

Brain tissue temperature dynamics during functional activity and possibilities for Imaging

by

GREGGORY H. ROTHMEIER

Under the Direction of A. G. Unil Perera

ABSTRACT

Regional tissue temperature dynamics in the brain is determined by the balance of the metabolic heat production rate and heat exchange with blood flowing through capillaries embedded in the tissue, the surrounding tissues and the environment. Local changes in blood flow and metabolism during functional activity can upset this balance and induce transient temperature changes. Invasive experimental studies in animal models have established that the brain temperature changes during functional activity are observable and a definitive relationship exists between temperature and brain activity. Here, we present a theoretical framework that links tissue temperature dynamics with hemodynamic activity allowing us to noninvasively estimate brain temperature changes from experimentally measured blood-oxygen level dependent (BOLD) signals. With this unified approach, we are able to pinpoint the mechanisms for hemodynamic activity-related temperature increases and decreases. Applying this approach to experimentally measured fMRI timeseries recorded in a finger-movement task, we also provide an estimate of the temperature change during a functional activity in the brain.

INDEX WORDS: fMRI, BOLD, Temperature, etc.

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

Copyright by
Greggory Rothmeier
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	4
2.2 Modeling the BOLD Response	5
2.3 Modeling Temperature	6
2.3.1 The Approach	6
How the temperature is calculated	6
Calculating the equilibrium temperature	6
Calculating Metabolism and Blood Flow Changes	6
Calculating the change in temperature in the active brain	6
2.3.2 Results	6
Using Theoretical BOLD Data	6
Using Experimental BOLD Data	7
3 Detector Applications to measuring the active brain	10
3.1 Functional Near-Infrared fNIR Imaging	10
3.2 Temperature Measurements	12
4 Conclusion	13

References	13
Appendices	15
A Code	15
A.1 Creating the Head Matrix	20
A.1.1 ImportSegmentedT1()	20
A.1.2 fillAir()	23
A.1.3 fillHoles()	24
A.1.4 build_skin()	25
A.1.5 repair_headdata()	26
A.2 Loading the fMRI Data	28
A.2.1 sample_bold_import()	28
A.2.2 avg_NII_rest()	31
A.2.3 avg_NII_normalize()	34
A.2.4 BOLDtoMF()	37
A.2.5 lambw() and lambw_mex()	40
A.3 Calculating the Equilibrium Temperature	42
A.3.1 tempCalcEquilibrium()	42
A.4 Calculating the Temperature Change	48
A.4.1 tempCalcDynMF	48
B Visualization Tools	54
B.0.2 TempPlot()	54
B.0.3 tsliceplot	57

List of Tables

2.1 Parameters used in the single-voxel approximation	5
-----------------------------------------------------------------	---

List of Figures

2.1	Generation of the fMRI BOLD response and a corresponding temperature change	3
2.2	Procedure used to calculate temperature change	6
2.3	Slice of the segmented head. Each color represents a different tissue type.	7
2.4	Temperature changes: simulated BOLD data	8
2.5	Temperature changes: experimental BOLD data	9
3.1	Absorption spectra of water, deoxyhemoglobin and oxyhemoglobin	11
A.1	temptools: main window	16
A.2	temptools: calculate the equilibrium temperature window	17
A.3	temptools: calculate temperature during activity	18
A.4	temptools: visualize the data	19
B.1	Visualization using tsliceplot	57
B.2	Visualization using tsliceplot (z v. t plane)	58

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

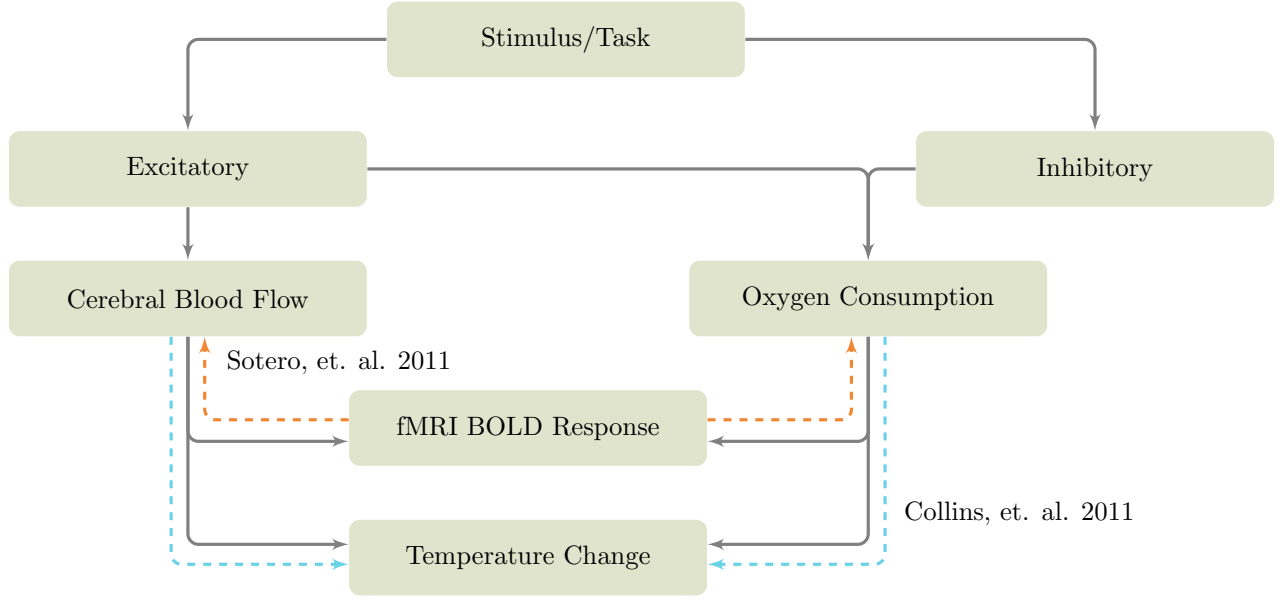


Figure 2.1: Generation of the fMRI BOLD response from changes in neuronal activity. Black arrows indicate a causal relationship while colored dashed-arrows indicate existing models for the relationship. The orange line (●) shows the model proposed by Sotero and Iturria-Medina [3] to calculate cerebral blood flow and metabolism and the blue line (●) shows how the model proposed by Collins et al. [4] is used to calculate temperature.

iteration of a single-voxel model was published by Sotero and Iturria-Medina [3]. The basis of this model is a modification of the Penne's Bioheat Equation [5, 3].

$$C_t \frac{dT(t)}{dt} = (\Delta H^\circ - \Delta H_b) CMRO_2|_0 m(t) - \rho_b C_b CBF|_0 f(t)(T(t) - T_a) - \frac{C_t}{\tau}(T(t) - T_0) \quad (2.1)$$

where BLA BLA BLA. One advantage of using eq. (2.1) is that the resting state temperature can be analytically determined by substituting $\frac{dT(t)}{dt} = 0$ [3].

$$T_0 = T_a + \frac{(\Delta H^\circ - \Delta H_b) CMRO_2|_0}{\rho_B C_B CBF|_0} \quad (2.2)$$

If the values provided in table 2.1 are substituted 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.

$$\text{change in temperature} = \text{heat generated by metabolism} - \text{heat lost to convection} - \text{heat lost to conduction} \quad (2.3)$$

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.

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

$$m(t) = af^{c+1}(t)e^{-bf(t)} \quad (2.5)$$

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

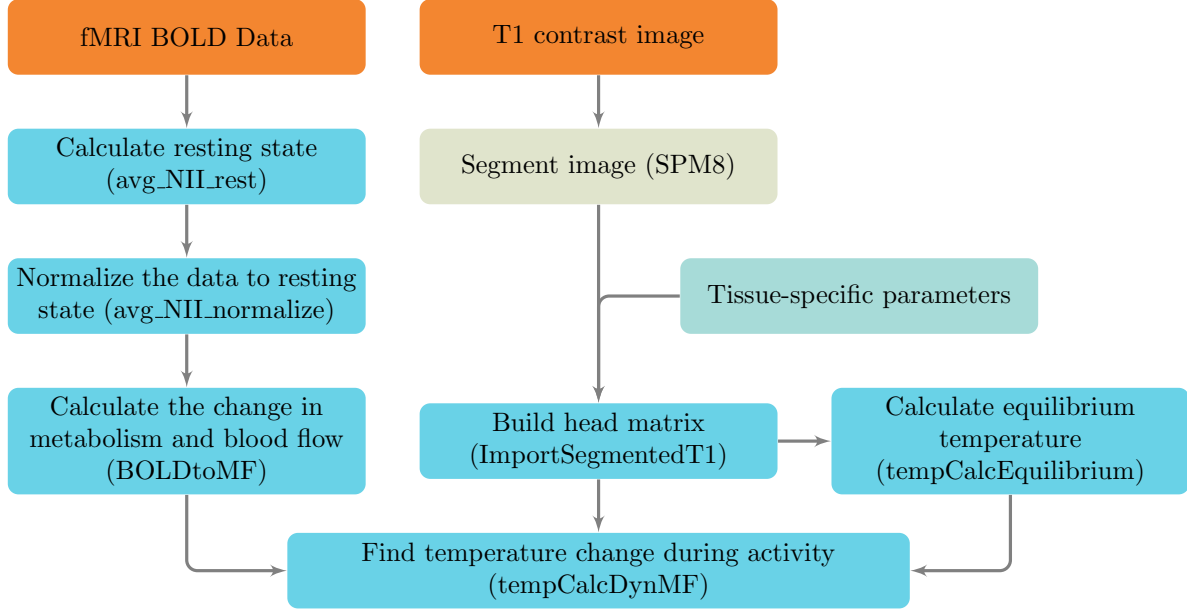
Multi-Voxel Approach

Table 2.1: Parameters used to solve the single-voxel Penne’s Bioheat Equation. (modified from Sotero and Iturria-Medina [3])

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 ₀	Cerebral metabolic rate of O ₂ consumption at rest	0.026310 ⁻⁶ mol/(gs)
CBF 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	

2.2 Modeling the BOLD Response

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).



2.3 Modeling Temperature

2.3.1 The Approach

How the temperature is calculated

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

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

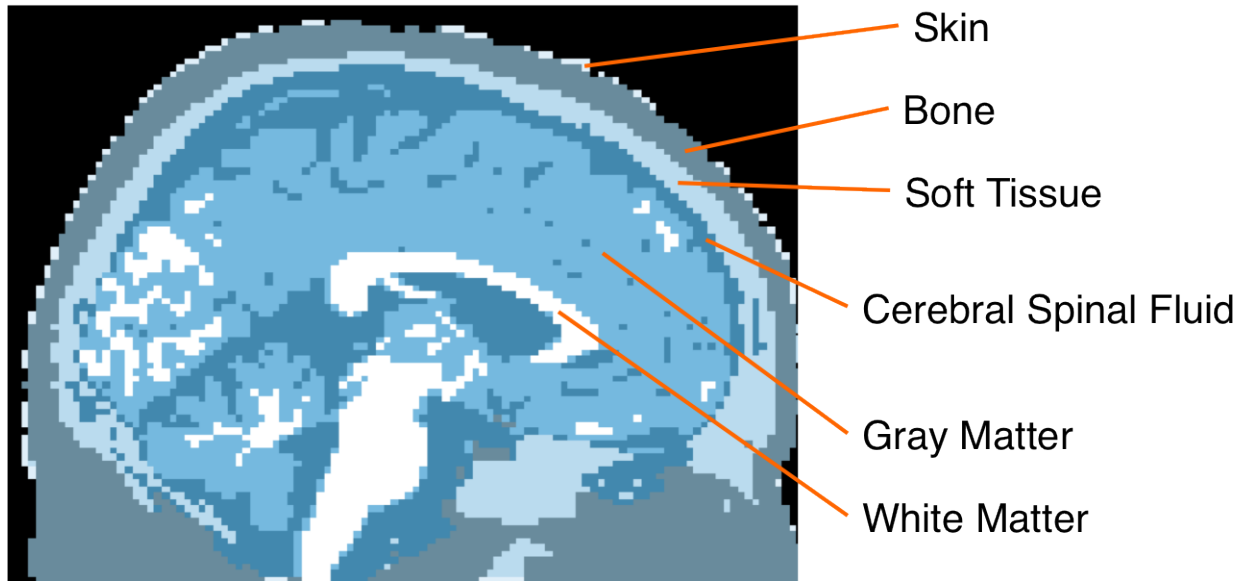


Figure 2.3: Slice of the segmented head. Each color represents a different tissue type.

Using Experimental BOLD Data

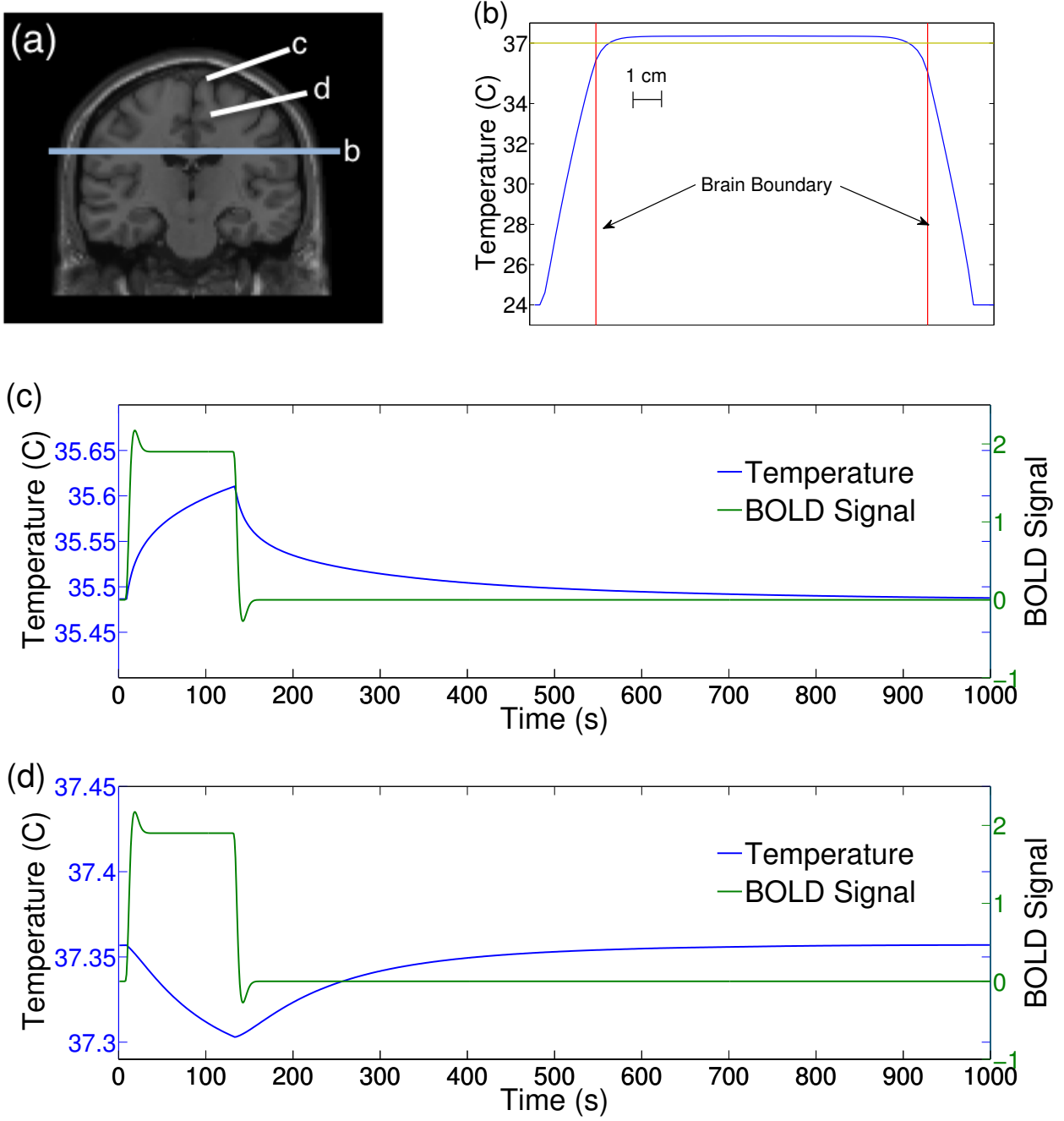


Figure 2.4: Temperature changes using simulated BOLD signals. (a) Slice of the head ($y = -12$) with indicators of the locations for parts (b)-(d). (b) Equilibrium temperature along a line through the head. Red lines indicate the brain boundary and the gold line indicates the blood temperature (37°C) 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.

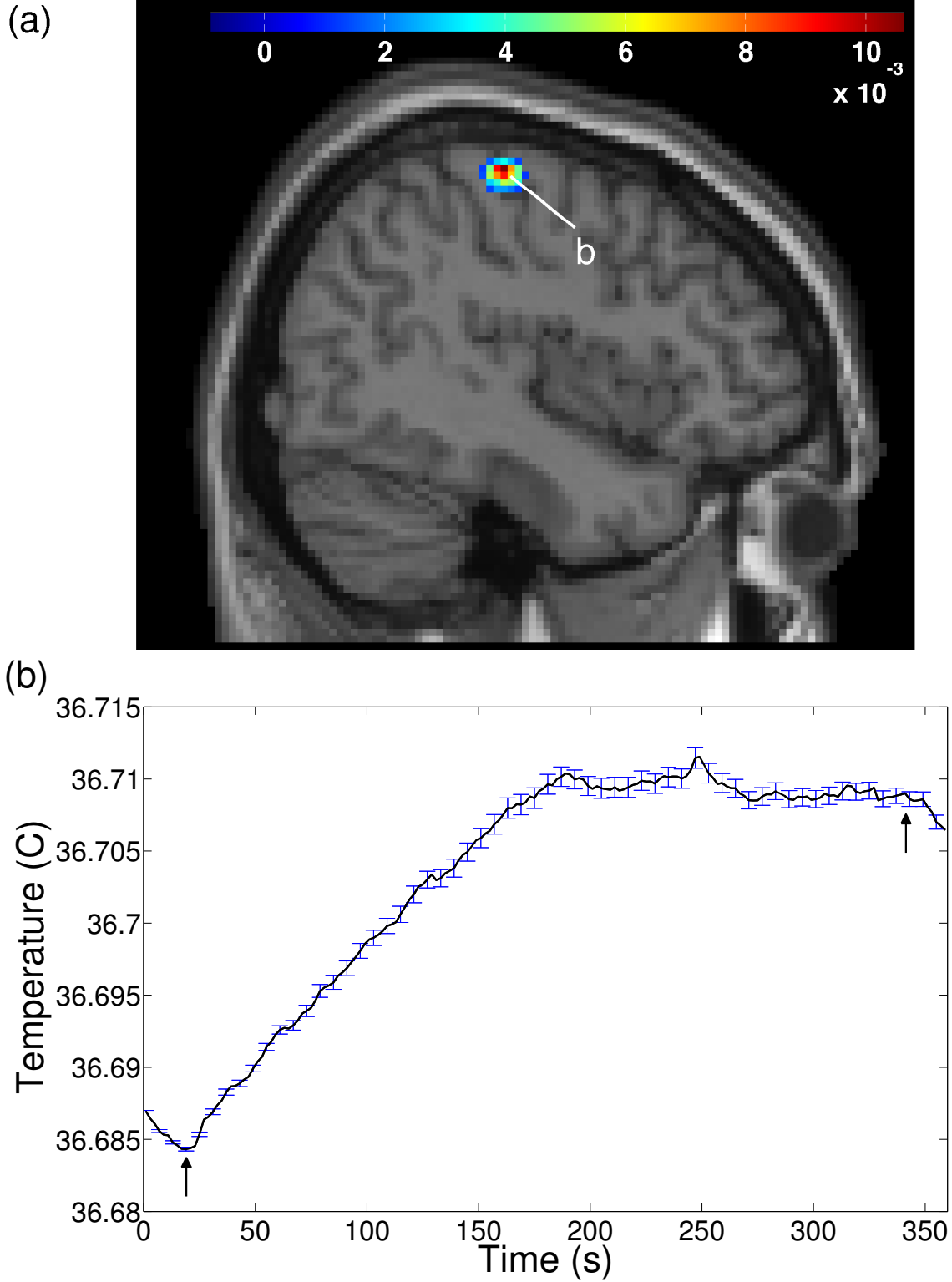


Figure 2.5: Temperature calculated from a voxel within the motor cortex. (a) A slice ($x = -44$) showing the motor cortex warming during a finger-tapping task. (b) Temperature at a voxel within the motor cortex ($-44, -24, 60$) with standard error indicated by blue error bars (Arrows indicate task onset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

Chapter 3

Detector Applications to measuring the active brain

3.1 Functional Near-Infrared fNIR Imaging

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

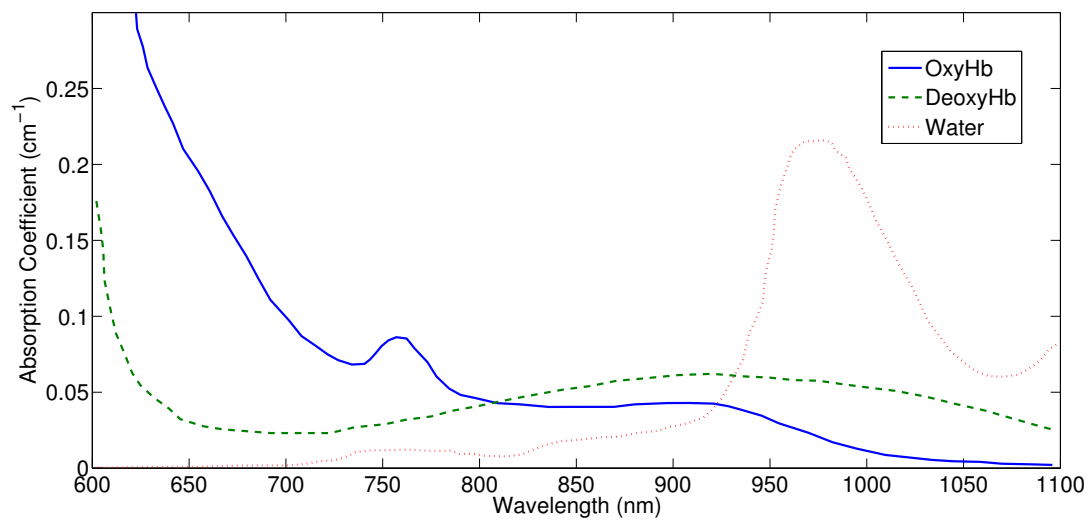


Figure 3.1: Absorption spectra of water, Hb and Dhb. From Cope [6] and HB stuff from Horecker [7]

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 Yasser Iturria-Medina. From Blood Oxygen Level Dependent (BOLD) signals to brain temperature maps. *Bulletin of Mathematical Biology*, 73(11):2731–2747, 2011.
- [4] Christopher M. Collins, Michael B. Smith, and Robert Turner. Model of local temperature changes in brain upon functional activation. *Journal of Applied Physics*, 97:2051–2055, 2004.
- [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 R2011b 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/temptools>). Additionally, many of the tasks can be completed using the temptools gui (figs. A.1 to A.4) which can be invoked by running

```
temptools
```

at the Matlab command prompt (make sure the temptools directory and subdirectories have been added to the Matlab path).

Figure A.1: The main window of temptools. From here, you can go through the calculation steps and launch the visualization tool.

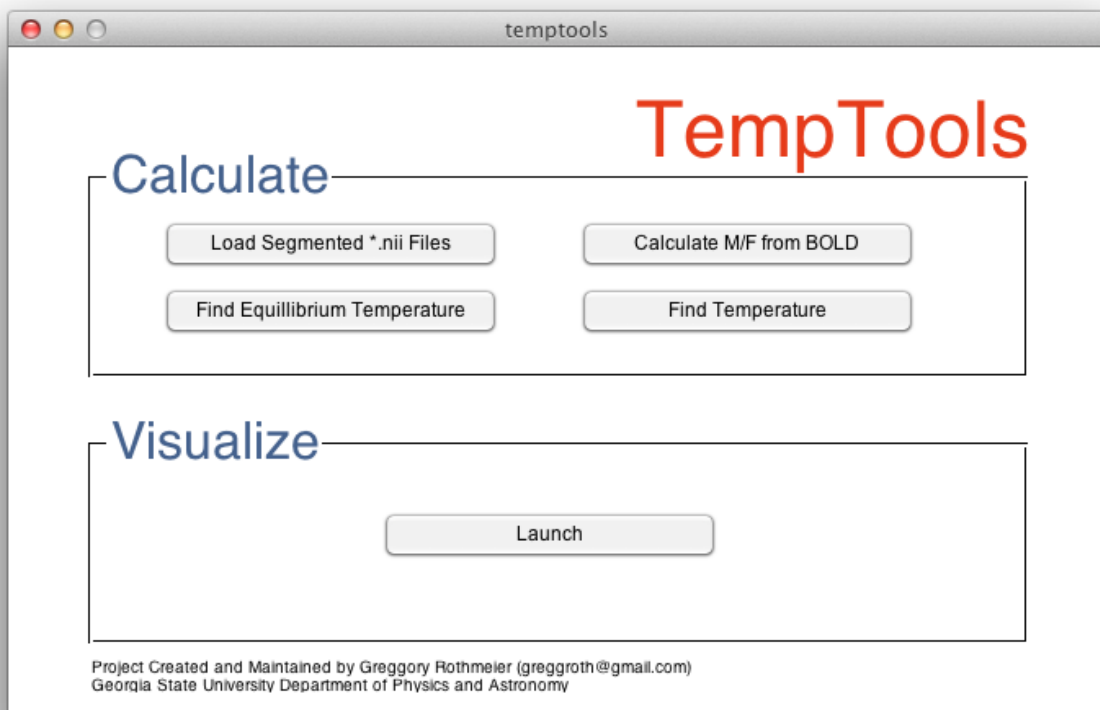


Figure A.2: This is the interface for calculating the equilibrium temperature (method explained in appendix A.3) under certain conditions.

findequilfigure

Find Equilibrium Temperatures

Load Head Data

Blood Temperature

Air Temperature

Max Simulation Time Allowed (s)

Steps per second

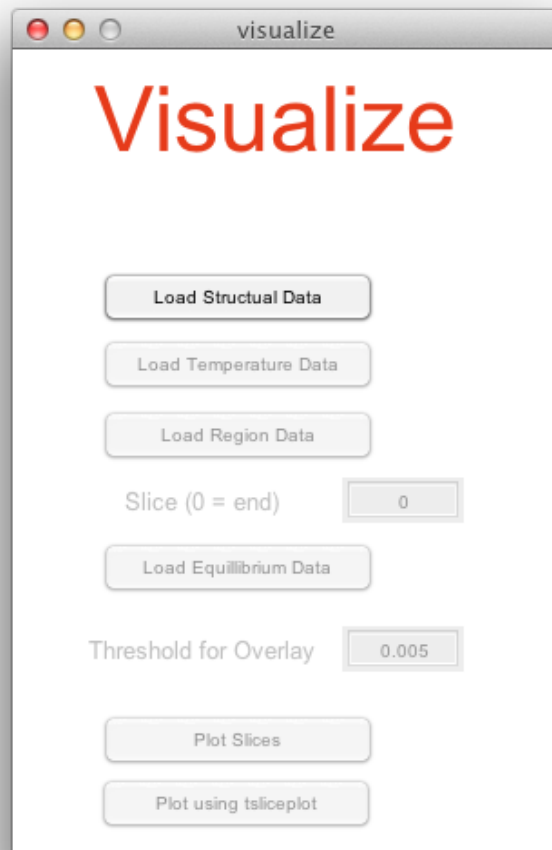
Start

Figure A.3: The interface for calculating temperature changes when blood flow and metabolism are time dependent. This can be achieved by either loading metabolism and blood flow datasets or by using generated activity.

The interface is titled "Temperature with Activity" and is part of a program named "findtempactivation". It contains the following elements:

- Data Loading Buttons:**
 - Load Head Data
 - Load Equilibrium Temp Data
 - Load Region of Interest
 - Load Metabolism Timecourse
 - Load Blood Flow Timecourse
- Use Generated Activity Section:**
 - Normalized Change in Metabolism: 1
 - Normalized Change in Blood Flow: 1
 - Start (Steps): 5 (1.67 s)
 - Stop (Steps): 10 (3.33 s)
 - Activation Duration: 1.67 s
- Input Fields and Labels:**
 - Blood Temperature: 37
 - Air temperature: 24
 - Max Time: 30
 - Step Frequency: 3
 - Save Every [6] Steps: 15 steps saved
 - Total Steps: 90
- Start Button:** A large button at the bottom labeled "Start".

Figure A.4: Visualize your data using the temptools visualization window. This loads all of the required data and launches a slice browser or tsliceplot (see appendix B for more details).



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" algorithm 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  %
5  % Output: head data as single with variables stored in the 4th dimension.
6  %
7  % Author: Gregory Rothmeier (greggroth@gmail.com)
8  % Georgia State University
9  % Created: 5/31/11
10
11  statusbar = waitbar(0,'Initializing');
12
13  if size(varargin) == 1
14      oldFolder = cd(varargin{1});
15  end
16
17
18  % =====
19  % = Tissue Parameters =
20  % =====
21  % Each tissue type is assigned an integer index (i.e. gray matter -> 11) such that
```

```

22 % tissue-specific parameters can be found by looking at that element within the
23 % corresponding storage matrix (i.e. QmSTORE(11) -> gray matter Qm)
24
25 % Parameters taken from Colins, 2004
26
27 tisorder = [11 15 5 13 3]; % Using: [GM WM CSF Muscle Bone]
28
29 QmSTORE = [0 0 26.1 11600 0 26.1 697 0 0 302 15575 0 697 1100 5192];
30 cSTORE = [1006 4600 2110 3640 3800 1300 3720 3000 4200 2300 3680 3500 3720 3150
31           3600];
32 rhoSTORE = [1.3 1057 1080 1035.5 1007 1850 1126 1076 1009 916 1035.5 1151 1041
33             1100 1027.4];
34
35 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
36           .342 .503];
37
38 wSTORE = [0 1000 3 45.2 0 1.35 40 0 0 2.8 67.1 3.8 3.8 12 23.7];
39
40 % =====
41 % = Import the pre-segmented T1 files =
42 % =====
43 % The T1 contrast image should be segmented using SPM8.
44 % This loop needs to complete before the next one can begin
45 % Import all of the datat and store as 'cdat1','cdat2', etc.
46
47 for i = 1:5
48     eval(strcat('dat',num2str(i),' = loadNII(''rc', num2str(i), 'single_subj_T1.
49             nii'');'))
50
51     % Preallocate
52     eval(strcat('out', num2str(i),' = zeros(cat(2,size(dat', num2str(i),'),7));'))
53 end
54
55 % =====
56 % = Populate the head matrix =
57 % =====
58 % For each data file, it fills in the data from the data storage arrays
59 % for that particular type of tissue. It picks which ever tissue is the
60 % most likely candidate for that voxel based on the segmented data

```



```

53
54 % PROBLEM: It returns 0 (later filled with air) if there is equal
55 % probability of a voxel being two or more different types of tissue.
56 % SOLVED BY fillHoles()
57
58
59 for i = 1:5
60     % Preallocate
61     holder = zeros(cat(2,size(dat1),7),'single');
62     mask = zeros(size(dat1));
63     final = zeros(size(holder),'single');
64
65     % Create a mask that indicates whether it is the mostly likely tissue type
66     guide = [1 2 3 4 5 1 2 3 4 5]; % This guides it through the data correctly
67     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
3 % Once you import all of the data using loadNII(), run it through this to
4 % fill in the remaining spaces with air.
5
6 airdata = [1 0 0 1006 1.3 0.026 0];
7
8 % Picks out air spots
9 a = find(tissue(:,:, :,1) == 0);
10 [x y z t] = ind2sub(size(tissue),a);
11
12 for i = 1:length(a)
13     tissue(x(i),y(i),z(i),:) = airdata;
14 end
15
16 output = tissue;
17
18 end

```

A.1.3 fillHoles()

```
1  function [ out_head ] = fillHoles( in1,in2,in3,in4,in5,headin)
2  % fillHoles() checks for misassigned voxels
3  %
4  % Solves an issue where a voxel with two equally probable tissue
5  % types resulted in being assigned as air. This checks for air
6  % voxels that are surrounded by tissue and decides a tissue it
7  % it would be best suited as
8
9  head = squeeze(headin(:,:,:,:),1)); % I only need the tissue indices so this makes
    things easier down the line
10
11 %% Data Storage
12 QmSTORE = [0 0 26.1 11600 0 26.1 697 0 0 302 15575 0 697 1100 5192];
13 cSTORE = [1006 4600 2110 3640 3800 1300 3720 3000 4200 2300 3680 3500 3720 3150
    3600];
14 rhoSTORE = [1.3 1057 1080 1035.5 1007 1850 1126 1076 1009 916 1035.5 1151 1041
    1100 1027.4];
15 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];
16 wSTORE = [0 1000 3 45.2 0 1.35 40 0 0 2.8 67.1 3.8 3.8 12 23.7];
17
18 %% Get locations of holes
19 % Where two tissue types have the same probability
20
21 idx1 = (in1==in2 | in1 == in3 | in1==in4 | in1==in5) & logical(in1);
22 idx2 = (in1==in2 | in2 == in3 | in2==in4 | in2==in5) & logical(in2);
23 idx3 = (in1==in3 | in2 == in3 | in3==in4 | in3==in5) & logical(in3);
24 idx4 = (in1==in4 | in2 == in4 | in3==in4 | in4==in5) & logical(in4);
25 idx5 = (in1==in5 | in2 == in5 | in3==in5 | in4==in5) & logical(in5);
26 % This array will have a zero anywhere there were two or more common
27 % elements between any of the five arrays.
28 idx = idx1|idx2|idx3|idx4|idx5;
29
30 [xmax ymax zmax] = size(in1)
```

```

31 [x y z] = ind2sub(size(in1),find(idx)); % get x, y and z coordinates of the
    holes
32
33 for i = 1:length(x) % go to each hole and do work
34     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
35         [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)]));
36         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.
37             commonesttissue = secondbest{1}(2);
38         end
39         headin(x(i),y(i),z(i),:) = [commonesttissue 0 QmSTORE(commonesttissue)
            cSTORE(commonesttissue) rhoSTORE(commonesttissue) kSTORE(
            commonesttissue) wSTORE(commonesttissue)];
40     end
41 end
42
43 out_head = headin;
44
45 end

```

A.1.4 build_skin()

```

1 function [ head_out ] = build_skin( head_in )
2 % build_skin() Creates a layer of skin around the head
3 %
4 % This will check all voxels that were previously labeled
5 % as soft tissue and checks if it has a neighbor which is air.
6 % If so, then it is reassigned as skin.
7
8 if ndims(head_in)==4
9     head_in = head_in(:,:,:,1);

```

```

10 end
11
12 % Get a list of all voxels labeled as muscle
13 muscle_voxels = find(head_in==13);
14
15 % Go through each of them and check for neighboring air voxels
16 for i=1:length(muscle_voxels)
17     [x y z] = ind2sub(size(head_in), muscle_voxels(i));
18     % makes sure we're not at a voxel at the boundry of the dataset
19     if (x~=1) && (x~=size(head_in,1)) && (y~=1) && (y~=size(head_in,2)) && (z~=1)
        && (z~=size(head_in,3))
20         % Looks for neighboring voxels that are air
21         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)
            )
22             head_in(x,y,z) = 14;
23         end
24     end
25 end
26
27 head_out = repair_headdata(head_in);
28
29 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.

```

1 function [ head_out ] = repair_headdata( head_in )
2 % repaid_headdata repopulates the headdata matrix
3 % If any changes are made to the index column in the headdata matrix, use
4 % this function to repopulate and correct the parameter values before running
5 % any other functions.
6 % head_in can be either 3 or 4 dimenisions

```

```

7
8
9 % =====
10 % = Parameter Storage =
11 % =====
12
13 QmSTORE = [0 0 26.1 11600 0 26.1 697 0 0 302 15575 0 500 1100 5192];
14 cSTORE = [1006 4600 2110 3640 3800 1300 3720 3000 4200 2300 3680 3500 3010 3150
            3600];
15 rhoSTORE = [1.3 1057 1080 1035.5 1007 1850 1126 1076 1009 916 1035.5 1151 978.5
              1100 1027.4];
16 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];
17 wSTORE = [0 1000 3 45.2 0 1.35 40 0 0 2.8 67.1 3.8 3.3 12 23.7];
18
19 if ndims(head_in)==4
20     head_in = head_in(:,:,:,1);
21 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 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.

```
1  %%=====
2  %%      How to process preprocessed BOLD data to calculate temperature
3  %%=====
4
5  % This Matlab script was used to automate the the process of using BOLD data
6  % stored in NIFTI (*.nii) format to calculate temperature changes. The
7  % particulars of the code may be specific to this case, but the procedure
8  % should be the same when doing these calculations on other datasets. All
9  % required functions are included as an attachment to my thesis and are
10 % available on my github (https://github.com/greggroth/tempcalc)
11
12 cd('/Users/Greggory/Documents/Data/fmri_rhythmic_tapping01/NIFTI')
13
14 directories = dir('*01');
15
16 %% Move coregistered files to new Directory
17 for i = 1:length(directories)
18     dir_name = directories(i).name;
19     main_path = cd( [dir_name filesep dir_name '_NIFTI_1'] );
20     mkdir 'Coregistered'
21     movefile('r*.nii','Coregistered')
22     main_path = cd( [dir_name filesep dir_name '_NIFTI_2'] );
23     mkdir 'Coregistered'
24     movefile('r*.nii','Coregistered')
25     cd(main_path)
26 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'
42         ], fullfile(dir_name, [dir_name '_NIFTI_1'], 'Coregistered', 'RestState', '
43         RestStateAvg.nii'), 'fullBrainMask.nii');
44     avg_NII_normalize([dir_name filesep dir_name '_NIFTI_2' filesep 'Coregistered'
45         ], fullfile(dir_name, [dir_name '_NIFTI_2'], 'Coregistered', 'RestState', '
46         RestStateAvg.nii'), 'fullBrainMask.nii');
47 end
48
49
50 %% Calculate metabolism and blood flow change
51 disp('Calculate metabolism and blood flow change')
52 for i = 1:length(directories)
53     dir_1 = [ directories(i).name filesep directories(i).name '_NIFTI_1' filesep
54         'Coregistered' filesep 'Normalized_to_rest'];
55     dir_2 = [ directories(i).name filesep directories(i).name '_NIFTI_2' filesep
56         'Coregistered' filesep 'Normalized_to_rest'];
57     BOLDtoMF(dir_1);
58     BOLDtoMF(dir_2);
59 end
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```



```

56 %% Calculate the change in temperature based on metabolism and blood flow
57
58 % load('equil.mat'); % equilibriumT
59 % load('tt_headdata.mat'); % headdata
60 mask = loadNII('fullBrainMask.nii');
61
62 for i = 1:length(directories)
63     disp([int2str(i) '-1 started'])
64     tic
65     % Part I
66     actResult.dat = tempCalcDynMF(headdata, 37, 24, 720, 360, equilibriumT, ...
67         fullfile(directories(i).name,[directories(i).name '_NIFTI_1'],'
68             Coregistered', 'Normalized_to_rest', 'Output_18-Sep-2011', 'm.mat'),
69         ...
70         fullfile(directories(i).name,[directories(i).name '_NIFTI_1'],'
71             Coregistered', 'Normalized_to_rest', 'Output_18-Sep-2011', 'f.mat'),
72         ...
73         4, mask);
74
75 % Store the parameters used for the calculations for reference in the future
76 [c lmax] = max(actResult.dat(:));
77 [likelymax x y z] = ind2sub(size(actResult.dat),lmax);
78 actResult.likelymaxslice = round(likelymax/2);
79 actResult.bloodT = 37;
80 actResult.airT = 24;
81 actResult.tmax = 360;
82 actResult.stepf = 2;
83 actResult.savestepf = 4;
84 actResult.metabandflowdata = 'From Dataset';
85 save(fullfile(directories(i).name,[directories(i).name '_NIFTI_1'],'
86     Coregistered', 'Normalized_to_rest', 'Output_18-Sep-2011','tt_act_res.mat')
87     , 'actResult');
88
89 old = cd([directories(i).name,filesep,[directories(i).name '_NIFTI_1'],filesep
90     , 'Coregistered', filesep, 'Normalized_to_rest', filesep, 'Output_18-Sep-2011'
91     ]);
92 writeT_to_nii(actResult, equilibriumT, 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, equilibriumT, ...
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, equilibriumT, exp_nii);
103    cd(old)
104    clear actResult
105    disp([int2str(i) ' finished in ' num2str(toc)])
106 end

```

A.2.2 avg_NII_rest()

```

1  function [ ] = avg_NII_rest( varargin )
2  %UNTITLED4 Summary of this function goes here
3  %   Detailed explanation goes here
4
5  %% Setup
6  switch length(varargin)
7      case 0
8          fold_name = uigetdir;
9          if ~fold_name % Cancel Button
10             return
11         end
12      case 1
13          fold_name = varargin{1};
14      otherwise
15  end
16
17  % Go to the folder containing the files
18  oldfold = cd(fold_name);
19  file_list = dir('*.nii');
20
21  % We're only interested in the rest period
22  % (first and last 10 steps in this case)
23  file_list = file_list([1:10 170:180]);
24  file_count = length(file_list);
25
26  % Cell array to store all of the datasets in.
27  datHolder = cell(file_count,1);
28
29  statusbar = waitbar(0,'Initializing');
30
31  for j=1:file_count
32      try
33          waitbar(j/file_count,statusbar,sprintf('%d%%',round((j/file_count)*100)));
34      catch err
35          return

```

```

36     end
37     fi = load_nii(file_list(j).name);
38     datHolder{j} = fi.img;
39 end
40
41 %% Calculate the mean
42 ymax = size(datHolder{1},2);
43 zmax = size(datHolder{1},3);
44 output = zeros(size(datHolder{1}));
45
46 for i=1:ymax
47     try
48         waitbar(i/ymax,statusbar,sprintf('%d%%',round((i/ymax)*100)));
49     catch err
50         return
51     end
52     for k=1:zmax
53         excStr = cell(length(datHolder),1);
54         for l=1:length(datHolder)
55             excStr{l} = [',datHolder{' int2str(l) '}'(:,' int2str(i) ',' int2str(k)
                    ''),''];
56         end
57         comb = eval(['cat(1' cell2mat(excStr) ' ')]);
58         output(:,i,k) = mean(comb);
59     end
60 end
61
62 close(statusbar)
63
64 fi.img = output;
65 mkdir('RestState')
66 save_nii(fi,fullfile('RestState','RestStateAvg.nii'));
67
68 cd(olddir)
69

```

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
14              File');
15          switch rest_index
16              case 0
17                  return
18              case 1
19                  rest_dat = load_nii(fullfile(rest_path,rest_file));
20                  rest_dat = double(rest_dat.img);
21              otherwise
22                  error('An error has occured loading the resting state data')
23          end
24
25          [mask_file mask_path mask_index] = uigetfile('*.nii','Mask');
26          switch mask_index
27              case 0
28                  return
29              case 1
30                  mask_dat = load_nii(fullfile(mask_path, mask_file));
31                  mask_dat = logical(mask_dat.img);
32                  if max(size(mask_dat)) ~= size(rest_dat))
```

```

32         error('The Mask and Resting State files must have the same
33             size')
34     end
35     otherwise
36         error('An error has occurred loading the resting state data')
37     end
38 case 1
39     fold_name = varargin{1};
40     [rest_file rest_path rest_index]= uigetfile('*.nii','Resting State NIFTI
41         File');
42     switch rest_index
43     case 0
44         return
45     case 1
46         rest_dat = load_nii(fullfile(rest_path,rest_file));
47         rest_dat = double(rest_dat.img);
48     otherwise
49         error('An error has occurred loading the resting state data')
50     end
51 case 2
52     fold_name = varargin{1};
53     rest_dat = loadNII(varargin{2});
54     [mask_file mask_path mask_index] = uigetfile('*.nii','Mask');
55     switch mask_index
56     case 0
57         return
58     case 1
59         mask_dat = load_nii(fullfile(mask_path, mask_file));
60         mask_dat = logical(mask_dat.img);
61         if max(size(mask_dat) ~= size(rest_dat))
62             error('The Mask and Resting State files must have the same
63                 size')
64         end
65     otherwise
66         error('An error has occurred 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 directory 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, divide 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/
90             file_count)*100))] );
91     catch err
92         return
93     end
94     [file_path file_name file_ext] = fileparts(file_list(i).name);
95     file_hold = load_nii(file_list(i).name);
96     file_hold.img = double(file_hold.img)./rest_dat - 1;
97     file_hold.img(~mask_dat) = 0; % set everything outside the mask to
98     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

```

```

56 % appended to 'm.mat' and 'f.mat' within a new folder created
57 % within the directory containing the data.
58
59 statusbar = waitbar(0,'Initializing');
60
61 maxBOLD = 0.22;
62
63 % Required Parameters
64 % [alpha beta a      b      c      A      ]
65 p = [0.4 1.5 0.1870 0.1572 -0.6041 maxBOLD];
66
67 % Calc flow and metabolism for when BOLD = 1
68 s = 0;
69 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)
    ));
70 fNOACT = -((p(1)+p(2)*p(5))/(p(4)*p(2)))*lambertw(y);
71 mNOACT = p(3)*fNOACT^(p(5)+1)*exp(-p(4)*fNOACT);
72
73
74 %% Calc flow and metabolism
75 disp(fold_name)
76 for j=1:file_count
77     try
78         waitbar(j/file_count, statusbar, sprintf('%d%%', round((j/file_count)*100)))
79         ;
80     catch err
81         return
82     end
83     s = loadNII(file_list(j).name); % Load up the file
84     s(isnan(s)) = 1;
85     s(isinf(s)) = 1;
86     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)));
87     if (size(y,1)==91)&&(size(y,2)==109)&&(size(y,3)==91)
88         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

```

```

4  %   Dwapr() doesn't work any arrays over Nx1, so this steps through the
5  %   full matrix and gives the rows to wapr.  Works pretty fast.
6
7  %#codegen
8
9  if ndims(array_in) ~= 3
10     error('This only works (for now) with a three dimensional array.')
11 end
12
13 xmax = size(array_in,1);
14 ymax = size(array_in,2);
15
16 array_out = zeros(size(array_in));
17 for ix=1:xmax
18     for iy=1:ymax
19         array_out(ix,iy,:) = wapr(array_in(ix,iy,:));
20     end
21 end
22
23 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 = tempCalcEquilibrium(tissue,bloodT,airT,nt,tmax,
    pastCalc,printprogress)
2  % tempCalcEquilibrium Find the equilibrium values
3  %   tissue: holds all of the structural information
4  %   bloodT: Temperature of the blood
5  %   airT:   Temperature of the surrounding air
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 it 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 course because it can take a lot of steps to
16 %   find the equilibrium. It outputs the last time step.
17 %
18 %   Written by Gregory 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.
41             Consider increasing the number of steps or reducing tmax.')
42 end
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 (
56 % default 24)
57 % The condition squeeze(tissue(:,:,,:).~=airIndex picks out the elements that are
58 % tissue
59
60 temperature = ones(3,xmax,ymax,zmax,'single')*airT;

```

```

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

```

```

91 zeropoint = 2.5e-7; % point at which the slope between two *steps* is considered
    essentially zero
92
93
94 goal = numElements - tolerance*numElements;
95 goon = numElements; % Forces the while loop to run the first time
96 format shortG;
97 % temperature(1,:,:,:) = Current Temperature
98 % temperature(2,:,:,:) = Next Temperature
99 % Resets after each update
100 if printprogress
101     disp(['Goal: ', num2str(goal), ' remaining voxels'])
102 end
103 t2 = 1;
104 while goon(1)>goal && t2<=nt % runs until either 'goal' elements have a slope
    greater than 'zeropoint' or it exceeds nt
105     if printprogress
106         disp([t2 goon(1) ((numElements-goony(1))/numElements)*100]) % progress
107     end
108     temperature(2,:,:,:) = squeeze(temperature(1,:,:,:)) + ...
109         dt/(tissue(:,:,:5).*tissue(:,:,:4)).* ...
110         ((averagedk/dx^2).*...
111         (circshift(squeeze(temperature(1,:,:,:)),[1 0 0])-2*squeeze(temperature
            (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[-1 0 0])+... %
            shift along x
112         circshift(squeeze(temperature(1,:,:,:)),[0 1 0])-2*squeeze(temperature
            (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[0 -1 0])+... %
            shift along y
113         circshift(squeeze(temperature(1,:,:,:)),[0 0 1])-2*squeeze(temperature
            (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[0 0 -1]))... %
            shift along z
114         -(1/6000)*rhoblood*tissue(:,:,:7)*cblood.*(squeeze(temperature
            (1,:,:,:))-bloodT)+tissue(:,:,:3));
115     % resets the air temperature back since it's also modified above, but
116     % 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     % equilibriumT = zeros([1 size(temperature(1,:,:,:))]);
148     % equilibriumT(1,:,:,:) = temperature(1,:,:,:); %might be better to swtich
        equilT to be 3d rather than 4d
149     equilibriumT = temperature;

```

```
150     save('equiltempAbortDump.mat','equilibriumT');
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:fmripprocessing) 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

```
1  function temperatureOut = tempCalcDynMF(tissue,bloodT,airT,nt,tmax,pastCalc,metab,
    flow,savesteps,region)
2  % tempCalcChaning Metabolism How does changin metabolism
3  % affect things?
4  %
5  % tissue: holds all of the strucual information
6  % bloodT: Temperature of the blood
7  % airT: Temperature of the surrounding ait
8  % nt: Number of time steps
9  % tmax: Total amount of time the simulation should run over
10 %
11 % region: logical matrix same size as head
12 %
13 % Writen by Gregory Rothmeier (greggroth@gmail.com)
14 % Georgia State University Dept. Physics and Astronomy
15 % May, 2011
16
17 statusbar = waitbar(0,'Initializing');
18
19 %% Default Values
20 if nargin<2, bloodT = 37; end
21 if nargin<3, airT = 24; end
22 if nargin<4, nt = 3; end
23 if nargin<5, tmax = 1; end
24 if nargin<6, pastCalc = 0; end
25
26
27 % Length of one side of a voxel (m)
```

```

28 dx = 2*10^-3;
29
30 if nt < (2*tmax),
31     warning('Time step size is not large enough. Results will be unreliable.
32         Consider increasing the number of steps or reducing tmax.')
33 end
34
35 % Constants used that aren't already stored in tissue
36 [xmax ymax zmax t] = size(tissue);
37 clear t;
38 dt = ones([xmax ymax zmax])*(tmax/(nt-1));
39 % rhoBlood = 1057;
40 % wBlood = 1000;
41 % cBlood = 3600;
42
43 %% Determine Metab/Flow Data Storage System
44 if ischar(metab)&&ischar(flow)
45     % if file locations are given rather than data
46     option = 1;
47 else
48     % Preallocate matrices for holding metabolism and blood flow data
49     metabMulti = ones([xmax ymax zmax], 'single');
50     flowMulti = ones([xmax ymax zmax], 'single');
51     option = 0;
52 end
53
54 %% Maps
55 % Creates a map that identifies where there is tissue
56 % the condition squeeze(tissue(:,:,,:))~=airIndex picks out the
57 % elements that are tissue
58
59 tmax = ceil((nt-1)/savesteps);
60 temperatureOut = ones(tmax,xmax,ymax,zmax, 'single');
61 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 equilibrium 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     ;
105     eval(strcat('load(fullfile(flow))','-mat','f',sprintf('%04d',t3),''));
106     eval(strcat('metabMulti = m',sprintf('%04d',t3),''));
107     eval(strcat('flowMulti = f',sprintf('%04d',t3),''));
108     eval(strcat('clear f', sprintf('%04d',t3),' m',sprintf('%04d',t3)))
109 else
110     metabMulti(region) = metab(t2); % region is hardcoded here
111     flowMulti(region) = flow(t2);
112 end
113
114 temperature(2,:,:,:) = squeeze(temperature(1,:,:,:)) + ...
115     dt./((tissue(:,:,:,5).*tissue(:,:,:,4)).* ...
116     ((averagedk/dx^2).*...
117     (circshift(squeeze(temperature(1,:,:,:)),[1 0 0])-2*squeeze(temperature
118         (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[-1 0 0])+... %
119         shift along x
120     circshift(squeeze(temperature(1,:,:,:)),[0 1 0])-2*squeeze(temperature
121         (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[0 -1 0])+... %
122         shift along y
123     circshift(squeeze(temperature(1,:,:,:)),[0 0 1])-2*squeeze(temperature
124         (1,:,:,:))+circshift(squeeze(temperature(1,:,:,:)),[0 0 -1]))... %
125         shift along z
126     -(1/6000)*rhoblood*flowMulti.*tissue(:,:,:,7)*cblood.*(squeeze(
127         temperature(1,:,:,:))-bloodT)+metabMulti.*tissue(:,:,:,3));
128 % resets the air temperature back since it's also modified above,

```

```

121     % but it needs to be kept constant throughout the calculations
122     temperature(2,squeeze(tissue(:,:,,1))==1) = airT;
123     temperatureOut(ceil(t2/savesteps),:,:,:) = temperature(2,:,:,:);
124     temperature(1,:,:,:) = temperature(2,:,:,:); % moves 2 back to 1
125     clear metabMulti flowMulti
126 end
127 close(statusbar);
128
129 % =====
130 % = Old Code =
131 % =====
132 % This is what used to be used. It's much slower (~60 times slower),
133 % but it's much easier to understand compared to the above code. If any
134 % changes need to be made above, first look through this code to ensure
135 % you understand what's happening before making changes. It's really
136 % easy to mess up the code above and nearly impossible to figure out
137 % where.
138 %
139 % good luck.
140
141 % for t2 = 1:nt-1
142 %     for x2 = 2:xmax-1
143 %         for y2 = 2:ymax-1
144 %             for z2 = 2:zmax-1
145 %                 if tissue(x2,y2,z2,1) ~= 1,
146 %                     temperature(t2+1,x2,y2,z2) = temperature(t2,x2,y2,z2) + (dt
147 %                         /(tissue(x2,y2,z2,5)*tissue(x2,y2,z2,4)))*((tissue(x2,y2,z2,6)/dx^2)*...
148 %                         (temperature(t2,x2+1,y2,z2)-2*temperature(t2,x2,y2,z2)+
149 %                         temperature(t2,x2-1,y2,z2)+...
150 %                         temperature(t2,x2,y2+1,z2)-2*temperature(t2,x2,y2,z2)+
151 %                         temperature(t2,x2,y2-1,z2)+...
152 %                         temperature(t2,x2,y2,z2+1)-2*temperature(t2,x2,y2,z2)+
153 %                         temperature(t2,x2,y2,z2-1))...
154 %                         -(1/6000)*rhoBlood*wBlood*cBlood*(temperature(t2,x2,y2,z2)
155 %                         -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/temptools/tree/master/lib/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(structureddata)
4 % TempPlot(structureddata,temperaturedata)
5 % TempPlot(structureddata,temperaturedata,highlightRegion)
6 % TempPlot(structureddata,temperaturedata,highlightRegion,slice)
7 % TempPlot(structureddata,temperaturedata,highlightRegion,slice,
    EquillibriumData)
8 %
9 % This function with determine which type of data it is and then plot it
10 % appropriately.
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/temptools/tree/master/lib/tsliceplot>).

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

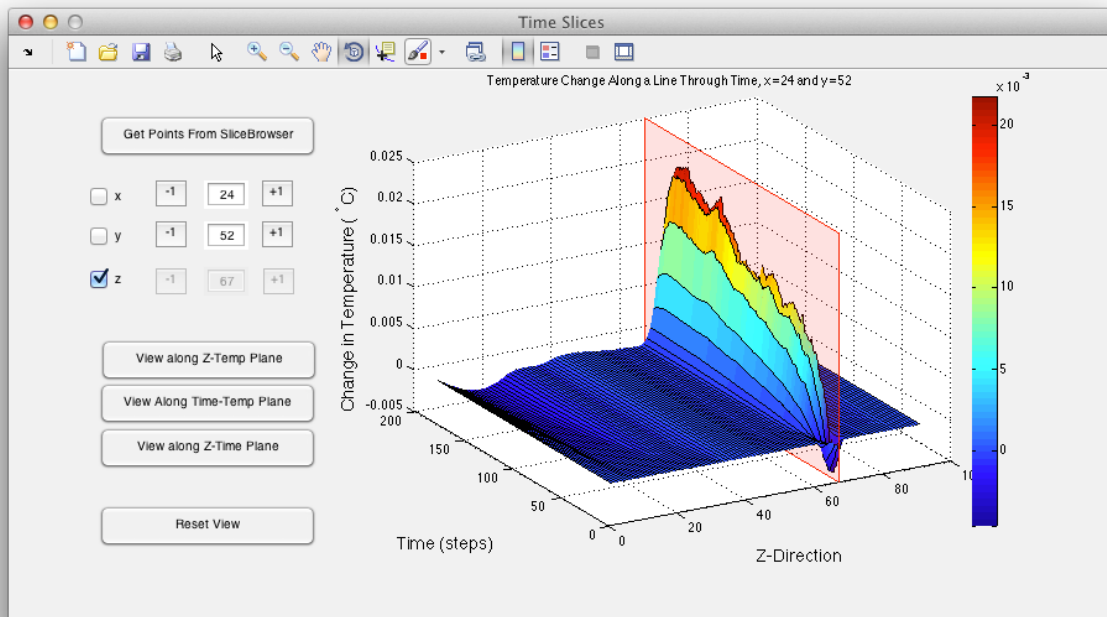


Figure B.2: The same data as is presented in fig. B.1, but viewed flat-on along the z vs. time plane.

