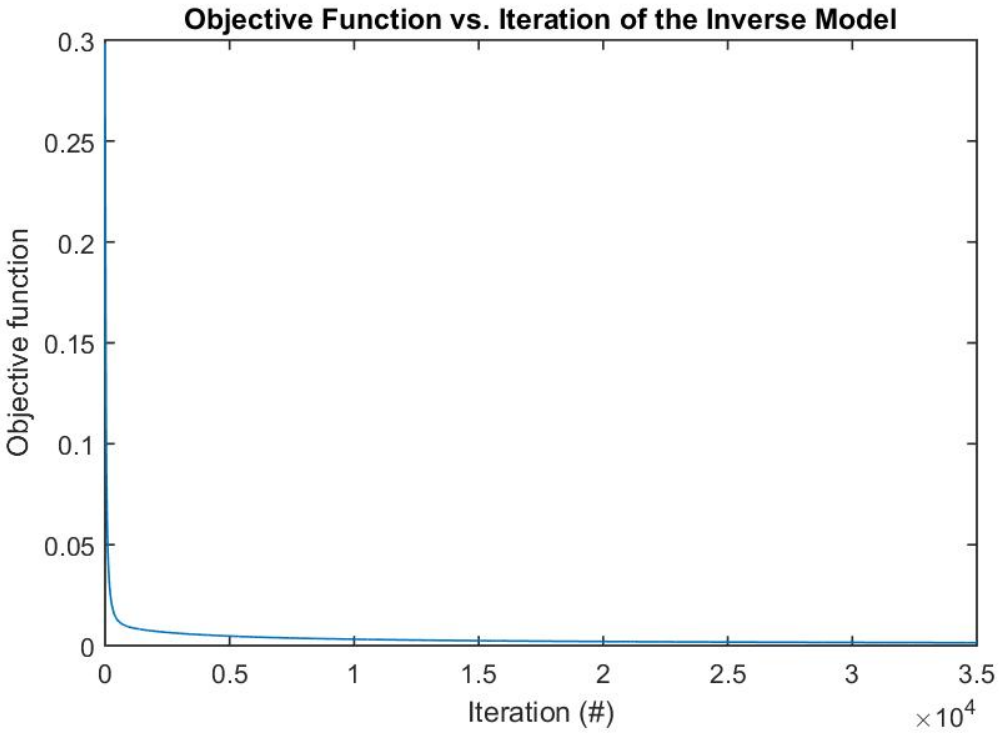

Figure 3. The μ_a profile (left) and the gradient (right) at the 100th iteration. Color bar of μ_a is in the unit of cm^{-1}.

The output of the inverse model at various iteration steps with the initial guess of $0.05491657cm^{-1}$ are shown in Figure 4 - 9. A circular shape object (shown as the red blob in Figure 4) appeared in the absorption profile during the very beginning of the iterative process. At the 100th iteration, the blob was estimated to have a μ_a of 0.07 cm^{-1}. As the number of iteration increased, the blob's estimated μ_a continued to increase. However, the convergence to the true μ_a was extremely slow. The last iteration tested in this project was the 36,056th iteration. The maximum μ_a inside of blob was 0.1203 cm^{-1}, which is only half of the true μ_a, and it took approximately 5 hours for the program to reach this value.

The other thing observed in the μ_a profiles was the uneven medium background. The medium was supposed to have a homogeneous (i.e. constant) absorption coefficient. In the reconstructed μ_a profile, there were some uneven artifacts/features arranged in circle around the red blob. This became more pronounced as the number of iterations increased.

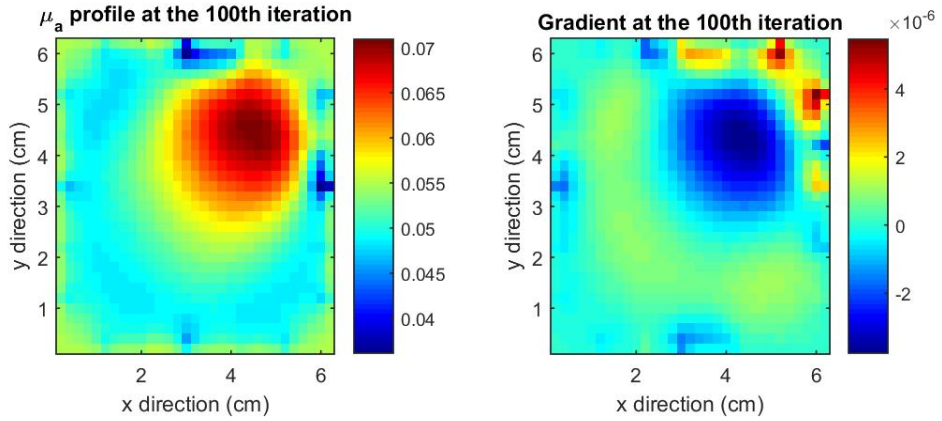

Figure 4. The μ_a profile (left) and the gradient (right) at the 100th iteration. Color bar of μ_a is in the unit of cm^{-1}.

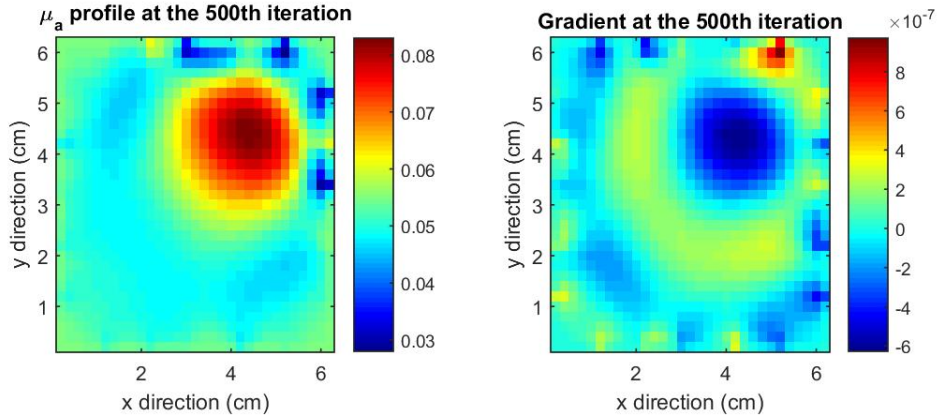

Figure 5. The μ_a profile (left) and the gradient (right) at the 500th iteration.

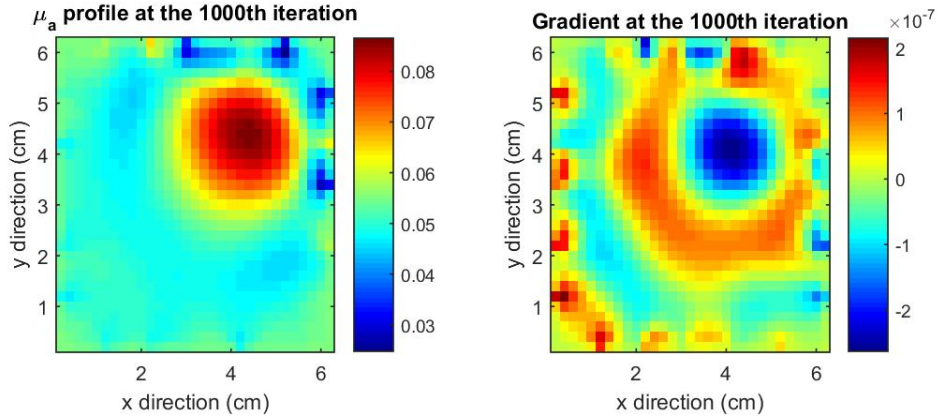

Figure 6. The μ_a profile (left) and the gradient (right) at the 1000th iteration.

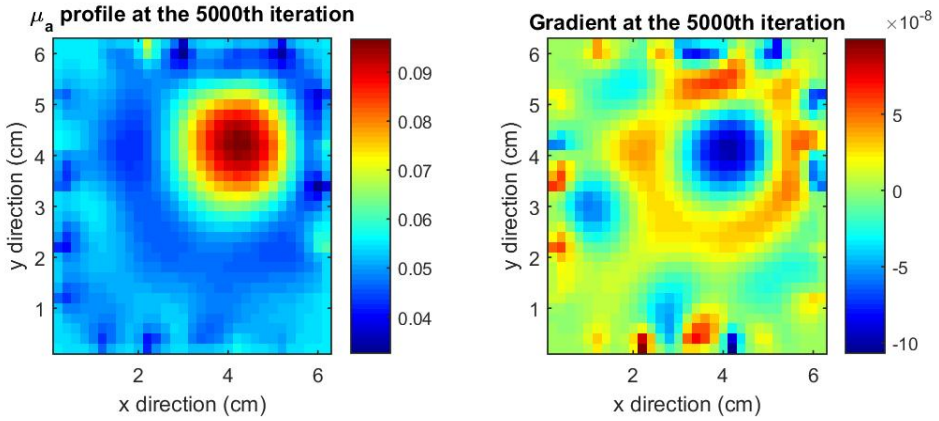

Figure 7. The μ_a profile (left) and the gradient (right) at the 5000th iteration.

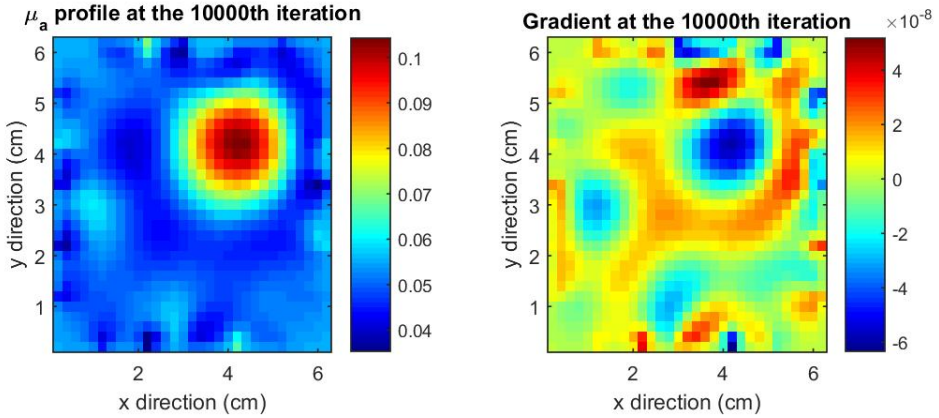

Figure 8. The μ_a profile (left) and the gradient (right) at the 10000th iteration.

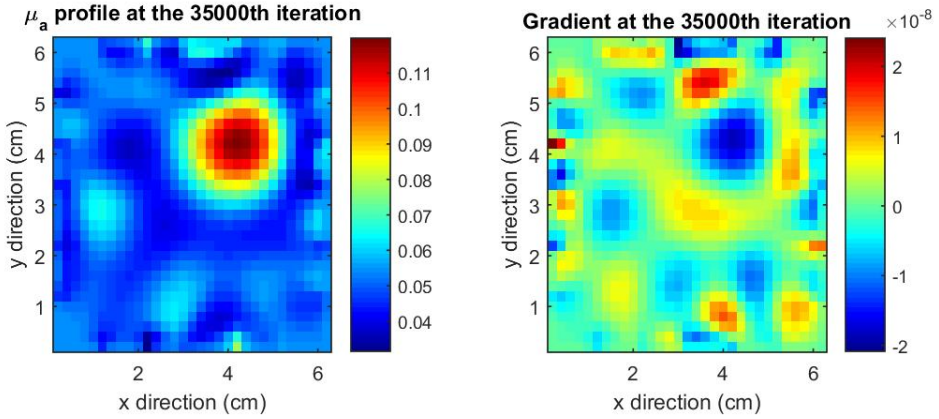

Figure 9. The μ_a profile (left) and the gradient (right) at the 35000th iteration.

4 Discussion

The reconstruction algorithm used in this project successfully finds the location of the small square within the medium. However, a few improvements could be made to enhance the performance of the algorithm and a few modifications would be needed to apply to various types of clinical data.

One problem shown in the result is the uneven features in the background medium. This arises from the fact that there was not enough of measurements. The algorithm was used 64 measurements to recover 961 unknowns. This could be solved by using more than 4 sources to obtain more than 4 projections. In addition, the uneven background can be reduced with a regularization matrix that does a smoothing on the inverse solution. For quality image reconstruction, this smoothing matrix is used in combination with inverse model's reconstruction algorithm.

The other problem was in the computational time. Steepest descent method is a simple but slow method to find the minimum of the objective function. This is because the Δx needs to be made small enough to ensure the validity of linear approximation. The speed could be improved by using Newton methods which have quadratic convergence.

It was mentioned in the introduction that the reconstruction algorithm used in this project can be applied to extract optical properties of soft tissues. A few modifications or improvements would be made in order to implement this algorithm and use it various clinical settings.

The first modification is to make the algorithm adaptable for any geometry. The algorithm is currently restricted to square shapes but most tissues in reality have curvature to them. Therefore, the matrix created within the algorithm would be different for different geometries. In addition, the Q matrix that describes the position and index of detectors would be modified accordingly. For something like breast, one simply solution would be compressing it to a square shape.

The second modification is to introduce another dimension to create 3D tomographic images of optical properties. This requires re-deriving the discretized finite volume equations for the 3D case.

The third modification is to expand the algorithm so that it can extract more than one optical property. It is easier and faster to solve for one unknown (the absorption coefficient in this case) as opposed to multiple unknowns. Therefore, the algorithm assumes that all the optical properties of the object and the small square within the object are the same, except for the absorption coefficient. Right now, this algorithm is limited to tissues with absorption as contrast. If the contrast comes from scattering, then the algorithm won't be able to detect the contrast. For example, this algorithm may not applicable to finger imaging. Bone has a different scattering coefficient compared to muscle or skin. Then, the absorption profile given by the algorithm won't provide us with much information on the contrast. By expanding the algorithm to solve for multiple unknowns, we can infer more information on the optical properties of tissue from the same set of measurements.

In summary, a lot of future work can be done to improve the performance of the reconstruction algorithm and apply this algorithm to real clinical data such as those collected from breast imaging or finger imaging.

Appendices

A Synthetic Measurements

0.001440	0.000019	0.000553	0.000033
0.009369	0.000029	0.000406	0.000088
0.004110	0.000030	0.000084	0.000414
0.000779	0.000020	0.000031	0.000560
0.000558	0.000023	0.001432	0.000025
0.000340	0.000025	0.000024	0.001441
0.000410	0.000061	0.009343	0.000035
0.000279	0.000064	0.000034	0.009370
0.000078	0.000318	0.004028	0.000026
0.000070	0.000262	0.000029	0.004109
0.000026	0.000480	0.000726	0.000016
0.000027	0.000322	0.000017	0.000778
0.000018	0.001306	0.000285	0.000021
0.000027	0.008900	0.000207	0.000059
0.000032	0.003895	0.000049	0.000270
0.000021	0.000728	0.000020	0.000332

Table 2. The synthetic measurements in the `meas.txt` file. The 1st column corresponds to the source on the west boundary. The 2nd column corresponds to the source on the east boundary. The 3rd column corresponds to the source on the north boundary. The 4th column corresponds to the source on the south boundary.

B MATLAB Programs

Listing 1. The script that uses evolution strategy to find the optimal initial guess.

```
% detector readings, ie. measurements
filename = 'meas.txt';
fid = fopen(filename,'r');
meas = fscanf(fid,'%f');
fclose(fid);

L = 6;                  % cm, thickness of slab
dx = 0.2;               % C.V. size in x-dir
dy = dx;                % C.V. size in y-dir
x = 0.0:dx:L;           % x-coord
y = 0.0:dy:L;           % y-coord
N = size(x,2);          % number of grid points

m = matfile('evolution_strategy.mat', 'Writable', true);

% initilization: create the parents
mua_bound = [0.001 2];
a = mua_bound(1);
b = mua_bound(2);
Np = 10;                          % number of parents
parent = (b-a).*rand(1,Np)+a;     % set of parents
Nt = Np*Np;                       % number of offsprings

iter = 0;       % number of iteration; larger Np, smaller iter
tol = 1e-6;     % tolerance
err = 10000000;

while iter<100 && min(err)>tol
    % generation (recombination and mutation) of offspring for pair of
 ↪  parents
    for i=1:Np
        for j=1:Np
            combine = (parent(i)+parent(j))/2;   % recombination
            sigma = parent(i)/3;                 % mutation size
            k = (i-1)*Np+j;                      % index from 1:100
            child(k) = combine + sigma*randn;
            mua(1:N*N) = child(k);
            pred = solve2DForward_all(N,mua);    % prediction of
 ↪  detector readings given the mua from offspring
            objFunc(k) = 0.5*sum((pred./meas-1).^2);
        end
    end
```

```
    [err,index] = sort(objFunc);
    parent = child(index(1:Np));
    iter = iter + 1;
    fprintf('Absorption coefficient = %4f, error = %4f, iter =
↪   %i\n',parent(1), min(err), iter)
    m.objFunc(iter,1) = min(err);
    m.mua(iter,1) = parent(1);
end
```

Listing 2. The function that solves the 2D forward model.

```
function [M,u] = solve2DForward(N,mua,source_loc)
%SOLVE2DFORWARD solves the 2D forward model of the diffusion eqn using
 ↪  the
%finite volume method.
%   Inputs:
%       N: number of grid points
%       mua: N^2-by-1 vector of the mua profile
%       source_loc: string that tells where the source is located
%         'W' = west
%         'E' = east
%         'N' = south
%         'S' = north
%   Outputs:
%       M: the matrix in Mu = b
%       u: a fluence vector with dimension N^2-by-1, it needs to be
%       reshapedto get the orignal fluence profile: reshape(u,N,N).

global L x dx dy Reff qin itermax tol Q

% construct control volume (C.V.) and calculate other necessary
 ↪  parameters
D = 1/15;               % assume D is constant over the medium
Nt = N*N;               % total number of grid points
delx = dx;              % distance between two adjacent nodes in x-dir
dely = dy;              % distance between two adjacent nodes in y-dir
A = (1+Reff)/(1-Reff);  % reflection correction factor A in Robin B.C.

% initialize variables
M = zeros(Nt,Nt);       % matrix M in Mu=b
b = zeros(Nt,1);        % right-hand side vector b in Mu=b
u = zeros(Nt,1);        % solution vector u
sp = zeros(Nt,1);       % internal source s
qin = zeros(Nt,1);      % boundary source qin [W/cm^2] (e.g., laser
 ↪  illumination)

% place light sources to nodes accordingly
switch source_loc
    case 'W'
        j=1;i=(N-1)/2;          % light source located in the middle of
 ↪  west side
        %sp(1) = 1;
    case 'E'
```

```
        j=N;i=(N-1)/2;
    case 'N'
        j=(N-1)/2;i=1;
    case 'S'
        j=(N-1)/2;i=N;
end
qin((j-1)*N+i) = 1.0;    % qin = 1 W/cm^2 directed into the medium

% construct matrix M and vector b in Mu=b according to fomulations
% in lecture note: ap*Up = ae*Ue+aw*Uw+as*Us+an*Un+b.
for i=1:N
    for j=1:N
        row = (j-1)*N+i; % row index of matrix M
        % interior node
        if (i>1 && j>1 && i<N && j<N)
            ae = D*dy/delx;
            aw = D*dy/delx;
            as = D*dx/dely;
            an = D*dx/dely;
            ap = ae + aw + as + an + mua(sub2ind([N,N],i,j))*dx*dy;

            col = row;
            M(row,col) = ap;
            M(row,col+1) = -ae;
            M(row,col-1) = -aw;
            M(row,col+N) = -as;
            M(row,col-N) = -an;
            b(row) = sp(row)*dx*dy;
        % east boundary:
        elseif (i==N && j>1 && j<N)
            aw =  D*dy/delx;
            as = D*(dx/2)/dely;
            an = D*(dx/2)/dely;
            ap = dy/(2*A)+ aw + as + an +
↪   mua(sub2ind([N,N],i,j))*(dx/2)*dy;

            col = row;
            M(row,col) = ap;
            M(row,col-1) = -aw;
            M(row,col+N) = -as;
            M(row,col-N) = -an;
            b(row) = sp(row)*(dx/2)*dy+4*qin(row)*dy/(2*A);
        % west boundary node
        elseif (i==1 && j>1 && j<N)
            ae = D*dy/delx;
```

```
            as = D*(dx/2)/dely;
            an = D*(dx/2)/dely;
            ap = ae + dy/(2*A)+ as + an +
↪   mua(sub2ind([N,N],i,j))*(dx/2)*dy;

            col = row;
            M(row,col) = ap;
            M(row,col+1) = -ae;
            M(row,col+N) = -as;
            M(row,col-N) = -an;
            b(row) = sp(row)*(dx/2)*dy+4*qin(row)*dy/(2*A);
↪
        % south boundary node
        elseif (j==N && i>1 && i<N)
            ae = D*(dy/2)/delx;
            aw =  D*(dy/2)/delx;
            an = D*dx/dely;
            ap = ae + aw + dx/(2*A) + an +
↪   mua(sub2ind([N,N],i,j))*dx*dy;

            col = row;
            M(row,col) = ap;
            M(row,col+1) = -ae;
            M(row,col-1) = -aw;
            M(row,col-N) = -an;
            b(row) = sp(row)*dx*(dy/2)+4*qin(row)*dy/(2*A);
        % north boundary node
        elseif (j==1 && i>1 && i<N)
            ae = D*(dy/2)/delx;
            aw =  D*(dy/2)/delx;
            as = D*dx/dely;
            ap = ae + aw + as + dy/(2*A) +
↪   mua(sub2ind([N,N],i,j))*dx*(dy/2);

            col = row;
            M(row,col) = ap;
            M(row,col+1) = -ae;
            M(row,col-1) = -aw;
            M(row,col+N) = -as;
            b(row) = sp(row)*dx*(dy/2)+4*qin(row)*dx/(2*A);
        % corner nodes
        elseif i==1 && j==1
            ae = D*(dy/2)/delx;
            as = D*(dx/2)/dely;
```

```
            ap = ae + (dy/2)/(2*A)+ as + (dx/2)/(2*A) +
↪  mua(sub2ind([N,N],i,j))*(dx/2)*(dy/2);

            col = row;
            M(row,col) = ap;
            M(row,col+1) = -ae;
            M(row,col+N) = -as;
            b(row) = sp(row)*(dx/2)*(dy/2)+4*qin(row)*(dy/2)/(2*A)...
                                          +4*qin(row)*(dx/2)/(2*A);
        elseif j==1 && i==N
            aw = D*(dy/2)/delx;
            as = D*(dx/2)/dely;
            ap = (dy/2)/(2*A)+ aw + as + (dx/2)/(2*A) +
↪  mua(sub2ind([N,N],i,j))*(dx/2)*(dy/2);

            col = row;
            M(row,col) = ap;
            M(row,col-1) = -aw;
            M(row,col+N) = -as;
            b(row) = sp(row)*(dx/2)*(dy/2)+4*qin(row)*(dy/2)/(2*A)...
                                          +4*qin(row)*(dx/2)/(2*A);
        elseif j==N && i==1
            ae = D*(dy/2)/delx;
            an = D*(dx/2)/dely;
            ap = ae + (dy/2)/(2*A)+ (dx/2)/(2*A) + an +
↪  mua(sub2ind([N,N],i,j))*(dx/2)*(dy/2);

            col = row;
            M(row,col) = ap;
            M(row,col+1) = -ae;
            M(row,col-N) = -an;
            b(row) = sp(row)*(dx/2)*(dy/2)+4*qin(row)*(dy/2)/(2*A)...
                                          +4*qin(row)*(dx/2)/(2*A);
        elseif j==N && i==N
            aw = D*(dy/2)/delx;
            an = D*(dx/2)/dely;
            ap = (dy/2)/(2*A) + aw + (dx/2)/(2*A) + an +
↪  mua(sub2ind([N,N],i,j))*(dx/2)*(dy/2);

            col = row;
            M(row,col) = ap;
            M(row,col-1) = -aw;
            M(row,col-N) = -an;
            b(row) = sp(row)*(dx/2)*(dy/2)+4*qin(row)*(dy/2)/(2*A)...
                                          +4*qin(row)*(dx/2)/(2*A);
```

```
        end
    end
end

% solve Ax = b to get fluence
u = M\b;

end
```

Listing 3. The function that calculate the detector readings from all four sources.

```
function [pred] = solve2DForward_all(N,mua)
%SOLVE2DFORWARD_ALL solves the 2D forward model of the diffusion eqn
 ↪  using
%the finite volume method. This solves for all 4 sources where the
 ↪  sources
%are located in the middle of each bounary (west, east, north, south).
%   Inputs:
%       N: number of grid points
%       mua: N^2-by-1 vector of the mua profile
%   Outputs:
%       pred: 64-by-1 vector with
%           the first 16 measurements correspond to the source on the
 ↪  west boundary
%           the second 16 measurements correspond to the source on the
 ↪  east boundary
%           the third 16 measurements correspond to the source on the
 ↪  north boundary
%           the fouth 16 measurements correspond to the source on the
 ↪  south boundary

global L x y dx dy Reff qin itermax tol Q

% all source locations
source_locs = ['W' 'E' 'N' 'S'];

% the prediction of the detector readings given a mua profile
pred = [];
for i=1:length(source_locs)
    source_loc = source_locs(i);
    [~,u] = solve2DForward(N,mua,source_loc);    % solve forward to get
 ↪  fluence
    P = Q*u;                                     % prediction of 1
 ↪  source's partial currents
    pred = [pred;P];                             % prediction of all
 ↪  sources
end

end
```

Listing 4. The function that solves the 2D inverse model.

```
function [objFunc_vec,mua_r] = solve2DInverse(N,mua,meas)
% solve inverse problem with steepest descent method
%SOLVE2DINVERSE solves the 2D inverse model of the diffusion eqn using
%the finite volume method. The reconstruction is done with the steepest
%descent method.
%   Inputs:
%       N: number of grid points
%       mua: N^2-by-1 vector of the mua profile
%       meas: 64-by-1 measurement data
%   Outputs:
%       objFunc_vec: the objective function values of all iterations
%       u is a fluence vector with dimension N^2-by-1, it needs to be
↪  reshaped
%       to get the orignal fluence profile: reshape(u,N,N).
%       mua_r: the solution of the mua profile that yields the lowest
%       objective function

% common parameters
global L x y dx dy Reff qin itermax tol Q

% function evaluation with initial guess
iter = 0;

% store the objFunc from every iteration
objFunc_vec = [];

% save progess data to file
filename='output.mat';
m = matfile(filename, 'Writable', true);

% start finding the mua profile
while iter < itermax
    iter = iter + 1;

    % solve adjoint problem
    source_locs = ['W' 'E' 'N' 'S'];
    grad = zeros(N*N,1);
    for i=1:length(source_locs)
        source_loc = source_locs(i);
        [M,u] = solve2DForward(N,mua,source_loc);     % solve forward to
↪  get fluence
        P = Q*u;                                      % partial currents
```

```
        % define volume size for each fluence position
        dAw = [(dx/2)*(dy/2);ones(N-2,1)*(dx/2)*dy;(dx/2)*(dy/2)];
        dAm = [dx*(dy/2);ones(N-2,1)*dx*dy;dx*(dy/2)];
        dAe = dAw;
        dA = [dAw;repmat(dAm,N-2,1);dAe];

        % calculate gradient
        eta = inv(M')*Q'*(P./meas((1+(i-1)*16):16*i)-1);
        grad = grad - (u.*eta).*dA;
    end

    % initialize parameters for line search
    pred = solve2DForward_all(N,mua);
    objFunc = 0.5*sum((pred./meas-1).^2);
    t = 20;
    mua_temp = mua-(t*grad)';
    pred = solve2DForward_all(N,mua_temp);
    objFunc_new = 0.5*sum((pred./meas-1).^2);

    % perform line search
    while objFunc_new > objFunc-(1.0e-4)*t*grad*grad'
        t = 0.5*t;
        mua_temp = mua-(t*grad)';
        pred = solve2DForward_all(N,mua_temp);
        objFunc_new = 0.5*sum((pred./meas-1).^2);
    end

    % update
    mua = mua-(t*grad)';
    objFunc_vec(iter) = objFunc_new;

    % save progress data to file
    m.muaprofile(iter,1:N*N) = mua;
    m.gradient(iter,1:N*N) = grad';
    m.objFunc(iter,1) = objFunc_new;

    % display progress
    fprintf('iter= %d: f(x) = %e and step size t =
↪   %f\n',iter,objFunc,t);
    if ~mod(iter,10)
        FigHandle = figure('Position', [100, 100, 1300, 500]);
        subplot(121);imagesc([dx dy*N],[dy dx*N],reshape(mua,N,N));
        title(sprintf('\\mu_a profile at the %dth iteration',iter));
        xlabel('x direction (cm)');ylabel('y direction (cm)');
```

```
        colorbar;colormap(jet);drawnow;

        subplot(122);imagesc([dx dy*N],[dy dx*N],reshape(grad,N,N));
        title(sprintf('Gradient at the %dth iteration',iter));
        xlabel('x direction (cm)');ylabel('y direction (cm)');
        colorbar;colormap(jet);drawnow;
    end

end
mua_r = mua;

end
```

Listing 5. The script that runs the 2D forward and inverse model given a set of parameters.

```
% Solve forward and inverse problems of 2D FV diffusion eqn

clear; close all;clear global;

% common parameters
global L x y dx dy Reff qin itermax tol Q

% system parameters and problem setup
probType = 'inv';        % problem type set to either 'fwd' or 'inv'
filename = 'meas.txt';   % detector readings as input to inverse code
L = 6;                   % cm, thickness of slab
dx = 0.2;                % C.V. size in x-dir
dy = dx;                 % C.V. size in y-dir
x = 0.0:dx:L;            % x-coord
y = 0.0:dy:L;            % y-coord
N = size(x,2);           % number of grid points

qin = 1;                 % laser power, W/cm^2, as light source on west
nindex = 1.414;          % refractive index of medium
Reff = -1.4399/nindex^2 + 0.7099/nindex + 0.6681 + 0.0636*nindex;

tol = 1.e-4;             % tol for inverse problem
itermax = 100000;        % max. iter num for inverse problem

% detector locations and constants for partial fluence
Q = zeros(16,N*N);
Q(1,sub2ind([N,N],1/dx+1,1)) = 0.5*((1-Reff)/(1+Reff));
Q(2,sub2ind([N,N],2/dx+1,1)) = 0.5*((1-Reff)/(1+Reff));
Q(3,sub2ind([N,N],4/dx+1,1)) = 0.5*((1-Reff)/(1+Reff));
Q(4,sub2ind([N,N],5/dx+1,1)) = 0.5*((1-Reff)/(1+Reff));
Q(5,sub2ind([N,N],1,1/dx+1)) = 0.5*((1-Reff)/(1+Reff));
Q(6,sub2ind([N,N],N,1/dx+1)) = 0.5*((1-Reff)/(1+Reff));
Q(7,sub2ind([N,N],1,2/dx+1)) = 0.5*((1-Reff)/(1+Reff));
Q(8,sub2ind([N,N],N,2/dx+1)) = 0.5*((1-Reff)/(1+Reff));
Q(9,sub2ind([N,N],1,4/dx+1)) = 0.5*((1-Reff)/(1+Reff));
Q(10,sub2ind([N,N],N,4/dx+1)) = 0.5*((1-Reff)/(1+Reff));
Q(11,sub2ind([N,N],1,5/dx+1)) = 0.5*((1-Reff)/(1+Reff));
Q(12,sub2ind([N,N],N,5/dx+1)) = 0.5*((1-Reff)/(1+Reff));
Q(13,sub2ind([N,N],1/dx+1,N)) = 0.5*((1-Reff)/(1+Reff));
Q(14,sub2ind([N,N],2/dx+1,N)) = 0.5*((1-Reff)/(1+Reff));
Q(15,sub2ind([N,N],4/dx+1,N)) = 0.5*((1-Reff)/(1+Reff));
Q(16,sub2ind([N,N],5/dx+1,N)) = 0.5*((1-Reff)/(1+Reff));
```

```
% switch
switch probType
    case 'fwd'
        % small square obejct center at 4cm*4cm of saize 1cm*1cm
        sqr_obj_center_x = find(x==4);
        sqr_obj_center_y = find(y==4);
        sqr_ojb_x_ind =
↪   sqr_obj_center_x-floor(.5/dx):sqr_obj_center_x+floor(.5/dx);
        sqr_ojb_y_ind =
↪   sqr_obj_center_y-floor(.5/dx):sqr_obj_center_y+floor(.5/dx);

        % optical properties - matrix of mua
        mua = 0.05*ones(N,N);                          % background
↪   absorption = 0.05 cm-1
        mua(sqr_ojb_x_ind,sqr_ojb_y_ind) = 0.2;     % square object
↪   absorption = 0.2 cm-1
        mua = flipud(mua);

        % write detector readings
        pred = solve2DForward_all(N,mua(:));
        fid = fopen(filename,'w');
        fprintf(fid,'%f\n',pred);
        fclose(fid);

    case 'inv'
        % read measurements
        fid = fopen(filename,'r');
        meas = fscanf(fid,'%f');
        fclose(fid);

        % initial guess from evolution_strategy.m
        mua(1:N*N) = 0.054916567129257;

        % solve inverse problem
        [objFunc_vec,mua_r] = solve2DInverse(N,mua,meas);

    otherwise
        warning('unexpected problem type');
end
```