Optical Tomography Using the GIIR Method

Alex P. Martinez

April 28, 2010

Abstract

The purpose of this report is to explain the theory behind the Gradient-based Iterative Image Reconstruction (GIIR) method and its application in optical tomography. The method is also implemented in MATLAB. Design issues related to the code and its performance are also discussed.

Contents

1 Acknowledgements

I would like to thank Prof. Nachman and Navid Samavati for their help and support.

2 The problem

The problem those solution is discussed in this report and in [1] and [2] is to find approximate optical properties in a medium using a finite set of measurements in a computationally efficient manner. This is done by simulating radiative transfer in the tissue and comparing it with measurements taken at the boundary of the tissue area. The radiative transfer being simulated is with light in the infra-red range. The simulation and the measurements can be used to find optical properties; this is referred to as an inverse problem.

Advantages compared to other imaging modalities:

- Infrared light is non-ionizing (it can be used for continuous monitoring)
- Faster image acquisition (compared to MRI for example)
- Optical imaging systems can be made portable ($>$ 1T magnets not needed like in MRI)

Radiation is non-ionizing if the energy of its photons is lower than the energy needed to detach electrons from atoms normally found in living tissue. The energy of a photon is given by:

$$E = h\nu \tag{1}$$

Therefore, radiation above a certain frequency will be ionizing. This frequency is about 10^{16} Hz (ultraviolet light), higher than the frequency of infrared light (10^{12} to 10^{14} Hz).

3 Physics of optical diffusion in tissue

Infra-red light is used because its absorption is minimal in tissue compared to other wavelengths of light, therefore the radiative transport equation (RTE) is applicable. Under more assumptions applicable for tissue, the radiative transport equation can be simplified to a diffusion equation. As shown below, this equation can be derived from energy conservation on a ray of light travelling though tissue. This equation can be solved to model radiative transfer but a Monte Carlo method modelling actual photons yields better solutions for most real applications. But with some approximations, the diffusion approximation to (RTE) can yield accurate results in less processing time. The following derivations are based on [3].

3.1 Radiative transfer equation

The radiative transfer equation describes quantitatively the conservation of energy of an ideal beam of light with an infinitesimal cross-section. The beam losses energy by absorption and scattering, and gains energy from other beams and scattering towards the beam. These processes are described at different coordinates specified by using a position vector $\vec{r}$, using a unit propagation vector $\hat{s}$, and a time t. The equation is as follows:

$$\frac{\partial L}{\partial tc} = -\hat{s} \cdot \nabla L - \mu_T L + \mu_s \int_{4\pi} L P d\Omega' + S \tag{2}$$

Where $L = L(\vec{r}, \hat{s}, t)$ is the radiance defined as the energy flow per unit normal area per unit solid angle per unit time, c is the speed of light in the tissue, $\mu_T = \mu_a + \mu_s$ is the sum of the absorption and scattering coefficients. $P = P(\hat{s}', \hat{s})$ is defined as a phase function, since it describes the probability of light with direction $\hat{s}'$ being scattered into a solid angle close to $\hat{s}$. $S = S(\vec{r}, \hat{s}, t)$ is another light source.

3.2 Diffusion approximation

We would like to simplify eq. (2) to an equation where one variable is diffused thought the tissue. As explained below, this variable will be the fluence rate light intensity, given by $U = U(\vec{r}, t) = \int_{4\pi} L(\vec{r}, \hat{s}, t) d\Omega$. As seen from the integration, the light intensity has the same units as L but it is only energy flow per unit volume (since the integration was done over the whole solid angle). The diffusion approximation to the radiative transfer equation, also referred to as the delta-Eddington approximation, follows from the two approximations applicable to living tissue:

1. The transport of radiance can be modelled by its first order spherical harmonics

2. The fractional change in current density in one transport mean free path (distance between scattering events) is much less than 1.

Where current density is defined as $J = \vec{J}(\vec{r}, t) = \int_{4\pi} \hat{s} L d\Omega$. Applying the first approximation to eq. (2) yields:

$$\frac{\partial U}{\partial tc} = -\nabla \cdot J - \mu_a U + S \tag{3}$$

The second approximation can be written as the following equation:

$$\left(\frac{l}{c}\right)\left(\frac{1}{|J|}\frac{\partial J}{\partial t}\right) \ll 1 \tag{4}$$

That can be rewritten as:

$$\left|\frac{\partial J}{\partial tc}\right| \ll (\mu_a + \mu_s')|J| \tag{5}$$

Under this condition, J can be expressed as a Frick's Law:

$$J = -D \nabla U \tag{6}$$

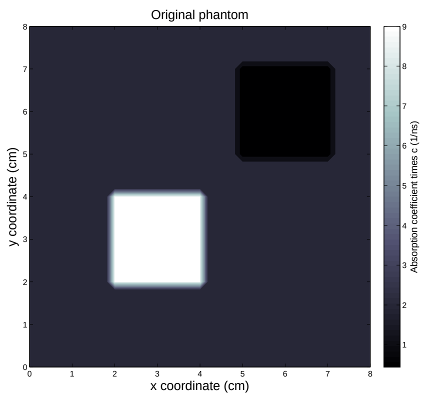

Figure 1: Original phantom

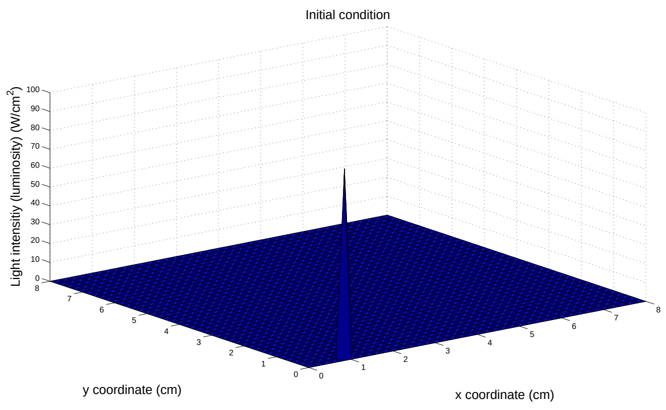

Figure 2: Initial condition: an impulse function at source/receiver pair 1

The negative sign is included to indicate that diffusion of J happens along a negative gradient. The final form of the equation in 2 dimensions is:

$$\frac{\partial U}{\partial t} = \frac{\partial}{\partial x}\Big(D\frac{\partial U}{\partial x}\Big) + \frac{\partial}{\partial y}\Big(D\frac{\partial U}{\partial y}\Big) - c\mu_a U + S \tag{7}$$

With: $D = \frac{1}{3(\mu_a+\mu_s')}$ and $\mu_s' = (1-g)\mu_s$. The value g is the scattering anisotropy and it is equal to the average cosine of the scattering angle distribution. In the optimizations carried out in this report, only the $c\mu_a$ is optimized. D is assumed constant and uniform.

Figures 1 to 7 show an example of the simulated forward problem using an ADI scheme.

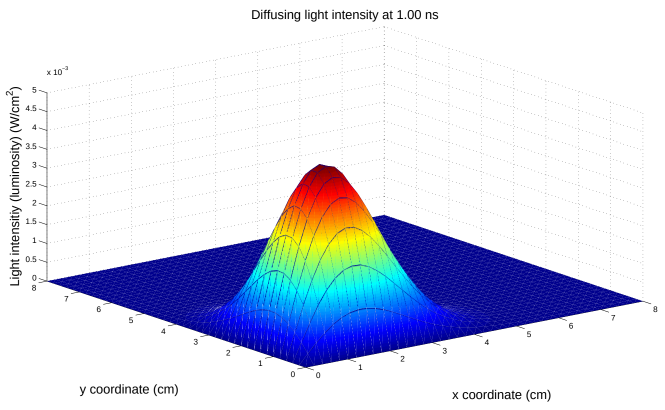

Figure 3: Difussion of light intensity at 1 ns

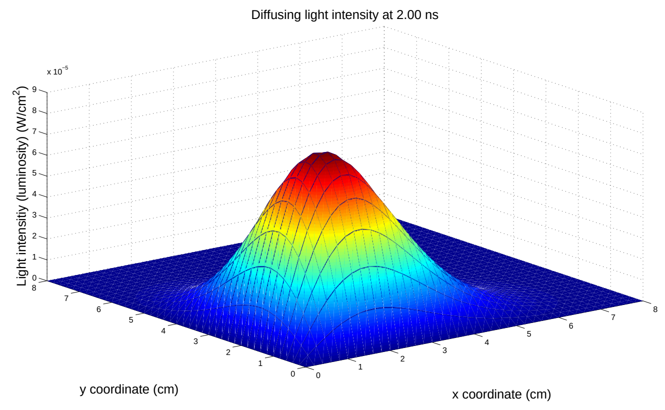

Figure 4: Difussion of light intensity at 2 ns

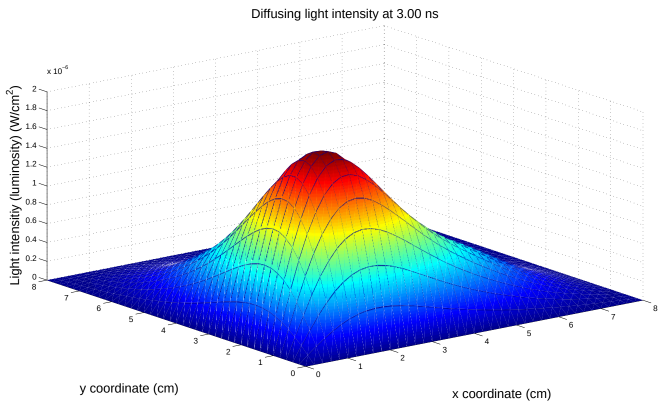

Figure 5: Difussion of light intensity at 3 ns

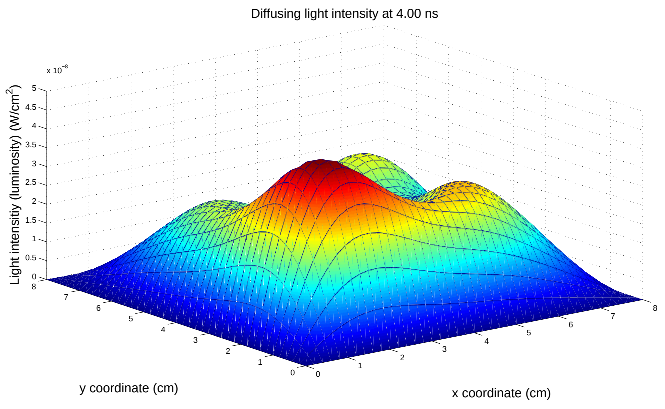

Figure 6: Difussion of light intensity at 4 ns

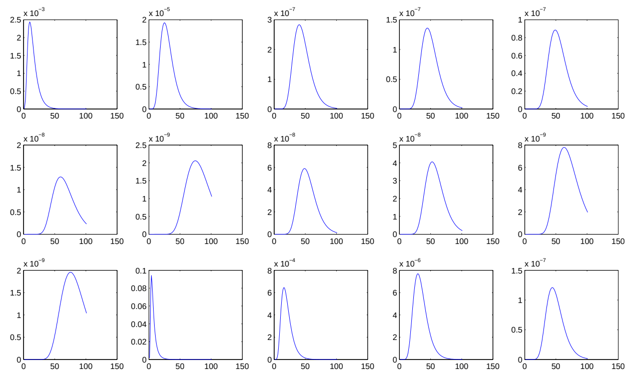

Figure 7: Readings form source/receiver pairs 2 to 16. The x axis is the time in ns and the y axis is the light intensity in W/cm^2

4 Perturbation methods

A conceptually simple approach to solving the inverse optical diffusion problem are perturbation methods. This approach solves a linearised inverse problem. The relation between measurements and estimated optical properties can be written as follows:

$$\mathbf{M}_p = \mathbf{F}[\theta_e] \tag{8}$$

A Taylor expansion about θ_e is:

$$\mathbf{M} = \mathbf{F}[\theta_e] + \mathbf{F}'\theta_e(\theta - \theta_e) + \mathbf{F}''(\theta - \theta_e)^T(\theta - \theta_e) + ... \tag{9}$$

The difference between actual and estimated optical properties is:

$$\Delta\mathbf{M} = \mathbf{M} - \mathbf{F}[\theta_e] = \mathbf{F}'\theta_e(\theta - \theta_e) + ... \tag{10}$$

A linear estimate of this difference is:

$$\Delta\mathbf{M} \approx \mathbf{J}[\theta_e]\Delta\theta \tag{11}$$

Where:

$$\mathbf{J}[\theta_e] = \mathbf{F}'[\theta_e]$$

and:

$$\Delta\theta = \theta - \theta_e$$

A solution to the inverse problem is then:

$$\Delta\theta \approx \mathbf{J}^{-1}[\theta_e]\Delta\mathbf{M} \tag{12}$$

Where the result is used to find the optical properties:

$$\theta = \Delta\theta + \theta_e \tag{13}$$

This linear approximation is "good" only if $\theta_e \approx \theta$: the standard deviation between the average and the predicted optical values from the measured values is relatively low. Finding the inverse of the Jacobian in eq. (12) is hard since it is full and ill-conditioned. The previous issue can be partially resolved by minimizing the following function:

$$\|\mathbf{J}[\theta_e]\Delta\theta - \Delta\mathbf{M}\| \tag{14}$$

The Jacobian can also be made more diagonally dominant using a preconditioning matrix and then solving an equivalent problem that takes less computational effort if the pre-conditioner is well designed. This method can also be used recursively if the linear approximation is not applicable. But this requires finding more than one Jacobian inverse like the one in eq. (12).

5 The GIIR method

GIIR stands for gradient-based iterative image reconstruction. The GIIR Method has the following steps:

1. Find an initial guess for the optical properties
2. Using a numerical approximation to the diffusion equation, find the gradient of a cost function with respect to a finite amount of optical parameters
3. Use a gradient based method to find a new guess for the optical properties
4. Rinse and repeat

The cost function used in [1] and [2] also referred as the sensitivity is the following:

$$\phi = \frac{1}{2}\sum(Y_s^n - U_s^n)^2 \ if \ s \in M \ and \ n \in T \tag{15}$$

Where M are the locations where the measurements were recorded, and T are the times when measurements were taken. In [1] and [2], eq. (15) is used together with Bayesian estimation to take into account the uncertainty for different measurement locations and times. In this report, all measurement uncertainties are assumed equal. Also in [1] and [2], a positivity constraint is imposed on the optical properties (since they are physical quantities that should be greater than 0). This can be taken into account by using a Lagrange multiplier and solving an equivalent problem by minimizing the magnitude of the gradient of a new equation that uses the Lagrange multiplier. This was not included in the current implementation. A common scheme to solve a diffusion equation is implicit finite differences. Solving an implicit scheme takes, per time step, about $O(N^2)$ computations. Here N is the numerical scheme's matrix side length. A method normally used to find gradients of a numerical scheme is numerical differentiation. Besides having truncation and cancellation errors, it takes $O(N^2)$ computations per outer iteration as shown in fig. Both can actually be done in $O(N)$ computations This drops the total computation cost from $O(N^2)$ to $O(N)$ per outer iteration

6 Finite differences

Finite difference schemes can be used to discretize the diffusion equation. Finite differences are useful in rectangular domains with structured grids. In all the following results, a zero boundary condition is used. There are 4 commonly used schemes in 2D. These 4 methods have different stability criteria and grid dependent errors.

6.1 Estimating the stability and the error of numerical schemes

To estimate the error, the discritization is carried out with higher order terms and the order of the error corresponds to the order of those terms [4]. To estimate stability, the Von Neumann or Fourier stability analysis is usually used. In Fourier analysis, a recurrence relation is found for the error. Then the error is assumed to be composed of a finite Fourier series:

$$\epsilon = \sum e^{at} e^{ik_m x} \tag{16}$$

Then the error in some evaluation of the recurrence relation is:

$$\epsilon_m = e^{at} e^{ik_m x} \tag{17}$$

Eq. 17 can be substituted back into the recurrence relation and the equation can be rearranged to be equal to the amplification factor G:

$$G = \frac{\epsilon_j^{n+1}}{\epsilon_j^n} \tag{18}$$

This factor should be lower than 1 for all t. Notice that this analysis does not take into account bounded oscillations. As shown below, the ADI scheme can show bounded oscillations if the following condition is not true:

$$\Delta t < \frac{\Delta^2}{2(maxD_s)} \tag{19}$$

Which corresponds to the explicit step in ADI done on 1 dimension.

6.2 Explicit

In this method, the future light intensity is determined from the same node at a previous time step and its closest 4 adjacent nodes (done in $O(N)$ per iteration):

$$U^{n+1} = BU^n \tag{20}$$

This scheme is stable under the following condition:

$$\Delta t < \frac{\Delta^2}{4(maxD_s)} \tag{21}$$

The order of the time and grid errors for this scheme are the following: $O(\Delta t)$, $O(\Delta^2)$. Where Δ is the distance between nodes in the x and y directions (the grid used is structured and square).

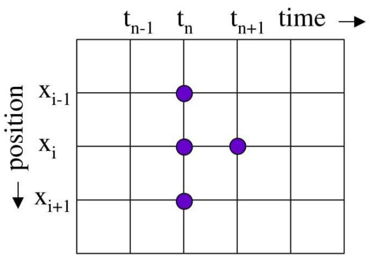

Figure 8: Numerical stencil used in the explicit scheme (from [5])

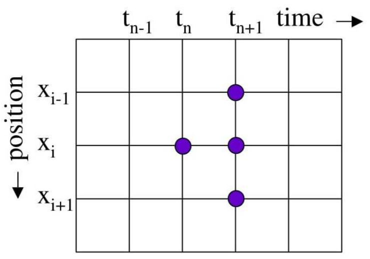

Figure 9: Numerical stencil used in the implicit scheme (from [5])

6.3 Implicit

In this method, the future light intensity is determined from the same node at a previous time step and its closest 4 adjacent nodes at the same time step (done in $O(N)$ per iteration):

$$AU^{n+1} = U^n \tag{22}$$

This scheme is unconditionally stable. The order of the time and grid errors for this scheme are the following: $O(\Delta t)$, $O(\Delta^2)$.

6.4 Crank-Nicholson

Although Crank-Nicholson has a lower time error than implicit and explicit methods, but it is computationally expensive to use since it is of the form:

$$AU^{n+1} = BU^n \tag{23}$$

Where both matrices A and B are pentadiagonal. Therefore it will not be considered in this report.

6.5 Alternating directions implicit

In this method, the future light intensity is determined implicitly in one direction, and explicitly in the other. This is done in one half a time step, then the directions are reversed by rearranging the node numbering:

$$AU^{n+1/2} = BU^{n} \tag{24}$$

$$AU^{n+1}_{reordered} = BU^{n+1/2}_{reordered} \tag{25}$$

This can be done in $O(N)$ per iteration. Half a time step means one time step but the Δt being used is $\Delta t/2$. This is shown in the equation by using n+1/2 instead or n+1. As explained in [2], U is first row ordered, then column ordered, then row ordered and so on. The vector U is row ordered if U with the same x values are enumerated first before increasing the y values ($U = [u_{x1,y1}, u_{x2,y1}, ..., u_{xn,y1}, u_{x1,y2}, u_{x2,y2}, ...]^T$) and column ordered if U with the same y values are enumerated first before increasing the x values ($U = [u_{x1,y1}, u_{x1,y2}, ..., u_{x1,yn}, u_{x2,y1}, u_{x2,y2}, ...]^T$). From now on, when the half time step notation is used, the U is assumed to be ordered as needed ($U := U_{reordered}$)

This scheme is unconditionally stable. However, as explained in the results, the solution can be oscillatory on the first time steps. Surprisingly, the error for this scheme is the following: $O(\Delta t)$, $O(\Delta^2)$.

7 Differentiation methods

As mentioned before, one of the steps of the GIIR method requires the computation of the gradient of a numerical scheme. Below are the 3 methods used for differentiation.

7.1 Symbolic differentiation

In this method, a program finds the derivative of a symbolic function that is imputed in the program. This is done when the expression being imputed is too complicated to differentiate manually. However, if the expression is like a computer program (with if statements for example) and it requires many iterations, the corresponding symbolic statement of the derivative will require too much memory. Also note that the computed numbers are kept with infinite precision (meaning their representation will usually grow larger). Therefore this method is not appropriate to find the gradient of the numerical scheme. MAPLE can be used to differentiate symbolic statements. MATLAB can also do symbolic differentiation by using a MAPLE library.

7.2 Numerical differentiation

Finite difference approximations of any accuracy can be obtained from the Taylor series expansion of the equation:

$$f(x+h) = f(x) + f^{'}(x)h + \frac{f^{''}(x)}{2!}h^2 + ... \tag{26}$$

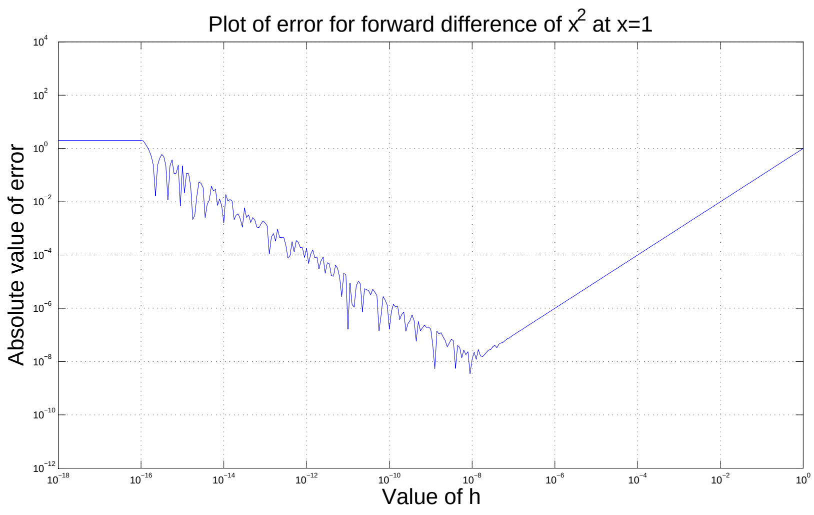

Figure 10: Error of x^2 evaluated at 1 using a forward difference with different values of h

Rearranging using the first term yields a forward difference:

$$f^{'}(x) = \frac{f(x+h) - f(x)}{h} \tag{27}$$

One can also do the Taylor series approximation about $-h$, which yields a backward difference:

$$f^{'}(x) = \frac{f(x) - f(x-h)}{h} \tag{28}$$

As can be seen from Fig.10, the total error is dominated by the truncation error and then by round-off error, levelling off at MATLAB's machine epsilon value as h is decreased. Finding the h value with the minimum error is nontrivial if an analytic solution is not known. The problem is harder if the function is mutivalued. Even if the total error was not a concern, to find the gradient of a function each input variable has to be disturbed individually, therefore the function or program should be run N times. Therefore the total computation time is $O(N^2)$.

7.3 Automatic differentiation

Automatic differentiation (AD) is also referred to as algorithmic differentiation. It can be used to find derivatives of program outputs if the input is given (if the adjoint model as explained below is derived manually then an expression for any input can be obtained). For some functions this can be done with the same computational cost as evaluating the function itself. There are 2 types:

forward mode or backward mode (adjoint). AD is similar but not equivalent to automatic symbolic differentiation or numerical differentiation. An excellent introduction to AD is [6]. When a program is written, more variables than the input and output variables are declared (either manually or by the compiler). This increases memory demands slightly but it increase the execution speed of the code. All variables being calculated in a running program will be refereed to as traces. For example, the following equation is evaluated to yield the answer shown:

$$y = x_1x_2 + x_1x_2sin(x_2) + x_2x_3^2 \tag{29}$$

$$y_{\mathbf{x}=[2\ 0\ 3]^T} = 0 \tag{30}$$

A trace that reduces the number of computations is the following:

$$\begin{aligned} v_{-1} &= x_1 = 2 \\ v_{-2} &= x_2 = 0 \\ v_{-3} &= x_3 = 3 \\ v_1 &= v_{-1}v_{-2} = 0 \\ v_2 &= v_1 sin(v_{-2}) = 0 \\ v_3 &= v_{-2}v_{-3}^2 = 0 \\ v_4 = y &= v_1 + v_2 + v_3 = 0 \end{aligned} \tag{31}$$

Using the product rule, while the trace variables are being evaluated, the derivatives of the trace variables with respect to the input variables can be evaluated using the previously evaluated trace derivatives:

$$\begin{aligned} \nabla v_{-1} &= [1\ 0\ 0]^T \\ \nabla v_{-2} &= [0\ 1\ 0]^T \\ \nabla v_{-3} &= [0\ 0\ 1]^T \\ \nabla v_1 = v_{-1}\nabla v_{-2} + \nabla v_{-1}v_{-2} &= [0\ 2\ 0]^T \\ \nabla v_2 = v_1cos(v_{-2})\nabla v_{-2} + \nabla v_1 sin(v_{-2}) &= [0\ 0\ 0]^T \\ \nabla v_3 = 2v_{-2}v_{-3}\nabla v_{-3} + \nabla v_{-2}v_{-3}^2 &= [0\ 9\ 0]^T \\ \nabla v_4 = \nabla y = \nabla v_1 + \nabla v_2 + \nabla v_3 &= [0\ 11\ 0]^T \end{aligned} \tag{32}$$

This is exactly like symbolic differentiation except that the intermediate derivatives are being evaluated while the total derivative is obtained. This decreases the amount of memory needed substantially since the representation of expressions for every trace is not being stored. This is referred to as forward mode AD.

Instead of taking a gradient for all input variables, we could take the derivative of each trace variable with the output variable. This is done working backwards from y, to each of the input variables. If we want to find $\overline{v}_1 \equiv \partial y/\partial v_1$ (called an adjoint) from a trace with $y = v_4$, we apply the multi-variable chain rule:

$$y = v_1 + v_2 + v_3 \tag{33}$$
$$v_2 = v_1 sin(v_2) \tag{34}$$
$$\overline{v}_1 = \overline{v}_4 + \overline{v}_2 sin(v_{-2}) \tag{35}$$

Instead of finding which functions contain v_1 before running the program to obtain the complete adjoint, the terms of eq. (35) can be added individually as the program is running:

$$\begin{aligned} \overline{v}_1 &= \overline{v}_4 \\ \overline{v}_1 &= \overline{v}_1 + \overline{v}_2 sin(v_{-2}) \end{aligned} \tag{36}$$

This is called accumulation [6]. The calculation of all adjoints is as follows:

$$\overline{v}_4 = y = 1 \tag{37}$$

$$\begin{aligned} y &= v_1 + v_2 + v_3 = 0 \\ \overline{v_3} &= \overline{v_4} = 1 \\ \overline{v_2} &= \overline{v_4} = 1 \\ \overline{v_1} &= \overline{v_4} = 1 \end{aligned} \tag{38}$$

$$\begin{aligned} v_3 &= v_{-2} v_{-3}^2 = 0 \\ \overline{v}_{-2} &= \overline{v}_3 v_{-3}^2 = 9 \\ \overline{v}_{-3} &= \overline{v}_3 2 v_{-2} v_{-3} = 0 \end{aligned} \tag{39}$$

$$\begin{aligned} v_2 &= v_1 sin(v_{-2}) = 0 \\ \overline{v}_1 &= \overline{v}_1 + \overline{v}_2 sin(v_{-2}) = 1 \\ \overline{v}_{-2} &= \overline{v}_{-2} + \overline{v}_2 v_1 cos(v_{-2}) = 9 \end{aligned} \tag{40}$$

$$\begin{aligned} v_1 &= v_{-1} v_{-2} = 0 \\ \overline{v}_{-2} &= \overline{v}_{-2} + \overline{v}_1 v_{-1} = 11 \\ \overline{v}_{-1} &= \overline{v}_1 v_{-2} = 0 \end{aligned} \tag{41}$$

This is referred to as backward mode AD or adjoint mode AD. The amount of computations is about 3 times higher if the gradient is calculated using forward mode. For functions of the form $f : \mathbf{R}^n \Rightarrow \mathbf{R}$, forward mode derivation takes $O(np)$ computations. In reverse mode, it takes only $O(p)$ computations. Where n is the number of input variables and p is the number of trace variables. The opposite is true if the function is of the form $f : \mathbf{R} \Rightarrow \mathbf{R}^m$. For general functions $F : \mathbf{R}^n \Rightarrow \mathbf{R}^m$ where a Jacobian is calculated, the two methods can be combined. Doing it in the least amount of time referred to as the optimal Jacobian accumulation problem [6]. Out of the 3 options discussed for computing the gradient of the numerical scheme, AD has the least memory and processing demands. AD is an even more attractive option since eq. (15) contains one output variable and N input variables, decreasing the computation time from $O(N^2)$ to $O(N)$ compared to numerical differentiation.

A program can be differentiated using AD by transforming the code automatically. The most obvious technique is source transformation where the function is sent to a program that outputs the derivative of the function. This method incurs the least overhead. Another method is operator overloading; in this method, language features are used that allow the programmer to use the same operation like + for different purposes depending on the types or classes being manipulated.

The FADBAD++ header files that overload the double type in C++ where included in an implementation of the ADI scheme using the Thomas algorithm for solving systems with tridiagonal matrices [7]. Unfortunately the program could not run due to a lack of memory. This issue could be resolved by having a header file that had more AD types. The programmer would tell the compiler what solution strategy to apply by using different AD types. This could yield a program with less memory requirements.

8 Adjoint mode AD of finite difference schemes

For one view, the sensitivity with respect to light intensity at a node and given time step is as follows:

$$\frac{d\phi}{dU_s^n} = \sum_{r \in S} \frac{d\phi}{dU_r^{n+1}} \frac{\partial U_r^{n+1}}{\partial U_s^n} + \frac{\partial \phi}{\partial U_s^n} \tag{42}$$

$$\frac{\partial \phi}{\partial U_s^n} = (Y_s^n - U_s^n)\ if\ s \in M\ and\ n \in T \tag{43}$$

This means the light intensity depends on the sensitivity at the recording positions and on previous (or future values since a forward simulation is performed beforehand) For all nodes:

$$\frac{d\phi}{dU^n} = [\frac{\partial U^{n+1}}{\partial U^n}]^T \frac{d\phi}{dU^{n+1}} + \frac{\partial \phi}{\partial U^n} \tag{44}$$

Differentiating the update rule:

$$\frac{\partial U^{n+1}}{\partial U^n} = A^{-1}B \tag{45}$$

$$\frac{d\phi}{dU^n} = B^T (A^{-1})^T \frac{d\phi}{dU^{n+1}} + \frac{\partial \phi}{\partial U^n} \tag{46}$$

For the final sensitivity:

$$\frac{d\phi}{d\theta_p} = \sum_n \sum_{r \in S} \frac{d\phi}{dU_r^n} \frac{\partial U_r^n}{\partial \theta_p} \tag{47}$$

For all optical properties being optimized:

$$\frac{d\phi}{d\theta} = \sum_n \Big[\frac{\partial U^n}{\partial \theta}\Big]^T \frac{d\phi}{dU^n} \tag{48}$$

Differentiating the update rule:

$$\frac{dA}{d\theta_p}U^{n+1} + A\frac{\partial U^{n+1}}{\partial \theta_p} = \frac{dB}{d\theta_p}U^n \tag{49}$$

$$\frac{\partial U^{n+1}}{\partial \theta_p} = A^{-1}\Big[\frac{dB}{d\theta_p}U^n - \frac{dA}{d\theta_p}U^{n+1}\Big] = A^{-1}X_p \tag{50}$$

$$\frac{\partial U^{n+1}}{\partial \theta} = A^{-1}[X_1 ... X_P] = A^{-1}X \tag{51}$$

Where P is the total number of optical properties being optimized. In this report only the $c\mu_a$ values are being optimized, therefore: $P = N$. The final expression for the sensitivity is:

$$\frac{d\phi}{d\theta} = \sum_n X^T(A^{-1})\frac{d\phi}{dU^n} \tag{52}$$

This equation accumulates the contributions of eq. 46 with:

$$\frac{d\phi}{dU^N} = \frac{\partial \phi}{\partial U^N} \tag{53}$$

Where N is the last time step.

9 Conjugate gradient descent

The conjugate gradient method is one of the most effective methods for solving large sparse matrix problems. Solving a linear system is equivalent to computing the minimum of a quadratic from. It can be shown that conjugate gradient descent is faster than steepest gradient descent in most cases. The above results can be used to find the minimum for other non-linear forms if their gradients can be computed. The following definitions and derivations are based on [8].

9.1 Quadratic form

A quadratic form is the following:

$$f(x) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c \tag{54}$$

$$\mathbf{x}^T\mathbf{A}\mathbf{x} > 0 \ \mathit{for\ all}\ \mathbf{x} \neq 0 \tag{55}$$

If $\mathbf{A}$ is square, symmetric and positive definite (condition (55)), $f(x)$ is minimized by the solution to the related matrix problem:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{56}$$

9.2 Steepest gradient descent

The residual is defined as the following:

$$\mathbf{r}_{(i)} = \mathbf{b} - \mathbf{A}\mathbf{x}_{(i)} = \mathbf{A}(\mathbf{x} - \mathbf{x}_{(i)}) = -\mathbf{A}\mathbf{e}_{(i)} = -f'(\mathbf{x}_{(i)}) \tag{57}$$

To minimize $f(x)$, we can move in the direction of the residual to a new point:

$$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + \alpha\mathbf{r}_{(i)} \tag{58}$$

Then "rinse and repeat" until a norm of the residual is below a certain tolerance, usually chosen as a fraction of the same norm applied to the original residual:

$$||\mathbf{r}_{(i)}|| < \varepsilon||\mathbf{r}_{(0)}|| \tag{59}$$

The value of α should be a value that minimizes along a line with direction $\mathbf{r}_{(i)} = -f'(\mathbf{x}_{(i)})$. A value where its derivative should be zero:

$$\frac{d}{d\alpha}f(\mathbf{x}_{(i)} + \alpha\mathbf{r}_{(i)}) = 0 \tag{60}$$

Using the chain rule:

$$\frac{d}{d\alpha}f(\mathbf{x}_{(i+1)}) = f'(\mathbf{x}_{(i+1)})^T\frac{d}{d\alpha}\mathbf{x}_{(i)} = \mathbf{r}_{(i+1)}^T\mathbf{r}_{(i)} \tag{61}$$

And:

$$\alpha = \frac{\mathbf{r}_{(i)}^T\mathbf{r}_{(i)}}{\mathbf{r}_{(i)}^T\mathbf{A}\mathbf{r}_{(i)}} \tag{62}$$

What eq. 61 means is that steepest descent (SD), will be able to choose previous directions instead of always picking new ones. This makes the convergence of the method much slower than conjugate gradient descent [8].

9.3 Conjugate gradient method

Conjugate gradient descent is derived by decomposing the solution into a basis of vectors that are conjugate to one another. This increases the convergence properties of the method compared to steepest descent [8].

10 MATLAB implementation

Adjoint differentiation using the implicit method and ADI was implemented in MATLAB. A section of the code is shown below:

```
function dSdM = AD(numPs,delta,deltat,tmax,impulse,source,numSs,D,...
                   reorder,optMu_a,Yn)
%GET Un
%Constants
numPs2 = numPs*numPs;
c      = D*deltat/2/delta/delta;
%Solution matrix
Uzero = zeros(numPs2,1);

%What receivers are being used?
receivers = 1:numSs;
receivers(source) = [];
%Get source index (in natural ordering) and apply initial condition
sInd = getSourceInd(source);
Uzero(sInd) = impulse;
%Get receiver indices
rInd = getReceiverInd(receivers);

%Fill A
[Aw,Ap,Ae] = fillA(c,optMu_a,deltat,numPs);
%Fill B
[Bw,Bp,Be] = fillB(c,numPs);

%Run ADI scheme
tnum = 0;
while tnum < tmax
    %First half time step: x implicit, y explicit
    b = timesTri(Bw,Bp,Be, Uzero,numPs);
    Uhalf = tdma(Aw,Ap,Ae, b);

    %Store info
    Un(:,tnum*2+1) = Uhalf;
    %Reorder last solution
    Uhalf = Uhalf(reorder);

    %Second half time step: y implicit, x explicit
    b = timesTri(Bw,Bp,Be, Uhalf,numPs);
    Uzero = tdma(Aw,Ap,Ae, b);

    %Store info
    Un(:,tnum*2+2) = Uzero;
    %Reorder obtained solution
    Uzero(reorder) = Uzero;

    %Increase timestep
    tnum = tnum + 1;
end

%GET delSdelU
numRs = numSs-1;
delSdelU = zeros(numPs2,tmax*2);
for n = 1:tmax
    delSdelU(rInd,n*2) = Un(rInd,n*2) - Yn((1:numRs)'+(n-1)*numRs);
end

%GET dSdU
dSdU(:,tmax*2) = delSdelU(:,tmax*2);
for n = (tmax*2-1):-1:1
    %Calculate derivative at previous time step
    C = tdma(Ae,Ap,Aw, dSdU(:,n+1));
    dSdU(:,n) = timesTri(Be,Bp,Bw, C,numPs) + delSdelU(:,n);
    %Reorder last solution
    if (mod(n,2) == 1)
        dSdU(reorder,n) = dSdU(:,n);
    else
        dSdU(:,n)       = dSdU(reorder,n);
    end
end

%GET intPs
%These vector is used to determine if a diagonal point is in the interior
intPs = zeros(numPs2,1);
for j = 2:numPs-1
    ind = (2:numPs-1)' + (numPs-j)*numPs;
    intPs(ind) = 1;
end

%GET dSdM
dSdM = zeros(numPs2,1);
for n = 1:tmax*2
    %Get X for given timestep
    X = diag(intPs.*(-deltat/2));
    X = X.*repmat(Un(:,n),1,numPs2);

    %Accumulate results
    dSdM = dSdM + X'*tdma(Ae,Ap,Aw, dSdU(:,n));
end
```

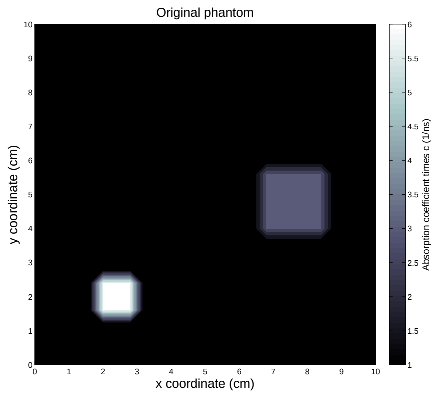

Figure 11: Phantom used to test convergence of SD and NCG schemes

The slowest part of the code is the multiplication of the X matrices. This part of the code was optimized by multiplying the components of X explicitly with U. The only difference between this function and the one that uses the implicit scheme are the reordering steps and the derivative of A with respect to the optical parameters.

11 Phantom and brain image reconstruction

The sensitivity and gradient calculations were included in a MATLAB function and the MATLAB function was used in a steepest descent algorithm and a non-linear conjugate gradient algorithm. The phantom used is the one shown in Fig 11 . The following figures show the reconstruction of the phantom using steepest descent and non-linear conjugate gradient. For the non-linear conjugate gradient a function written by Carl Rasmussen was used. It can be found at: http://www.kyb.tuebingen.mpg.de/bs/people/carl/code/minimize/ The conjugate gradient method used in the function is Polak-Ribiere [8].

12 Regularization

Regularization can be applied in the gradient computation by adding a formula of the following form to the sensitivity:

$$R = b \sum |D_s - D_r|^p \tag{63}$$

This function penalizes high gradients in the solution. But this changes the value of the solution. Therefore there are optimal values of b and p that minimize the error between the obtained and actual solution. In [1] it is reported that a value of p of 1.1 is better than a value of 2 in minimizing the difference between the obtained and actual solutions.

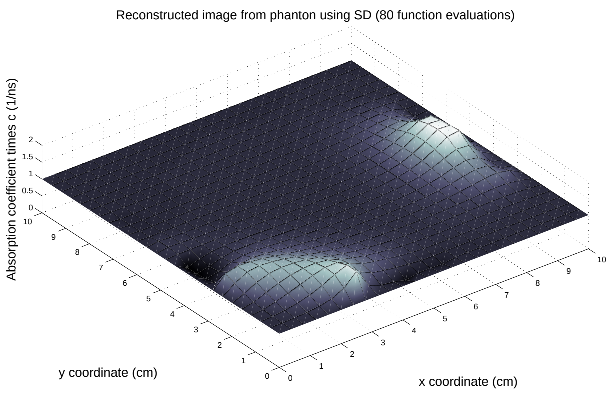

Figure 12: Surface of reconstructed phantom after 80 function evaluations using SD

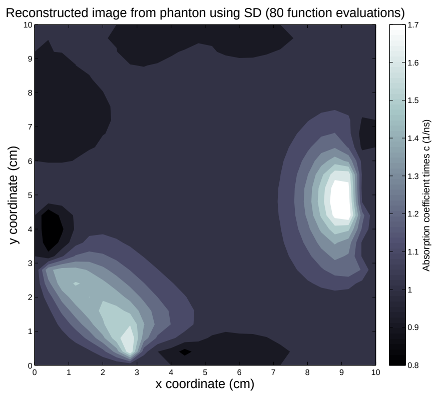

Figure 13: Contour plot of reconstructed phantom after 80 function evaluations using SD

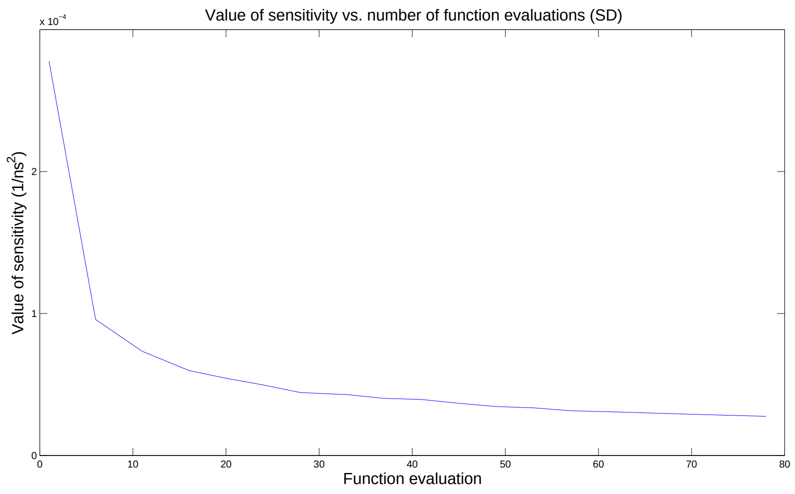

Figure 14: Graph of the sensitivity value as a function of function evaluation using SD

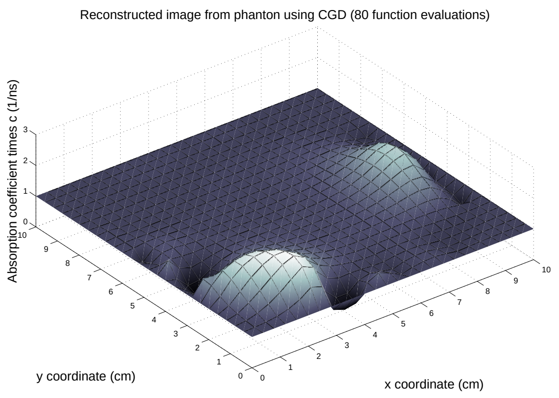

Figure 15: Surface of reconstructed phantom after 80 function evaluations using NCG

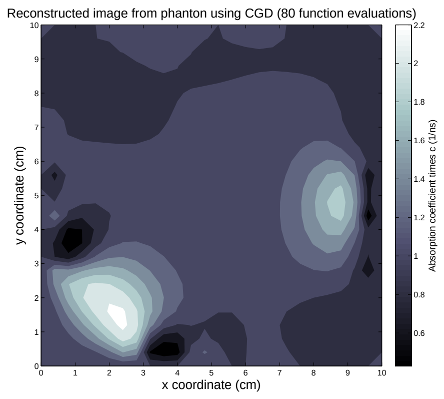

Figure 16: Contour plot of reconstructed phantom after 80 function evaluations using NCG

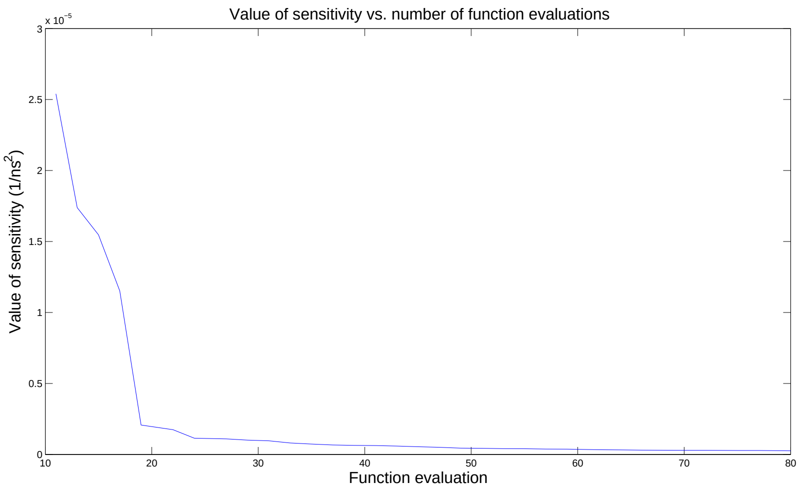

Figure 17: Graph of the sensitivity value as a function of function evaluation using CGD

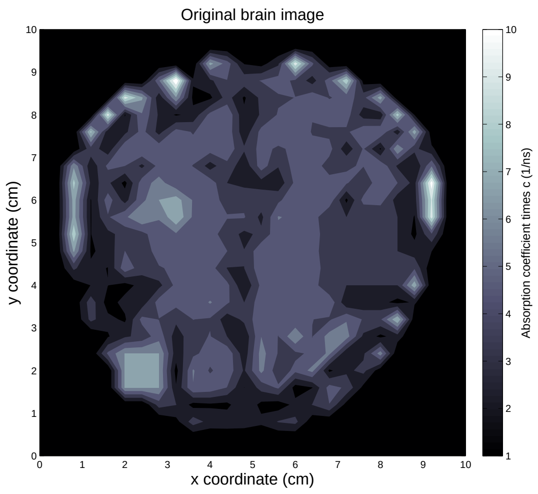

Figure 18: Original low resolution brain image

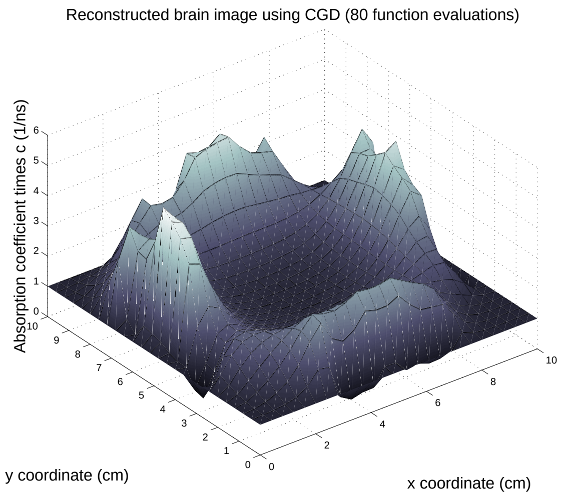

Figure 19: Reconstructed surface of low resolution brain image

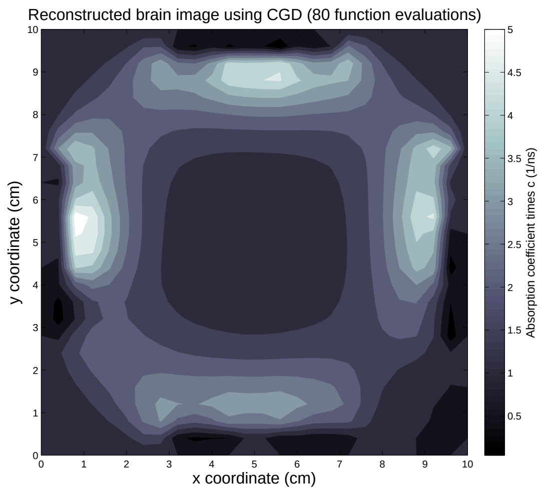

Figure 20: Reconstructed image of low resolution brain image

13 Conclusion

The ADI scheme and adjoint differentiation allow the fastest optimization of optical properties compared to other traditional methods. However, to get better results with the GIIR method, more work should be done in optimization and regularization before a reconstruction is attempted.

References

[1] Suhail S. Saquib et al. Model-based image reconstruction from time-resolved diffusion data. *SPIE proceedings series*, 3034(2):369–380, 1997.

[2] Andreas H. Hielscher et al. Gradient-based iterative image reconstruction scheme for time-resolved optical tomography. *IEEE Transactions on Medical Imaging*, 18(3):262–271, 1999.

[3] Lihong V. Wang. *Biomedical Optics: Principles and Imaging.* Wiley-Interscience, 1st edition, 2007.

[4] Robert D. Richtmyer. *Difference Methods for Initial-Value Problems.* Interscience Publishers, 2nd edition, 1967.

[5] Kenneth M. Hanson. Optical tomography reconstruction: Inversion based on adjoint differentiation. 2000.

[6] Andreas Griewank. *Evaluating derivatives: principles and techniques of algorithmic differentiation.* SIAM, 2nd edition, 2008.

[7] William H. Press et al. *Numerical Recipes in C.* Cambridge University Press, 1st edition, 1988.

[8] Jonathan Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. 1994.