MEG/EEG Source Localization Using

Spatio-Temporal Sparse Representations

Alexey Polonsky and Michael Zibulevsky

May 14, 2004

1. Abstract

Inverse MEG/EEG problem is known to be ill-posed and no single solution can be found without utilizing some prior knowledge about the nature of signal sources, the way the signals are propagating and finally collected by the sensors. The signals are assumed to have a sparse representation in appropriate domain, e.g. wavelet transform, and spatial locality of sources is assumed, the fact that MEG/EEG data comes from physiological source justifies such assumption. Spatial information is utilized through MEG/EEG forward model, which is used when looking for an inverse solution. Finally, we formulate an optimization problem that incorporates both the sparsity and the locality assumptions, and physical considerations about the model. The optimization problem is solved using an augmented Lagrangian framework with truncated Newton method for the inner iteration.

2. Introduction

2.1 MEG/EEG inverse problem

Neural activities of the brain are accompanied with ionic currents that produce weak magnetic and electric fields. Those fields are non invasively measured by highly sensitive sensors. The data collected on the sensors reflects some distribution of brain activity. The final goal is to localize those activities and possibly recover their time courses. In MEG/EEG inverse problem the brain is modeled as a mesh of voxels, each voxel represents a current dipole source [1]. The inverse problem should be solved given the forward model [3], [4] and the response of the sensors.

2.2 Spatial-temporal approach

The voxels activity can be described by an $N:T$ matrix S where N is the number of voxels and T is a number of coefficients that represent the activity of each voxel. Assuming that number of active voxels is relatively small, even if we stay in time domain (the coefficients are time samples), the solution would have a sparse structure. The left plot on Figure 1 depicts time samples of synthetic voxels activity $S^{(t)}$ ($N = 200$ voxels, $T^{(t)} = 60$ time samples, only 10 voxels are active).

It is possible to achieve a sparser representation by applying appropriate transform to time courses of each voxel. A short time Fourier transform and a wavelet transform are known to produce good results when compressing signals from various natural sources [9]. By applying a wavelet transform separately to each voxel in $S^{(t)}$, a much sparser structure is obtained. The right plot on Figure 1 depicts a resulting matrix $S^{(WT)}$.

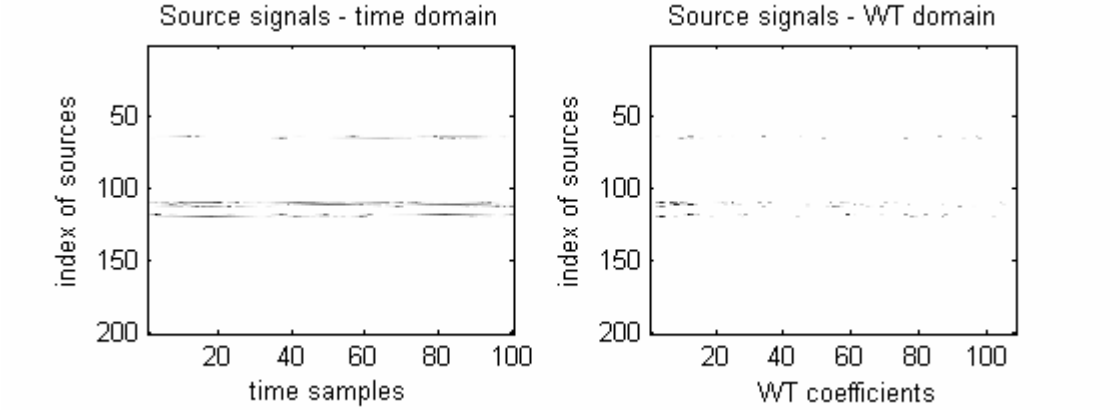

Figure 1: left - time course of voxels activity, right - WT coefficients of voxels activity

Notice, that this view on voxels activity utilizes both temporal and spatial information. Temporal information is utilized once all time courses (in a raw or transformed form) are used to form the matrix *S*. Spatial information can be utilized by considering a known forward model for MEG/EEG (i.e. the propagation rule that tells what is the response of sensors given voxels activity). Finally, sparsity of the solution is assumed, thus we should be able to achieve a good separation of the sources.

3. Problem formulation

Let's denote:

$s = vec(S)$ - matrix S stacked column wise

$S = mat(s)$ - the reverse of vec

s_{ij} - i'th row-j'th column element of S

S_i - i'th row of S reshaped as a column vector

A - block diagonal $MT \times NT$ matrix consisting of T diagonal blocks, where each diagonal block equals A (size $M \times N$)

The MEG/EEG inverse problem can be stated in terms of optimization problem $\min_S f_{obj}(S)$ with an objective function of the following general form:

$$f_{obj} = \frac{1}{2} \cdot \|AS - X\|_F^2 + w_1 \cdot \Psi_1(S) + \ldots + w_k \cdot \Psi_k(S), \qquad (3.1)$$

where $\|\square\|_F^2$ is the Frobenius matrix norm, Ψ_k are some convex scalar functions with minimum at 0 and w_k are positive scalar weights. While $\frac{1}{2}\|AS-X\|_F^2$ forces the forward model response AS to be close to the sensors readings X, functions Ψ_k reflect a priori knowledge and assumptions about S.

In this work we used two such functions. The first one enforces sparsity of the solution:

$$\Psi_1(S)=\|s\|_1=\sum_{i,j}|s_{ij}|,$$

We further assume locality of sources. Active sources can be represented by several non-zero coefficients while non-active sources ideally have all-zero coefficients. Hence, we want to give a high penalty for stand-alone coefficients in random locations of matrix *S* and low penalty for coefficients that represent the same voxel, i.e. belong to the same row of matrix *S*.

One possible choice is a sum of row-wise L2 norms $\sum_{i=1}^{N}\|S_i\|_2$, where S_i is an *i*'th row of *S*. This expression is not differentiable at 0. In order to make it smooth, we use the following technique:

$$\Psi_3(S)=\sum_{i=1}^{N}\sqrt{\|S_i\|_2^2+\varepsilon},$$

The resulting objective function is

$$f_{obj}=\frac{1}{2}\cdot\|AS-X\|_F^2+w_1\cdot\|s\|_1+w_2\cdot\sum_{i=1}^{N}\sqrt{\|S_i\|_2^2+\varepsilon}, \qquad (3.2)$$

where A is an $M:N$ gain matrix of forward model, S is an $N:T$ matrix of sources coefficients, X is an $M:T$ matrix of sensors coefficients and w_1, w_2, ε are empiric values that are tuned during simulations.

Let $C=S\Phi$ be a row-wise wavelet or other transform coefficients of S, which we expect to be sparse for true solution. Denote $Y=X\Phi$ the coefficients of the sensor signal *X*. Our objective can be then reformulated as

$$f_{obj}=\frac{1}{2}\cdot\|AC\Phi^{-1}-Y\Phi^{-1}\|_F^2+w_1\cdot\|c\|_1+w_2\cdot\sum_{i=1}^{N}\sqrt{\|C_i\|_2^2+\varepsilon},$$

where Φ^{-1} is the corresponding inverse transform. In the case of the orthonormal operator Φ^{-1}, we can write equivalently

$$f_{obj} = \frac{1}{2} \cdot \left\| AC - Y \right\|_F^2 + w_1 \cdot \left\| c \right\|_1 + w_2 \cdot \sum_{i=1}^{N} \sqrt{\left\| C_i \right\|_2^2 + \varepsilon} \,, \tag{3.3}$$

Note, that both optimization problems with respect to S and with respect to C have identical objective functions (3.2 and 3.3). For convenience, in the further text S will denote a variable of a generic optimization problem, i.e. S will stand for S or C and X will stand for X or Y.

Typically, in an EEG/MEG problem there are thousands of voxels, hundreds of sensors and hundreds of time course samples/coefficients, which results in 10^5 – 10^6 optimization variables (matrix S or C). This is a large-scale optimization problem, and special optimization techniques should be used.

4. Solution

4.1 Moving from unconstrained non-smooth objective function to smooth but constrained objective.

The objective function 3.2 (or 3.3) is non-differentiable because of the non-smooth second term (the sum of the absolute values). A common way to make it smooth is to express S (or C) as a difference of two non-negative terms $S = S_+ - S_-$ where $S_+ \geq 0$ and $S_- \geq 0$. The resulting optimization problem is:

$$\min\left(\frac{1}{2} \cdot \left\| A\left(S_+ - S_-\right) - X \right\|_F^2 + w_1 \cdot \left\| vec\left(S_+ - S_-\right) \right\|_1 + w_2 \cdot \sum_{i=1}^{N} \sqrt{\left\| \left(S_+ - S_-\right)_i \right\|_2^2 + \varepsilon} \right)$$

$$s.t.$$

$$S_+ \geq 0,$$

$$S_- \geq 0$$

Or in a more compact formulation:

$$\min\left(w_1 \cdot \left\| \tilde{A}\tilde{S} - X \right\|_F^2 + w_1 \cdot \left\| vec\left(\tilde{S}\right) \right\|_1 + w_2 \cdot \sum_{i=1}^{N} \sqrt{\left\| \tilde{S}_i \right\|_2^2 + \left\| \tilde{S}_{i+N} \right\|_2^2 + \varepsilon} \right)$$

$$s.t. \tag{4.2}$$

$$\tilde{S} \geq 0$$

where

$$\tilde{A}_{Mx2N} \equiv \left[A_{MxN} \quad -A_{MxN} \right]$$

$$\tilde{S}_{2N \times T} \equiv \begin{bmatrix} S_+ \\ S_- \end{bmatrix}$$

We choose to solve this constrained problem in a framework of augmented Lagrangian. For convenience, in the further text S will stand for $\tilde{S}$ and A will stand for $\widetilde{A}$.

4.2 Augmented Lagrangian method

Consider an optimization problem with inequality constraints:

$$\min_s f(s), \qquad (4.3)$$

$$\text{s.t. } g_j(s) \le 0,\ 1 \le j \le r$$

where $g_j(s),\ 1 \le j \le r$ is a set of active inequality constraints.

The Lagrangian function of this problem is:

$$L(s,\mu) = f(s) + \sum_{j=1}^{r} \mu_j \cdot g_j(s) \qquad (4.4)$$

Denote a vector of constraints $g(s) \equiv [g_1(s),\ \ldots\ , g_r(s)]^T$ and a matrix of gradients of constraints $\nabla g(s) \equiv [\nabla g_1(s),\ \ldots\ , \nabla g_r(s)]$. The expression (4.4) can be rewritten in a vector form:

$$L(s,\mu) = f(s) + \mu^T g(s) \qquad (4.5)$$

According to Lagrange multipliers theory, if the first Karush-Kuhn-Tucker necessary condition is met (i.e. $f(s), g(s) \in C^1$, s^* is a minimum of (4.3) and s^* is a regular point), then there exists a unique vector of optimal Lagrange multipliers μ^* such that:

$$\nabla L(s^*,\mu^*) = \begin{pmatrix} \nabla_s L(s^*,\mu^*) \\ \nabla_\mu L(s^*,\mu^*) \end{pmatrix} = \begin{pmatrix} \nabla f(s^*) + \mu^{*T} \cdot \nabla g(s^*) \\ g(s^*) \end{pmatrix} = 0$$

s^* is a regular point, if $\nabla g_1(s^*),\ \ldots\ , \nabla g_r(s^*)$ are linearly independent.

If the vector of optimal Lagrange multipliers μ^* was already known, we could find s^* by solving the unconstrained optimization problem $\min_s L(s,\mu^*)$. Augmented Lagrangian algorithm is a numeric implementation of this idea. It iteratively searches both for s^* and for μ^*, the saddle point of $L(s,\mu)$.

The Lagrangian function (4.4) is approximated by an aggregate function of augmented Lagrangian algorithm:

$$F_p(s,\mu) = f(s) + \sum_{j=1}^{r} \varphi_p(g_j(s),\mu_j), \qquad (4.6)$$

where $\varphi_p(g_j(s),\mu_j)$ is a penalty function such that $\varphi'(0,\mu) = \mu$.

At i'th iteration of augmented Lagrangian algorithm, the unconstrained optimization problem $\min_s L\left(s,\mu^{(i)}\right)$ is solved. Then the vector of Lagrangian multipliers is updated $\mu^{(i+1)}=\varphi_p'\left(g\left(s^{(i)}\right),\mu^{(i)}\right)$.

One possible penalty function for inequality constraints is [6]:

$$\tilde{\varphi}(t)=\begin{cases}\dfrac{t^2}{2}+t, & t\geq -\dfrac{1}{2}\\ -\dfrac{1}{4}\ln(-2t)-\dfrac{3}{8}, & t<-\dfrac{1}{2}\end{cases} \qquad (4.7)$$

An extended version of inequality penalty $\varphi_p\left(t,\mu\right)=\mu\cdot\dfrac{1}{p}\cdot\tilde{\varphi}\left(p\cdot t\right)$, where p is a penalty parameter and μ is the slope at 0, provides a better control over the shape of penalty function and is useful in iterative algorithms such as augmented Lagrangian.

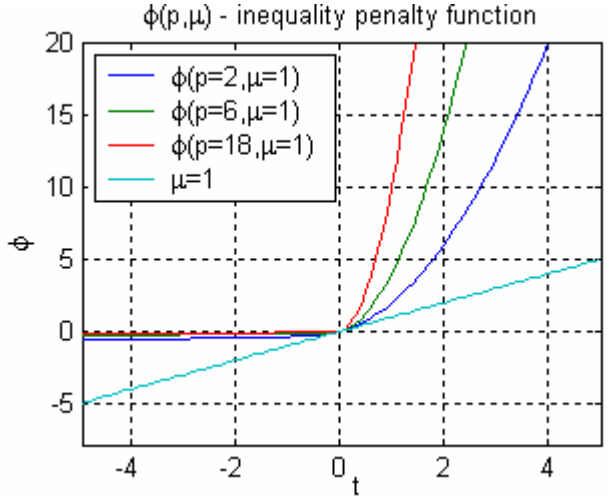

Figure 2: extended inequality penalty function plotted with fixed slope parameter and different penalty parameters

4.3 The actual algorithm

We solve the constrained optimization problem (4.2) in a framework of augmented Lagrangian [5], [6]. At each iteration, an unconstrained problem should be solved, its objective function is an aggregate of augmented Lagrangian (4.6). This is a large-scale problem and appropriate techniques should be used to solve it.

4.4 Truncated Newton method

When dealing with large scale problems, two major obstacles arise. One is that data storage beyond $O(N)$ becomes prohibitively high (N is a number of variables). The other is that computational load needed to find a search direction per iteration should also be

proportional to N. Thus even if second order derivative information is theoretically available, the explicit calculation may not be feasible. This draws Newton and Quasi Newton methods out of consideration since they both use $O(N^2)$ storage and computational load per iteration is $O(N^3)$ and $O(N^2)$ respectively.

Truncated Newton method [2] features linear storage and linear computational load per iteration, while exploiting second order information. All this makes truncated Newton an appealing choice for large scale problems.

At each iteration, a search direction d is found by approximately solving Newton equations $\nabla^2 f \cdot d \approx -\nabla f$ using some iterative method, then a line search along direction d is performed. The line search ensures a global convergence. Newton equations are a linear system with respect to d, which is most effectively optimized by conjugate gradients method. The optimization is "truncated" after a certain (fixed) number of iterations hoping that an approximate solution is sufficiently good. The good thing is that explicit calculation of the Hessian is not needed, only a Hessian-vector product of the form $\nabla^2 f \cdot v$ needs to be calculated for an arbitrary vector v.

For line search we use a back tracking method [2] (Armijio rule). This is an inexact method, and it is significantly faster than exact line search methods.

4.5 Hessian-vector product calculation

A common way of calculating Hessian-vector product is by one sided finite difference approximation. By rearranging the Taylor expansion of $\nabla f(x)$ around x_k

$$\nabla f(x_k + \varepsilon v) = \nabla f(x_k) + \varepsilon \cdot \nabla^2 f(x_k) \cdot v + O(\varepsilon^2)$$

We obtain that

$$\nabla^2 f(x_k) \cdot v = \lim_{\varepsilon \to 0} \frac{\nabla f(x_k + \varepsilon v) - \nabla f(x_k)}{\varepsilon}$$

The resulting finite difference formula is:

$$\nabla^2 f(x_k) \cdot v \approx \frac{\nabla f(x_k + hv) - \nabla f(x_k)}{h} \qquad (4.8)$$

This formula calculates a simple approximation of $\nabla^2 f \cdot v$ at a cost of just one extra gradient computation, assuming that the gradient at x_k we already have.

This formula is good for numerical analysis when an analytic equation cannot be derived. However, it suffers from some numerical problems. The choice of h is rather empirical, a thorough discussion can be found in [2].

Fortunately, an analytic expression for Hessian-vector product can be derived. This product is still calculated at the cost similar to gradient calculation. The Hessian itself is not calculated and only linear storage is needed. See appendices **A.2**, **A.3** for details.

5. Simulations

5.1 Generated data – simulations input

We generated sparse coefficients that represent a synthetic brain activity. These coefficients can be treated as a result of a preprocessing that produces sparse coefficients when applied to signals originating from a natural source. Such preprocessing may incorporate some sort of wavelet transform.

The synthetic sources activity was further used to obtain a sensors response, which is calculated by $X = A \cdot S_{synthetic}$, where $S_{synthetic}$ is an $N : T$ matrix of generated sources, X is an $M : T$ matrix of sensors coefficients and A is an $M : N$ gain matrix of forward model. In these simulations we used MEG forward model publicly available from BrainStorm group [4].

The sensors response was further spoiled by additive background noise. The ultimate goal of the optimization algorithm is to recover synthetic sources in all conditions, with or without background noise.

Finally, the optimization algorithm has several empirical parameters. During the simulations we compared performance with different values of those parameters. Here is a list of empirical values:

- w_2 – weight of sum of absolute values
- w_3 – weight of row-wise L2 norm
- ε – smoothing parameter for row-wise L2 norm
- Stopping conditions of conjugate gradients method
- Stopping conditions of truncated Newton method
- Stopping condition for back-propagation line search
- Stopping conditions of augmented lagrangian method

We generated data for 20 sensors, 200 dipole sources. The generated signals were 50 samples long. Active sources were chosen at random locations and non-zero coefficients were spread with 10% sparsity. The simulations were run with and without additive temporally and spatially white Gaussian noise with $\sigma = 0.01 \cdot \frac{\|x\|_1}{MT}$.

The Figure 3 depicts coefficients of 6 active synthetic sources (on the left) and a plot of 200 synthetic sources all together (active and passive).

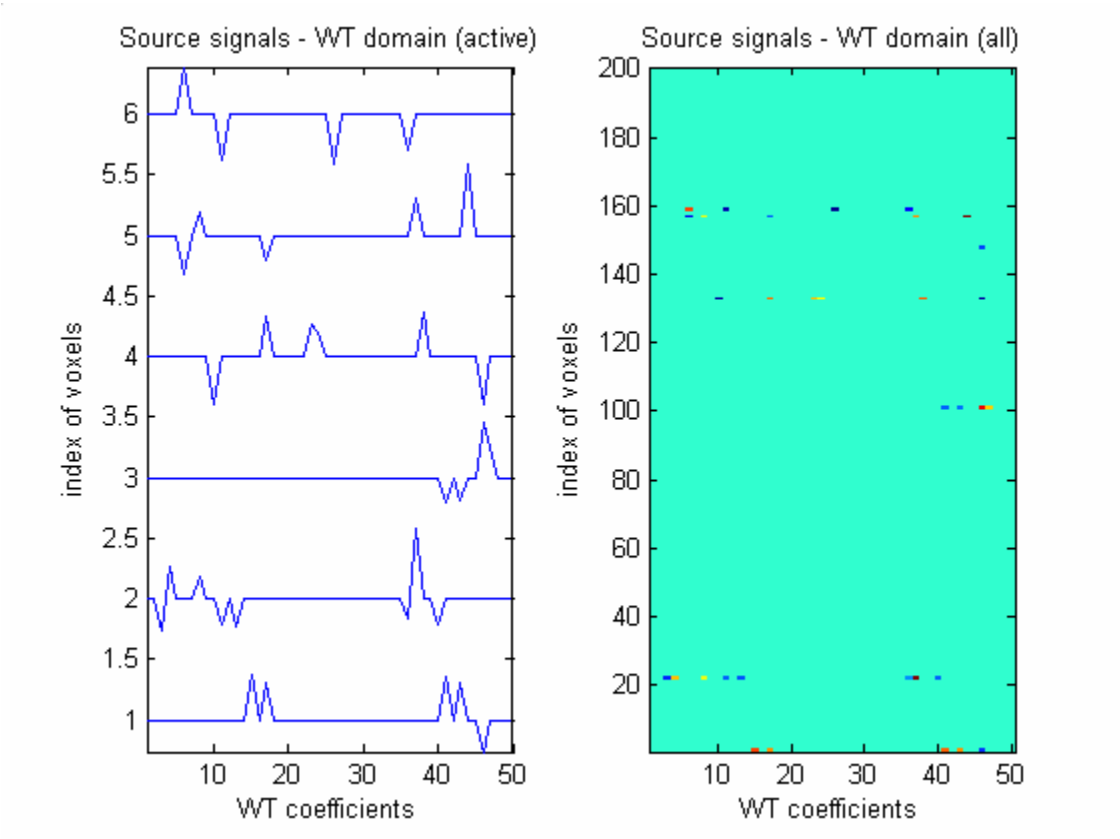

Figure 3: generated source signals – WT coefficents

The dipole sources are randomly spread over the cortex

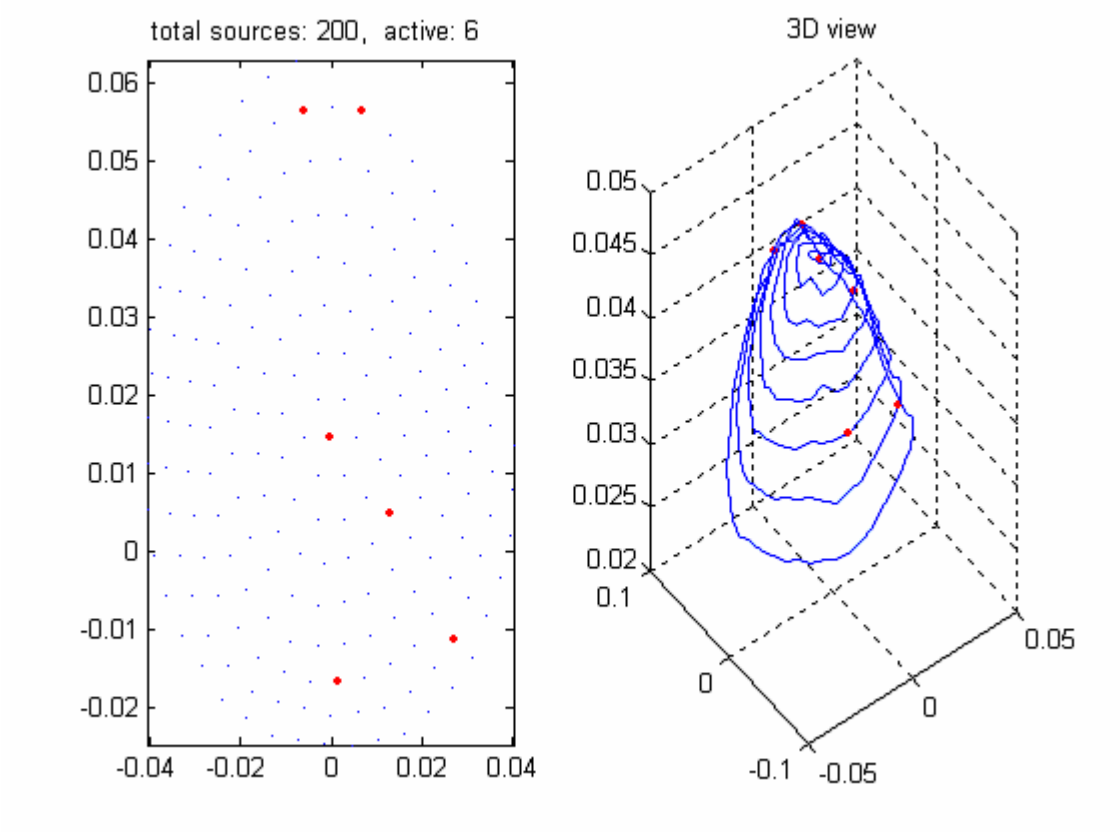

Figure 4: locations of generated dipole sources

The resulting sensors response

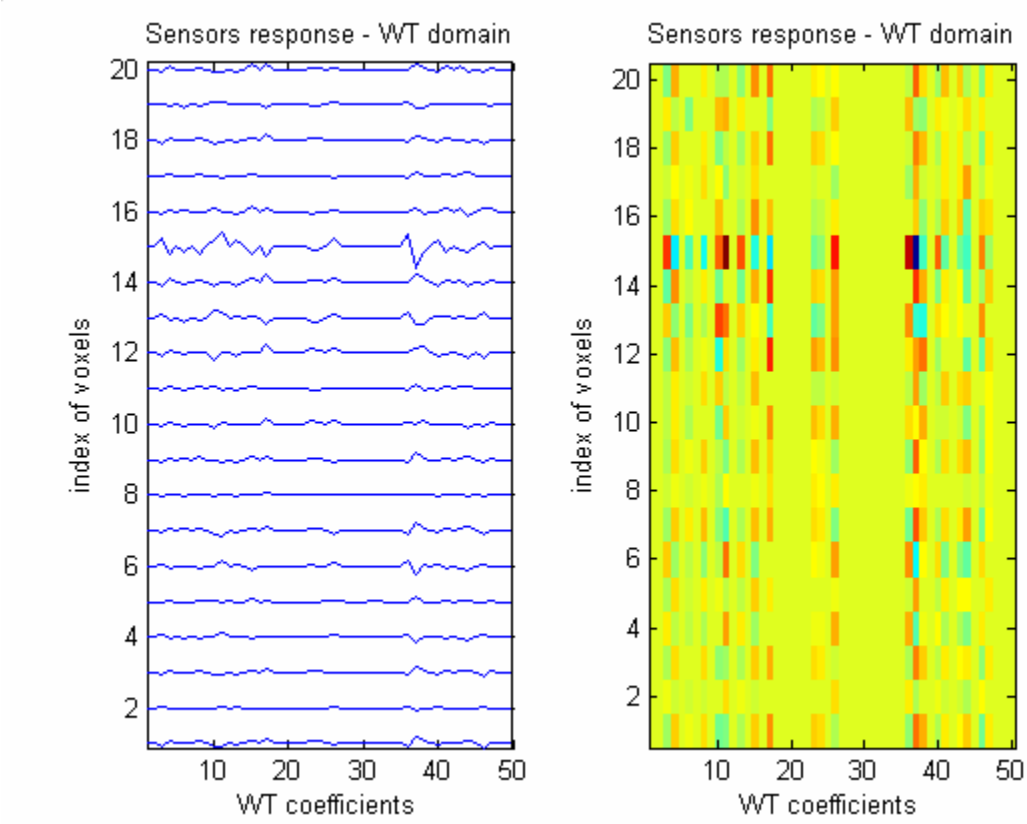

Figure 5: generated sensors signals – WT coefficents

5.2 Sources recovery

The output of the optimization algorithm is a recovered matrix of coefficients of sources signals. If the optimization would obtain a perfect solution, our quest would end there. However, since the original problem is solved only approximately, we need to further refine the solution by deciding what signals are active according to some criteria.

In this work we used a very simple and straight-forward recovery technique: voxels with signal energy above a predefined threshold were chosen to be active sources. Alternatively, voxels with maximum L1 norm could be chosen. Both techniques produce very similar results.

5.3 Results

Figures 6 and 7 depict recovered sources for simulations without noise, while Figures 8 and 9 depict recovered sources for simulations with noise. We can see that without noise all the signals are recovered and localized almost correctly, except for one weak signal

that was not located at all. With noisy simulations sources are recovered with distortions but are localized with quality similar to the former case.

Note, that both simulations rejected a signal with the least number of non-zero coefficients, you can see that signal 3 on Figure 3 has only 3 non-zero coefficients. This is an expected behavior, it happens because of the sources locality assumption, which is incorporated into the optimization problem through the third term of objective function. As a result of this assumption signals with small number of non-zero coefficients are suppressed.

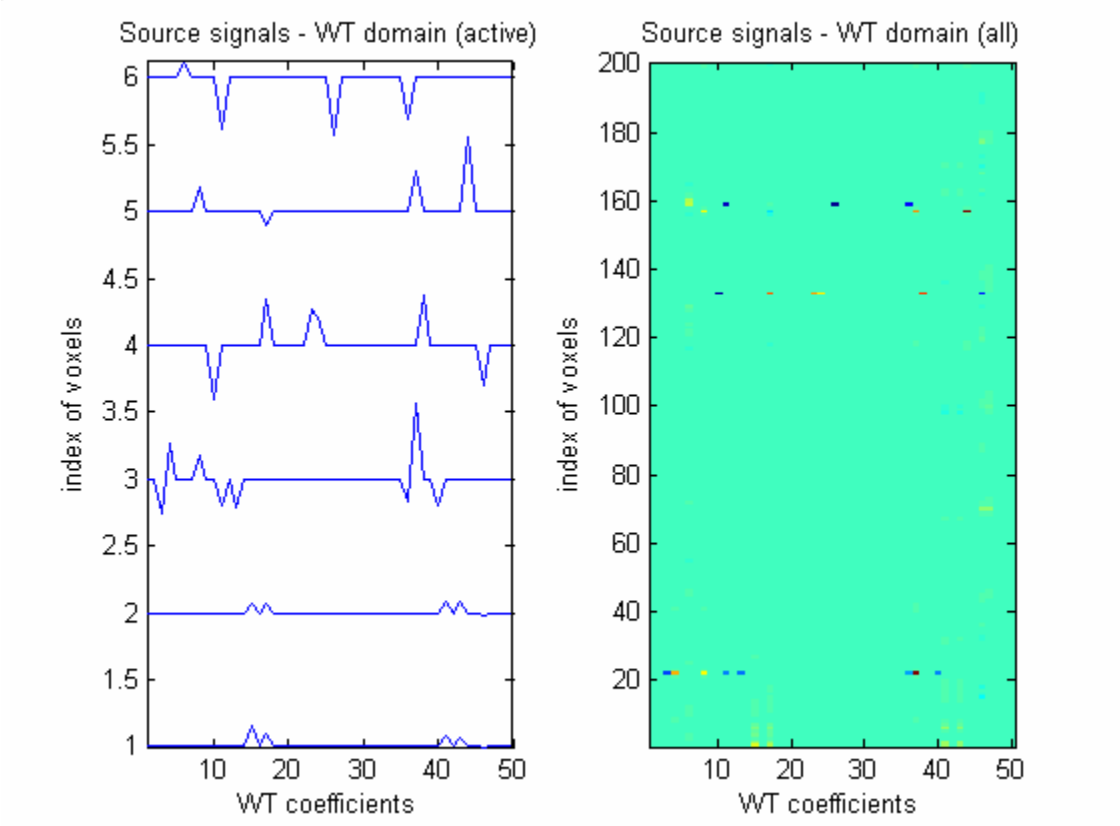

Figure 6: signals of recovered dipole sources - without background noise

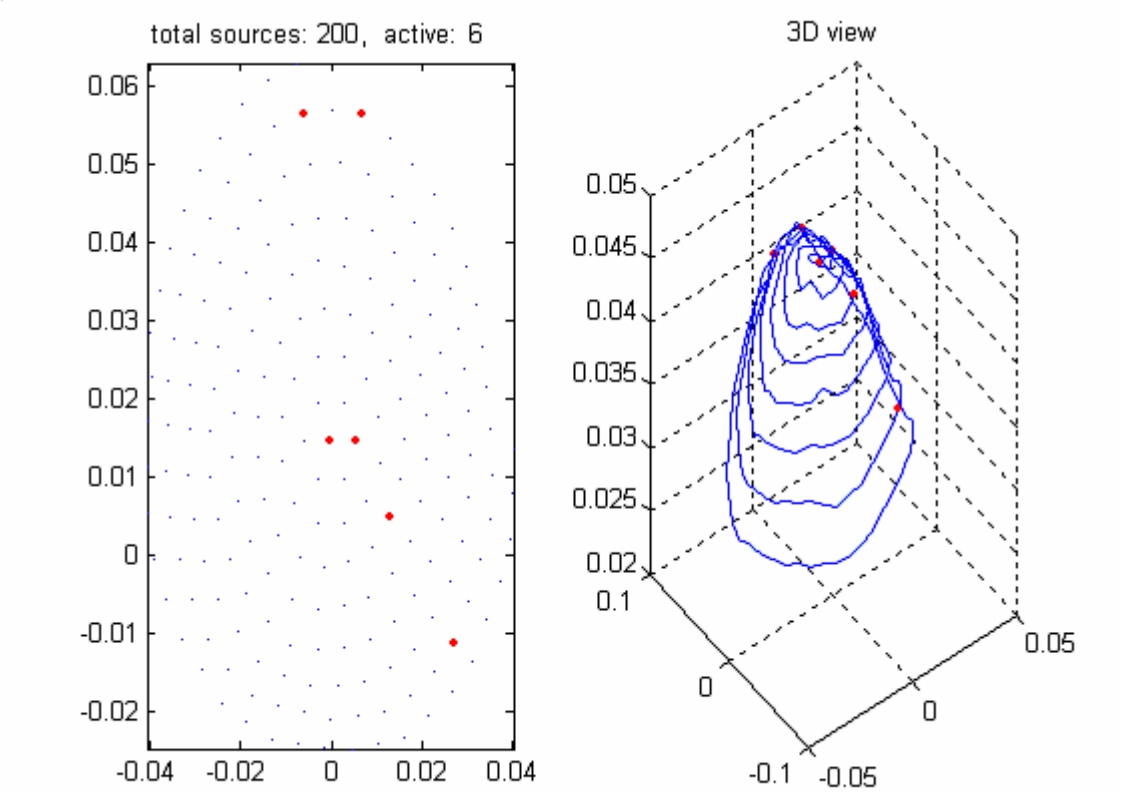

Figure 7: locations of recovered dipole sources - without background noise

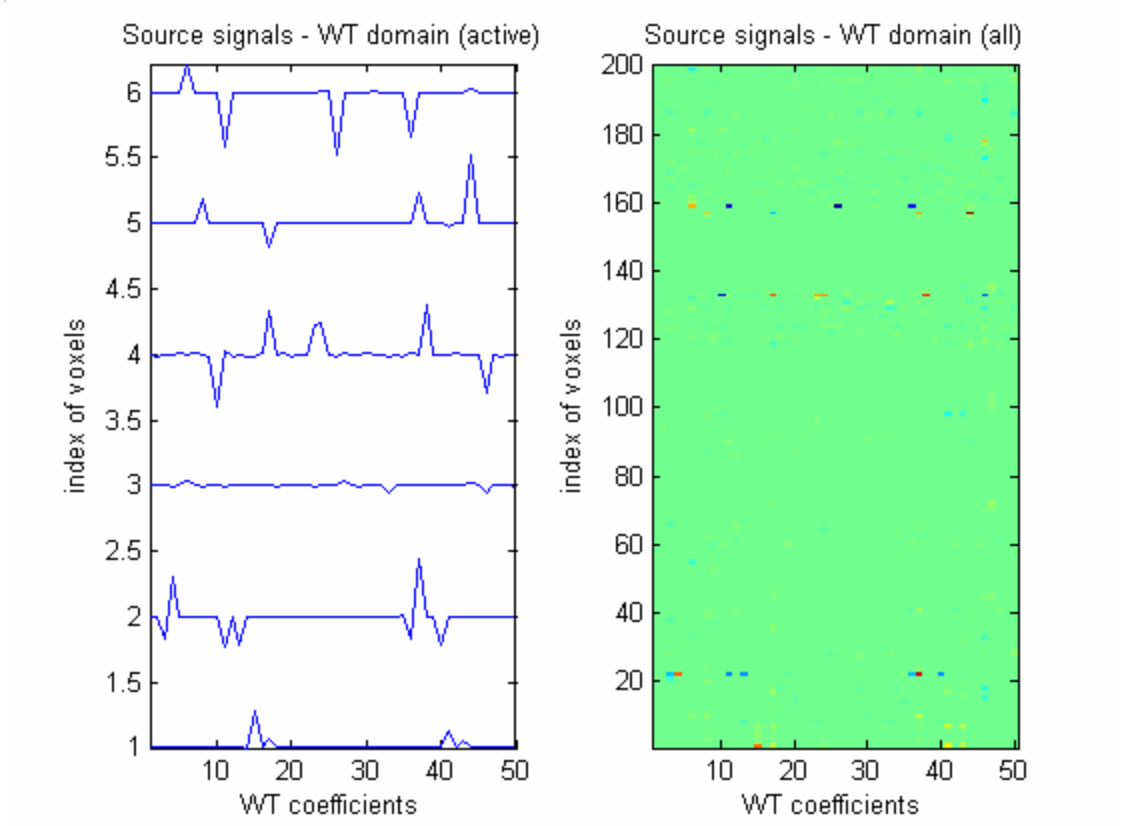

Figure 8: signals of recovered dipole sources - with a background noise

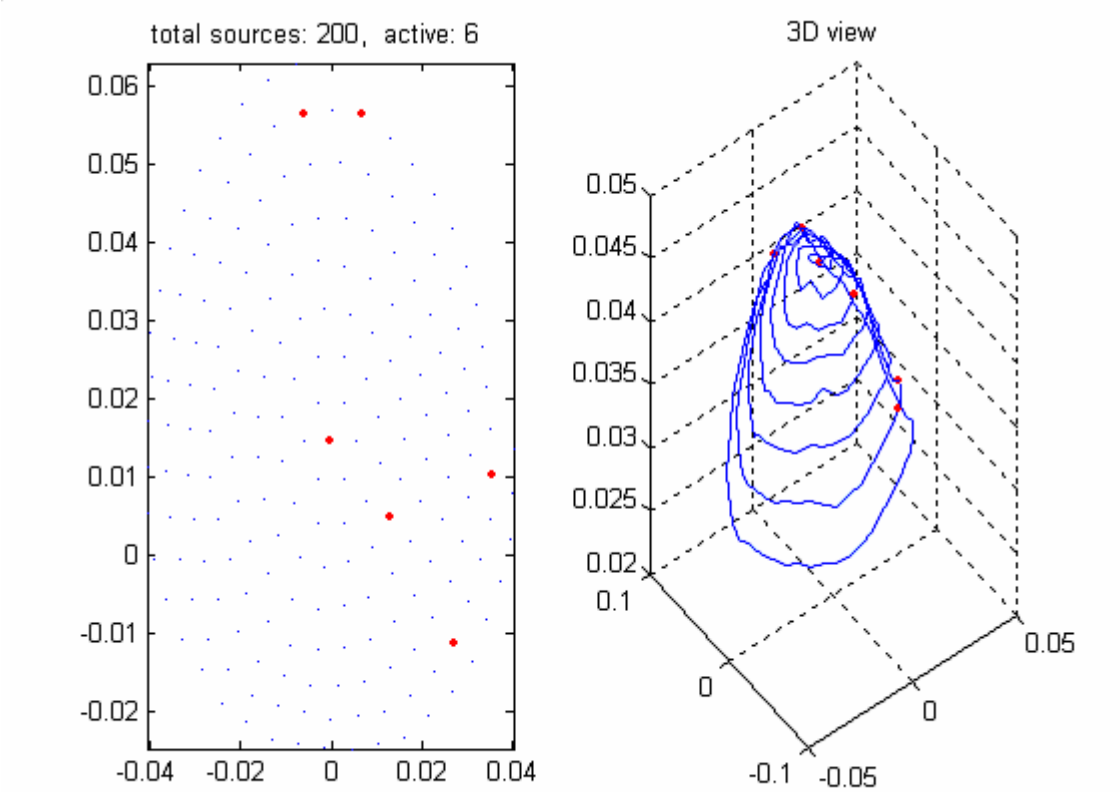

Figure 9: locations of recovered dipole sources - with a background noise

6. Conclusions

We have explored a spatial-temporal approach for a solution of an inverse MEG/EEG problem. The solution was based on two physiological assumptions: sparsity of wavelet coefficients of the signals and spatial locality of sources. We have formulated a corresponding optimization problem that incorporates the physiological assumptions and forces the forward model response to be close to the observed data.

The simulation results confirm that the framework of optimization problem allows to efficiently utilize the available temporal and spatial information. The simulations showed that the solution of optimization problem is close to generated sources signals even when sensors readings are spoiled with noise. Sparsity assumption proved to be a driving force that takes an optimization to the right solution. The simulations also showed that as a result of sources locality assumption, signals with small number of coefficients might be suppressed. Recall, that this assumption is incorporated into optimization problem through the third term of objective function.

A clear advantage of spatial-temporal approach is that it utilizes as much of the available information as possible. And a framework of optimization problem allows incorporating physiological and physical assumptions and considerations into a single model.

A. Appendices

A.1 Conjugate gradients algorithm

We use conjugate gradients method to approximately solve a set of Newton equations $\nabla^2 f \cdot d \approx -\nabla f$. This set of equations is a linear system of the form $Ax = b$, where $A \equiv \nabla^2 f$, $b \equiv -\nabla f$, $x \equiv d$.

Here is the outline of the method:

Initialize:

$x_0 = 0$ *// initial guess*

$r_0 = b - Ax$ *// residual that is minimized*

$v_0 = r_0$

$\rho_0 = 1$

$\beta = 0$

Repeat for $k = 0,1,2,...$ *until* $\|\rho_k\|_2$ *is sufficiently small:*

$\rho_{k+1} = r_k^T \cdot r_k$

$if(k > 0), \quad set \ \beta = \rho_{k+1} / \rho_k$

$v_{k+1} = r_k + \beta \cdot v_k$

$$\alpha = \frac{\rho_{k+1}}{v_{k+1}^T \cdot A \cdot v_{k+1}}$$

$x_{k+1} = x_k + \alpha \cdot v_{k+1}$

$r_{k+1} = r_k - \alpha \cdot A \cdot v_{k+1}$

end

Although all variables are written with an index of iteration, only the current values of variables have to be stored, except for inner product $\rho_k = r_{k-1}^T \cdot r_{k-1}$ which has to be remembered from previous iteration.

Note, that a Hessian-vector product Ax should be calculated at each step of the method. This is a potential bottle neck and it should be implemented efficiently. The exact calculation of Hessian-vector product can be found in A.3 and A.4 .

It can be shown, that conjugate gradients method converges in a number of steps equal to the number of distinct eigen values of the matrix *A*. When dealing with MEG inverse problem with *M* sensors and *N* sources, $M < N$, there are at most *M* degrees of freedom,

i.e. $rank(A) \leq M$. This consideration can be used when setting the maximum number of iterations of the conjugate gradients method.

A.2 Gradient, Hessian and Hessian-vector product of the objective function

The objective function of optimization problem 4.2 can be rewritten as a sum of standard QCQP objective function $f_q(S)$ and a smoothed row-wise L2 norm $f_{L2}(S)$:

$$f_{obj}(S)=\frac{1}{2}\cdot\|AS-X\|_F^2+w_1\cdot\|vec(S)\|_1+w_2\cdot\sum_{i=1}^{N}\sqrt{\|S_i\|_2^2+\|S_{i+N}\|_2^2+\varepsilon}=f_q(S)+f_{L2}(S)$$

Thus gradient, Hessian and Hesian-vector product can also be expressed as sums:

$$\nabla f_{obj}(S)=\nabla f_q(S)+\nabla f_{L2}(S)$$

$$\nabla^2 f_{obj}(S)=\nabla^2 f_q(S)+\nabla^2 f_{L2}(S)$$

$$\nabla^2 f_{obj}(S)\cdot v=\nabla^2 f_q(S)\cdot v+\nabla^2 f_{L2}(S)\cdot v$$

Gradient and Hessian of $f_q(S)$

Recall that A denotes a block diagonal matrix of T blocks, each block equals A. The function $f_q(S)$ can be rewritten in the following vector form:

$$f_q(s)=\frac{1}{2}\cdot\|\mathrm{A}\cdot s-x\|_2^2+w_1\cdot 1^T\cdot s=\frac{1}{2}\cdot(\mathrm{A}\cdot s-x)^T\cdot(\mathrm{A}\cdot s-x)+w_1\cdot\underline{1}^T\cdot s=$$

$$=\frac{1}{2}\cdot\left[(\mathrm{A}\cdot s)^T(\mathrm{A}\cdot s)-(\mathrm{A}\cdot s)^T x-x^T(\mathrm{A}\cdot s)+x^T x\right]+w_1\cdot\underline{1}^T\cdot s=$$

$$=\frac{1}{2}\cdot s^T\mathrm{A}^T\mathrm{A}s+\left(w_1\cdot 1^T-x^T\mathrm{A}\right)\cdot s+\frac{1}{2}\cdot x^T x$$

$\mathrm{A}^T\mathrm{A}$ is a symmetric matrix $\Rightarrow$ $\frac{1}{2}\left(\mathrm{A}^T\mathrm{A}+(\mathrm{A}^T\mathrm{A})^T\right)=\mathrm{A}^T\mathrm{A}$

$$\nabla_s f_q(s)=\frac{1}{2}\left(\mathrm{A}^T\mathrm{A}+(\mathrm{A}^T\mathrm{A})^T\right)\cdot s+\left(w_1\cdot 1^T-x^T\mathrm{A}\right)^T=\mathrm{A}^T\mathrm{A}\cdot s+w_1\cdot\underline{1}-\mathrm{A}^T\cdot x$$

$$H_q\equiv\nabla_s^2 f_q(s)=\mathrm{A}^T\mathrm{A}$$

Hessian-vector product is calculated by

$$H_q\cdot v=\mathrm{A}^T\mathrm{A}\cdot v$$

where V is an arbitrary $2N\times T$ matrix and $v=vec(V)$

An appropriate line of MATLAB code is: `Hqv=A'*(A*v)`

Note that `A'*(A*v)` is much more efficient than `(A'*A)*v`, where A is an *MT:2NT* matrix and `v` is an *2NT:1* column vector. The former expression requires $(2NT)^2$ multiplications, and the latter requires $(2NT)^3$ multiplications.

Gradient and Hessian of smoothed row-wise L2 norm $f_{L2}(S)$

$$f_{L2}(S)=\sum_{i=1}^{N}\sqrt{\|S_i\|_2^2+\|S_{i+N}\|_2^2+\varepsilon}=\sum_{i=1}^{N}\left(\sqrt{S_i{}^T S_i+S_{i+N}^T S_{i+N}+\varepsilon}\right)=\sum_{i=1}^{N} l2(S)$$

Where we denote $l2(S)=\sqrt{S_i{}^T S_i+S_{i+N}^T S_{i+N}+\varepsilon}$. Obviously, a gradient of $f_{L2}(S)$ is just a sum of gradients of $l2(S)$, and a Hessian of $f_{L2}(S)$ is a block diagonal matrix, where *i*'th diagonal block is a Hessian of $l2_i(S)$.

Let's calculate the gradient, the Hessian and Hessian-vector product for $l2(S_i)$.

$l2_i(S)=\varphi(h_i(S))$,
$\varphi(\cdot)=\sqrt{\cdot}$ is a scalar to scalar mapping.
$h_i(S)=\begin{cases} S_i{}^T S_i+S_{i+N}^T S_{i+N}+\varepsilon, & 1\le i\le N \\ S_i{}^T S_i+S_{i-N}^T S_{i-N}+\varepsilon, & N+1\le i\le 2N \end{cases}$ is a vector to scalar mapping

derivatives of $h(S_i)$:

$$dh_i=2S_i{}^T dS_i,\qquad \nabla h=2S_i,\qquad \nabla^2 h=2\mathrm{I}\quad (\mathrm{I}-\text{identity matrix})$$

derivatives of φ:

$$\varphi'=\frac{1}{2}h^{-1/2},\qquad \varphi''=-\frac{1}{4}h^{-3/2}$$

gradient:

$$d(l2_i)=\varphi'\cdot dh_i=\varphi'\cdot\nabla h_i{}^T\cdot dS_i=\frac{1}{2}(h_i(S))^{-1/2}\cdot 2S_i{}^T\cdot dS_i=(h_i(S))^{-1/2}\cdot S_i{}^T\cdot dS_i$$

$$\nabla l2_i(S)=(h_i(S))^{-1/2}\cdot S_i$$

Hessian:

$$d(\nabla l2)=d(\varphi'\cdot\nabla h)=d\varphi'\cdot\nabla h+\varphi'\cdot d\nabla h=\varphi''\cdot\nabla h^T dS_i\cdot\nabla h+\varphi'\cdot\nabla^2 h\cdot dS_i$$
$$=\varphi''\cdot\nabla h\cdot\nabla h^T dS_i+\varphi'\cdot\nabla^2 h\cdot dS_i$$

$$H_{L2}(S_i)\equiv\nabla^2 l2(S_i)=\varphi''\cdot\nabla h\cdot\nabla h^T+\varphi'\cdot\nabla^2 h$$
$$=-(h_i(S))^{-3/2}\cdot S_i{}^T S_i+(h_i(S))^{-1/2}\cdot\mathrm{I}=p_i\left(\mathrm{I}-p_i{}^2 S_i S_i{}^T\right)$$

where $p_i\equiv\varphi(h_i(S))^{-1}=(h_i(S))^{-1/2}$

Hessian-vector product:

$$H_{L2}(S_i)\cdot V_i = p_i\left(\mathrm{I} - {p_i}^2 S_i {S_i}^T\right)\cdot V_i = p_i\cdot V_i - {p_i}^3 S_i\left({S_i}^T V_i\right)$$

where V is an arbitrary $2N\times T$ matrix, $v = vec(V)$ and V_i is an i'th row of V

By combining the expressions obtained for $f_q(S)$ and $f_{L2}(S)$ we get the following expressions for $f_{obj}(s)$:

$$\nabla f_{obj}(S) = \nabla f_q(S) + \begin{bmatrix} \nabla l2(S_1) \\ \nabla l2(S_2) \\ \dots \\ \nabla l2(S_{2N}) \end{bmatrix}_{2NT\times 1} = \mathrm{A}^T\mathrm{A}\cdot s + w_1\cdot \underline{1} - \mathrm{A}^T\cdot x + w_2\cdot \begin{bmatrix} \left({S_1}^T S_1 + {S_{N+1}}^T S_{N+1} + \varepsilon\right)^{-1/2}\cdot S_1 \\ \dots \\ \left({S_N}^T S_N + {S_{2N}}^T S_{2N} + \varepsilon\right)^{-1/2}\cdot S_N \\ \left({S_1}^T S_1 + {S_{N+1}}^T S_{N+1} + \varepsilon\right)^{-1/2}\cdot S_{N+1} \\ \dots \\ \left({S_N}^T S_N + {S_{2N}}^T S_{2N} + \varepsilon\right)^{-1/2}\cdot S_{2N} \end{bmatrix}$$

$$\nabla^2 f_{obj}(S) = \nabla^2 f_q(S) + \begin{bmatrix} \nabla^2 l2(S_1) & 0 & 0 & 0 \\ 0 & \nabla^2 l2(S_2) & 0 & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \nabla^2 l2(S_{2N}) \end{bmatrix}_{2NT\times 2NT}$$

$$= \mathrm{A}^T\mathrm{A} + w_2\cdot \begin{bmatrix} p_1\left(\mathrm{I} - {p_1}^2 S_1 {S_1}^T\right) & 0 & \dots & \dots \\ 0 & p_2\left(\mathrm{I} - {p_2}^2 S_2 {S_2}^T\right) & 0 & \dots \\ \dots & 0 & \dots & 0 \\ \dots & \dots & 0 & p_{2N}\left(\mathrm{I} - {p_{2N}}^2 S_{2N} {S_{2N}}^T\right) \end{bmatrix}$$

$$\nabla^2 f_{obj}(S)\cdot v = \nabla^2 f_q(S)\cdot v + \nabla^2 f_{L2}(S)\cdot v = \mathrm{A}^T\mathrm{A}\cdot v + w_2\cdot \begin{bmatrix} p_1\cdot V_1 - {p_1}^3 S_1\left({S_1}^T V_1\right) \\ p_2\cdot V_2 - {p_2}^3 s_2\left({s_2}^T V_2\right) \\ \dots \\ p_{2N}\cdot V_{2N} - {p_{2N}}^3 S_{2N}\left({S_{2N}}^T V_{2N}\right) \end{bmatrix}_{2NT\times 1}$$

where V is an arbitrary $2N\times T$ matrix, $v = vec(V)$, v_i is an i'th row of V and $p_i \equiv \varphi(h_i(S))^{-1} = (h_i(S))^{-1/2}$

A.3 Aggregate function for augmented Lagrangian

General expressions of the aggregate function and its derivatives:

$$F_p(x) = f_{obj}(x) + \sum_{j=1}^{r} \varphi_p\left(g_j(x), \mu_j\right)$$

$$\nabla F_p(x) = \nabla f_{obj}(x) + \sum_{j=1}^{r} \varphi_p'\left(g_j(x), \mu_j\right) \cdot \nabla g_j(x) = \nabla f_{obj}(x) + \nabla \underset{\rightarrow}{g}(x) \cdot \underset{\rightarrow}{\varphi_p'}\left(\underset{\rightarrow}{g}(x), \underset{\rightarrow}{\mu}\right)$$

$$\nabla^2 F_p(x) = \nabla^2 f_{obj}(x) + \sum_{j=1}^{r} \varphi_p''\left(g_j(x), \mu_j\right) \cdot \nabla g_j(x) \cdot \nabla g_j(x)^T + \sum_{j=1}^{r} \varphi_p'\left(g_j(x), \mu_j\right) \cdot \nabla^2 g_j(x)$$

After substituting the gradient and the Hessian of the constraints vector $\underset{\rightarrow}{g}(s) = -s$

$$\nabla \underset{\rightarrow}{g}(s) = \left[\nabla g_1(s) \quad \nabla g_2(s) \quad \ldots \quad \nabla g_{2NT}(s)\right] = -\mathrm{I}$$

$$H_{g_j}(s) \equiv \nabla^2 g_j(s) = 0\,, \quad 1 \le j \le 2NT$$

We obtain the following expressions:

$$\nabla F_p(s) = \nabla f_{obj}(s) - \left[\varphi_p'(g_1(s), \mu_1) \quad \varphi_p'(g_2(s), \mu_2) \quad \ldots \quad \varphi_p'(g_{2NT}(s), \mu_{2NT})\right]^T$$

$$H(s) = \nabla^2 F_p(s) = \nabla^2 f_{obj}(s) + \begin{bmatrix} \varphi_p''(g_1(s), \mu_1) & 0 & \ldots & 0 \\ 0 & \varphi_p''(g_2(s), \mu_2) & \ldots & \ldots \\ \ldots & \ldots & \ldots & 0 \\ 0 & \ldots & 0 & \varphi_p''(g_{2NT}(s), \mu_{2NT}) \end{bmatrix}$$

Hessian-vector product for the aggregate function:

$$H(s) \cdot v = \nabla^2 f_{obj}(s) \cdot v + \begin{bmatrix} \varphi_p''(g_1(s), \mu_1) \cdot v_1 \\ \varphi_p''(g_2(s), \mu_2) \cdot v_2 \\ \ldots \\ \varphi_p''(g_{2NT}(s), \mu_{2NT}) \cdot v_{2NT} \end{bmatrix}$$

where V is an arbitrary $2N \times T$ matrix, $v = vec(V)$ and v_i is an *i*'th element of *v*.

The same expression can be derived using the technique described in [8] by introducing a differential operator $\Re_d(f(s)) = \dfrac{\partial}{\partial r} f(s + rv)\Big|_{r=0}$ and applying this operator to $\nabla f(s)$. It can be easily shown that $\Re_d(\nabla f(s)) = H(s) \cdot v$.

6. References

[1] M. Hamalainen, R. Hari, R. J. IImoniemi, J. Knuutila, and O. V. Lounasmaa, "Magnetoencephalography – theory, instrumentation, and applications of noninvasive studies of the working human brain", Rev.Mod. Phys., vol. 65, pp. 413–497, Mar. 1993.

[2] Stephen G. Nash and Ariela Sofer, "Linear and Nonlinear Programming", McGraw-Hill, New York, 1996.

[3] J.P. Ary, S.A. Klein, and D.H. Fender, "Location of sources of evoked scalp potentials: corrections for skull and scalp thickness", IEEE Trans. Biomed. Eng. 28:447-452, 1981.

[4] Brain Storm group, public data of MEG experiments with a human skull phantom http://neuroimage.usc.edu/.

[5] Dimitri P. Bertsekas, "Nonlinear Programming", 2'nd edition, Athena Scientific, 1999.

[6] A. Ben-Tal, M. Zibulevsky, "Penalty/Barrier Multiplier Methods for Convex Programming Problems", SIAM Journal on Optimization v. 7 # 2, pp. 347-366, 1997.

[7] R. M. Leahy, J. C. Mosher, M. E. Spencer, M. X. Huang, and J. D.Lewine, "A study of dipole localization accuracy for MEG and EEG using a human skull phantom", Electroencephalogr. Clin. Neurophysiol., vol. 107, pp. 159–173, Aug. 1998.

[8] Barak A. Pearlmutter, "Fast Exact Multiplication by the Hessian", Neural Computation, 1994.

[9] Stephane Mallat, "A Wavelet Tour of Signal Processing", 2'nd edition, Academic Press, 1999.