Brain tissue temperature dynamics during functional activity and possibilities for Imaging

by

Greggory H. Rothmeier

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Masters of Science

in the College of Arts and Sciences

Georgia State University

2012

Brain tissue temperature dynamics during functional activity and possibilities for Imaging

by

Greggory H. Rothmeier

Committee Chair: A. G. Unil Perera

Committee: Mukesh Dhamala

Brian Thoms

D. Michael Crenshaw

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2012

Dedication

Mama.

Acknowledgements

Perera, Dhamala, Brooke, Lab Mates, Dhamala's Lab

Contents

Acknowledgments . . . . . **iv**

List of Tables . . . . . **vii**

List of Figures . . . . . **viii**

1 Introduction **1**

2 Calculating Temperature Changes using the fMRI BOLD Response **2**

2.1 Background 2

2.1.1 Previously Proposed Temperature Models 2

Single-Voxel Approach 2

Multi-Voxel Approach 5

2.2 Modeling the BOLD Response 5

2.3 Modeling Temperature 5

2.3.1 The Approach 5

How the temperature is calculated 5

Calculating the equilibrium temperature 5

Calculating Metabolism and Blood Flow Changes 5

Calculating the change in temperature in the active brain 5

2.3.2 Results 5

Using Theoretical BOLD Data 5

Using Experimental BOLD Data 6

3 Detector Applications to measuring the active brain **9**

3.1 Functional Near-Infrared fNIR Imaging 9

3.2 Temperature Measurements 11

4 Conclusion **12**

References **12**

Appendices **14**

A Code **14**

A.1 Creating the Head Matrix 14

A.1.1 ImportSegmentedT1() 14

A.1.2 fillAir() 17

A.1.3 fillHoles() 18

A.1.4 build_skin() 20

A.1.5 repair_headdata() 21

A.2 Loading the fMRI Data 23

A.2.1 sample_bold_import() 23

A.2.2 avg_NII_rest() 26

A.2.3 avg_NII_normalize() 29

A.2.4 BOLDtoMF() 32

A.2.5 lambw() and lambw_mex() 35

A.3 Calculating the Equilibrium Temperature 37

A.3.1 tempCalcEquilibrium() 37

A.4 Calculating the Temperature Change 43

A.4.1 tempCalcDynMF 43

B Visualization Tools **49**

B.0.2 TempPlot() 49

B.0.3 tsliceplot 52

List of Tables

2.1 Parameters used in the single-voxel approximation 4

List of Figures

2.1 Generation of the fMRI BOLD Response 3

2.2 Procedure used to calculate temperature change 5

2.3 Slice of the segmented head. Each color represents a different tissue type. 6

2.4 Temperature changes: simulated BOLD data 7

2.5 Temperature changes: experimental BOLD data 8

3.1 Absorption spectra of water, deoxyhemoglobin and oxyhemoblogin 10

B.1 Visualization using tsliceplot 52

B.2 Visualization using tsliceplot (z v. t plane) 53

Brain tissue temperature dynamics during functional activity and possibilities for Imaging

A thesis

presented in Partial Fulfilment of Requirements for the Degree of Master of Science in the

College of Arts and Sciences

Georgia State University

2012

by

Greggory Rothmeier

Committee:

A. G. Unil Perera, Chair

Mukesh Dhamala, Member

Brian Thoms, Member

D. Michael Crenshaw, Member

April 1, 2012

Date

Dick Miller
Department Chair

Chapter 1

Introduction

Chapter 2

Calculating Temperature Changes using the fMRI BOLD Response

2.1 Background

Since its invention in the 1950's [1] and later development in the 1970's [2], Magnetic Resonance Imaging (MRI) has allowed physicians and scientists a detailed view within the human body.

2.1.1 Previously Proposed Temperature Models

Current efforts to model temperature changes be can categorized into two classes. The first class approaches the problem by considering a single voxel deep within the brain (single-voxel approach) while the second approach considers the brain and head as an entire system (multi-voxel approach). Each of these methods has their own pros and cons which will be discussed below.

Single-Voxel Approach

A single-voxel model of temperature was first proposed by SOMEONE, but has been refined over the past HOWLONG years CITEABUNCH to include more terms. Although different approaches consider different contributions to the temperature change, they all narrow the problem down to a single voxel which is usually 2mm x 2mm x 2mm. By simplifying the model, the heat equation can be simplified and the calculation is much easier to undertake. However, since the brain is not homogenous, the values used for parameters such as heat production and thermal conductivity are taken from an average of the tissues. As a result, this reduces the possible accuracy of such a model when applied to a subject. The most recently published

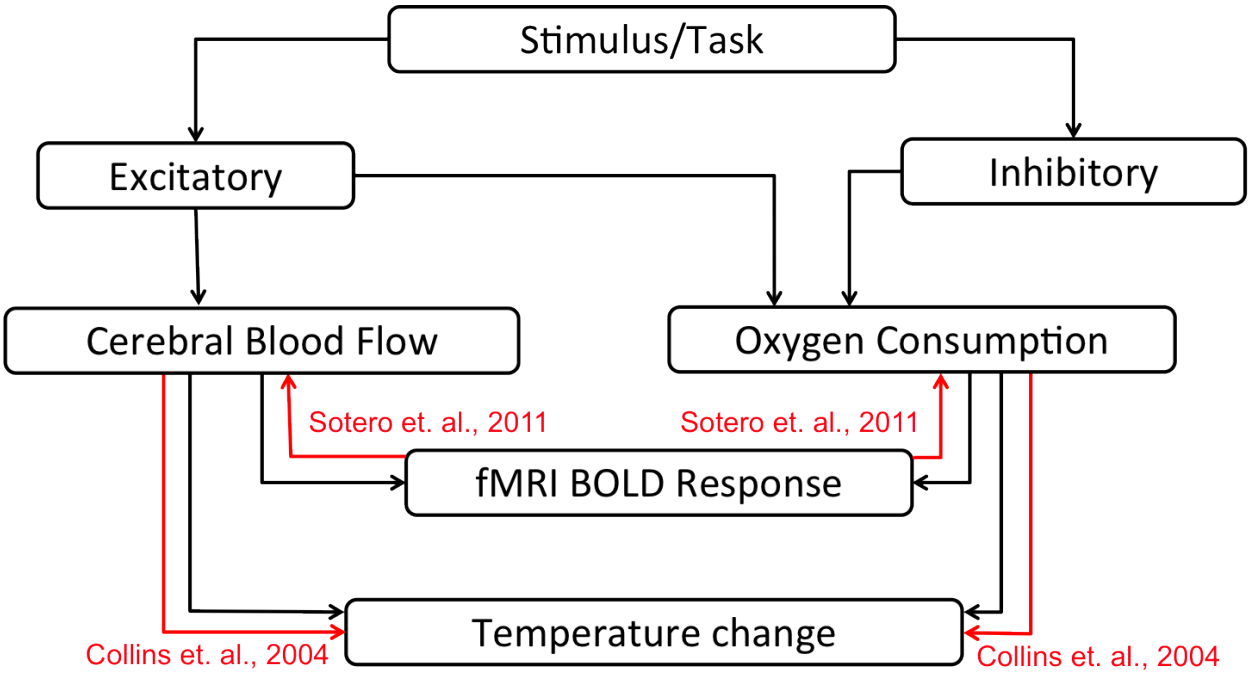


Figure 2.1: Generation of the fMRI BOLD response from changes in neuronal activity. Black arrows indicate a causal relationship while red arrows indicate existing models for the relationship. Modified from Sotero and Trujillo-Barreto [3]

iteration of a single-voxel model was published by Sotero and Iturria-Medina [4]. The basis of this model is a modification of the Penne's Bioheat Equation [5, 4].

$$C_t \frac{dT(t)}{dt} = (\Delta H^\circ - \Delta H_b) CMRO_2 \mid_0 m(t) - \rho_b C_b CBF \mid_0 f(t)(T(t) - T_a) - \frac{C_t}{\tau}(T(t) - T_0) \tag{2.1}$$

where BLA BLA BLA. One advantage of using eq. (2.1) is that the resting state temperature can be analytically determined by substituting $\frac{dT(t)}{dt} = 0$ [4].

$$T_0 = T_a + \frac{(\Delta H \mid^\circ - \Delta H_b) CMRO_2 \mid_0}{\rho_B C_B CBF \mid_0} \tag{2.2}$$

If the values provided in table 2.1 are substitued into eq. (2.2), a resting temperature of 37.3057°C is found. Since the resting temperature is always greater than the arterial blood temperature, it limits the ability of the model to account for all experimental results.

While eq. (2.1) is appears complicated, conceptually the equation can be easily understood.

$$\textit{change in temperature} = \textit{heat generated by metabolism} - \textit{heat lost to convection} - \textit{heat lost to conduction} \tag{2.3}$$

Table 2.1: Parameters used to solve the single-voxel Penne's Bioheat Equation. (modified from Sotero and Iturria-Medina [4])

Parameter	Meaning	Value
T_a	Arterial blood temperature	37°C
C_{tissue}	Tissue Heat Capacity	3.664 J/(gK)
ΔH°	Enthalpy released by oxidation of glucose	4.710^5 J
ΔH_b	Enthalpy used to release O_2 from hemoglobin	2.810^4 J
$CMRO_2 \mid_0$	Cerebral metabolic rate of O_2 consumption at rest	0.026310^{-6} mol/(gs)
$CBF\mid_0$	Cerebral blood flow at rest	0.0093 cm^3/(gs)
ρ_b	Blood density	1.05 g/cm^3
C_B	Blood heat capacity	3.894 J/(gK)
τ	Time constant for conductive heat loss from the ROI to the surrounding tissue	190.52 s
a, b, c	Parameters of the gamma function fitted from E(f) vs. f	0.4492, 0.2216, -0.9872
A	Maximum BOLD signal change	0.22
α	Steady state flow-volume relation	0.4
β	Field-strength dependent parameter	1.5
Variable	**Meaning**	
m(t)	$CMRO_2$ normalized to baseline	
f(t)	CBF normalized to baseline	
T(t)	Temperature	
W(t)	Lambert W Function	
$frac\Delta S(t)S_0$	Change in BOLD signal normalized to rest	

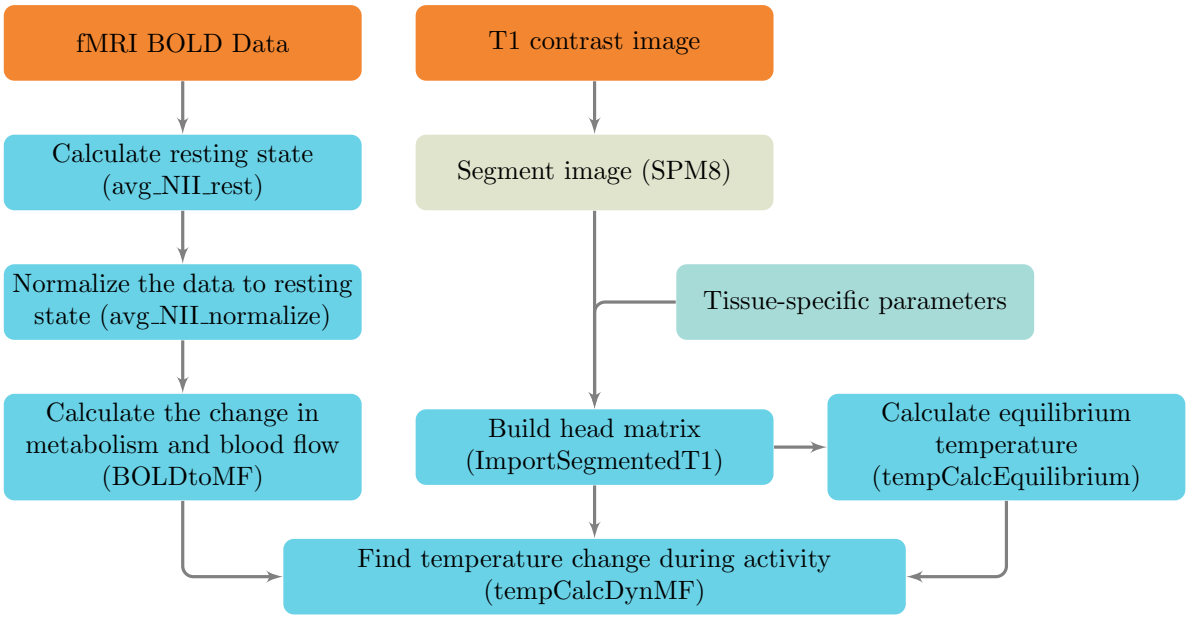
The system is a balance between heat generation (metabolism) and heat transfer (conduction and convection). The direction of heat transfer by convection is determined by the difference between the voxel temperature and the arterial blood temperature $(T(t) - T_a)$. Similarly, the direction of heat transfer by conduction is determined by the difference between the voxel temperature and the temperature of the surrounding tissue $(T(t) - T_0)$. Since T_a is less than $T(0)$, an increase in blood flow $(f(t))$ will remove heat from the voxel thereby decreasing the temperature. Conversely, an increase in metabolism $(m(t))$ without a corresponding change in blood flow, will result in tissue warming.

$$f(t) = \frac{\alpha + \beta c}{b\beta} W(y(t)) \tag{2.4}$$

$$m(t) = a f^{c+1}(t) e^{-bf(t)} \tag{2.5}$$

$$y(t) = -\frac{b\beta}{\alpha + \beta c} \left[\frac{(A - \frac{S(t)}{S_0} - 1)}{A a^{\beta}} \right]^{\left(\frac{1}{\alpha + \beta c}\right)} \tag{2.6}$$

Figure 2.2: The procedure used to calculate temperature from BOLD data. Orange blocks (●) represent data, the sandy-colored block (●) is a step done using SPM8 and the teal blocks (●) are steps done using a function provided within temptools (appendix A).



Multi-Voxel Approach

2.2 Modeling the BOLD Response

2.3 Modeling Temperature

2.3.1 The Approach

How the temperature is calculated

$$\rho c \frac{dT}{dt} = k\nabla^2 T - \rho_{blood} f(t) w c_{blood} (T - T_{blood}) + m(t) Q_m \tag{2.7}$$

Calculating the equilibrium temperature

Calculating Metabolism and Blood Flow Changes

Calculating the change in temperature in the active brain

2.3.2 Results

Using Theoretical BOLD Data

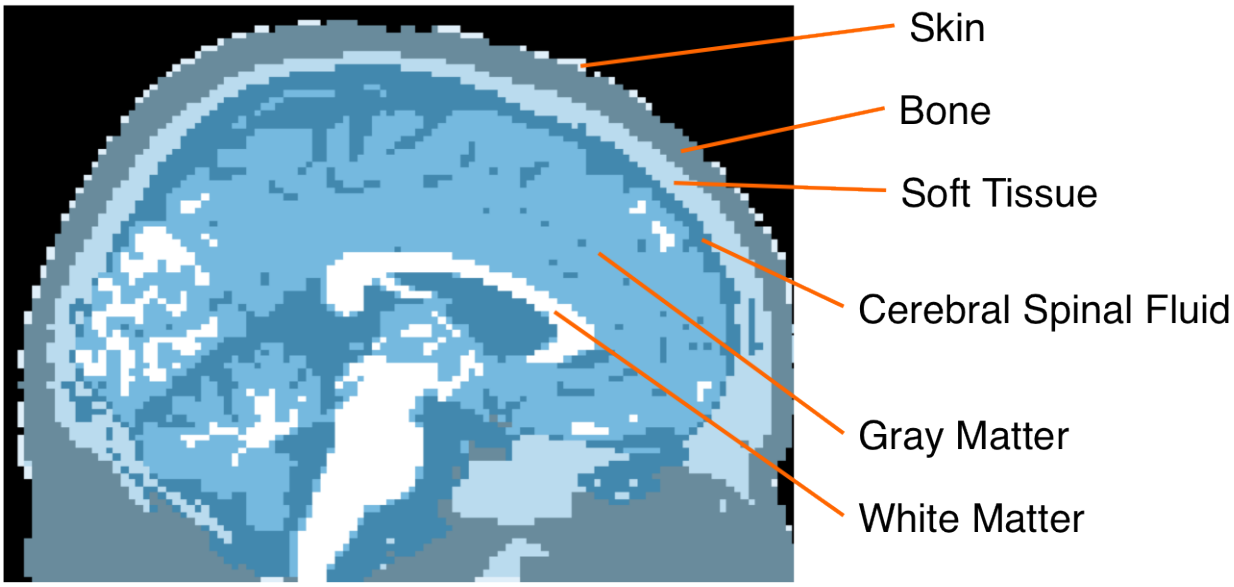


Figure 2.3: Slice of the segmented head. Each color represents a different tissue type.

Using Experimental BOLD Data

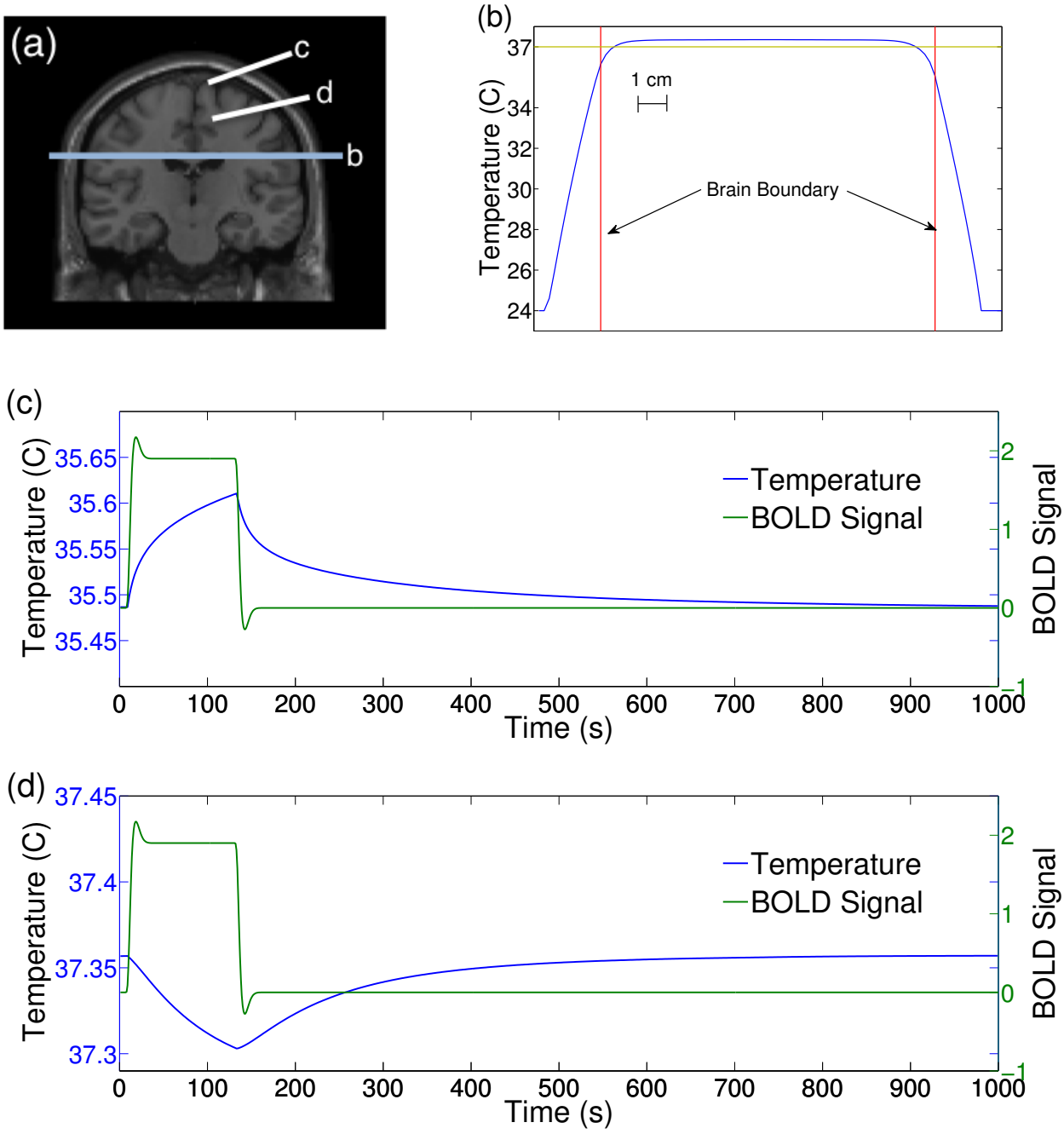


Figure 2.4: Temperature changes using simulated BOLD signals. (a) Slice of the head (y = -12) with indicators of the locations for parts (b)-(d). (b) Equilibrium temperature along a line through the head. Red lines indicate the brain boundary and the gold line indicates the blood temperature (37°C) used for calculations. Inside the brain, a 4-6 mm thick shell is created where the equilibrium temperature is less than the blood temperature. Within this shell, (c) the temperature rises with increased brain activity while (d) tissue deeper in the brain experiences the opposite effect.

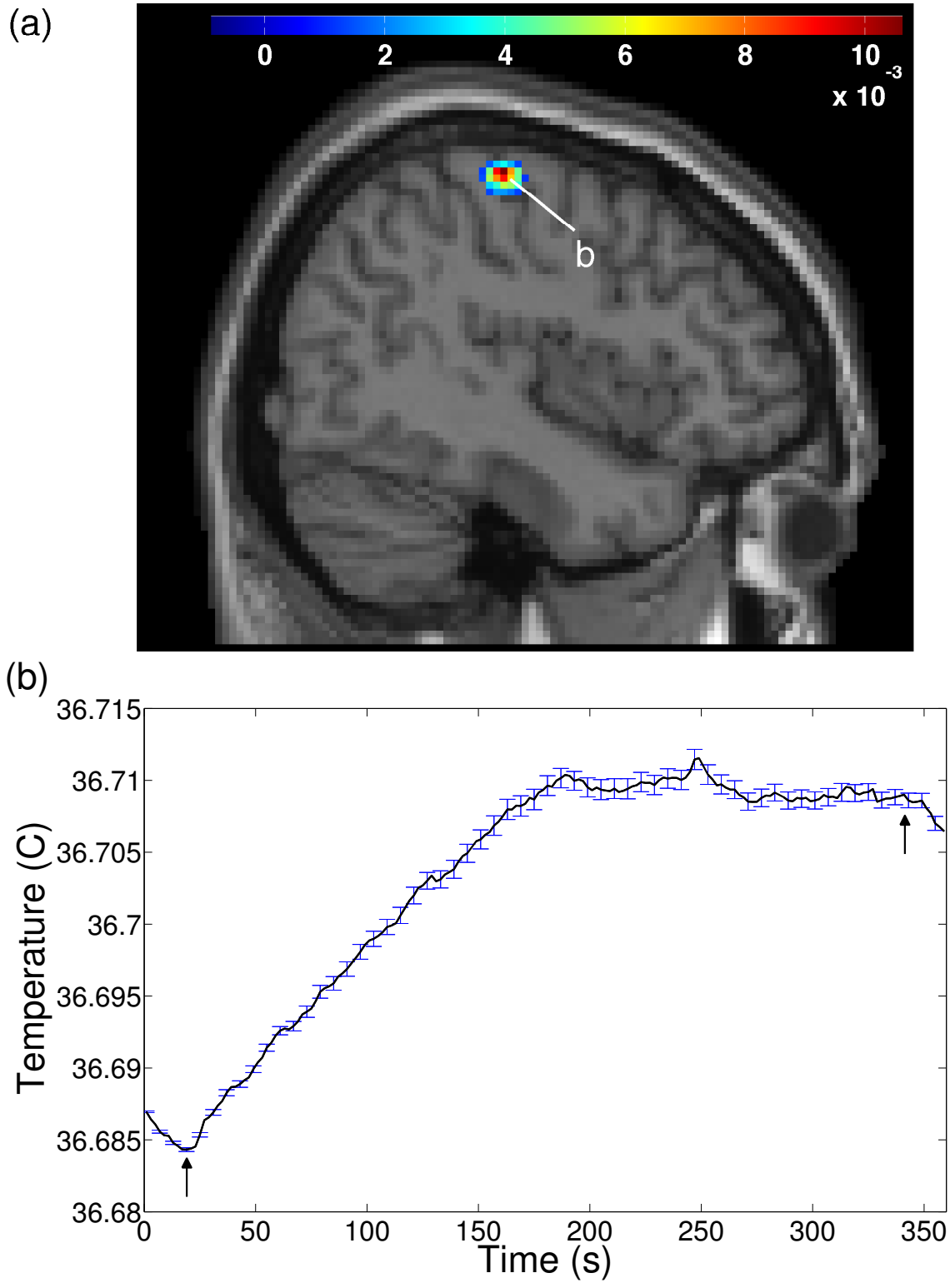


Figure 2.5: Temperature calculated from a voxel within the motor cortex. (a) A slice (x = -44) showing the motor cortex warming during a finger-tapping task. (b) Temperature at a voxel within the motor cortex (-44, -24, 60) with standard error indicated by blue error bars (Arrows indicate task onset and conclusion, N=24).

Chapter 3

Detector Applications to measuring the active brain

3.1 Functional Near-Infrared fNIR Imaging

$$I = I_0 e^{-\alpha x} \tag{3.1}$$

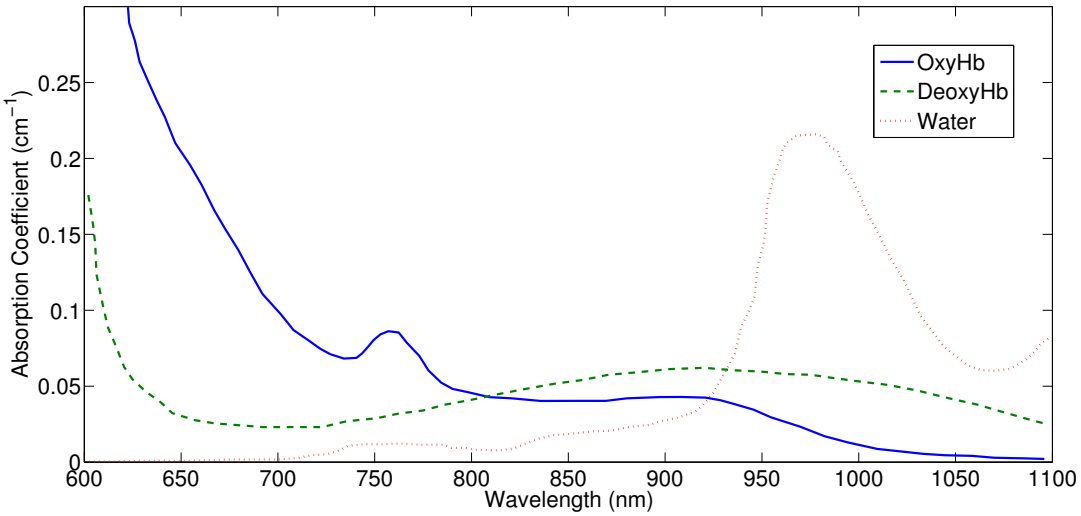


Figure 3.1: Absorption spectra of water, Hb and Dhb. From Cope [6] and HB stuff from Horecker [7]

3.2 Temperature Measurements

From the Beer-Lambert law eq. (3.1), the penetration depth, δ_p can be expressed as

$$\delta = \frac{1}{\alpha} \tag{3.2}$$

where α is the absorption coefficient. At body temperature (37°) the peak wavelength in the blackbody spectrum is approximately BLA. For water at this wavelength, α is approximately HUGE, so δ is VERY SMALL.

Chapter 4

Conclusion

Bibliography

[1] Herman Y Carr and E M Purcell. Effects of diffusion on free precession in nuclear magnetic resonance experiments. *Physical Review*, 94:630–638, 1954.

[2] Paul Lauterbur. Image formation by induced local interactions: Examples employing nuclear magnetic resonance. *Nature*, 242:190–191, 1973.

[3] Roberto C Sotero and Nelson J Trujillo-Barreto. Modeling the role of excitatory and inhibitory neuronal activity in the generation of the bold signal. *NeuroImage*, 35:149–165, 2007.

[4] Roberto C Sotero and Yasser Iturria-Medina. From Blood Oxygen Level Dependent (BOLD) signals to brain temperature maps. *Bulletin of Mathematical Biology*, 73(11):2731–2747, 2011.

[5] H H Pennes. Analysis of tissue and arterial blood temperatures in the resting human forearm. *Journal of Applied Physiology*, 1(2):93–122, 1948.

[6] M Cope. *The development of a near infrared spectroscopy system and its application for non invasive monitoring of cerebral blood and tissue oxygenation in the newborn infants*. PhD thesis, London University, 1991.

[7] B L Horecker. The absorption spectra of hemoglobin and its derivatives in the visible and near infra-red regions. *The Journal of Biological Chemistry*, 1942.

[8] Mukeshwar Dhamala, Giuseppe Pagnoni, Kurt Wiesenfeld, Caroline Zink, Megan Martin, and Gregory Berns. Neural correlates of the complexity of rhythmic finger tapping. *NeuroImage*, 20:918–926, 2003.

Appendix A

Code

The following sections include the code used. It was written for Matlab 2011(b) and requires SPM8 to run. Additionally, it is recommended that you have at least 4 GB of RAM in order to work with the large datasets that are produced. For information about how to visualize the data produced, see appendix B. All of the code is available through the temptools github page (`https://github.com/greggroth/tempcalc`)

A.1 Creating the Head Matrix

Before any calculations can be done, a matrix containing tissue-specific parameters must be created. First, a T1 contrast image should be segmented using SPM8 (`http://www.fil.ion.ucl.ac.uk/spm/software/spm8/`). For ease of consistency, the one provided by SPM8 in ./canonical/ is best to use. Using SPM's "New Segmentation" algorithim will segment the image into five different tissue types (gray matter, white matter, cerebral spinal fluid, soft tissue and bone). Once this is complete, run ImportSegmentedT1() within this directory and it will return a matrix that has been populated with the tissue-specific parameters required for accurate temperature calculations. The functions fillAir() (A.1.2), fillHoles() (A.1.3), build_skin() (A.1.4) and repair_headdata() (A.1.5) are functions required by BulkImportNII(). More information about this procedure is in section 2.3.1.

A.1.1 ImportSegmentedT1()

```
1 function [ total ] = ImportSegmentedT1(varargin)
2 %  ImportSegmentedT1 Import NII files from a directory
3 %   Must be run within the directory containing the files
4 %
```

```
%   Output: head data as single with variables stored in the 4th dimension.
%
%   Author:  Greggory Rothmeier (greggroth@gmail.com)
%   Georgia State University
%   Created:  5/31/11

statusbar = waitbar(0,'Initializing');

if size(varargin) == 1
    oldFolder = cd(varargin{1});
end


% =====================
% = Tissue Parameters =
% =====================
% Each tissue type is assigned an integer index (i.e. gray matter -> 11) such that
% tissue-specific parameters can be found by looking at that element within the
% corresponding storage matrix (i.e. QmSTORE(11) -> gray matter Qm)

% Parameters taken from Colins, 2004

tisorder = [11 15 5 13 3];  %  Using:  [GM WM CSF Muscle Bone]

QmSTORE = [0 0 26.1 11600 0 26.1 697 0 0 302 15575 0 697 1100 5192];
cSTORE = [1006 4600 2110 3640 3800 1300 3720 3000 4200 2300 3680 3500 3720 3150
   3600];
rhoSTORE = [1.3 1057 1080 1035.5 1007 1850 1126 1076 1009 916 1035.5 1151 1041
   1100 1027.4];
kSTORE = [0.026 0.51 0.65 0.534 0.5 0.65 0.527 0.4 0.594 0.25 0.565 0.4975 0.4975
   .342 .503];
wSTORE = [0 1000 3 45.2 0 1.35 40 0 0 2.8 67.1 3.8 3.8 12 23.7];

% ====================================
% = Import the pre-segmented T1 files =
```

```
% ====================================
% The T1 contrast image sould be segmented using SPM8.
%   This loop needs to complete before the next one can begin
%  Import all of the datat and store as 'cdat1','cdat2', etc.
for i = 1:5
    eval(strcat('dat',num2str(i),' = loadNII(''rc', num2str(i), 'single_subj_T1.
       nii'');'))
    %  Preallocate
    eval(strcat('out', num2str(i),' = zeros(cat(2,size(dat', num2str(i),'),7));'))
end

% ============================
% = Populate the head matrix =
% ============================
%   For each data file, it fills in the data from the data storage arrays
%   for that particular type of tissue.  It picks which ever tissue is the
%   most likely candidate for that voxel based on the segmented data

%   PROBLEM:  It returns 0 (later filled with air) if there is equal
%   probability of a voxel being two or more different types of tissue.
%   SOLVED BY fillHoles()


for i = 1:5
    %  Preallocate
    holder = zeros(cat(2,size(dat1),7),'single');
    mask = zeros(size(dat1));
    final = zeros(size(holder),'single');

    %  Create a mask that indicates whether it is the mostly likely tissue type
    guide = [1 2 3 4 5 1 2 3 4 5];  %  This guides it through the data correctly
    eval(strcat('mask = (dat',num2str(i),'>dat',num2str(guide(i+1)),') & (dat',
       num2str(i),'>dat',num2str(guide(i+2)),') & (dat',num2str(i),'>dat',num2str(
       guide(i+3)),') & (dat',num2str(i),'>dat',num2str(guide(i+4)),') & (dat',
       num2str(i),'~=0);'))
```

```
68      holder(:,:,:,1) = mask;                         %  move structure data to new
            matrix
69      a = find(holder(:,:,:,1) == 1);                 %  get indicies of tissues
70      [x y z t] = ind2sub(size(holder),a);            %  gets coordinates from index
71
72      %  go to each tissue point and store the info
73      for j = 1:length(a)
74          final(x(j),y(j),z(j),:) = [tisorder(i) 0 QmSTORE(tisorder(i)) cSTORE(
                tisorder(i)) rhoSTORE(tisorder(i)) kSTORE(tisorder(i)) wSTORE(tisorder(
                i))];
75      end
76
77      %  Saves the result to a unique output variable (out1, out2, etc)
78      eval(strcat('out',num2str(i),'= final;'))
79
80      clearvars a x y z t holder final;
81      waitbar(i/6,statusbar,sprintf(['File ',num2str(i),' Import Compete']));
82  end
83
84  % The filleAir() function checks for any voxels which were not assigned a
85  % tissue type and fills them in with air
86  almostthere = fillAir(out1+out2+out3+out4+out5);
87  % The fillHoles() function corrects for a voxel having two equally-probable
88  % tissue types
89  total = single(buildskin(fillHoles(dat1,dat2,dat3,dat4,dat5,almostthere)));
90  waitbar(1,statusbar,'Saving Data')
91
92  cd(oldFolder);
93  close(statusbar);
94
95  end
```

A.1.2 fillAir()

```
1  function [ output ] = fillAir( tissue )
2  % fillAir() fills gaps in data with air
```

```
%   Once you import all of the data using loadNII(), run it thought this to
%   fill in the remaining spaces with air.

airdata = [1 0 0 1006 1.3 0.026 0];

%  Picks out air spots
a = find(tissue(:,:,:,1) == 0);
[x y z t] = ind2sub(size(tissue),a);

for i = 1:length(a)
    tissue(x(i),y(i),z(i),:) = airdata;
end

output = tissue;

end
```

A.1.3 fillHoles()

```
function [ out_head ] = fillHoles( in1,in2,in3,in4,in5,headin)
% fillHoles() checks for misassigned voxels
%
% Solves an issue where a voxel with two equally probable tissue
% types resulted in being assigned as air.  This checks for air
% voxels that are surrounded by tissue and decides a tissue it
% it would be best suited as

head = squeeze(headin(:,:,:,1));  %  I only need the tissue indices so this makes
   things easier down the line

%%  Data Storage
QmSTORE = [0 0 26.1 11600 0 26.1 697 0 0 302 15575 0 697 1100 5192];
cSTORE = [1006 4600 2110 3640 3800 1300 3720 3000 4200 2300 3680 3500 3720 3150
   3600];
rhoSTORE = [1.3 1057 1080 1035.5 1007 1850 1126 1076 1009 916 1035.5 1151 1041
   1100 1027.4];
```

```
kSTORE = [0.026 0.51 0.65 0.534 0.5 0.65 0.527 0.4 0.594 0.25 0.565 0.4975 0.4975
   .342 .503];
wSTORE = [0 1000 3 45.2 0 1.35 40 0 0 2.8 67.1 3.8 3.8 12 23.7];

%%  Get locations of holes
%   Where two tissue types have the same probability

idx1 = (in1==in2 | in1 == in3 | in1==in4 | in1==in5) & logical(in1);
idx2 = (in1==in2 | in2 == in3 | in2==in4 | in2==in5) & logical(in2);
idx3 = (in1==in3 | in2 == in3 | in3==in4 | in3==in5) & logical(in3);
idx4 = (in1==in4 | in2 == in4 | in3==in4 | in4==in5) & logical(in4);
idx5 = (in1==in5 | in2 == in5 | in3==in5 | in4==in5) & logical(in5);
%  This array will have a zero anywhere there were two or more common
%  elements between any of the five arrays.
idx = idx1|idx2|idx3|idx4|idx5;

[xmax ymax zmax] = size(in1)
[x y z] = ind2sub(size(in1),find(idx));  %  get x, y and z coordinates of the
   holes

for i = 1:length(x)  %  go to each hole and do work
    if (x(i)~=1)&&(y(i)~=1)&&(z(i)~=1)&&(x(i)~=xmax)&&(y(i)~=ymax)&&(z(i)~=zmax)
       &&(headin(x(i),y(i),z(i),1)==1)  %  keeps away from the edge and only looks
        at voxels that were assigned air
        [commonesttissue nouse secondbest] = mode([head(x(i)+1,y(i),z(i)) head(x(i
           )-1,y(i),z(i)) head(x(i),y(i)+1,z(i)) head(x(i),y(i)-1,z(i)) head(x(i),
           y(i),z(i)+1) head(x(i),y(i),z(i)-1)]);
        if commonesttissue == 1 && length(secondbest{1})>=2  % if air and
           something else are equally common, it'll choose air.  This forces it to
            pick the tissue if possible.
            commonesttissue = secondbest{1}(2);
        end
        headin(x(i),y(i),z(i),:) = [commonesttissue 0 QmSTORE(commonesttissue)
           cSTORE(commonesttissue) rhoSTORE(commonesttissue) kSTORE(
           commonesttissue) wSTORE(commonesttissue)];
```

```
    end
end

out_head = headin;

end
```

A.1.4 build_skin()

```
function [ head_out ] = build_skin( head_in )
% build_skin() Creates a layer of skin around the head
%
% This will check all voxels that were previously labeled
% as soft tissue and checks if it has a neighbor which is air.
% If so, then it is reassigned as skin.

if ndims(head_in)==4
    head_in = head_in(:,:,:,1);
end

% Git a list of all voxels labeled as muscle
muscle_voxels = find(head_in==13);

% Go through each of them and check for neighboring air voxels
for i=1:length(muscle_voxels)
   [x y z] = ind2sub(size(head_in), muscle_voxels(i));
   % makes sure we're not at a voxel at the boundry of the dataset
   if (x~=1) && (x~=size(head_in,1)) && (y~=1) && (y~=size(head_in,2)) && (z~=1)
      && (z~=size(head_in,3))
     % Looks for neighboring voxels that are air
     if ((head_in(x+1,y,z)==1) || (head_in(x-1,y,z)==1) || (head_in(x,y+1,z)==1)
        || (head_in(x,y-1,z)==1) || (head_in(x,y,z+1)==1) || (head_in(x,y,z-1)==1)
        )
         head_in(x,y,z) = 14;
     end
   end
```

```
end

head_out = repair_headdata(head_in);

end
```

A.1.5 repair_headdata()

This function will go through the dataset and make sure the tissue-specific parameters are correct for the tissue type assigned for that voxel. fillAir(), fillHoles() and build_skin() all correct mislabeled voxels, but they only correct the tissue assignment. After using any of these functions, the data must be passed through repair_headdata to update the stored parameters.

```
function [ head_out ] = repair_headdata( head_in )
% repaid_headdata repopulates the headdata matrix
%   If any changes are made to the index column in the headdata matrix, use
%   this function to repopulate and correct the parameter values before running
% any other functions.
%  head_in can be either 3 or 4 dimenisions


% =====================
% = Parameter Storage =
% =====================

QmSTORE = [0 0 26.1 11600 0 26.1 697 0 0 302 15575 0 500 1100 5192];
cSTORE = [1006 4600 2110 3640 3800 1300 3720 3000 4200 2300 3680 3500 3010 3150
   3600];
rhoSTORE = [1.3 1057 1080 1035.5 1007 1850 1126 1076 1009 916 1035.5 1151 978.5
   1100 1027.4];
kSTORE = [0.026 0.51 0.65 0.534 0.5 0.65 0.527 0.4 0.594 0.25 0.565 0.4975 0.3738
   .342 .503];
wSTORE = [0 1000 3 45.2 0 1.35 40 0 0 2.8 67.1 3.8 3.3 12 23.7];

if ndims(head_in)==4
    head_in = head_in(:,:,:,1);
end
```

```
22
23  % Reassign the parameter values
24  head_out = cat(4,head_in, zeros(size(head_in)), QmSTORE(head_in), cSTORE(head_in),
        rhoSTORE(head_in), kSTORE(head_in), wSTORE(head_in));
25
26  end
```

A.2 Loading the fMRI Data

The following sections details the processing required to convert the BOLD data (in NIFTI format) to metabolism and blood flow time-courses that can then be used to calculate temperature.

A.2.1 sample_bold_import()

The following code automates the procedure of processing and doing all the calculations on the dataset reported in Dhamala et al. [8]. It's is written for my data on my machine, but it can be used to gain a better understanding of the procedure. For a conceptual explanation, see section 2.3.1.

```
%%=====================================================================
%%     How to process preprocessed BOLD data to calculate temperature
%%=====================================================================

% This Matlab script was used to automate the the process of using BOLD data
% stored in NIFTI (*.nii) format to calculate temperature changes.  The
% particulars of the code may be specific to this case, but the procedure
% should be the same when doing these calculations on other datasets.  All
% required functions are included as an attachment to my thesis and are
% available on my github (https://github.com/greggroth/tempcalc)

cd('/Users/Greggory/Documents/Data/fmri_rhythmic_tapping01/NIFTI')

directories = dir('*01');

%%  Move coregistered files to new Directory
for i = 1:length(directories)
    dir_name = directories(i).name;
    main_path = cd( [dir_name filesep dir_name '_NIFTI_1'] );
    mkdir 'Coregistered'
    movefile('r*.nii','Coregistered')
    main_path = cd( [dir_name filesep dir_name '_NIFTI_2'] );
    mkdir 'Coregistered'
    movefile('r*.nii','Coregistered')
    cd(main_path)
end
```

```
27
28  %%  Calculate Rest State
29  disp('Calculating Rest State')
30  for i = 1:length(directories)
31      dir_name = directories(i).name;
32      avg_NII_rest([dir_name filesep dir_name '_NIFTI_1' filesep 'Coregistered']);
33      avg_NII_rest([dir_name filesep dir_name '_NIFTI_2' filesep 'Coregistered']);
34  end
35
36
37  %%  Normalize to Rest and Mask
38  disp('Normalize to Rest and Mask')
39  for i = 1:length(directories)
40      dir_name = directories(i).name;
41      avg_NII_normalize([dir_name filesep dir_name '_NIFTI_1' filesep 'Coregistered'
          ], fullfile(dir_name, [dir_name '_NIFTI_1'], 'Coregistered', 'RestState', '
          RestStateAvg.nii'), 'fullBrainMask.nii');
42      avg_NII_normalize([dir_name filesep dir_name '_NIFTI_2' filesep 'Coregistered'
          ], fullfile(dir_name, [dir_name '_NIFTI_2'], 'Coregistered', 'RestState', '
          RestStateAvg.nii'), 'fullBrainMask.nii');
43  end
44
45
46  %%  Calculate metabolism and blood flow change
47  disp('Calculate metabolism and blood flow change')
48  for i = 1:length(directories)
49      dir_1 = [ directories(i).name filesep  directories(i).name '_NIFTI_1' filesep
          'Coregistered' filesep 'Normalized_to_rest'];
50      dir_2 = [ directories(i).name filesep  directories(i).name '_NIFTI_2' filesep
          'Coregistered' filesep 'Normalized_to_rest'];
51      BOLDtoMF(dir_1);
52      BOLDtoMF(dir_2);
53  end
54
55
```

```
%%  Calculate the change in temperature based on metabolism and blood flow

% load('equil.mat');  % equillibriumT
% load('tt_headdata.mat');  % headdata
mask = loadNII('fullBrainMask.nii');

for i = 1:length(directories)
    disp([int2str(i) '-1 started'])
    tic
    % Part I
    actResult.dat = tempCalcDynMF(headdata, 37, 24, 720, 360, equillibriumT, ...
        fullfile(directories(i).name,[directories(i).name '_NIFTI_1'],'
           Coregistered', 'Normalized_to_rest', 'Output_18-Sep-2011', 'm.mat'),
           ...
        fullfile(directories(i).name,[directories(i).name '_NIFTI_1'],'
           Coregistered', 'Normalized_to_rest', 'Output_18-Sep-2011', 'f.mat'),
           ...
        4, mask);
    % Store the parameters used for the calculations for reference in the future
    [c lmax] = max(actResult.dat(:));
    [likelymax x y z] = ind2sub(size(actResult.dat),lmax);
    actResult.likelymaxslice = round(likelymax/2);
    actResult.bloodT = 37;
    actResult.airT = 24;
    actResult.tmax = 360;
    actResult.stepf = 2;
    actResult.savestepf = 4;
    actResult.metabandflowdata = 'From Dataset';
    save(fullfile(directories(i).name,[directories(i).name '_NIFTI_1'],'
       Coregistered', 'Normalized_to_rest', 'Output_18-Sep-2011','tt_act_res.mat')
       , 'actResult');
    old = cd([directories(i).name,filesep,[directories(i).name '_NIFTI_1'],filesep
       ,'Coregistered', filesep,'Normalized_to_rest', filesep,'Output_18-Sep-2011'
       ]);
    writeT_to_nii(actResult, equillibriumT, exp_nii);
```

```
83      cd(old)
84      clear actResult
85      % Part II
86      disp([int2str(i) '-2 started'])
87      actResult.dat = tempCalcDynMF(headdata, 37, 24, 720, 360, equillibriumT, ...
88          fullfile(directories(i).name,[directories(i).name '_NIFTI_2'],'
               Coregistered', 'Normalized_to_rest', 'Output_18-Sep-2011', 'm.mat'),
               ...
89          fullfile(directories(i).name,[directories(i).name '_NIFTI_2'],'
               Coregistered', 'Normalized_to_rest', 'Output_18-Sep-2011', 'f.mat'),
               ...
90          4, mask);
91      [c lmax] = max(actResult.dat(:));
92      [likelymax x y z] = ind2sub(size(actResult.dat),lmax);
93      actResult.likelymaxslice = round(likelymax/2);
94      actResult.bloodT = 37;
95      actResult.airT = 24;
96      actResult.tmax = 360;
97      actResult.stepf = 2;
98      actResult.savestepf = 4;
99      actResult.metabandflowdata = 'From Dataset';
100     save(fullfile(directories(i).name,[directories(i).name '_NIFTI_2'],'
           Coregistered', 'Normalized_to_rest', 'Output_18-Sep-2011','tt_act_res.mat')
           , 'actResult');
101     old = cd([directories(i).name,filesep,[directories(i).name '_NIFTI_2'],filesep
           ,'Coregistered', filesep,'Normalized_to_rest', filesep,'Output_18-Sep-2011'
           ]);
102     writeT_to_nii(actResult, equillibriumT, exp_nii);
103     cd(old)
104     clear actResult
105     disp([int2str(i) ' finished in ' num2str(toc)])
106 end
```

A.2.2 avg_NII_rest()

```
function [ ] = avg_NII_rest( varargin )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

%% Setup
switch length(varargin)
    case 0
        fold_name = uigetdir;
        if ~fold_name  % Cancel Button
            return
        end
    case 1
        fold_name = varargin{1};
    otherwise
end

%  Go to the folder containing the files
oldfold = cd(fold_name);
file_list = dir('*.nii');

%  We're only interested in the rest period
%  (first and last 10 steps in this case)
file_list = file_list([1:10 170:180]);
file_count = length(file_list);

%  Cell array to store all of the datasets in.
datHolder = cell(file_count,1);

statusbar = waitbar(0,'Initializing');

for j=1:file_count
    try
        waitbar(j/file_count,statusbar,sprintf('%d%%',round((j/file_count)*100)));
    catch err
        return
```

```
    end
    fi = load_nii(file_list(j).name);
    datHolder{j} = fi.img;
end

%%  Calculate the mean
ymax = size(datHolder{1},2);
zmax = size(datHolder{1},3);
output = zeros(size(datHolder{1}));

for i=1:ymax
    try
        waitbar(i/ymax,statusbar,sprintf('%d%%',round((i/ymax)*100)));
    catch err
        return
    end
    for k=1:zmax
        excStr = cell(length(datHolder),1);
        for l=1:length(datHolder)
            excStr{l} = [',datHolder{' int2str(l) '}(:,' int2str(i) ',' int2str(k)
                ')'''];
        end
        comb = eval(['cat(1' cell2mat(excStr') ')']);
        output(:,i,k) = mean(comb);
    end
end

close(statusbar)

fi.img = output;
mkdir('RestState')
save_nii(fi,fullfile('RestState','RestStateAvg.nii'));

cd(oldfold)

```

```
70 end
```

A.2.3 avg_NII_normalize()

```
1  function [  ] = avg_NII_normalize( varargin )
2  %UNTITLED6 Normalize to rest state
3  %   Detailed explanation goes here
4
5  %% Setup
6  switch length(varargin)
7      case 0
8          fold_name = uigetdir('Directory Containing Data');
9          if ~fold_name  % Cancel Button
10             return
11         end
12
13         [rest_file rest_path rest_index]= uigetfile('*.nii','Resting State NIFTI
               File');
14         switch rest_index
15             case 0
16                 return
17             case 1
18                 rest_dat = load_nii(fullfile(rest_path,rest_file));
19                 rest_dat = double(rest_dat.img);
20             otherwise
21                 error('An error has occured loading the resting state data')
22         end
23
24         [mask_file mask_path mask_index] = uigetfile('*.nii','Mask');
25         switch mask_index
26             case 0
27                 return
28             case 1
29                 mask_dat = load_nii(fullfile(mask_path, mask_file));
30                 mask_dat = logical(mask_dat.img);
31                 if max(size(mask_dat) ~= size(rest_dat))
```

```
32                      error('The Mask and Resting State files must have the same
                            size')
33                  end
34              otherwise
35                  error('An error has occured loading the resting state data')
36          end
37      case 1
38          fold_name = varargin{1};
39          [rest_file rest_path rest_index]= uigetfile('*.nii','Resting State NIFTI
                File');
40          switch rest_index
41              case 0
42                  return
43              case 1
44                  rest_dat = load_nii(fullfile(rest_path,rest_file));
45                  rest_dat = double(rest_dat.img);
46              otherwise
47                  error('An error has occured loading the resting state data')
48          end
49      case 2
50          fold_name = varargin{1};
51          rest_dat = loadNII(varargin{2});
52          [mask_file mask_path mask_index] = uigetfile('*.nii','Mask');
53          switch mask_index
54              case 0
55                  return
56              case 1
57                  mask_dat = load_nii(fullfile(mask_path, mask_file));
58                  mask_dat = logical(mask_dat.img);
59                  if max(size(mask_dat) ~= size(rest_dat))
60                      error('The Mask and Resting State files must have the same
                            size')
61                  end
62              otherwise
63                  error('An error has occured loading the resting state data')
```

```
64          end
65      case 3
66          fold_name = varargin{1};
67          rest_dat = loadNII(varargin{2});
68          mask_dat = loadNII(varargin{3});
69      otherwise
70          return
71  end
72
73  %  Go to the folder containing the files
74  oldfold = cd(fold_name);
75  file_list = dir('*.nii');
76  file_count = length(file_list);
77
78  %  Make a directoy to save the normalized data to
79  saveDir = 'Normalized_to_rest';
80  if ~isdir(saveDir)
81      mkdir(saveDir);
82  end
83
84  statusbar = waitbar(0,'Initializing');
85
86  % for each file: load it, devide by the rest state and save it
87  for i=1:file_count
88      try
89          waitbar(i/file_count,statusbar,[fold_name sprintf('%d%%',round((i/
                file_count)*100))]);
90      catch err
91          return
92      end
93      [file_path file_name file_ext] = fileparts(file_list(i).name);
94      file_hold = load_nii(file_list(i).name);
95      file_hold.img = double(file_hold.img)./rest_dat - 1;
96      file_hold.img(~mask_dat) = 0;              % set everything outside the mask to
            0
```

```
97      file_hold.img(isnan(file_hold.img)) = 0;  % set all NaN's to 0
98      file_hold.img(isinf(file_hold.img)) = 0;  % set all inf's to 0
99      file_hold.img(file_hold.img == -1) = 0;   % correct these for voxels that are
           giving me problems
100     file_hold.hdr.dime.datatype = 16;  % set the datatype to single
101     file_hold.hdr.dime.bitpix = 16;
102     save_nii(file_hold,fullfile(saveDir,[file_name '_rn' file_ext]))
103 end
104
105 close(statusbar)
106 cd(oldfold)
107
108 end
```

A.2.4 BOLDtoMF()

```
1  function [ ] = BOLDtoMF( varargin)
2  %BOLDtoMF Calculate metabolism and blood from from BOLD reponse
3  %
4  %   Input: Directory containing a series of *.nii files of the BOLD
5  %   response.
6  %
7  %   Output: Two new files will be created in a new subdirectory with a
8  %   variable for each time step.
9  %
10 %   Usage:
11 %       BOLDtoMF
12 %       BOLDtoMF(directory)
13 %
14 %   If a directory is not provided, one will be requested.
15 %
16 %   Method from Sotero, et. al. 2010
17
18 % =========
19 % = Setup =
20 % =========
```

```
21
22 % if a folder isn't an argument, it'll prompt for one
23 switch length(varargin)
24     case 0
25         fold_name = uigetdir;
26         if ~fold_name  % Cancel Button pressed
27             return
28         end
29     case 1
30         fold_name = varargin{1};
31     otherwise
32         error('Input is not understood')
33 end
34
35 %  Go to the folder containing the files
36 oldfold = cd(fold_name);
37 file_list = dir('*.nii');
38 file_count = length(file_list);
39
40 %  Set up a directory for the outputs
41 newFolder = ['Output_',datestr(clock,1)];
42 mkdir(newFolder)
43
44 %  Make *.mat files to append the data to
45 m0001 = 0; f0001 = 0;
46 save(['./' newFolder '/m.mat'],'m0001');
47 save(['./' newFolder '/f.mat'],'f0001');
48
49 s = loadNII(file_list(1).name);
50 norm = ones(size(s));
51
52 % ===========
53 % = Do Work =
54 % ===========
55 % This will calculate the metabolism and blood flow.  The output is
```

```
% appended to 'm.mat' and 'f.mat' within a new folder created
% within the directory containing the data.

statusbar = waitbar(0,'Initializing');

maxBOLD = 0.22;

% Required Parameters
%   [alpha beta a      b      c      A    ]
p = [0.4 1.5 0.1870 0.1572 -0.6041 maxBOLD];

% Calc flow and metabolism for when BOLD = 1
s = 0;
y = -((p(4)*p(2))/(p(1)+p(2)*p(5)))*((p(6)-s)/(p(6)*p(3)^p(2)))^(1/(p(1)+p(2)*p(5)
   ));
fNOACT = -((p(1)+p(2)*p(5))/(p(4)*p(2)))*lambertw(y);
mNOACT = p(3)*fNOACT^(p(5)+1)*exp(-p(4)*fNOACT);


%%  Calc flow and metabolism
disp(fold_name)
for j=1:file_count
    try
      waitbar(j/file_count, statusbar, sprintf('%d%%', round((j/file_count)*100)))
         ;
    catch err
        return
    end
    s = loadNII(file_list(j).name);  %  Load up the file
    s(isnan(s)) = 1;
    s(isinf(s)) = 1;
    y = -((p(4)*p(2))/(p(1)+p(2)*p(5))).*((p(6)-s)./(p(6)*p(3)^p(2))).^(1/(p(1)+p
       (2)*p(5)));
    if (size(y,1)==91)&&(size(y,2)==109)&&(size(y,3)==91)
        f = -((p(1)+p(2)*p(5))/(p(4)*p(2))).*lambw_mex(real(y));
```

```
88      else
89          f = -((p(1)+p(2)*p(5))/(p(4)*p(2))).*lambw(y);
90      end
91      m = p(3)*f.^(p(5)+1).*exp(-p(4)*f);
92      % Clean up NaNs that may have popped up
93      m(isnan(m))=1;
94      f(isnan(f))=1;
95      % Normalize to resting m and f
96      m = m./mNOACT;
97      f = f./fNOACT;
98
99      % Rename and save the data
100     eval(['m' sprintf('%04d',j) ' = m;']);
101     eval(['f' sprintf('%04d',j) ' = f;']);
102     eval(['save(''./' newFolder '/m.mat'', ''m' sprintf('%04d',j) ''',''-append'')
          ;']);
103     eval(['save(''./' newFolder '/f.mat'', ''f' sprintf('%04d',j) ''',''-append'')
          ;']);
104     clear m0* f0*
105 end
106
107 close(statusbar)
108 cd(oldfold)
109
110 end
```

A.2.5 lambw() and lambw_mex()

The lambw() function is a wrapper for the wapr() function available on Matlab FileExchange (`http://www.mathworks.com/matlabcentral/fileexchange/3644-real-values-of-the-lambert-w-function/content/Lambert/wapr.m`). A compiled version of this function (lambw_mex()) runs much faster and is recommended. This function is used over Matlab's built-in Lambert-W function for the sake of performance.

```
1 function [ array_out ] = lambw( array_in )
2 % lambw Wrapper for wapr()
3 % Available:  http://www.mathworks.com/matlabcentral/fileexchange/3644-real-values
    -of-the-lambert-w-function/content/Lambert/wapr.m
```

```
%   Dwapr() doesn't work any arrays over Nx1, so this steps through the
%   full matrix and gives the rows to wapr.  Works pretty fast.

%#codegen

if ndims(array_in) ~= 3
    error('This only works (for now) with a three dimensional array.')
end

xmax = size(array_in,1);
ymax = size(array_in,2);

array_out = zeros(size(array_in));
for ix=1:xmax
    for iy=1:ymax
        array_out(ix,iy,:) = wapr(array_in(ix,iy,:));
    end
end

end
```

A.3 Calculating the Equilibrium Temperature

In order to determine the temperature fluctuations due to changes in activity, the baseline temperature must first be established for each voxel. The function tempCalcEquilibrium() will update the temperature using the Penne's bioheat equation (eq. (2.7)) until the change in temperature for each voxel falls below a certain threshold. Details about this procedure are available in section 2.3.1.

A.3.1 tempCalcEquilibrium()

```
1  function temperature_Out = tempCalcEquillibrium(tissue,bloodT,airT,nt,tmax,
       pastCalc,printprogress)
2  % tempCalcEquillibrium  Find the equillibrium values
3  %   tissue: holds all of the strucual information
4  %   bloodT: Temperature of the blood
5  %   airT:   Temperature of the surrounding ait
6  %   nt:     Max number of time steps
7  %   tmax:   Total amount of time the simulation should run over
8  %
9  %   This is based off of tempCalc() but loops until the rate of change of
10 %   a each voxel is sufficiently small then outputs what's
11 %   calculated.  If if takes too long to do all at once, split it up into
12 %   smaller time chunks and use the last step from the previous dataset as
13 %   pastCalc in order to resume.
14 %
15 %   Note: This does not save the time corse because it can take a lot of step to
16 %   find the equillibrium.  It outputs the last time step.
17 %
18 %   Writen by Greggory Rothmeier (greggroth@gmail.com)
19 %   Georgia State University Dept. Physics and Astronomy
20 %   May, 2011
21 tic
22 %%   Default Values
23 if nargin<2, bloodT = 37;       end
24 if nargin<3, airT = 24;         end
25 if nargin<4, nt = 100;          end
26 if nargin<5, tmax = 50;         end
```

```
27  if nargin<6, pastCalc = 0;       end
28  if nargin<7, printprogress = 1; end
29
30  % These rescue the data if the calculation is interrupted.
31  global temperature
32  global dirty
33
34  c = onCleanup(@InterCatch);
35  dirty = 1;
36
37  dx = 2*10^-3;         %  Voxel size (m)
38
39  if nt<(2*tmax),
40     warning('Time step size is not large enough.  Results will be unreliable.
       Consider increasing the number of steps or reducing tmax.')
41  end
42
43
44  % Constants used that aren't already stored in tissue
45  [xmax ymax zmax t] = size(tissue);
46  clear t;
47  dt = tmax/(nt-1);
48  % rhoBlood = 1057;
49  % wBlood = 1000;
50  % cBlood = 3600;
51
52  % =========
53  % = Setup =
54  % =========
55  %   Starts all tissue voxels at bloodT (default 37) and maintains air at airT (
    default 24)
56  %   The condition squeeze(tissue(:,:,:,)~=airIndex picks out the elements that are
57  %   tissue
58
59  temperature = ones(3,xmax,ymax,zmax,'single')*airT;
```

```
if pastCalc == 0
    temperature(1,squeeze(tissue(:,:,:,1))~=1) = bloodT;
else
    temperature(1,:,:,:) = pastCalc;
end
numElements = numel(temperature(1,:,:,:));

% ===========
% = Do Work =
% ===========
%   This is a vectorized version of the next section.  For the love of god
%   don't make any changes to this without first looking below to make sure
%   you know what you're changing.  This is [nearly] impossible to
%   understand, so take your time and don't break it.
%   data is stored in 'tissue' as such :
%   [tissuetype 0 Qm c rho k w];  <--  second element is blank for all.
%   [    1      2  3 4  5  6 7

%  This makes an array that has smoothed out variations in k by averaging all
%  of the k's around each voxel (including itself).  This is a hap-hazard
%  solution to the problem that if you only take the value of k for the voxel
%  without considering what surrounds it, it doesn't matter whether the head
%  is surrounded by air, water or anything else.  Since water is a better
%  thermal conductor than air, we need a way of accounting for this.  This is
%  one way:
averagedk = (circshift(tissue(:,:,:,6),[1 0 0])+circshift(tissue(:,:,:,6),[-1 0
   0])+circshift(tissue(:,:,:,6),[0 1 0])+circshift(tissue(:,:,:,6),[0 -1 0])+
   circshift(tissue(:,:,:,6),[0 0 1])+circshift(tissue(:,:,:,6),[0 0 -1])+tissue
   (:,:,:,6))/7;
rhoblood = 1057;
cblood = 3600;

%%  Specify Percision Goal
tolerence = 1;     % fraction of voxels have a slope less than 'zeropoint'
```

```
zeropoint = 2.5e-7;  % point at which the slope between two *steps* is considered
   essentially zero


goal = numElements - tolerence*numElements;
goon = numElements; %  Forces the while loop to run the first time
format shortG;
% temperature(1,:,:,:) = Current Temperature
% temperature(2,:,:,:) = Next Temperature
% Resets after each update
if printprogress
    disp(['Goal:  ', num2str(goal),' remaining voxels'])
end
t2 = 1;
while goon(1)>goal && t2<=nt  % runs until either 'goal' elements have a slope
   greater than 'zeropoint' or it exceeds nt
   if printprogress
    disp([t2 goon(1) ((numElements-goon(1))/numElements)*100]) % progress
   end
   temperature(2,:,:,:) = squeeze(temperature(1,:,:,:)) + ...
        dt/(tissue(:,:,:,5).*tissue(:,:,:,4)).* ...
        ((averagedk/dx^2).*...
        (circshift(squeeze(temperature(1,:,:,:)),[1 0 0])-2*squeeze(temperature
           (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[-1 0 0])+...  %
           shift along x
         circshift(squeeze(temperature(1,:,:,:)),[0 1 0])-2*squeeze(temperature
           (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[0 -1 0])+...  %
           shift along y
         circshift(squeeze(temperature(1,:,:,:)),[0 0 1])-2*squeeze(temperature
           (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[0 0 -1]))...  %
           shift along z
            -(1/6000)*rhoblood*tissue(:,:,:,7)*cblood.*(squeeze(temperature
              (1,:,:,:))-bloodT)+tissue(:,:,:,3));
    %   resets the air temperature back since it's also modified above, but
    %   it needs to be kept constant throughout the calculations
```

```
117     temperature(2,squeeze(tissue(:,:,:,1))==1) = airT;
118     %   checks how quickly the temperature is changing and if it is close
119     %   enough to zero to be considered stopped ('zeropoint')
120     goon = size(temperature(abs(squeeze(temperature(2,:,:,:)-temperature(1,:,:,:))
            )>zeropoint));
121     temperature(1,:,:,:) = temperature(2,:,:,:);  % moves 2 back to 1
122     t2 = t2 + 1;
123 end
124
125 temperature_Out = temperature(2,:,:,:);  % Only outputs the last time step
126 dirty = 0;
127
128 % equilTemperature = temperature_Out;
129 % save('equil.mat','equilTemperature');
130
131 %% To Combine Datasets
132 %  use this technique if there are seperate datasets that need combining
133 %    vertcat(squeeze(res1(:,:,:,:)),squeeze(res2(2:end,:,:,:)))
134 %  Where for all by the first dataset, you need to do the time from 2:end
135 %  so that there are no repeats (remember that the last timestep from the
136 %  previous dataset serves as the first for the new one)
137
138
139 time = toc;
140 end
141
142 function InterCatch
143 global dirty
144 if dirty
145     disp('Interupt Intercepted.  Inprepretating Interworkspace Data.')
146     global temperature
147     % equillibriumT = zeros([1 size(temperature(1,:,:,:))]);
148     % equillibriumT(1,:,:,:) = temperature(1,:,:,:);   %might be better to swtich
            equilT to be 3d rather than 4d
149     equillibriumT = temperature;
```

```
150         save('equiltempAbortDump.mat','equillibriumT');
151         % setappdata(0,'InterpOut',temperature);
152     end
153     end
```

A.4 Calculating the Temperature Change

The following function inputs the head data matrix (appendix A.1), the metabolism and blood flow time courses (crefsec:fmriprocessing) and the equilibrium temperatures (appendix A.3) and calculates the temperature time-course. More details about this algorithm can be found in section 2.3.1.

A.4.1 tempCalcDynMF

```
function temperatureOut = tempCalcDynMF(tissue,bloodT,airT,nt,tmax,pastCalc,metab,
   flow,savesteps,region)
% tempCalcChaning Metabolism  How does changin metabolism
% affect things?
%
%   tissue: holds all of the strucual information
%   bloodT: Temperature of the blood
%   airT:   Temperature of the surrounding ait
%   nt:     Number of time steps
%   tmax:   Total amount of time the simulation should run over
%
%   region: logical matrix same size as head
%
%   Writen by Greggory Rothmeier (greggroth@gmail.com)
%   Georgia State University Dept. Physics and Astronomy
%   May, 2011

statusbar = waitbar(0,'Initializing');

%%   Default Values
if nargin<2,  bloodT = 37;          end
if nargin<3,  airT = 24;            end
if nargin<4,  nt = 3;               end
if nargin<5,  tmax = 1;             end
if nargin<6,  pastCalc = 0;         end


% Length of one side of a voxel (m)
```

```
dx = 2*10^-3;

if nt<(2*tmax),
   warning('Time step size is not large enough.  Results will be unreliable.
      Consider increasing the number of steps or reducing tmax.')
end


% Constants used that aren't already stored in tissue
[xmax ymax zmax t] = size(tissue);
clear t;
dt = ones([xmax ymax zmax])*(tmax/(nt-1));
% rhoBlood = 1057;
% wBlood = 1000;
% cBlood = 3600;

%%  Determine Metab/Flow Data Storage System
if ischar(metab)&&ischar(flow)
  % if file locations are given rather than data
     option = 1;
else
  % Preallocate matrices for holding metabolism and blood flow data
     metabMulti = ones([xmax ymax zmax],'single');
     flowMulti = ones([xmax ymax zmax],'single');
     option = 0;
end

%%  Maps
% Creates a map that identifies where there is tissue
% the condition squeeze(tissue(:,:,:,)~=airIndex picks out the
% elements that are tissue

tmax = ceil((nt-1)/savesteps);
temperatureOut = ones(tmax,xmax,ymax,zmax,'single');
temperature = ones(2,xmax,ymax,zmax,'single')*airT;
```

```
62  if pastCalc == 0
63      temperature(1,squeeze(tissue(:,:,:,1))~=1) = bloodT;
64  else
65    % Starts everything off at the pre-determined equilibium temperatures
66      temperature(1,:,:,:) = pastCalc(end,:,:,:);
67  end
68  temperatureOut(1,:,:,:) = temperature(1,:,:,:);
69
70
71  % ===========
72  % = Do Work =
73  % ===========
74  %   This is a vectorized version of the next section.  For the love of
75  %   god don't make any changes to this without first looking below to
76  %   make sure you know what you're changing.  This is [nearly]
77  %   impossible to understand, so take your time and don't break it.
78  %   data is stored in 'tissue' as such :
79  %  [tissuetype 0 Qm c rho k w] <--  second element is blank for all.
80  %  [    1      2  3 4  5  6 7]
81
82  % This makes an array that has smoothed out variations in k by
83  % averaging all of the k's around each voxel (including itself). This
84  % is a hap-hazard solution to the problem that if you only take the
85  % value of k for the voxel without considering what surrounds it, it
86  % doesn't matter whether the head is surrounded by air, water or
87  % anything else. Since water is a better thermal conductor than air, we
88  % need a way of accounting for this. This is one way:
89
90  averagedk = (circshift(tissue(:,:,:,6),[1 0 0])+circshift(tissue(:,:,:,6),[-1 0
        0])+circshift(tissue(:,:,:,6),[0 1 0])+circshift(tissue(:,:,:,6),[0 -1 0])+
        circshift(tissue(:,:,:,6),[0 0 1])+circshift(tissue(:,:,:,6),[0 0 -1])+tissue
        (:,:,:,6))/7;
91  rhoblood = 1057;
92  cblood = 3600;
93
```

```
94  %%  Only saves every 4 steps to reduce the final matrix size
95  for t2 = 1:nt-1
96      waitbar(t2/(nt-1),statusbar,sprintf('%d%%',round(t2/(nt-1)*100)));
97
98  % if a variable needs to be used multiple times for the same time step.
99      t3 = floor((t2-1)/4)+1;  % 1 1 1 1 2 2 2 2 3 3 . . .
100
101     % if a file is specified, pulls the data from the file for each step
102     if option
103         eval(strcat('load(fullfile(metab),''-mat'',''m',sprintf('%04d',t3),''');'))
              ;
104         eval(strcat('load(fullfile(flow),''-mat'',''f',sprintf('%04d',t3),''');'));
105         eval(strcat('metabMulti = m',sprintf('%04d',t3),';'));
106         eval(strcat('flowMulti = f',sprintf('%04d',t3),';'));
107         eval(strcat('clear f', sprintf('%04d',t3),' m',sprintf('%04d',t3)))
108     else
109         metabMulti(region) = metab(t2);   %  region is hardcoded here
110         flowMulti(region) = flow(t2);
111     end
112
113     temperature(2,:,:,:) = squeeze(temperature(1,:,:,:)) + ...
114          dt./(tissue(:,:,:,5).*tissue(:,:,:,4)).* ...
115          ((averagedk/dx^2).*...
116          (circshift(squeeze(temperature(1,:,:,:)),[1 0 0])-2*squeeze(temperature
               (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[-1 0 0])+...  %
               shift along x
117           circshift(squeeze(temperature(1,:,:,:)),[0 1 0])-2*squeeze(temperature
               (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[0 -1 0])+...  %
               shift along y
118           circshift(squeeze(temperature(1,:,:,:)),[0 0 1])-2*squeeze(temperature
               (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[0 0 -1]))...  %
               shift along z
119              -(1/6000)*rhoblood*flowMulti.*tissue(:,:,:,7)*cblood.*(squeeze(
                   temperature(1,:,:,:))-bloodT)+metabMulti.*tissue(:,:,:,3));
120      % resets the air temperature back since it's also modified above,
```

```
    % but it needs to be kept constant throughout the calculations
    temperature(2,squeeze(tissue(:,:,:,1))==1) = airT;
    temperatureOut(ceil(t2/savesteps),:,:,:) = temperature(2,:,:,:);
    temperature(1,:,:,:) = temperature(2,:,:,:);  % moves 2 back to 1
    clear metabMulti flowMulti
end
close(statusbar);

% ============
% = Old Code =
% ============
% This is what used to be used. It's much slower (~60 times slower),
% but it's much easier to understand compared to the above code. If any
% changes need to be made above, first look through this code to ensure
% you understand what's happening before making changes. It's really
% easy to mess up the code above and nearly impossible to figure out
% where.
%
%  good luck.

% for t2 = 1:nt-1
%     for x2 = 2:xmax-1
%         for y2 = 2:ymax-1
%             for z2 = 2:zmax-1
%                 if tissue(x2,y2,z2,1) ~= 1,
%                     temperature(t2+1,x2,y2,z2) = temperature(t2,x2,y2,z2) + (dt
   /(tissue(x2,y2,z2,5)*tissue(x2,y2,z2,4)))*((tissue(x2,y2,z2,6)/dx^2)*...
%                       (temperature(t2,x2+1,y2,z2)-2*temperature(t2,x2,y2,z2)+
   temperature(t2,x2-1,y2,z2)+...
%                       temperature(t2,x2,y2+1,z2)-2*temperature(t2,x2,y2,z2)+
   temperature(t2,x2,y2-1,z2)+...
%                       temperature(t2,x2,y2,z2+1)-2*temperature(t2,x2,y2,z2)+
   temperature(t2,x2,y2,z2-1))...
%                       -(1/6000)*rhoBlood*wBlood*cBlood*(temperature(t2,x2,y2,z2)
   -bloodT)+tissue(x2,y2,z2,3));
```

```
151 %                     end
152 %                 end
153 %             end
154 %       end
155 % end
156
157 end
```

Appendix B

Visualization Tools

The temperature data is a four dimensional dataset (time, x, y and z), so good visualizations tools are necessary to analyzing the results. The primary tool I use is a modification of SliceBrowser (`http://www.mathworks.com/matlabcentral/fileexchange/20604`) and is provided as part of temptools (`https://github.com/greggroth/tempcalc/tree/master/tt/supportfcts/SliceBrowser`). In working with this, I also created a function (TempPlot()) to act as a wrapper and handle possible plotting situations depending on the number of inputs.

B.0.2 TempPlot()

```
1  function [ ] = TempPlot( head, tempdata, highlightRegion, slice, equil,threshold,
       point)
2  %TempPlot Plot data from tempCalc() or BulkImportNII()
3  %   INPUT TempPlot(structuredata)
4  %         TempPlot(structuredata,temperaturedata)
5  %         TempPlot(structuredata,temperaturedata,highlightRegion)
6  %         TempPlot(structuredata,temperaturedata,highlightRegion,slice)
7  %         TempPlot(structuredata,temperaturedata,highlightRegion,slice,
       EquillibriumData)
8  %
9  %   This function with determine which type of data it is and then plot it
10 %   appropiately.
11 %
12 %   equil - Equillibrium state data
```

```
13  %   threshold - threshold value for being displayed as an overlay
14  %   REQUIRES:  SliceBrowser (http://www.mathworks.com/matlabcentral/fileexchange
        /20604)
15  %%  Error checking and data restructuring where necessary
16  if ndims(head) == 4
17      head = head(:,:,:,1);
18  elseif ndims(head) ~= 3
19      error('Input ''head'' must have either 3 or 4 dimensions');
20  end
21
22  if nargin > 1
23      if ndims(tempdata) == 3  % should only happen when comparing two equilibrium
            datasets
24      temp = tempdata;
25      tempdata = zeros([1 size(temp)]);
26      tempdata(1,:,:,:) = temp;
27      elseif ndims(tempdata) ~= 4
28      error('Input ''tempdata'' must have either 3 or 4 dimensions');
29      end
30      tempdataShort = squeeze(tempdata(end,:,:,:));
31  end
32
33  if nargin > 2
34      if ndims(highlightRegion) ~= 3
35      error('Input ''highlightRegion'' must have 3 dimensions');
36      end
37      if size(highlightRegion) ~= size(head)
38      error('Input ''highlightRegion'' must be of the same size as ''head''');
39      end
40      tempdataShort = squeeze(tempdata(end,:,:,:));
41  end
42
43  if nargin > 3
44      if slice > size(tempdata,1)
```

```
45      error('Input ''slice'' must be less or equal to the length of the first
            dimension of ''tempdata''');
46      end
47      tempdataShort = squeeze(tempdata(slice,:,:,:));
48  end
49
50  if nargin > 4
51      if ndims(equil) == 3
52          eq = equil;
53      elseif ndims(equil) == 4
54          eq = squeeze(equil(1,:,:,:));
55      else
56          error('Input ''equil'' must have either 3 or 4 dimensions');
57      end
58      clear 'equil';
59  end
60
61  %%  Pick how to format the call of SliceBrowser()
62  switch nargin
63      case 1
64      SliceBrowser(head,1,head);
65      colormap(gray);
66      case 2
67      %SliceBrowser(squeeze(tempdata(size(tempdata,1),:,:,:)),tempdata,head);
68      SliceBrowser(tempdataShort,tempdata,head);
69      case 3
70      SliceBrowser(tempdataShort,tempdata,head,highlightRegion);
71      case 4
72      SliceBrowser(tempdataShort,tempdata,head,highlightRegion);
73      case 5
74      SliceBrowser(tempdataShort-eq,tempdata,head,highlightRegion);
75      case 6
76      SliceBrowserOverlay(tempdataShort-eq,tempdata,head,highlightRegion,threshold);
77      case 7
78      imgoverlay(head,tempdataShort-eq,point,threshold)
```

```
79  end
80
81  end
```

B.0.3 tsliceplot

This is a visualization tool I wrote that allows you to view the change in temperature versus time for a line passing through the head. Screenshots of the tool can be seen in figs. B.1 and B.2.

Usage:

```
tsliceplot(temperature_data, equilibrium_temperature_data)
```

The script is available as part of temptools (`https://github.com/greggroth/tempcalc/tree/master/tt/supportfcts/tsliceplot`).

Figure B.1: Experimental data for activity in the motor cortex visualized with tsliceplot.

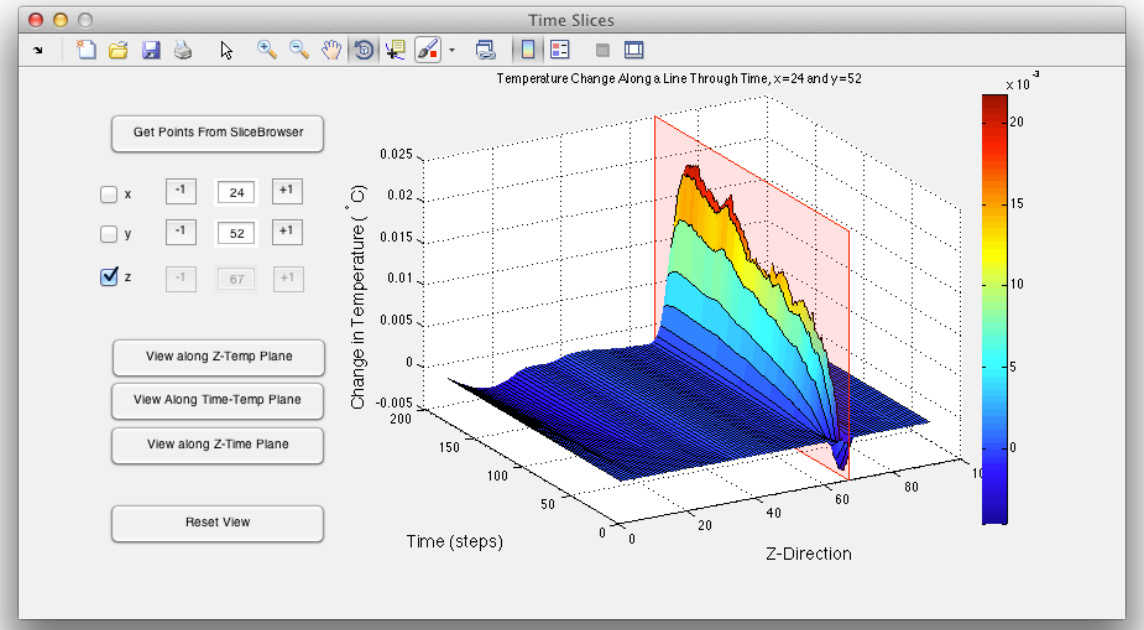


Figure B.2: The same data as is presented in fig. B.1, but viewed flat-on along the z vs. time plane.