Design and Implementation of Convex Analysis of Mixtures Software Suite

Fan Meng

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science

In

Computer Engineering

Yue Wang (Chair)
Jianhua Xuan
Chang-Tien Lu

August 07, 2012
Arlington, VA

Keywords: R script, Graphic User Interface, Convex Analysis of Mixtures, Compartment Modeling
Design and Implementation of Convex Analysis of Mixtures
Software Suite

Fan Meng

Abstract

Various convex analysis of mixtures (CAM) based algorithms have been developed to address real world blind source separation (BSS) problems and proven to have good performances in previous papers. This thesis reported the implementation of a comprehensive software CAM-Java, which contains three different CAM based algorithms, CAM compartment modeling (CAM-CM), CAM non-negative independent component analysis (CAM-nICA), and CAM non-negative well-grounded component analysis (CAM-nWCA). The implementation works include: translation of MATLAB coded algorithms to open-sourced R alternatives. As well as building a user friendly graphic user interface (GUI) to integrate three algorithms together, which is accomplished by adopting Java Swing API.

In order to combine R and Java coded modules, an open-sourced project RCaller is used to handle the establishment of low level connection between R and Java environment. In addition, specific R scripts and Java classes are also implemented to accomplish the tasks of passing parameters and input data from Java to R, run R scripts in Java environment, read R results back to Java, display R generated figures, and so on. Furthermore, system stream redirection and multi-threads techniques are used to build a simple R messages displaying window in Java built GUI.

The final version of the software runs smoothly and stable, and the CAM-CM results on both simulated and real DCE-MRI data are quite close to the original MATLAB version algorithms. The whole GUI based open-sourced software is easy to use, and can be freely distributed among the communities. Technical details in both R and Java modules implementation are also discussed, which presents some good examples of how to develop software with both complicate and up to date algorithms, as well as decent and user friendly GUI in the scientific or engineering research fields.
First and foremost, I should thank my family. My parents give me unconditional supports not only during the master degree studying period, but also all through my life. It is their love and confidence to me make all these happen. Also, a special thanks to my friend Lisa, her encouragement during my thesis writing is priceless to me.

I will also give my deepest gratitude to my advisor, Dr. Yue Wang. His consistent guidances and advices during two years of master studying are always the most important things for me. And without his enlightened and detailed modification of my thesis, this paper cannot reach to what it looks now.

I also sincerely appreciate the help from Dr. Jianhua Xuan and Dr. Chang-Tien Lu. Their advice to my thesis and final exam illuminate me a lot. Also thank for all the works they accomplished as to be my committee members.

Finally, my special thanks go to all my lab mates in the Computational Bioinformatics and Bioimaging Laboratory. Dr. Niya Wang and Dr. Li Chen offered me a lot of suggestions of my thesis project. Dr. Guoqiang Yu and Dr. Ye Tian kindly offered their apartments for me to live in during the first and the last semesters, which did great help for my start up as well as final graduation. Other lab mates, Dr. Bai Zhang, Dr. Jinhua Gu, Dr. Xi Chen, Dr. Xiao Wang, and Ji Wang, etc., are also so kind to help me a lot in everyday life in the lab.
# Table of Contents

Acknowledgements .................................................................................................................. iii

Table of Contents .................................................................................................................... iv

List of Figures ........................................................................................................................... v

List of Tables ............................................................................................................................. vi

Chapter 1 Introduction ............................................................................................................... 1
  1.1 Background ....................................................................................................................... 1
  1.2 Motivation ......................................................................................................................... 4
  1.3 Organization of the Thesis ............................................................................................... 6

Chapter 2 Implementation of Major Algorithms by R ................................................................ 8
  2.1 Structure of the Algorithms .......................................................................................... 8
  2.2 Implementation Details ................................................................................................ 10
  2.3 Cases Studies and Algorithm Verifications .................................................................... 14

Chapter 3 Implementation of Graphic User Interface by Java ..................................................... 18
  3.1 Structure and Design of GUI Software ........................................................................... 18
  3.2 Packages Description ..................................................................................................... 19
    3.2.1 GUI displaying and Event Handling Package ......................................................... 19
    3.2.2 Data Modeling Package ......................................................................................... 21
  3.3 Illustration of the Software Usage ................................................................................ 23

Chapter 4 Interactive Integration of R and Java Modules ........................................................... 27
  4.1 Works in R Module ......................................................................................................... 27
  4.2 Works in Java Module .................................................................................................... 30
    4.2.1 Importing Data and Parameters to R Environment ................................................. 30
    4.2.2 Running R Scripts and Read Results from R Environment .................................... 31
    4.2.3 Display Figures Drawn by R .................................................................................. 32
    4.2.4 Showing R Generated Information during Calculation .......................................... 33

Chapter 5 Discussion and Future Work ................................................................................... 37
  5.1 Discussion ....................................................................................................................... 37
  5.2 Future Work .................................................................................................................... 39

References .................................................................................................................................... 41
List of Figures

Figure 2.1 Normalized tracer concentration results ................................................................. 17
Figure 3.1 The layout of the main frame of CAM-Java with default look and feel ......................... 20
Figure 3.2 Input Dialog for receiving filepath of “Rscript.exe” in CAM-Java ............................... 24
Figure 3.3 Results of applying CAM-CM algorithm to data of a typical DCE-MRI case .................... 25
Figure 3.4 Contrast/tracer concentration results and a scatter plot of final demixing signal ............... 26
Figure 4.1 Sketch map of system streams redirections in CAM-Java ......................................... 34
Figure 4.2 Sketch map of multi-threads used in CAM-Java ...................................................... 36
List of Tables

Table 2.1 Parameter estimated obtained by the R code and MATLAB code of the CAM-CM algorithm, tested on the simulation dataset 1................................................................. 14
Table 2.2 Parameter estimated obtained by the R code and MATLAB code of the CAM-CM algorithm, tested on the simulation dataset 2................................................................. 15
Table 2.3 Parameter estimated obtained by the R code and MATLAB code of the CAM-CM algorithm, tested on the simulation dataset 3................................................................. 15
Table 2.4 Parameter estimated obtained by the R code and MATLAB code of the CAM-CM algorithm, tested on the simulation dataset 4................................................................. 15
Chapter 1 Introduction

1.1 Background

Blind source separation (BSS) has proven to be a powerful and widely-applicable tool for the analysis and interpretation of composite patterns in engineering and science, where both source patterns and mixing proportions are of interest but unknown. BSS is often described by a linear latent variable model $\mathbf{X} = \mathbf{A}\mathbf{S}$, where $\mathbf{X}$ is the observation data matrix, $\mathbf{A}$ is the unknown mixing matrix, and $\mathbf{S}$ is the unknown source data matrix. The fundamental objective of BSS is to estimate both the unknown mixing proportions and source signals based on only the observed mixtures. Most BSS algorithms aim to find a matrix factorization of data under certain assumptions (e.g., source independence or sparse solution) which may be invalid for real-world BSS problems. Convex analysis of mixtures (CAM) method has been recently developed and implemented via various algorithms for different real-world applications, including CAM compartment modeling (CAM-CM) implemented in MATLAB, CAM non-negative independent component analysis (CAM-nICA) implemented in R, and CAM non-negative well-grounded component analysis (CAM-nWCA) implemented in MATLAB.

Dynamic contrast-enhanced magnetic resonance imaging (DCE-MRI) provides a noninvasive method for evaluating tumor vasculature patterns based on contrast accumulation and washout. DCE-MRI can potentially depict intratumor heterogeneity of vascular permeability which reflects tumor angiogenic activity. However, the quantitative application of DCE-MRI has been hindered by its inability to accurately resolve vascular compartments of distinct pharmacokinetics due to limited imaging resolution. This indistinction between contributions of different compartments to the mixed tracer signals could confound compartment modeling and genotype-phenotype association studies. The goal of the CAM-CM was to discern vascular heterogeneity and its changes in tumors using DCE-MRI and novel mathematical modeling tools, for personalized cancer diagnosis and treatment.

CAM-CM algorithm has been developed for deconvolving intratumor vascular heterogeneity and identifying pharmacokinetics changes in many biological contexts [1-2]. This method works by exploiting convex analysis of mixtures that enables geometrically-principled delineation of distinct vascular structures from DCE-MRI data. Tumors to be analyzed by CAM-CM contain unknown numbers of distinct vascular compartments. In these tumors, the contributed signal of pixel-wise tracer concentration in a particular vascular compartment is modeled as being proportional to the local volume transfer constant of the vascular compartment (Online Method). Because there are often significant numbers of partial-volume pixels, CAM-CM instead estimates
pharmacokinetics parameters (flux rate constants) via the time-activity curves of pure-volume pixels (pixels whose signal is highly enriched in a particular vascular compartment). Convex analysis of mixtures automatically identifies pure-volume pixels resided at the vertices of the clustered pixel time series scatter simplex, without any knowledge of compartment distribution. When the number of underlying vascular compartment is appropriately chosen by the minimum description length (MDL) criterion, CAM-CM signifies a completely unsupervised approach for characterizing intratumor heterogeneity.

Furthermore, CAM-nWCA/nICA algorithm has also been developed to directly address non-negative BSS problems. Assume that sources contain sufficient number of well-grounded points (WGPS) at which signals are highly expressed in one source relative to each of the remaining sources, the goal is to estimate the column vectors of mixing matrix by identifying WGPS located at the corners of mixture observation scatter simplex and subsequently recover the hidden source signals. However, in these techniques there are three potential limitations. First, the method is lack of theoretical proof in model identifiability and solution optimality [1]. Second, the solution (including model selection) is sensitive to data noise and outliers. Third, the computational complexity is high when analyzing high dimensional data. Based on a geometrical latent variable model, CAM learns the mixing matrix by identifying the lateral edges of convex data scatter plot. The algorithm is supported theoretically by a well-grounded mathematical framework and practically by plug-in noise filtering using sector-based clustering, efficient convex analysis scheme, and stability-based model selection.

Main steps of CAM algorithm in a typical CAM-CM analysis are as follows:

1) Masking the regions of interest (ROIs) in the image and adopting vector-norm filtering to remove noise or outlier pixels.
2) Projecting the pixel-time series onto standard scatter simplex.
3) Three additional core algorithms are implemented to processing the data in order to further reduce noise or outliers. Including multivariate clustering based on standard finite normal mixture (SFNM) model, affinity propagation clustering (APC), and the expectation maximization (EM) algorithm.
4) Pure volume pixels can then be picked by identifying corner cluster centers of the scatter plot convex hull. The way of finding corner cluster centers is using exhaustive combinatorial search (works when the number of tissues and number of pixels are relative small).

The MATLAB implementation of the CAM-CM algorithm have already been accomplished and available from the website of Computational Bioinformatics & Bio-imaging Laboratory (CBIL).

http://www.cbil.ece.vt.edu/software.htm
In addition to CAM-CM algorithm, other CAM based algorithms are also being development in CBIL. These algorithms are aimed to make use of the CAM algorithm to different real-world applications. CAM-nICA and CAM-nWCA algorithms have also been implemented by R and MATLAB, respectively.
1.2 Motivation

As aforementioned, original CAM based algorithms (i.e., CAM-CM, CAM-nICA, and CAM-nWCA) have been implemented in MATLAB or R code. One important issue is that the MATLAB license may limit the distribution of implemented software, since the software cannot be used without MATLAB environment, and intended users may not able to (or may not want to) buy the expensive commercial software. This fact motivates us to implement the algorithms with advanced open-source programming languages, when combined R and Java implementation has been taken into consideration.

R is available as free software under GNU General Public License [3]. It has been popular in the statistical analysis field because of its abundant statistical libraries and high quality plotting functions. Compared to other statistical tools, like SPSS, SAS and Stata, one of the main differences is that R is not just a “tool” for specific usages, but is designed to be a general purposed system environment, which means R can be easily extended by packages. In the current version of R (up to August 2012, the newest version is 2.15.1), numerous R packages make it competitive with MATLAB, and easily cooperate with other programming languages (for example, users are able to write C or C++ code to manipulate R objects directly). These features convince us to choose R as an open-source “MATLAB” alternative.

In addition to translating the original MATLAB code, another effort to improve the usability of the algorithms is to add a neat and easy use graphic user interface (GUI) and integrate all algorithms into it. Some generally used programming languages that are effective in designing and implementing GUI include Visual Basic, Visual C++, Java, etc. Among these languages Java is not only an open-sourced language, but also among the most popular programming languages up to now (according to TIOBE Programming Community Index, Java enjoys No. 1 popularity in 2011, and No.2 popularity in 2012 when C is No.1 [4]). Also Java is designed to be platform independence, which makes the program written in Java being easily transferred to other platforms (while R can also run on multiple commonly used platforms). All of these make Java an ideal language for us to adopt.

So our goal here is to re-implement the CAM algorithms in R code, and integrate them with a Java based GUI. Some important issues should be considered before the actual implementation:

1) How to translate MATLAB code to R code in an efficient way, and the execution efficiency and qualities of the resulting R code should also be considered. That is, we should consider and make more use of the unique and applicable features of R, rather than the command to simple command replacement during the translation.

2) How to connect R coded algorithms and Java GUI, this could be the main issue of the whole project, and it
will affect the overall performance and stability of the final program.

Our final product of the software is called “CAM-Java”, since Java module of the software takes in charge of the overall execution procedure. It successfully combined R and Java modules. The whole software runs smoothly and stably. In this thesis we will discuss several technical details and show how we achieve these goals.
1.3 Organization of the Thesis

We will discuss several implementation details and some important issues that we met during the work, and how we solve them in this thesis. We will mainly focus on the design strategies and the methodologies of solving particular problems rather than the actual codes themselves, as choosing a right strategy or methodology beforehand can be more important than the actual programming. Good design strategy also improves the efficiency of the software implementation, and debugging. If readers want to know more coding details, they may refer to the source code that we will also provide the information on how to locate particular code sections in a particular source file in this thesis.

Since our software contains two main parts: R module and Java module, we divide our discussion into three chapters. Chapter 2 will discuss the implementation of major algorithms by R. We first present the main structure of the whole R algorithms suite and describe the usage of every R function and script. We then discuss some important issues for writing R program with multiple R functions and scripts, and most importantly, what should be considered during the translation from MATLAB code to R code. Finally, we compare the results generated by both MATLAB and R version algorithms on the same simulated and real input data. Comparative studies show that the results obtained by the two versions are quite similar. The slight difference comes from the different precisions of floating numbers used in MATLAB and R.

In Chapter 3, we introduce the implementation of graphic user interface by Java Swing API. Again, we first present the whole structure of Java module, and the Model-Viewer-Controller (MVC) design pattern used in the module will be mainly discussed. We then discuss all Java classes which are packaged into two packages: “guiView” (contains implementations of Viewer and Controller parts), and “guiModel” (contains implementation of Model part as well as functionalities dealing with the interaction with R). Finally, we give some practical examples to show how the final CAM-Java software is used under different conditions.

Chapter 4 discusses the most important issue in this project: the interactive integration between R and Java modules. To assure a successfully interaction between R and Java, we adopt an open-source project called “RCaller”, with additional R scripts and Java classes being written in both modules. We first discuss these works done in R module, mainly focus on the design strategies used there. Then we present how the communications are achieved with R in Java environment. We divide the discussion of the overall communication task into four sections, with each section describing the communication between R and Java in a specific aspect.

Finally, in Chapter 5, further discussions about the integration of R and Java codes will be presented. We find
that the joint use of R and Java is an effective way to build software with both user-friendly GUI and analytically comprehensive core algorithms. This combination has some unique advantages, which make it even more suitable in the field of scientific and engineering software development. Future improvements on the current software have also been presented. In general, we may study deeper techniques of R and Java integration and make the R environment more integrated into Java GUI. For the specific program CAM-Java, two extended functionality and algorithm improvements are introduced.
Chapter 2 Implementation of Major Algorithms by R

2.1 Structure of the Algorithms

The R module in the CAM software is a suite of algorithms consisting of three major algorithms, namely, CAM–CM, CAM–nICA, and CAM–nWCA. Reflected in its name, CAM algorithm forms the base for all these algorithms. Specifically, though different algorithms are employed to estimate model parameters, they are all based on the data extracted by the CAM algorithm.

To implement the aforementioned algorithms in R language, we first divide them into several main functions, and we use R function object to represent each of them. We then divide the main functions further into some helper functions which make the main functions simpler and cleaner. Each of these helper functions performs an independent task, and can be called by the main functions. Finally, we develop R scripts (an aggregation of R commands) to perform each of the major algorithms, by calling one or two main functions as well as helper functions.

In the final product of the CAM software, we provide a set of R functions and R scripts in the folder “r_func”, representing the whole R module of the CAM software. Specifically, there are two subfolders within “r_func”, namely, “functions” and “functions2” containing various helper functions. A brief summary of these functions is given below.

The “functions” folder contains helper functions that are mainly used in CAM and CM algorithm:

- “affinity.R” performs Affinity Propagation Clustering algorithm.
- “SL_EM.R” performs Expectation-Maximization (EM) algorithm.
- “measure_conv.R” performs minMargin convex optimization algorithm.
- “PCA.R” performs principal component analysis for dimension reduction.
- “multinorm.R” evaluates a multidimensional Gaussian value at a specified point with given mean vector and covariance matrix.
- “ve_cov_Jain.R” deals with the singularity problem and the “realmin” problem, being used in “multinorm.R”.
- “nnls.R” implements the Lawson and Hanson method for solving the least squares problems with non-negativity constraints.
- “nnls_wrapper.R” is the wrapper function of the “nnls.R”.
The “functions2” folder contains additional helper functions that are mainly used in CAM–nICA algorithm. Once again, each helper function performs one independent task.

On top of these helper functions, there are four main functions:

- “CAM.R” performs CAM algorithm.
- “CM.R” performs CM algorithm.
- “CAM-ICA2dim.R” performs CAM-nICA algorithm with 2-dimensional data.
- “CAM-ICA3dim.R” performs CAM-nICA algorithm with more than 2-dimensional data.

We provide three R scripts, for the user, to run each of the three major algorithm, namely, “Java-runCAM-CM.R”, “Java-runCAM-ICA.R”, and “Java-runCAM-nWCA.R”.

The structure of the algorithms we discuss here is carefully designed to provide flexibility to the potential users of CAM software. For example, with these functions each performing a specific analytic task, the users can not only run the major algorithms, but also perform specific task(s). Furthermore, all the implemented R scripts are readily called by Java (as reflected explicitly by their names all started with “Java-“), and more details can be found in section 4.1. It shall be noted that all three major algorithms in R can be run without involving Java, where the R scripts “demo_for_simulation.R”, “demo_for_DCEMRI.R”, and “demo_for_gene_expression.R” perform these functions on different types of data in R environment alone.
2.2 Implementation Details

This section describes technical details about the implementation of the R module. We provide representative examples to illustrate the ways for dealing with various problems when building a relative small scale R program (about 5000 lines of R commands), as well as the translating from MATLAB code to R code.

1) Connecting different R functions and R scripts
As aforementioned, we use the implementation strategy of dividing big algorithms into several smaller independent tasks. Here the important issue is how to efficiently connect different parts of the software. The way we chose is to use R scripts at the higher level, and use R function object as a “wrapper” for the smaller independent tasks at the lower level. The R function object has the following format:

```
functionName <- function(parameter1, parameter2, ...) {
    body
    ...
}
```

When one R function (which has been defined) is being used in another file in the body of another R function or in one R script, the R command “source” is used as follows:

```
source("filepath/functionName.R")
```

This is an efficient way for the operation, since we can easily call different pre-defined R functions in a specific R function or R script. Also, we can pass the parameters when calling the R functions, making the R functions more adaptive to different situations.

2) Automatic R library installation
One unique feature of R language is that it uses numerous libraries which performs different tasks. Since R is an open-source language, people can easily download libraries written by others and make good use of them. CAM software development adopts this tradition, that is, we actively use several R libraries that are not contained in the base libraries (the libraries that will be installed together with R environment). Thus, the use of CAM software would require the installation of these “additional” R libraries, as the prerequisite. To ensure an convenient use of the CAM software by the end users, we provide an automatic R library installation capability in the R module. Specifically, before running the actual algorithms or functions, the R module will first detect whether a particularly required R library is installed. If not, the R module will automatically
download and install the library without needing any user’s intervention. To achieve this capability, the following R script section is needed when any functions outside of the base libraries are first time used.

```r
# load the library "MASS" for the function "ginv"

packageExist <- require("MASS")  # detect whether library “MASS” exists
if (!packageExist) {
  install.packages("MASS")        # if it doesn’t exist, install it
}
```

3) Translation from MATLAB code to R code

In this project, some of the analytic functions have been implemented previously by the MATLAB code, which in turn needs to be converted into R code with the same (or better) functionality. The basic and foremost criterion here is the translation accuracy. Each of the functionalities is verified when the translation is done. The details about functional verification can be found in section 2.3. Here we discuss the specific considerations we have used in the translation. We found that these considerations have made the translated R code much simpler and cleaner, as well as much more “R styled”. This effort also improves the readability and efficiency of the R code.

Although MATLAB language and R language share many common principles and features, for example, almost every commonly used MATLAB command has an R alternative; there are some differences that deserve special care when translating the codes between them. David Hiebeler’s manual “MATLAB / R Reference” [5] contains many good examples of command-to-command translation. Nevertheless, R language has its unique features, and the use of these features can efficiently translate several lines of MATLAB code into only one single line of R code, or transform one complex R command into another simpler R command.

For example, one of the main differences between MATLAB and R is that the data type in MATLAB is based on Matrix while in R it is based on Vector, although R can also handle the data type “matrix”. Consider the following MATLAB command:

```
X=X_mask./repmat(sum(X_mask,1),[M,1]);
```

where “X_mask” and “X” are all M*N size matrix. A straightforward way to implement the same function in R code would be:

```
X <- X_mask / matrix(rep(colSums(X_mask), M), M, N, byrow=TRUE)
```
However, in R, there is no need to follow the rule that “two matrices can only be divided if they have the same size”, as R consider every data as a vector (even its type is “matrix”). Specifically, when performing arithmetic operations with two vectors of different lengths, “Shorter vectors in the expression are recycled as often as need be (perhaps fractionally) until they match the length of the longest vector” [6]. Therefore, we can utilize this feature to translate the above MATLAB command into a much simpler R code as following:

\[ X \leftarrow t(t(X_{\text{mask}})/\text{colSums}(X_{\text{mask}})) \]

Other major unique features in R include:

1. R has an extremely useful data type “list”, which can hold data with different type together in one data set.
2. R always uses the result of the proceeding command in the R function as the function’s return value, so often there is no need to insert a specific “return” command in R functions.
3. In R functions, there is a unique concept of “lexical scope”. The use of this feature can create a function with “mutable state”, that is, a function more adaptive in different situation.

The availability and adoption of these unique features provide us with the opportunity to write more “R styled” code, rather than a “rigid” translation of MATLAB code. One can refer to the R online manual for more details [6].

4) Drawing more formatted figures

Figure is a good form to show the results. In this project, we need to draw some figures which are much formatted in order to display the results more clearly and precisely. In R, there are several ways to draw figures. The easiest one is to use “High-level plotting commands” that can automatically choose proper format to display the data. However, if we need to control the format at more detailed level, we need to use “Low-level plotting commands”. Generally, we need to write additional codes to draw a figure as compared to using high-level plotting commands, but we can control almost every part of a figure, such as lines, points, axes, titles, legends, and so on. The R scripts “Java-plot-eTCs.R” and “Java-plot-eTCs-withCF.R” show some examples of how to use this style to draw a figure.

5) Saving results on hard disk

R has some useful functions to save data in a file with various formats. In this project, we use R (rather than Java) to save data in CSV (comma-separated values) formatted files. Also we name our files with the current time when they are created, so the results of different rounds of calculation will have different file name. The specific R script to perform these tasks is as follows:
# Save the results (Aest, Sest) as CSV formatted files in the directory "results".
# The files will be named by the current time.

saveData <- function(Aest, Sest) {
  now <- format(Sys.time(), "%Y-%m-%d-%H-%M")
  AestName <- paste("../results/", now, "-Aest.csv", sep="")
  SestName <- paste("../results/", now, "-Sest.csv", sep="")
  write.csv(Aest, file=AestName, row.names=FALSE)
  write.csv(Sest, file=SestName, row.names=FALSE)
}

2.3 Cases Studies and Algorithm Verifications

In this project, the R code of CAM and CM algorithms are largely translated from the original MATLAB codes [1-2]. Logically, we test the performance of the R implementation of CAM and CM algorithms using the same datasets that were used by the MATLAB based experiments, and compare the consistency of the results obtained by R and MATLAB codes where the outcomes of MATLAB code serves as the reference.

We first test the R code of CAM–CM algorithm on the same simulation datasets used by the original MATLAB software [1-2]. These datasets include twelve simulated datasets with four different parameter settings (called “scenarios”), and with three different SNR (signal to noise ratio) levels. The datasets are stored in the folder “data/data_simulation”. Tables 2.1-2.4 summarize the results of CAM–CM algorithm on the simulation data (in terms of four estimated kinetic parameters) obtained by using the R code and MATLAB code of algorithms, respectively.

Table 2.1. Parameter estimated obtained by the R code and MATLAN code of the CAM-CM algorithm, tested on the simulation dataset 1.

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>SNR = 10dB</th>
<th>SNR = 15dB</th>
<th>SNR = 20dB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K^i_f$ (/min)</td>
<td>0.030</td>
<td>0.032</td>
<td>0.029</td>
</tr>
<tr>
<td>$K^o_f$ (/min)</td>
<td>0.484</td>
<td>0.495</td>
<td>0.465</td>
</tr>
<tr>
<td>$K^i_s$ (/min)</td>
<td>0.030</td>
<td>0.032</td>
<td>0.028</td>
</tr>
<tr>
<td>$K^o_s$ (/min)</td>
<td>0.096</td>
<td>0.099</td>
<td>0.145</td>
</tr>
</tbody>
</table>
Table 2.2. Parameter estimated obtained by the R code and MATLAB code of the CAM-CM algorithm, tested on the simulation dataset 2.

<table>
<thead>
<tr>
<th>Scenario 2</th>
<th>SNR = 10dB</th>
<th>SNR = 15dB</th>
<th>SNR = 20dB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K^I_f$ (/ min)</td>
<td>0.063</td>
<td>0.063</td>
<td>0.065</td>
</tr>
<tr>
<td>$K^O_f$ (/ min)</td>
<td>1.171</td>
<td>1.171</td>
<td>1.227</td>
</tr>
<tr>
<td>$K^I_s$ (/ min)</td>
<td>0.038</td>
<td>0.038</td>
<td>0.039</td>
</tr>
<tr>
<td>$K^O_s$ (/ min)</td>
<td>0.090</td>
<td>0.090</td>
<td>0.102</td>
</tr>
</tbody>
</table>

Table 2.3. Parameter estimated obtained by the R code and MATLAB code of the CAM-CM algorithm, tested on the simulation dataset 3.

<table>
<thead>
<tr>
<th>Scenario 3</th>
<th>SNR = 10dB</th>
<th>SNR = 15dB</th>
<th>SNR = 20dB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K^I_f$ (/ min)</td>
<td>0.078</td>
<td>0.078</td>
<td>0.069</td>
</tr>
<tr>
<td>$K^O_f$ (/ min)</td>
<td>1.298</td>
<td>1.298</td>
<td>1.329</td>
</tr>
<tr>
<td>$K^I_s$ (/ min)</td>
<td>0.092</td>
<td>0.092</td>
<td>0.082</td>
</tr>
<tr>
<td>$K^O_s$ (/ min)</td>
<td>0.526</td>
<td>0.526</td>
<td>0.548</td>
</tr>
</tbody>
</table>

Table 2.4. Parameter estimated obtained by the R code and MATLAB code of the CAM-CM algorithm, tested on the simulation dataset 4.

<table>
<thead>
<tr>
<th>Scenario 4</th>
<th>SNR = 10dB</th>
<th>SNR = 15dB</th>
<th>SNR = 20dB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K^I_f$ (/ min)</td>
<td>0.120</td>
<td>0.132</td>
<td>0.113</td>
</tr>
<tr>
<td>$K^O_f$ (/ min)</td>
<td>1.661</td>
<td>1.668</td>
<td>1.635</td>
</tr>
<tr>
<td>$K^I_s$ (/ min)</td>
<td>0.074</td>
<td>0.082</td>
<td>0.070</td>
</tr>
<tr>
<td>$K^O_s$ (/ min)</td>
<td>0.647</td>
<td>0.649</td>
<td>0.639</td>
</tr>
</tbody>
</table>
Based on the experiment results, it can be seen that the R code of the CAM-CM algorithm, translated from the MATLAB code, can accurately estimate the parameters values of the pharmacokinetics, as compared to the ground truth, and the parameter estimates are also very close to what obtained by the original MATLAB code.

It should be noted that, although the R and MATLAB codes of the CAM–CM algorithm follow exactly the same principles, the experimental results tested on the same dataset show some minor differences, independent of the expected random effects. There are some potential causes for such outcome differences. We have found that the R and MATLAB codes actually use slightly different precision of data type. Specifically, in the majority of MATLAB code, “double” precision is used when representing floating numbers, while in a small portion of the code (for example, affinity propagation clustering algorithm), it changes to “single” precision. In contrast, in the R code uses “double” precision exclusively. As the result, in the R code of the CAM–CM algorithm, the rounding error may be smaller at the cost of higher computational time (for example, R code needs 2-3 minute to complete the same task as compared to 1 minute used by MATLAB code). Nevertheless, the differences are considered insignificant.

We then apply the R code of the CAM–CM algorithm to real DCE-MRI datasets used in [1-2]. Specifically, we use a DCE-MRI dataset of an advanced breast cancer case [7-8] as the input data. Figure 2.1 show the results (normalized tracer concentration) in both version, we can see the shapes of both flow 1 and 2 are similar in different versions. Actually these two lines represent two compartments with different kinetic patterns, and both versions of CAM-CM algorithm successfully reveal these two compartments with similar kinetic patterns that are reasonable in the real world (refer to original MATLAB software [1-2] for more descriptions).
Figure 2.1 Normalized tracer concentration results by applying CAM-CM algorithm to a typical DCE-MRI dataset, algorithm is implemented with both R and MATLAB language.

(a) Results by using R version algorithm.
(b) Results by using MATLAB version algorithm.
3.1 Structure and Design of GUI Software

In this project, to provide the end users with graphic user interface for convenient use of the CAM algorithms, we develop Java script to build the GUI of the software, that is, the Java module. In addition to the interactive GUI for the users to run the software, the Java module also contains the required functions such as importing data, passing user selected parameters to the specified algorithm, calling R scripts, reading and displaying the earlier stored results produced by the R module, and so on. In order to integrate all these functions seamlessly and make the software implementation more efficient and in a good order, in this project, we adopt the MVC (model-view-controller) design strategy.

The MVC design strategy is an efficient and clear method for implementing GUI based application. The basic concept of this design strategy is to separate the task of displaying components (view) from the task of interactions with the user. In addition, the interactions may be designed and implemented independent of the underlying data (model) and how the user’s actions affecting the data (the task of controller), as well as how the changes of data affecting the current state of displaying components (the task of controller again). Practically, the MVC design strategy requires the programmer to divide the whole implementation task into three parts, so that the programmer can design and implement each part separately. Such strategy will improve the efficiency of the implementation process, as well as the readability of the whole software.

Following closely the MVC design strategy, the final product of the Java module includes two main packages. The “guiView” package includes the classes that handle the display of the main frame, menus, and dialogs; as well as the classes that handle the interactions between user’s specific commands or feedback for certain reactions (e.g., display changes or expansions); achieved via the “view” and “controller” components. The “guiModel” package includes the classes that represent the results and handle the interaction with the R module, achieved by the “Model” component. Based on such class division, we can make good use of code reuse and locate the original code for a specific task more easily.
3.2 Packages Description

In this section we will discuss the technical details on each of the classes in the two main packages, and their functions.

3.2.1 GUI displaying and Event Handling Package

First we discuss the classes in the “guiView” package. As aforementioned, this package contains “view” and “controller” components, mainly handling the tasks that display various GUI components, as well as handling user’s commands.

The whole GUI is implemented by using Java Swing API. Java Swing provides various types of GUI components and methods to control these GUI components. With the help of Java Swing, we can choose proper GUI components, add them to the main frame at particular locations, set their parameters, and select the overall “look and feel” of the whole interface. The following classes are all evolved in the task of GUI display.

1) “MainFrame” constructs the main frame for the display, and all the components within the frame. It controls the relative positions of the every component within the frame by using grid bag layout method. With this specific design, the relative positions of each component remain unchanged when the size of the main display frame changes. The MainFrame class also includes the principal method of the whole program. That is, every time when the user starts to run the CAM-Java software, this class will be called at first to display the initial main frame. The following figure shows the main frame produced by MainFrame class with default look and feel.
2) “GBC” contains helper functions for using grid bag layout in the main frame. These functions will make the use of grid bag layout much easier.

3) “MenuBarCreator” class includes only one static method, which is used for generating menu bar within the main frame. In the final product of CAM software, we have two menus named “Option” and “Help”. In “Option” menu, two sub menus are provided, one is used for modifying the file path of “Rscript.exe” (this function will be further discussed in section 4.2), and another one is used for changing the overall look and feel (or “skin”) of the whole GUI. In “Help” menu, only one sub menu “About…” is provided, which will display the “About” dialog when the user click on it.

4) “DialogCreator”, similar to “MenuBarCreator” class, contains only static methods. Three different model dialogs can be created. The first one is used when the user changing the file path of “Rscript.exe”, and has an input JTextArea component allowing the user to input new file path. After the user finishes the new file path input, it will first be verified – checking whether “Rscript.exe” exists according to the path, and it will only be saved if the new file path passes the verification. Otherwise, the second dialog with some error
messages will be created and displayed. The last dialog, as we mentioned in 3), is the “About” dialog, whose function is to provide some information to the user and request user to click “OK” button.

5) “ImagePreviewer” class creates an accessory of the file chooser, which can be used to preview the images. This component is only used when the user choose a data file of image type, with which the user can preview the images when needed.

6) “MyRPlotViewer” class creates a new frame used for displaying result figures.

Another class, named “ActionListenerCreator” in the package “GUI”, serves as the “controller” part of the Java module. Java Swing API adopts action listeners to handle various user input event (for example, clicking the button, changing the content in the text area, etc.). Specifically, for each possible event that a specific GUI component can generate, we can “register” a specific “listener” to the component and associate it with that event. So each time the event occurs, this listener will be referred and a proper method in the listener will be run. In the class “ActionListenerCreator”, it can generate three different action listeners used to handle the click events of three buttons “…” “Load”, and “Run” on the main frame. The actionPerformed (event e) method in each of these listeners takes charge in the actual reaction behavior of the whole program when a specific event occurs.

### 3.2.2 Data Modeling Package

In this section we focus our discussions on the classes in the “guiModel” package. Generally speaking, these classes are mainly used to model the data used in the analytic calculation. That is, these classes aim to construct specific abstract data type to represent data and assure the simple and convenient usage of the data. In addition, the classes in the “guiModel” package handle the interaction between Java GUI module and the R module.

1) “Results” is an abstract data type that represents the outcomes of analytic calculation. Specifically, this data type is a collection of the three different results: “aEst” represents the contrast/tracer concentration; “sEst” represents the spatial distribution maps; and “cmResults” represents the estimated kinetic parameter estimates produced by the CM algorithm (it is only applicable when using CAM–CM algorithm).

2) “ResultTableModel” class is used to describe the basic behaviors of the abstract model, where the stored data will be shown in tables. In the main frame, there are two basic tables for displaying the analytic calculation results, one for “aEst” and one for “cmResults”.
3) “MyRCaller” class handles the interaction with R module. More details about this class will be discussed in section 4.2.
3.3 Illustration of the Software Usage

As the final product of CAM software, all aforementioned Java classes are packaged together into one Jar file, implemented by the following Java command:

```
jar cvfm CAM-Java.jar manifest.txt guiModel guiView src HowTo.txt
```

The generated jar file “CAM-Java.jar” includes all compiled class files and source files, as well as a manifest file indicating essential options when running this jar file (for example, location of the main method, the class path). Practically, the R module and the Java module can be combined, that is, put the R functions, scripts and jar files all together in one folder. With carefully specified file path, the two modules can seamlessly work together, constituting the complete software.

Once the user has successfully installed Java SE (JRE or JDK, not lower than 6 u26) and R (not lower than 2.14.1), s/he should be able to run the whole software by double clicking the “CAM-Java.jar” file, or if s/he prefers to use command-line based shell, s/he can first go to the folder containing the whole software, and type the Java command as shown below. It is noted that these two methods have the same effect when starting the program.

```
java -jar CAM-Java.jar
```

When the user runs the software for the first time, a dialog will be shown (figure 3.2) allowing the user to enter the file path of the binary executable file “Rscript.exe”. This is an important tool provided by R to run R scripts, and one can easily find it in the installation folder of R.
After successfully entering the correct file path, the main frame of the software will appear. Generally speaking, the usage of the major algorithms is the same as we did in section 2.3. But now with a Java GUI, we do not need to write any extra R script to run the algorithm, in contrast, we can perform the analytic tasks by simply clicking several buttons.

For example, we can perform the same task as in section 2.3, to apply CAM–CM algorithm to a real DCE-MRI dataset. Here is the procedure:

- Select “Load Data file”
- Click “…” button
- In the file selection dialog, first select “R / Matlab Data (*.rda, *.mat)” in “Files of Type:”, then select one dataset “data / data_DCE_MRI / typical_case.rda”
- Click “Open”
- Click “Load”
- Select “CAM-CM”
- Set “number of organs” to 3, time interval to 0.5 min
- Check the box “Use multivariate clustering to denoise”, “Do visualization of the convexity”, and “Show concentration” results in a figure
- Click “Run”

Then after about 2 to 3 minutes, the results (“aEst” and “cmResults”) will be shown in the table areas (Figure 3.3 (a)), together with two figures (Figure 3.3 (b) and (c)) displayed in separated windows.
Figure 3.3 Results of applying CAM-CM algorithm to data of a typical DCE-MRI case

(a) Tables displaying contrast/tracer concentration results and CM estimated kinetic parameter.

(b) Figure showing convexity visualization

(c) Figure showing tracer concentration results
Another example is running the CAM–nICA algorithm on the real gene expression dataset. Similar to the procedure we detailed above, we first select the data file “data / data_gene_expression / mix_gene_expression_2dim.txt”, then select “CAM-nICA” algorithm, set the number of phenotypes to 2. After about half minute, we can get “aEst” result as well as a scatter plot of the final demixed signals (figure 3.4).

From the above examples, we can see that the “CAM-Java” software can be used in different situations, deal with different types of data, and has several parameters which can be adjusted when running the software. Moreover, no matter in what situation, the R module and Java module work together seamlessly, and the interaction between them is steady and smooth. In the next Chapter, we will focus on the issue of making R module and Java module work together, and some technical details of the interaction.
Chapter 4 Interactive Integration of R and Java Modules

One of the important yet challenging task in developing CAM software is ensure that the R and Java modules work together to implement the user’s requested analytic tasks and display the results. In our software design, we choose the Java module as the main “driver” of the CAM software, where the users do not need to open the local R environment in order to run the R module of the software. The development effort at hand is now on the method of calling the R module from/by the Java module, that is, establishing the connection or interaction between the R module and the Java module.

Here we adopt a well developed open-sourced project called “RCaller” [9], which provides a convenient and relative fast way to allow Java program to be able to associate with R program. In this project, we use RCaller 2.1.0 under GNU Lesser GPL license. RCaller 2.1.0 contains two parts, namely, RCaller-2.1.0-SNAPSHOT.jar and Runiversal” library. The jar file belongs to the Java module, and contains the classes and methods for implementing the specific R caller. The jar file needs to be added to the Java module classpath. The “Runiversal” library belongs to the R module, and contains functions for converting R list objects to Java or XML. This library should be installed in the R module before its use. In addition to using RCaller, additional R scripts and Java classes are also developed within both R and Java modules, assuring the whole CAM software runs smoothly and conveniently by the users. Section 4.1 and 4.2 present more technical details about these functionalities that are accomplished in both R and Java modules, respectively.

4.1 Works in R Module

In order to make the R calling process in Java easier, the R scripts in R module are all designed and arranged to be “Java friendly”. This means that when calling the R scripts in R module from the Java module, the only thing that is needed to be done by the Java module is to pass some algorithm parameters to R environment. After that, the R scripts will take care of all the subsequent calculation procedures. The concept behind this strategy is “letting one R script do as many things as it can”. Actually, the final CAM software contains only three main R scripts, namely, Java-runCAM-* R. Each of these three R scripts deals with one independent major algorithm and it is the only script being called by Java for one calculation, where three helper R scripts, namely, Java-plot-eTC.R, Java-plot-eTC-withCF.R, and Java-saveData.R, are activated to perform the tasks of drawing figures and save results on hard disk. It is noted that the users also have the option to call “Java-plot-eTC-withCF.R” to perform the extra curve fitting task based on the result figures directly in R environment.
Taking the “Java-runCAM-CM.R” as an example. In this script, the required functions are first loaded into R environment, then the input data are processed by the CAM algorithm. When results are produced (e.g., contrast/tracer concentration and spatial distribution maps), the concentration result is processed in order to draw a figure. After some preprocess procedures, the CM algorithm is then performed based on both the original input data and the concentration result. Finally when all analytic calculations are accomplished, the results are automatically saved on the hard disk. From these descriptions, we can see that the Java module does not need to know how the CAM-CM algorithm is actually performed, instead, the final CAM-Java software can only run the major algorithms as a whole and the user cannot interrupt or stop at a specific point during the process. Since the major algorithms is of focused interest in this software, this design strategy has significantly simplified the implementation of Java module and user GUI. For example, running the CAM-CM algorithm in Java module can be as simple as this:

```java
... 
code.R_source("Java-runCAM-CM.R");  // source the script
... 
caller.runAndReturnResult("final.result"); // run R scripts
... 
```

The CAM software mainly follows the “main strategy” of allowing the Java module to call one R script that performs all the works. While for importing data with different types, we use other strategies when applicable. For example, one cannot see any R script that performs data importing work in the R module, as all R scripts are integrated with Java code in the Java module. The reason we do not use the aforementioned main strategy is that we need to call different importing functions for different types of data, and the differentiation of the data file types can be easily done in Java rather than in R, to the best of our knowledge. Therefore, we integrate the importing codes (written in R) within the file processing codes (written in Java). That is, for each type of data, we pass a small section of R commands to R environment from Java (the way used to pass R commands from Java will be discussed in the next section). For example, the Java code of importing data stored in MATLAB formatted file “*.mat” is given below:

```java
// read Matlab data (*.mat), the R package "R.matlab" is needed.
else if (ext.equals("mat")) {
    code.addRCode("packageExist <- require("R.matlab")");
    code.addRCode("if(!packageExist){");
    code.addRCode("install.packages("R.matlab")");
    code.addRCode(""");
    command = String.format("list <- readMat("%s", file.getPath())");
```
\texttt{command = command.replaceAll("\\\", "/");
code.addRCode(command);
code.addRCode("X\_mask < - list\$X\_mask");
}

Noted that some R commands are integrated into the above Java code section. One may find that these Java codes are a little bit more complicate than those using the first design strategy. The current version of CAM-Java software can handle four types of different data files: MATLAB formatted data file (*.mat), R generated data file (*.rda), plain text file (*.txt), and comma-separated values file (*.csv). In the sample datasets provided by the final product of CAM software, we have all these four types of data file, and users can try them out.
4.2 Works in Java Module

Since we choose the Java module as the main “driver” of the CAM-Java software, more works need to be done in Java module for handling the interactions between the two modules. Furthermore, some additional and special features are provided by the Java GUI to assure the convenient use of the software by the end users. Below we will discuss the five major features we have developed within the Java module. Each of these features presents a good example of showing how to interact with R in a specific field, allowing the Java GUI integrated seamlessly into R environment.

4.2.1 Importing Data and Parameters to R Environment

There are two main classes we used from the project “RCaller”. The class “rcaller.RCode” uses a String buffer to store R commands the user may want to execute later, and one can add different types of R commands into it, delete any type of R commands, or modify R commands before actually executing them. The class “rcaller.RCaller” is able to establish a connection between R and Java, pass those R commands stored in an object of class RCode to R environment, running R code and save the results with Java readable format (done by “Runiversal” library in R, the format is mainly XML), then load back these results.

With the help of these two main classes and other auxiliary classes managing drawing figures, we build the specifically designed Java class “guiModel.MyRCaller.java”. It contains three major methods for running three major algorithms respectively. In each of these major methods, it takes care of the entire process, which can be divided as follows:

1) Initializing the RCaller object;
2) Passing parameters and input data;
3) Running specific R scripts and reading results back.

This strategy is similar to the one we have discussed in section 4.1: the caller calls the callee to perform the whole workflow without the need to know its details.

As described above, the first step of each major method is to initialize the RCaller object. This is required by “RCaller” project for making some initial settings before establishing the connection to R. In the class “guiModel.MyRCaller.java”, the “initialize” method takes in charge of this, whose code is as follows:
/**
 * Initialize the RCaller object.
 * This will start a new thread for running R codes.
 */

public void initialize()
{

caller = new RCaller(); // establish a new instance of class RCaller
caller.setRscriptExecutable(rScriptPath); // set the file path of “Rscript.exe”
caller.setGraphicsTheme(new DefaultTheme()); // set default theme of figures drawn by R
caller.redirectROutputToConsole(); // make R display all its messages to standard output

code = new RCode(); // establish a new instance of class RCode
code.clear();
}

Then the second step is to pass parameters and input data. When dealing with a parameter set by the user, what we actually passing to R is an R command that creates a specific variable (always with the type of double array) in R environment with the same value as that parameter. When dealing with input data (often in larger scale), as we discussed before, we pass proper R commands to let R environment import data itself, so actually Java module does not know the content of the input data.

4.2.2 Running R Scripts and Read Results from R Environment

After importing data, the next step is to run specific R scripts. We can simply pass another R command to let R environment run them. There are two details need to be mentioned. The first one is that, though R usually uses the command “source(filepath/script_name)” to load other helper functions or R scripts, but in RCaller project, a specified method “R_source ()” is recommended for the same operation, rather than using general method to pass an R command. The second one is that in RCaller, R commands being passed to R environment will not be executed until the user calls for particular methods in RCaller, like:

caller.runOnly();

However, if one also wants to read back some results, one must call “runAndReturnResult” method rather than “runOnly” method, like in our program:
caller.runAndReturnResult("final.result");

where we do read back some results from R to Java module. Although, as we described before, the R scripts being called will save all the results on the hard disk automatically and allow users to access them at a later time, we also needs part of the results (contrast/tracer concentration and CM estimated kinetic parameters) to be displayed on GUI (within table components) in real time, so that users can check them immediately right after the calculation is done. The way to accomplish this task is that “Runiversal” library in R will transform the R list type data (so we should change our results to this type) to XML formatted data, which can then be loaded by proper Java API. In our program, we store the results in a “Results” class instance by using the method “guiModel.MyRCaller.readResults”, like follows:

```java
/**
 * Read results from R.
 * @param type the type of the algorithm, 0 for CAM-CM, 1 for others.
 * @return the Results object.
 */
private Results readResults(int type)
{
    double[] aEst = caller.getParser().getAsDoubleArray("Aest");
    if (type == 0) {
        double[] cmResults =
            caller.getParser().getAsDoubleArray("cmResults");
        return new Results(aEst, cmResults);
    }
    else
        return new Results(aEst);
}
```

4.2.3 Display Figures Drawn by R

The way we accomplish this task is that we first let R save all figures drawn during the calculation as temporary image files on the hard disk, then load them back to Java environment (by using “RCode.startPlot” method, this method will return a File object that associated with the image file), and show each figures in a separated frame.
The Java class that creates a new frame to display the figure is “guiView.MyRPlotViewer”, which inherits from the class JFrame. Compared with the default plot viewer provided by RCaller project, the viewer used in our program has the following improvements:

1) Change the closing behavior of the frame. When the user closes one plot viewer frame, other frames created by this program (for example, the main frame) will remain open, rather than being closed together in the default viewer.

2) Change the image updating behavior. With the default viewer, when the user moves the viewer frame, or when other windows happen to cover the viewer temporarily, the image on the viewer may disappear unexpectedly. The reason is that, in default viewer, the image is painted on the frame directly, and image updating (re-paint onto the screen when the position of the image changing, or recovering from the overlaying of other images) interval is not sooner enough. So in our viewer, we add a new JComponent object on the frame, and paint the image on the object, by utilizing the inherent JComponent updating strategy in Java Swing API, so that the image can be updated much sooner.

4.2.4 Showing R Generated Information during Calculation

In the first version of the CAM-Java, when user presses the “Run” button, the GUI starting to let R run the algorithms and waiting for the results. This usually takes 2 to 5 minutes. During this time, the user cannot do anything and see any response messages (for example, to see the interim calculation progress). In the later version, we add a new feature to improve the user experience of the software.

The way we accomplished this new feature is that we first set the R environment to let it display every messages (no matter the results or warning messages) via the standard output (with the help of RCaller, one can find that command in the “initialize” method discussed in section 4.2.1). Then we build our own Java console by adding a JTextArea object on the main frame of GUI, and re-direct the system output stream (standard out) and system error stream (standard error) to that object, so that the user can see every message from the standard output in the GUI (Figure 4.1).
When we finish both of these steps, with the connection of standard output, we can treat our Java console as a simple R console (though one cannot input R commands into it), and see the response messages generated by R. However, these messages cannot be practically displayed on the GUI until R finishes all the tasks, and the whole GUI cannot respond any user’s input event during the calculation. That is because all system resources (specifically, CPU, memory, and so on) are blocked when R is running the algorithms. To overcome these disadvantages, we use multi-threads features in a Java Swing program.

It is well known that Java Swing API is actually not thread safe. While the designers of Java AWT (the foundation of Java Swing) introduces a thread called “Event Dispatch Thread (EDT)”, which controls the dispatching of tasks. Thus, it is recommended that one should follow the “single-thread rule” when using Java Swing, meaning put all tasks that may update any of the JComponent object on GUI into the EDT. For example, the main method in the class “guiView.MainFrame” creates the main frame when the software started, where we should put it into EDT, as follows:

```java
// Launch the main frame.
...
EventQueue.invokeLater(new Runnable() {
    public void run() {
        ...
        // task code
    }
});
...
```

Since displaying R messages on the GUI in real time needs the content update of a JTextArea object, one should also put it into EDT. When Java module calling R to run algorithms, the EDT thread is starving and cannot dispatch any task to be executed. That is why users can only wait until the calculation is completely done. To solve this problem, we can create a new thread and assign the task of calling R and running R scripts to this thread, rather than using the main thread. This design will give EDT the chance to perform its new tasks...
when another thread is performing the calculation. This design can also solve the problem that the software cannot respond any user’s input event during the calculating, because now this interaction task will be handled by the main thread. The following code shows how we start a new thread in our program:

```java
// create a new thread for the rest of tasks.
Thread t = new Thread(new Runnable() {
    public void run() {
        // run and get results.
        ...
        // return if an exception occurs.
        ...
        // put the rest of tasks to the event dispatch thread.
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                // display the results.
                ...
                System.out.println("Please load another data ...");
            }
        });
    }
    });

// start the thread.
t.start();
```

that can solve the aforementioned problems almost perfectly. It should be noted that when R finishes the calculation, the software will read back part of the results and display them (update contents) in JTable objects on GUI. Thus, this part of task (now it is in the new thread) should be put back into event dispatch thread, enabling it being seen in the above code frame. The sketch map of multi-threads used in CAM-Java is shown in Figure 4.2.
Figure 4.2 Sketch map of multi-threads used in CAM-Java.
Chapter 5 Discussion and Future Work

5.1 Discussion

In the field of scientific or engineering research, software development often involves the implementation of specific analytic and interface algorithms with relatively sophisticated mathematical principles and large scale dataset. In addition, when the algorithms are successfully implemented and verified, researchers may want to further integrate the newly developed algorithms into the existing and comprehensive software platform or pipeline. For example, adding a nice and user-friendly graphic user interface associate with the underlying algorithms, in order to improve the usability and stability of the software, and expand the utility of the software, ultimately for the wider applications and faster distributions of the software among the science and engineering community.

In order to fulfill this objective, one efficient way is to use two different programming languages for the implementation of a “corporative” software. One language can specifically focus on the design and implementation of complicated algorithms, often being a script based language and has considerable existing libraries providing numerous mathematical algorithms. Another language could be a universal programming language which has easier functionality of building good GUI, where Visual Basic, Visual C++, and Java are all good candidates. Logically, So one can make use of the advantages of both languages.

MATLAB may be the first choice by most researchers as a script based language. MATLAB is a commercial numerical computing programming environment, and has enjoyed its high popularity for about 20 years. The ever increasing new tool boxes and features make MATLAB even more powerful these days. On the other hand, however, MATLAB is a non-open source commercial software environment by MathWorks, and the price of MATLAB license is quite expensive that may not be affordable by small research groups or students. Moreover, the license of the product may prohibit people from sharing their results freely to the open world of the scientific community. Nowadays, some open-source script based languages are also available, one of which is the R language that has been known by more and more people, and it has already been widely used in statistical and bioinformatics community. Certainly, R is not limited to statistics or mathematics, while in fact, R is designed to be a universal programming language, and almost every commonly used MATLAB function has an R alternative, meaning that most algorithms written by MATLAB can also be written by R without much difficulty.
There have already been some successful examples of combining two different languages together, for example, to combine MATLAB and Java [10]. In this project, we are trying to combine R and Java. The resulting CAM-Java software shows the successful combination in that it runs smoothly and stable with a Java implemented graphic user interface, in which the core algorithms are written in R. Compared to other usually seen combinations (MATLAB and Java, MATLAB and C, etc.), we found that this combination has two major advantages:

1) Since both R and Java are open-source and cross-platform programming languages, the software written in R and Java can be easily run on multiple platforms (Windows, UNIX like Systems, and Mac OS, etc.) without caring much about compatibility issues, and can be freely distributed among the various communities (both languages are under GNU General Public License).

2) Interactive integration of R and Java is actually quite simple. In this project, what we really need are just one R library (Runiversal), and several Java class packages (RCaller). More conveniently, we do not need to know anything about compilers or drivers (for example, to connect MATLAB and Java, one has to first compile MATLAB functions and then create corresponding Java and C drivers [10]). With the support of abundant library functions in both R and Java languages, we can combine the two languages much coherently. In this project, we tried to create a simple R console on Java GUI, and it works quite well in the final product. This example shows that one is able to, not only pass commands and data between R and Java, but also integrate the whole R environment into Java GUI (or vice versa). This software implementation strategy will make the combined software even more powerful (for example, users can modify the underlying algorithms written by R and run them directly in Java GUI).
5.2 Future Work

As for the programming aspect, we plan to further study the issues of adopting R and Java to build the project. First, since we translate MATLAB code to R code, more computational efficiency issues in R version algorithms should be considered, comparing to MATLAB version algorithm. Especially when dealing with large scale data, more efforts should be made to reduce the calculation time in R version algorithm. This may be accomplished by considering R’s own data structures, or by adopting specific speedup techniques. Also, we are considering implementing specific tools to aid the translation (for example, an automatic MATLAB-R translation program).

Second, the current program is implemented as a standalone program and can only be used by one user at a time. In the future we plan to move the program onto website, making it work as a server and can handle requests from multiple users simultaneity. In order to achieve this, more complicated multi-threads design should be considered. Also the efficiency of using multi-threads should be taking into consideration, for example, to use one single thread to handle the displaying of GUI to multiple users.

Finally, the most important issue is to connect R and Java in a much deeper level. As described above, this may further improve the usability of the software, and make it a more powerful tool to assist researchers. Several aspects could be considered:

1) Making the program able to stop or pause at a specific point during the calculation, so the user can check intermediate results. And then with or without the user’s modification, the calculation process should be able to continue running from the stop point.

2) Adding move interactive features associated with underlying R algorithms in Java GUI. For example, a GUI based tool manipulating results figures (or input data as well) could be implemented, to let the user changes the selection of input data based on current results. This may need the underlying R algorithms respond the user’s choices as well.

3) More errors (or exceptions) which may be generated by R environment during the calculation should be identified and handled by Java module, and subsequent managing methods should be considered.

For the specific software CAM-Java described in this thesis, we also plan to improve its functionalities in two major directions:

1) One of main fields in which CAM-Java can be applied is the analysis of contrast-enhanced imaging. In this research area the original data are usually a series of images, while CAM algorithm requires that the input should be one number matrix, so we should first convert images to number matrices, and then combine them together into a big matrix. We plan to add this pre-processing procedure to our software, so
that it can load original images directly rather than requiring users to prepare the formatted data beforehand.

2) With the ongoing researches in our lab, these CAM based algorithms are still developing. So the CAM-Java software should also follow that development. Since we use the implementation strategy that dividing the underlying algorithms suite into different smaller yet independent modules, our algorithms can be easily updated or expanded. We are planning to add in alternative methods or schemes to the current algorithms suite. For example, CAM based algorithms in this software are often used to separate different sources (or compartments) from mixed input data. In the current version of the software, users have to set the number of sources before running the algorithms. While in the real world there are lots of blind source separation tasks, where users do not know how many sources are in the mixtures. We are developing a model selection algorithm to solve this problem, that is, to help users to determine the optimal number of sources contained in the mixtures. In the future version of the software, we plan to add this function. So there will be no need to set the number of sources as an algorithm parameter beforehand. Another example is in the clustering procedure of CAM algorithm, we plan to add more clustering methods and use ensemble clustering technique to combine the outcomes of all these methods together to generate the final clustering result, rather than to use the result of only one clustering method, in order to improve the adaptability, robustness, and stability of the clustering solution [11].
References


