Parallel De-Noising of Biological Signals

Emily Sosin

Final Paper for ENEE499

May 19, 2008

Introduction

Currently, Dr. Jonathan Simon is involved in research on how the human auditory cortex processes complex sounds in the brain. To trace the neural signals produced by the brain, he is using magnetoencephalography (MEG). MEG allows one to read the neural signals in close to real time without having to trade off on spatial resolution. In the Computational Sensorimotor Systems Lab, the MEG works by placing 157 sensors across one's head and recording the magnetic fields produced by the brain. However, the neural signal picked up by each sensor is a relatively small when compared with the noise the sensors also record. In addition to the desired neural signal, the MEG picks up magnetic signals from external noise, background brain noise as well as other biological signals. This means it is hard to identify the desired signal apart from the noise.[1] To improve the low signal to noise ratio, algorithms were developed to better read the desired signal.

There are currently two algorithms in place that filter out noise, TSPCA and SNS. De-noising based on time-shift principal components analysis (TSPCA)[2] is the technique used to remove the external noised picked up by the neural sensors. This external noise can be caused by the magnetic fields produced from nearby power lines, elevators, or cars to name a few. While there are several methods in which one can remove external noise, TSPCA is used here because it does not require the spectral or spatial filters that other techniques require, in order to keep the signal undistorted. The TSPCA algorithm requires at least three external reference signals, one for each spatial component of the noise signal. These sensors are placed a good distance from the subject wearing the neural sensors.

To understand the algorithm, one must realize that the environmental noise being read by the reference sensor is the same noise being read by the neural sensors, except for scalar multiplication or convolution from possible delays or filtering. To begin to remove the environmental noise, one must first time shift the three reference channels by a series of multiples of the sampling period. This is equivalent to putting the reference channels through a

---

[1] Ahmar, Nayef. Da Vinci's Encephalogram: in Search of Significant Brain Signals. Department of Electrical and Computer Engineering. College Park, MD: Digital Repository At the University of Maryland, 2005. 1-71.

[2] De Cheveigné, Alain, and Jonathan Z. Simon. "Denoising Based on Time-Shift PCA." Journal of Neuroscience Methods 165 (2007): 297-305.

finite impulse response (FIR) filter. This is done because it has been shown that residual noise power drops significantly as the number taps increases. Increasing the number of taps to the FIR filter or equivalently by time shifting by a larger series of multiples only removes the power from the noise signal and increase the signal to environmental noise ratio.

After the reference channels are time-shifted, principal component analysis (PCA) is applied to create an orthogonal basis with a number of coordinates equal to the number of neural channels multiplied by the sampling period. Principal component analysis is a technique used to identify patterns or trends in the data that would be hard to notice in its original form. It takes the greatest variance of any projection lies along the first coordinate of the basis, the second greatest variance lies on the second coordinate and so on. When one of the neural sensor channels is projected onto this basis, a vector expressing the environmental noise affecting that particular channel will be produced. In other words, when we project a neural channel onto this orthogonalized basis, we are given a vector, which is more closely related to the patterns found in the reference channel. The vector shows the components of the neural signal that best approximate the environmental noise picked up by the reference channels. Once the environmental noise in the channel is determined, the last step is to simply subtract the vector from that particular neural sensor channel. This process is of course repeated for each channel.

Once environmental noise is removed from the signal, the next step is to remove the noise produced from neural sensors themselves, using the second algorithm known as sensor noise suppression (SNS)[3]. To understand this algorithm, one must make the two assumptions about the brain activity signals and the sensors themselves. Firstly, one neural sensor picks up on several brain sources, and each individual brain source signal is read by several sensors. With this assumption, one can say that each sensor signal is linearly dependent. This means we equate each sensor or channel to a weighted sum of all of the other channels. The second assumption, we make is that the noise produced by a sensor is unique to that channel and is not correlated to the noise created by the other sensors.

After understanding these assumptions, we can conceptualize the algorithm. The first step is to isolate the channel that will be de-noised and then orthogonalize all the other channels by applying PCA. After, project the isolated channel onto the newly created basis. As described

[3] De Cheveigné, Alain, and Jonathan Z. Simon. "Sensor Noise Suppression." Journal of Neuroscience Methods 168 (2008): 195-202.

above, the PCA method will identify the principal components of a matrix, thus depicting a clearer picture of the behavior of a signal. When we project the isolated signal onto this basis, it will in turn produce a new vector that has values more closely resembling the patterns found in the across all channels. Because the noise is not correlated across channels, it will not be recognized as a pattern across all channels. This projection is the de-noised channel and will replace original channel. This process is repeated for each channel.

After gaining a clearer understanding of how the algorithms work, it became clear that for a large set of data, it would take a very long time to compute all these calculations. When both algorithms are run serially in MATLAB, it can take up to an hour to de-noise 30 minutes worth of data. The primary goal of this independent research project was redesign the algorithms; so neural signals could be de-noised in parallel. Meaning the data would be subdivided into blocks, and the de-noising algorithms would be run on each block simultaneously. The intention being to reduce the time it takes to de-noise the MEG neural signal data.

Approach

The first step in this project was to write a program in MATLAB that would divide the data into blocks. Each of these blocks had to be saved under a file name that identified its place within the larger data. In other words, if there was a block of data that had 400 time samples and 4 channels (400 x 4) and it was to be split into blocks of size 100 in length, samples from 1 to 100 would be saved under the heading “Data_0.mat”, while samples 101 to 200 would be titled “Data_1.mat”. This file was aptly named PreDivide.m.

In addition to being able to split the data into blocks, it was also important to be able to recombine the de-noised blocks into one collective set of data. This program entitled Combine.m would load each block of data, saved after de-noising, and store it into a larger matrix in accordance to the file number it was stored under.

After being able to divide and recombine the blocks of data, the next step was to understand how to de-noise each of these blocks simultaneously. This part of the project required an understanding of how the High Performance Computer Cluster (HPCC)[4] worked. HPCC essentially has one scheduling computer tied to several slave computers. The slave computers are

[4] "OIT High Performance Computing Cluster (HPCC)." Office of Information Technology. 27 Sept. 2007. 1 Feb. 2008 <http://www.oit.umd.edu/hpcc/>.

called nodes and on each node up to four processors can run simultaneously. Meaning if someone requested two nodes, they could be processing up to 8 blocks simultaneously. The main computer will read through the MATLAB code while assigning blocks of data and tasks to each of processors.

The first part of understanding the HPCC was to be able to identify each processor by number in MATLAB. This was important because later the processor identified as number 1 should run the first block of data, processor 2 should de-noise the second block, and so on. A small test code was written using 12 processors (3 nodes each with 4 processors). The only output of this code was to display the processor number. Each processor was labeled "vpid" and was numbered in order from 0 to 11. The test code then called on MATLAB asking it to run a program that simply echoed the "vpid" number. So essentially, processor 1 would run a program printing out the number 1, as would the other processors according to their assigned numbers.

After having an understanding of how the parallel computing cluster works, the next stage was to introduce a more complicated MATLAB task for the HPCC to distribute. The PreDivide.m file split a minimized set of artificial data into 40 different blocks. All forty blocks were transferred to the HPCC using a secure file transfer application (SFTP), called Fugu. In addition, to the data files, it was also necessary to transfer the files previously written for de-noise using SNS algorithm. Lastly, the current MATLAB code was rewritten, so instead of just printing out the processor number, it used that "vpid" value to open the block with corresponding number and de-noise the section of data, before saving to file with matching digit in the title. This MATLAB file was called Denoise.m.

The final steps to de-noising the data in parallel involved adding the TSPCA algorithm into the Denoise.m file. The TSPCA algorithm would run first, in such a way that the output data from TSPCA would be the input for the SNS. The concluding step was to include a command in the Unix shell to call on the PreDivide.m file before the de-noising file and the Combine.m file afterwards. Both PreDivide.m and Combine.m would not run in parallel, but would allow the user to simply run the file and only be required to transfer in the data directly from the source.

The conclusive step was replacing the artificial data with a real data taken from the MEG. The real data contained 252560 samples from 157 neural channels and 3 reference channels, producing a matrix that was 252560 by 160. Using real data was helpful in helped demonstrating a signal was truly being de-noised.

Conclusion

Once the real data, the de-noised data could be compared versus the original data so as to show that the algorithms were performing, as they should. The actual data was split into 56 different blocks. Each block was de-noised in parallel before being reunited in a 252560 by 157 matrix. (It is three columns shorter since the reference channels are removed). Plotting the results of one channel will show if the de-noising algorithm did in fact remove the environmental and sensor noise. Figure 1 below compares the original signal with the final de-noised signal (after both TSPCA and SNS). The graph shows how the de-noising affected one channel. The data is plotted versus frequency because it easier to see the difference in amplitude in the frequency domain. Clearly, the original data depicted by in green is larger than the de-noised data in blue. The second graph (Figure 2) compares the signal after it has only been de-noised with TSPCA and then after it has been completely de-noised by the TSPCA and the SNS algorithms. While the difference on this graph is harder to see, the data de-noised by both algorithms (depicted in red) is in fact smaller in amplitude of the data de-noised only be TSPCA (depicted in blue)

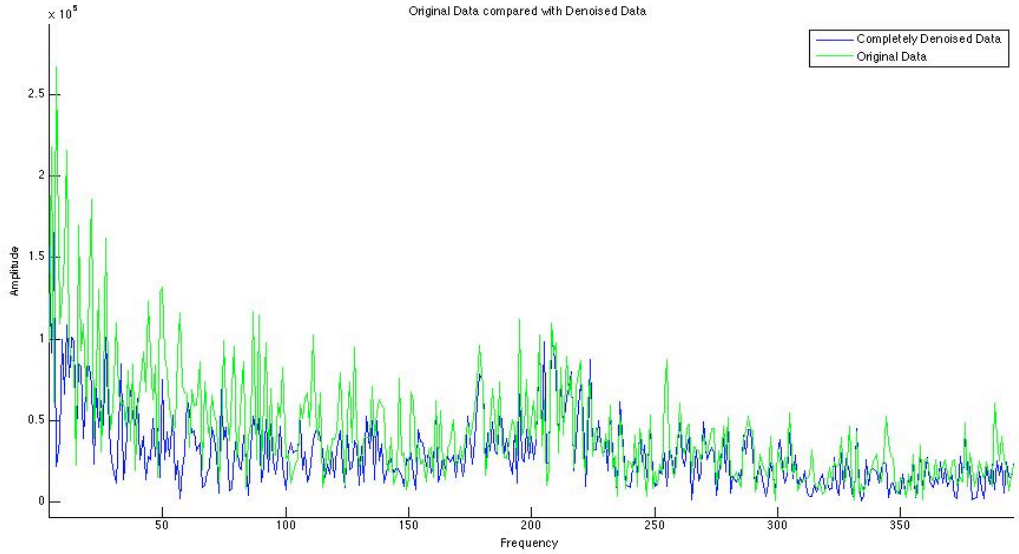

Figure 1. Original Data versus De-noised Data.

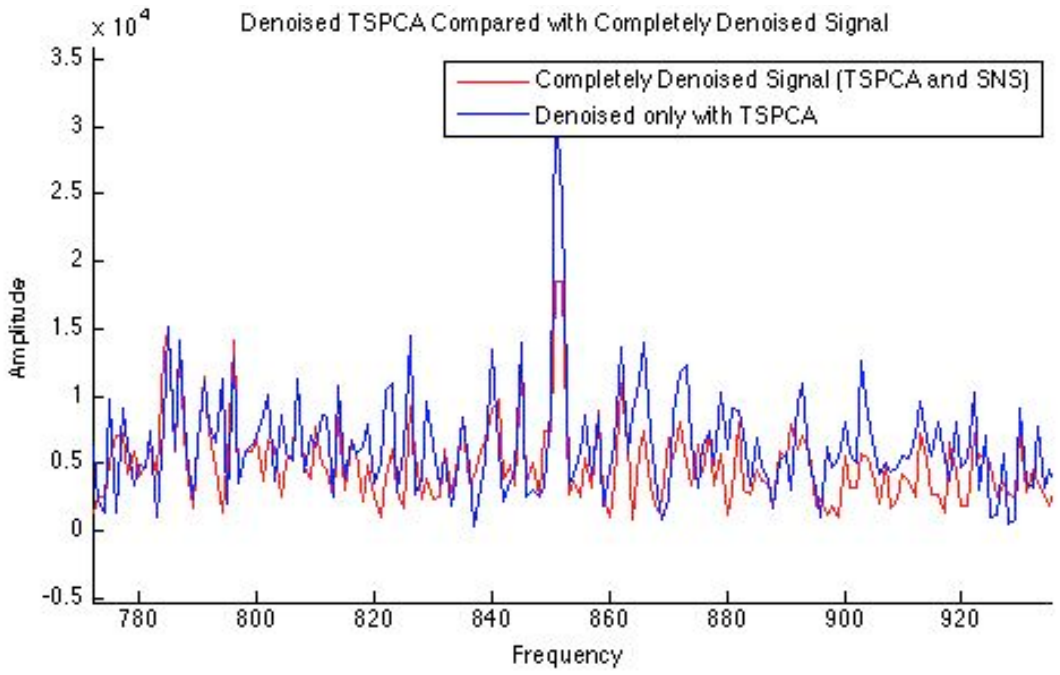

Figure 2. Data De-noised using only TSPCA versus Completely De-noised Data

Throughout the project, there were some minor setbacks that created some interesting limitations on the user. The first glitch was found when testing the SNS algorithm on the sample data. The data was 40,000 by 208, and was being divided in 40 separate blocks, thus making each block 1000 samples long. After the sample was de-noised in blocks and recombined to again create the 40,000 by 208 matrix, it was compared with data that was de-noised serial. A rather large difference was observed between the two signals. It was concluded that the block size was too small, and that was effecting the parallel de-noising. The block sizes should be around 4000 samples long in length in order to get signal closely resembling signal de-noised serially.

Another glitch was again found when using the sample data. This time it was noticed that if you put 40 blocks of length 1000 into the TSPCA algorithm and then recombine them, instead of getting the desired matrix of length 40,000, there was at matrix of length 32,000. It was then discovered that when one uses TSPCA, the algorithm truncates the last 200 samples. When TSPCA is being used serially only 200 samples are cut-off and it makes only a small impact. However, when you cut-off 200 samples from each block, it makes a much larger impression. This was fixed by added the 200 samples from the next block to the end of the current block.

Meaning if we want to have a signal block by 1000 samples, we have to make it 1200 samples. With these minor setbacks corrected the de-noising algorithms run smoothly in parallel.

The intention of this project was to save time by running the algorithms in parallel. It takes the algorithms approximately an hour to de-noise the signals serially, and the hope was to divide the time it takes by a factor proportional the number of processors used. Currently, to de-noise a signal in parallel comparable in size to the above-mentioned real signal, it takes under 10 minutes. This is when the signal is divided into 40 different blocks. However, the time it takes to de-noise drastically increases if you include the time it takes to transfer files from one's computer to the HPCC. Using Fugu, it takes approximately 35 minutes to import a file and about the same amount of time to export back to the original computer. This makes the total time to de-noise 1 hour and 20 minutes. The best way to improve the over all time and to truly take advantage of the time-saving abilities of parallel-processing, one would need to find a way to import and export data much more quickly from the HPCC.

APPENDIX A

MATLAB codes

The following is the file called PreDivide.m. If the data is given in the form of 160 channel matrix with the last 3 channels being the reference channels, this code will divide both the signal and the sensors into 40 different blocks.

```
load('Data.mat')
D=D(1:157); % D is the 157 neural sensor channels
R=D(158:160); % R is the 3 reference channels
[c,d]=size(D);
L=c/40; % represents the length of each block
D(c+1:c+200,:)=0; % This zeros pads the Data as well as the reference
channels, so that for the final block, there is an extra 200 samples,
that will be truncated by the TSPCA.
R(c+1:c+200,:)=0;
z=0;    % z is the index that keeps track of the number of blocks
for x = 1:L:c
    y= 1:L+200; % this extra 200 is added to avoid truncation by 200
from the TSPCA algorithm
    A(y,:)=D(x+y-1,:); % A is block of data
    r(y,:)=R(x+y-1,:); % r is an equivalent section of the ref.
channel
    save(['Divide_' num2str(z) '.mat'], 'A','r');
    save('MatNum.mat', 'z');
    z=z+1;
end
```

The following is the file called Denoise.m. It will open the saved blocks and assign them to a specific processor based on the "vpid" number assigned by the HPCC. For each block it will run the TSPCA algorithm followed by the SNS algorithm. Lastly, it will save blocks according "vpid" number.

```
[s,vpid] = unix('echo $OMPI_MCA_ns_nds_vpid');
vpid=vpid(1:end-1); % vpid identifies the processor that the block
below will run on
load(['Divide_' vpid '.mat']);
data=demean(A);
ref=demean(r);
shifts=-100:100;
wdata=[];
wref=[];
clean=tsr(data,ref,shifts,wdata,wref); % clean is the TSPCA de-noised
data
nneighbors=10;
```

```
skip=0;
w=[];
clean2=sns(clean,nneighbors,skip,w); % clean2 is the data after being
de-noised by SNS
save(['Clean_' num2str(vpid) '.mat'], 'clean2');
exit
```

The final MATLAB file called Combine.m. It takes the blocks saved after de-noising and recombines them to make one matrix. The reference channel is not recombined, since it no longer serves a purpose after TSPCA.

```
load('MatNum.mat')
% MatNum.mat was saved from PreDivide.m, it contains value z which is
how
% tells us how many blocks there are total
q=0; % q will serve as our index this time
load(['Clean_0.mat'])
[L,d]=size(clean2); % L tells of the length of each block
for b=1:L:((z*L)+L)
    load(['Clean_' num2str(q) '.mat'])
D(b:(b+L-1),:)=clean2;
q=q+1;
end
save('FinalAnswer.mat', 'D');
```

APPENDIX B

Unix shells

To run the program in deepthought.umd.edu, the parallel computer cluster, enter the command: **qsub –q serial test.sh.** This will prompt the shell test.sh to run. The test.sh file identifies the number of nodes as well as number of processors (ppn) that will run on each node. It also assigns a maximum amount of time it will run before stopping, (walltime). This file also opens MATLAB to run the PreDivide.m before the signal is de-noised and the Combine.m file after it is de-noised. Between those two commands it calls on another shell testnode.sh. This program will identify "vpid" as well as run the MATLAB program, Denoise.m, on parallel computing nodes in accordance to its assigned "vpid" number.

Test.sh

```
#PBS -l nodes=10:ppn=4
#PBS -l walltime=00:05:00

#hostname
#date
#setenv
echo $PBS_NODEFILE
cat $PBS_NODEFILE
# mpirun -np 40 testnode.sh
matlab -nodisplay -nojvm -r PreDivide
mpiexec -np 40 testnode.sh
matlab -nodisplay -nojvm -r Combine
```

Testnode.sh

```
#!/bin/csh
set vpid = $OMPI_MCA_ns_nds_vpid
set vpid0 = $OMPI_MCA_ns_nds_vpid_start
set vpidN = $OMPI_MCA_ns_nds_num_procs
echo vpid $vpid out of $vpidN starting at $vpid0
matlab -nodisplay -nojvm -r Denoise
```

References

Ahmar, Nayef. Da Vinci's Encephalogram: in Search of Significant Brain Signals. Department of Electrical and Computer Engineering. College Park, MD: Digital Repository At the University of Maryland, 2005. 1-71.

De Cheveigné, Alain, and Jonathan Z. Simon. "Denoising Based on Time-Shift PCA." Journal of Neuroscience Methods 165 (2007): 297-305.

De Cheveigné, Alain, and Jonathan Z. Simon. "Sensor Noise Suppression." Journal of Neuroscience Methods 168 (2008): 195-202.

"OIT High Performance Computing Cluster (HPCC)." Office of Information Technology. 27 Sept. 2007. 1 Feb. 2008 <http://www.oit.umd.edu/hpcc/>.