AN R-PACKAGE FOR THE ESTIMATION AND TESTING
OF MULTIPLE COVARIATES AND BIOMARKER
INTERACTIONS FOR SURVIVAL DATA
BASED ON LOCAL PARTIAL LIKELIHOOD

by

SIWEI ZHANG

A project submitted to the
Department of Mathematics and Statistics
in conformity with the requirements for
the degree of Master of Science

Queen’s University
Kingston, Ontario, Canada
September 2016

Copyright © Siwei Zhang, 2016
Abstract

When we study the variables that affect survival time, we usually estimate their effects by the Cox regression model. In biomedical research, effects of the covariates are often modified by a biomarker variable. This leads to covariates-biomarker interactions. Here biomarker is an objective measurement of the patient characteristics at baseline. Liu et al. (2015) has built up a local partial likelihood bootstrap model to estimate and test this interaction effect of covariates and biomarker, but the R code developed by Liu et al. (2015) can only handle one variable and one interaction term and can not fit the model with adjustment to nuisance variables. In this project, we expand the model to allow adjustment to nuisance variables, expand the R code to take any chosen interaction terms, and we set up many parameters for users to customize their research. We also build up an R package called “lplb” to integrate the complex computations into a simple interface.

We conduct numerical simulation to show that the new method has excellent finite sample properties under both the null and alternative hypothesis. We also applied the method to analyze data from a prostate cancer clinical trial with acid phosphatase (AP) biomarker.
Acknowledgments

Here I sincerely appreciate my supervisors professor Bingshu Chen and professor Wenyu Jiang for your patience, guidance and help for the past whole year of my Master’s program. Especially on this project, you did much more than I expected and most of your help will last long into the future. It is a milestone in my life to study Statistics at Queen’s after 5 years working in the finance industry. You provided me such a great opportunity to start learning what I really want and what I want to engage for in the future.

I also want to thank the Department of Mathematics and Statistics and Queen’s University for providing me such a perfect environment and facilities to let me enjoy this year. Thanks to the professors who lectured me and who I worked for as a Teaching Assistant. Thanks to Ru Zhang who helped me a lot on my study, and thanks to all my classmates.

Thanks to my wife, Wenjing Su, who is also a graduate student of Queen’s University graduated last year. It is great to experience such a perfect year with you.
Contents

Abstract i

Acknowledgments ii

Contents iii

List of Tables v

List of Figures vi

Chapter 1: Introduction 1

1.1 Background ................................................. 1
1.2 Model for Covariates-Biomarker Interaction ..................... 3
1.3 Contributions ............................................. 4

Chapter 2: Review of LPLE and LPLB test 5

2.1 Overview ................................................. 5
2.2 Local Partial Likelihood Estimation (LPLE) ..................... 5
2.3 Estimation of Variance for LPLE ................................ 8
2.4 Maximum Likelihood for Regular Cox Model .................... 10
2.5 Local Partial Likelihood Bootstrap Test (LPLB) ............... 12

Chapter 3: Computational Method and the R package 15

3.1 Overview ............................................... 15
3.2 Prepare the Data Set .................................. 16
3.3 Kernels .................................................. 17
3.4 Calculating $\text{Var} (\hat{\beta}(w_0) - \hat{\beta})$ ............... 18
3.5 The Test Statistics ..................................... 20
3.6 Fit Local Partial Likelihood Estimates (lple) ................ 21
3.7 The Bootstrap Procedure ................................ 22
3.8 R Package Description .................................. 23
3.9 Setup the “lplb” R package .............................. 25
List of Tables

4.1 Empirical test size under the null hypothesis $H_0$ and empirical test power under the alternative hypotheses $H_1$ for three different models. Results were based on $B = 100$ bootstrap and $R = 200$ simulation replications for $\beta \neq 0$ and $R = 500$ replications for $\beta_1 = \beta_2 = 0$. . . . 34
List of Figures

3.1 “lplb” R package at GitHub website (https://github.com/statapps/lplb) 26

4.1 R output of “lplb” (print), with 25 chosen $w_0$ points for estimation of
$\beta_1(W)$ and $\beta_2(W)$, $p_1 = 2$ ................................. 29

4.2 R output of “lplb” (plot) of $\hat{\beta}_1(W)$ and $\hat{\beta}_2(W)$ at 25 chosen $w_0$ points,
$p_1 = 2$ ................................................................. 30

4.3 R output of “lplb” (plot), compared with true values of $\beta_1(W)$ and
$\beta_2(W)$, $p_1 = 2$ ................................................................. 30

4.4 R output of “lplb” (print), with 99 chosen $w_0$ points for estimation of
$\beta_1(W)$ and $\beta_2(W)$, $p_1 = 2$ ................................. 31

4.5 R output of “lplb” (plot) of $\hat{\beta}_1(W)$ and $\hat{\beta}_2(W)$ at 99 chosen $w_0$ points,
p_1 = 2 ................................................................. 31

4.6 Model 1 estimated treatment effects as functions of biomarker $ap$ on
transformed scale. Estimation is based on 25 chosen $w_0$ points. .... 36

4.7 Model 2 estimated treatment effects as functions of biomarker $ap$ on
transformed scale. Estimation is based on 25 chosen $w_0$ points. .... 37
Chapter 1

Introduction

1.1 Background

In survival analysis, we focus on patients’ life time and the factors that may obviously or potentially affect it. One of the most commonly used models is the Cox proportional hazards (PH) model (Cox, 1972), which describes the distribution of the survival time $T$ through a linear combination of a vector of covariates $Z$, $(Z_1, Z_2, \ldots, Z_p)^T$, and a vector of coefficients $\beta$, $(\beta_1, \beta_2, \ldots, \beta_p)^T$. In the PH model, the covariate $Z$ will affect survival time $T$ by

$$
\lambda(t, Z) = \lambda_0(t) \exp\{\beta^T Z\},
$$

(1.1)

where $\lambda(t, Z)$ is the hazard function at time $t$, $\lambda_0(t)$ is the baseline hazard function at time $t$, and $\beta$ is the parameter vector we are hoping to estimate. For the regular PH model, a partial likelihood method is used to estimate $\beta$.

However, some examples (Hayes et al., 2005) demonstrate that some covariate effects vary for different patients with different levels of measurement of patient characteristics at baseline, which we call the “biomarker”, denoted as “$W$” in this project.
So assuming constant effects for the covariate vector $Z$ may not be appropriate for such cases, and we can build interaction terms between the biomarker and every variable of interest to investigate the relationship between the covariates $Z$ and the biomarker $W$ (Cox, 1972, Shuster and van Eys, 1983). Thus the Cox model becomes

$$
\lambda(t) = \lambda_0(t) \exp\{\beta^T Z + \alpha^T ZW + \gamma W\},
$$

(1.2)

where $\alpha$ and $\gamma$ are the coefficients of the interaction terms and $W$. By adding the interaction terms, the effect of $Z$ will change with $W$ of different values. Thus the PH model could be written as (Liu et al., 2015)

$$
\lambda(t) = \lambda_0(t) \exp\{\beta(W)^T Z + g(W)\},
$$

(1.3)

where $\beta(\cdot)$ and $g(\cdot)$ are functions of biomarker $W$, and $\beta(\cdot)$ is of our primary interest.

Since the biomarker $W$ can be either a continuous or discrete variable, to estimate the function of $\beta(W)$, local partial likelihood estimation (LPLE) method is used to estimate $\beta$ at every $w_0$, which is a series of estimated points chosen arbitrarily (Fan et al., 2006).

Furthermore, we can evaluate the goodness of model (1.3) with the null hypothesis that $\beta(W) = \beta_0$, the covariates $Z$ have constant effects, by using local partial likelihood bootstrap (LPLB) test (Liu et al., 2015).

For its importance and wide use, we wish to build a general computational tool to conveniently estimate the coefficients and conduct the LPLB test. However, the R code developed by Liu et al. (2015) can only handle one variable ($Z$ contains only treatment) and one interaction term (treatment and biomarker interaction), so
1.2 Model for Covariates-Biomarker Interaction

Let \( p \) be the total number of covariates in \( Z \). Assuming that only a subset of \( Z \) of \( p_1 \) covariates \( Z_1 = (Z_1, Z_2, \ldots, Z_{p_1})^T \), \( p_1 \leq p \), interact with \( W \) and the other \((p - p_1)\) covariates \( Z_2 = (Z_{p_1+1}, \ldots, Z_p)^T \) do not interact with \( W \), then the Cox model in this scenario can be written as

\[
\lambda(t, Z) = \lambda_0(t) \exp \{\beta_1(W)^T Z_1 + \beta_2^T Z_2 + g(W)\}, \tag{1.4}
\]

for the \( p \)-dimensional covariate vector, \( Z = (Z_1^T, Z_2^T)^T \).

Model (1.4) is more flexible than Liu et al. (2015)’s model (1.3) because we can choose any integer value of \( p_1 \) between 1 and \( p \). The primary objective of this project is to develop methods to estimate \( \beta_1(W) \) while treating \( \beta_2 \) and \( g(W) \) as nuisance parameters. We are interested in testing that the effects of covariates \( Z_1 \) do not vary for different biomarker \( W \) values, that is

\[
H_0 : \ \beta_1(W) = \beta_1. \tag{1.5}
\]

Under the null hypothesis (1.5), model (1.4) reduces to

\[
\lambda(t, Z) = \lambda_0(t) \exp \{\beta^T Z + \gamma W\}, \tag{1.6}
\]
where $\beta = (\beta_1^T, \beta_2^T)^T$, and $Z = (Z_1^T, Z_2^T)^T$.

### 1.3 Contributions

In this project, we use a more flexible model (1.4) to expand the original R code, so that any subset of covariates and interaction terms can be used and the estimated $\beta_1(W)$ can be adjusted for other nuisance covariates. Then we build the new model into an R package with a simple interface. We improve some of the methodologies by using matrix operations and save some computing time. Researchers may choose different models and parameters for different data, such as the number of covariates and interaction terms, kernel function types and bandwidths, a series of estimated points of the biomarker values $w_0$, etc.

We run simulations to show that when $Z_2$ has a very strong but constant effect on survival time $T$, fitting model (1.3) may introduce bias to $\beta_1(W)$. On the other hand, model (1.4) fits the data well and have higher statistical power in some settings.
Chapter 2

Review of LPLE and LPLB test

2.1 Overview

In this chapter, we review the theory and method of local partial likelihood estimation and bootstrap test (Liu et al., 2015), and adapt it into our new model (1.4), in which we are interested in the interaction term of biomarker $W$ and $Z_1$ only, where $Z_1$ is a user-specified subset of $Z$ as defined in chapter 1.

In section 2.2, we introduce the local partial likelihood estimation and re-define the variables based on model (1.4). In section 2.3, we further derive the variance of the estimator from the previous section. Then we discuss the maximum likelihood estimation of the coefficients and variance of regular Cox model in section 2.4. We will further construct the test for (1.5), $H_0 : \beta_1(W) = \beta_1$, by bootstrap method in section 2.5.

2.2 Local Partial Likelihood Estimation (LPLE)

For a study with $n$ subjects, we denote $\tilde{T}_i$ as the observed survival time, $Z_i$ as the vector of covariates, and $W_i$ as the biomarker, for subject $i = 1, 2, \cdots, n$. Because
LOCAL PARTIAL LIKELIHOOD ESTIMATION (LPLE)

censoring exists, the survival time $\tilde{T}_i = \min(T_i, C_i)$, where $T_i$ and $C_i$ denote the failure time and censoring time, respectively. Let $\delta_i$ be the indicator of censoring, i.e. $\delta_i = 0$ for censored survival time when $\tilde{T}_i = C_i$, and $\delta_i = 1$ for uncensored when $\tilde{T}_i = T_i$. Then we have our dependent variable $y_i$, determined by $\{\tilde{T}_i, \delta_i\}$, for subject $i$. The independent variables include $Z_i$ and $W_i$. If the dimension of $Z_i$ is $p$, then we have an $n \times (p + 1)$ covariate matrix $X$, whose $i^{th}$ row is a $(p + 1)$ dimension vector of $(Z_{1i}, Z_{2i}, \ldots, Z_{pi}, W_i)$.

With partial likelihood for Cox proportional hazards model, we need to maximize

$$L(\beta_1(\cdot), \beta_2, g(\cdot)) = \prod_{i=1}^{n} \left[ \frac{\exp\{\beta_1(W_i)^T Z_{1i} + \beta_2^T Z_{2i} + g(W_i)\}}{\sum_{j=1}^{n} Y_j(\tilde{T}_i) \exp\{\beta_1(W_j)^T Z_{1j} + \beta_2^T Z_{2j} + g(W_j)\}} \right]^{\delta_i}, \quad (2.1)$$

with respect to $\beta_1(\cdot)$, $\beta_2$ and $g(\cdot)$ from model (1.4), where $Y_j(t_i) = I(\tilde{T}_j \geq t_i)$ is an indicator of whether subject $j$ is at risk at time $t_i$.

In the above coefficients, $\beta_2$ are fixed parameters, but $\beta_1(\cdot)$ and $g(\cdot)$ have unknown functional forms and we need to estimate their values at different $w_0$. Nonparametric estimation was introduced by Fan et al. (2006). Assuming that $\beta_1(\cdot)$ and $g(\cdot)$ are smooth functions, we use the first order Taylor expansions at every $w_0$ within a narrow neighborhood of $w_0$ as follows

$$\beta_1(w) \approx \beta_1(w_0) + \beta'_1(w_0)(w - w_0) \equiv \zeta + \eta(w - w_0);$$

$$g(w) \approx g(w_0) + g'(w_0)(w - w_0) \equiv \alpha + \gamma(w - w_0). \quad (2.2)$$
In this way, we can have the following equivalent transformations,

\[
\beta_1(W_i)Z_{1i} + \beta_2Z_{2i} + g(W_i) \\
\approx (\zeta + \eta(W_i - w_0))Z_{1i} + \beta_2Z_{2i} + \gamma(W_i - w_0) \\
= \zeta^T Z_{1i} + \beta_2^T Z_{2i} + \eta^T Z_{1i}(W_i - w_0) + \gamma(W_i - w_0) \\
= \rho^T Z_i + \eta^T Z_{1i}(W_i - w_0) + \gamma(W_i - w_0) \\
= \xi^T \tilde{R}_i.
\]

Equation (2.4) is because the \(\alpha\) term in (2.2) is absorbed into the baseline hazard function \(\lambda_0(t)\) in model (1.4). For equation (2.5), \(\zeta\) and \(\beta_2\) are all fixed parameters of every covariates in \(Z_i\). If we define \(\rho = (\zeta^T, \beta_2^T)^T\) and \(Z_i = (Z_{1i}^T, Z_{2i}^T)^T\), then we can derive (2.6). Finally, we define \(\xi = (\rho^T, \eta^T, \gamma)^T\), and \(\tilde{R}_i = (Z_i^T, Z_{1i}^T(W_i - w_0), (W_i - w_0))^T\), thus we derive (2.7), which is a simple notation for the expression of (2.3), where \(\tilde{R}_i\) is the covariate vector for subject \(i\), and \(\xi\) is the regression coefficients for each covariate in \(\tilde{R}_i\). Note here that \(W_i\) was changed into \((W_i - w_0)\) in this new form, and the distance from \(W_i\) to \(w_0\) will be arranged to different kernel weight in the following part. Then we have a new matrix of \(X_R\) for the local regression at \(w_0\), with \(\tilde{R}_i^T\) as the \(i^{th}\) row and \(n\) rows in total. Now we have the interaction terms of \(Z_{1i}\) and \((W_i - w_0)\) in our model, which will be used in chapter 3.

As we mentioned, the Taylor expansion is only valid in a small local interval around \(w_0\), so we use Kernel weight to arrange contributions of different points of all the \(W_i\). Define \(K_h(W_i - w_0)\) as the kernel weight function of \(W_i\) at \(w_0\), with bandwidth \(h\). We will discuss more on different options for kernel functions in chapter 3.

We also define, for subject \(i\), a counting process \(N_i(t) = I(\tilde{T}_i \leq t, \delta_i = 1)\), which
2.3. ESTIMATION OF VARIANCE FOR LPLE

denotes the number of uncensored failure time before and at time \( t \).

By all the above (Liu et al., 2015), we can write the logarithm of partial likelihood function of (2.1) at \( w_0 \) as

\[
l_{w_0}(\xi) = \sum_{i=1}^{n} \int_{0}^{\infty} K_h(W_i - w_0) \left( \xi^T \tilde{R}_i - \log\left\{ \sum_{j=1}^{n} Y_j(u) K_h(W_j - w_0) \exp(\xi^T \tilde{R}_i) \right\} \right) dN_i(u),
\]

which is a local partial likelihood. In equation (2.8), \( dN_i(u) = 1 \) only when we observe a failure at time \( u = t_i \) for subject \( i \), otherwise it equals to 0, which means there is only one time point \( t_i \) for each subject \( i \), that makes \( dN_i(u) = 1 \). Therefore, in (2.8), we only sum on the value in the integral at one time point for each subject \( i \). By (2.8), we can have the estimated local partial likelihood coefficients of \( \xi \) (LPLE) at point \( w_0 \), which we denote as \( \hat{\xi}(w_0) \).

2.3 Estimation of Variance for LPLE

In this section, we review variance estimation for \( \hat{\xi}(w_0) \), using the method by Fan et al. (2006).

We define \( H \) to be a \((p + p_1 + 1) \times (p + p_1 + 1)\) diagonal matrix as follows

\[
H = \begin{pmatrix}
I_{p \times p} & 0 & 0 \\
0 & hI_{p_1 \times p_1} & 0 \\
0 & 0 & h
\end{pmatrix}
\]

Let \( Z^{\otimes k} = 1, Z, ZZ^T \) for \( k = 0, 1, 2 \), respectively.

We write \( \xi^T \tilde{R}_i \) as \( \xi^T \tilde{R}_i = \xi^T HH^{-1} \tilde{R}_i = \phi^T R_i \), where \( \phi^T = \xi^T H \), and \( R_i = \)
2.3. ESTIMATION OF VARIANCE FOR LPLE

\( H^{-1} \tilde{R}_i \). For a given kernel function \( K_h(\cdot) \), we define

\[
S^{(k)}_{w_0}(\phi, u) = \sum_{i=1}^{n} \left[ K_h(W_i - w_0) Y_i(u) \exp(\phi^T R_i) \right] (R_i)^{\otimes k}, \quad k = 0, 1, 2. \tag{2.10}
\]

By the above definitions, we can rewrite the logarithm of the local partial likelihood function (2.8) as

\[
l_{w_0}(\xi) = l_{w_0}(H^{-1}\phi) = \tilde{l}_{w_0}(\phi) = \sum_{i=1}^{n} \int_0^\infty K_h(W_i - w_0) \left( \phi^T R_i - \log S^{(0)}_{w_0}(\phi, u) \right) dN_i(u) \tag{2.11}
\]

Thus our target estimator of coefficients has changed from \( \hat{\xi}(w_0) \) to \( \hat{\phi}(w_0) \). The maximum likelihood estimates of \( \hat{\phi}(w_0) \) can be obtained by using Cox model with weighted function \( K_h(W_i - w_0) \) at each \( w_0 \).

By Fan et al. (2006), the variance of \( \hat{\phi}(w_0) - \phi_0(w_0) \), where \( \phi_0(w_0) \) is the true value of \( \phi \) at \( w_0 \), can be estimated by

\[
\hat{\text{Var}}(\hat{\phi}(w_0) - \phi_0(w_0)) = \hat{I}_{w_0}^{-1}(\hat{\phi}(w_0)) \hat{\Pi}_{w_0}(\hat{\phi}(w_0)) \hat{I}_{w_0}^{-1}(\phi(w_0)) \tag{2.12}
\]

where

\[
\hat{I}_{w_0}(\hat{\phi}(w_0)) = \sum_{i=1}^{n} \int_0^\infty K_h(W_i - w_0) \times \frac{S^{(2)}_{w_0}(\hat{\phi}(w_0), u) S^{(0)}_{w_0}(\hat{\phi}(w_0), u) - \{S^{(1)}_{w_0}(\hat{\phi}(w_0), u)\}^2}{\{S^{(0)}_{w_0}(\phi(w_0), u)\}^2} dN_i(u) \tag{2.13}
\]

\[
\hat{\Pi}_{w_0}(\hat{\phi}(w_0)) = \sum_{i=1}^{n} \int_0^\infty [K_h(W_i - w_0)]^2 \left\{ R_i - \frac{S^{(1)}_{w_0}(\hat{\phi}(w_0), u)}{S^{(0)}_{w_0}(\hat{\phi}(w_0), u)} \right\}^2 dN_i(u) \tag{2.14}
\]
2.4. Maximum Likelihood for Regular Cox Model

Before we start the next part, we will here define some similar notations for the no interaction model (1.1), where \( W \) is included in \( Z \).

2.4 Maximum Likelihood for Regular Cox Model

Recall section (1.2), under the null hypothesis (1.5) \( H_0 : \beta_1(W) = \beta_1 \), model (1.4) reduces to a regular Cox model of (1.6)

\[
\lambda(t, Z) = \lambda_0(t) \exp\{\beta^T Z + \gamma W\}. \tag{2.15}
\]

Define the coefficients as \( \theta = (\beta^T, \gamma)^T \). Define the covariates as \( G_i = (Z_i^T, W_i)^T \).

Thus \( G \) is a \( ((p + 1) \times n) \) matrix, i.e. the transpose of covariates matrix \( X \), as we mentioned at the beginning of section 2.2, without the interaction terms. Meanwhile, \( R \) is a \( ((p + p_1 + 1) \times n) \) matrix with interaction terms, as we denoted before \( X_R \).

Then similarly to \( S_{(k)}^{(0)}(\phi, u) \), we define an \( S^{(k)}(\theta, u) \) function

\[
S^{(k)}(\theta, u) = \sum_{i=1}^{n} \left[ Y_i(u) \exp(\theta^T G_i) \right] (G_i)^\otimes k, \quad k = 0, 1, 2. \tag{2.16}
\]

It is obvious that \( S^{(0)}(\theta, u) = \sum_{i=1}^{n} Y_i(u) \exp(\theta^T G_i) \), and \( S^{(1)}(\theta, u) = \frac{\partial}{\partial \theta} S^{(0)}(\theta, u) \), \( S^{(2)}(\theta, u) = \frac{\partial^2}{\partial \theta \partial \theta^T} S^{(0)}(\theta, u) \) are the first and second partial derivative, respectively.

The logarithm likelihood function for regular Cox model is

\[
\tilde{l}_1(\theta) = \sum_{i=1}^{n} \int_{0}^{\infty} \left( \theta^T G_i - \log S^{(0)}(\theta, u) \right) dN_i(u) \tag{2.17}
\]
Then the observed Fisher Information matrix for the partial likelihood is given by

\[
I(\hat{\theta}) = -\frac{\partial^2}{\partial \theta \partial \theta^T} \tilde{I}_1(\theta)|_{\theta = \hat{\theta}}
= -\frac{\partial}{\partial \theta} \left( \sum_{i=1}^{n} \int_{0}^{\infty} \left( G_i - \frac{S^{(1)}(\theta, u)}{S^{(0)}(\theta, u)} \right) dN_i(u) \right)|_{\theta = \hat{\theta}}
= \sum_{i=1}^{n} \int_{0}^{\infty} \frac{S^{(2)}(\hat{\theta}, u)S^{(0)}(\hat{\theta}, u) - \{S^{(1)}(\hat{\theta}, u)\} \otimes^2 \{S^{(0)}(\hat{\theta}, u)\}^2}{\{S^{(0)}(\hat{\theta}, u)\}^2} dN_i(u).
\] (2.18)

Let \( \Pi(\hat{\theta}) \) be the sum of \( \left( \frac{\partial \tilde{I}_1(\theta)}{\partial \theta_i} \right)^2 \) over \( i = 1, 2, \cdots, n \) as follows,

\[
\Pi(\hat{\theta}) = \sum_{i=1}^{n} \int_{0}^{\infty} \left\{ G_i - \frac{S^{(1)}(\hat{\theta}, u)}{S^{(0)}(\hat{\theta}, u)} \right\} \otimes^2 dN_i(u).
\] (2.19)

We also define \( \Pi^* \) for future calculation of the covariance of \( \hat{\phi}(w_0) \) and \( \hat{\theta} \) as follows,

\[
\Pi^*_{w_0}(\hat{\phi}(w_0), \hat{\theta}) = \sum_{i=1}^{n} \int_{0}^{\infty} K_h(W_i - w_0) \times \left\{ R_i - \frac{S^{(1)}_{w_0}(\hat{\phi}(w_0), u)}{S^{(0)}_{w_0}(\hat{\phi}(w_0), u)} \right\} \left\{ G_i - \frac{S^{(1)}(\hat{\theta}, u)}{S^{(0)}(\hat{\theta}, u)} \right\}^T dN_i(u).
\] (2.20)

Variances of \( \hat{\phi}(w_0), \hat{\theta} \) and covariance between \( \hat{\phi}(w_0) \) and \( \hat{\theta} \) will be used to construct the test with null hypothesis (1.5), \( H_0 : \beta_1(W) = \beta_1 \), in section 2.5.
2.5 Local Partial Likelihood Bootstrap Test (LPLB)

To test the difference between our LPLE $\beta_1(W)$ and a constant estimator $\beta_1$, we set null and alternative hypotheses

$$H_0 : \beta_1(W) = \beta_1$$
$$H_1 : \beta_1(W) \neq \beta_1.$$  \hspace{1cm} (2.21)

Then we define the test statistic

$$Q_1 = \max_{1 \leq j \leq m} |(\hat{\beta}_1(w_j) - \hat{\beta}_1)^T [\hat{\text{Var}}(\hat{\beta}_1(w_j) - \hat{\beta}_1)]^{-1} (\hat{\beta}_1(w_j) - \hat{\beta}_1)|,$$ \hspace{1cm} (2.22)

where $w_j \in w = (w_1, w_2, \ldots, w_m)$ are the $m$ different biomarker values that can be used to estimate $\beta_1(w_j)$, $\hat{\beta}_1(w_j)$ is the estimated $\beta_1(w_j)$, $\hat{\beta}_1$ is the estimated $\beta_1$, and $\hat{\text{Var}}(\cdot)$ is the estimated variance matrix under the null hypotheses $H_0$. By Liu et al. (2015), for a specific $w_0$,

$$\hat{\text{Var}}(\hat{\beta}_1(w_0) - \hat{\beta}_1) = \hat{\Sigma}_{11} + \hat{\Gamma}_{11} - 2\hat{\Omega}_{11},$$ \hspace{1cm} (2.23)

where $\hat{\Sigma}_{11}, \hat{\Gamma}_{11}$ and $\hat{\Omega}_{11}$ are the top left $p_1 \times p_1$ sub-matrices of matrices $\hat{\Sigma}_n, \hat{\Gamma}_n$ and $\hat{\Omega}_n$, respectively. The variance and covariance matrices can be estimated by,

$$\hat{\Sigma}_n = \hat{\text{Var}}(\hat{\theta} - \theta_0) = I^{-1}(\hat{\theta})\Pi(\hat{\theta})I^{-1}(\hat{\theta}),$$ \hspace{1cm} (2.24)

$$\hat{\Gamma}_n = \hat{\text{Var}}(\hat{\phi}(w_0) - \phi_0(w_0))$$
$$= I^{-1}_{w_0}(\hat{\phi}(w_0))\hat{\Pi}_{w_0}(\hat{\phi}(w_0))I^{-1}_{w_0}(\hat{\phi}(w_0)), $$ \hspace{1cm} (2.25)
2.5. LOCAL PARTIAL LIKELIHOOD BOOTSTRAP TEST (LPLB)

\[ \hat{\Omega}_n = C^\text{cov} \left\{ \{ \phi(w_0) - \phi_0(w_0) \}, (\theta - \theta_0) \right\} \]

\[ = \hat{I}^{-1}_{w_0} (\hat{\phi}(w_0)) \Pi_{w_0}^* (\hat{\phi}(w_0), \hat{\theta}) I^{-1}(\hat{\theta}). \] (2.26)

Consequently, we can calculate \( Q_1 \) for a given dataset. Because the distribution of \( Q_1 \) is unknown, we bootstrap the data to approximate the distribution of \( Q_1 \). We bootstrap on the dataset by resampling on the residual-indicator pairs. The residuals are martingale residuals given below, and \( \hat{\Lambda}(\cdot), \hat{\Lambda}_0(\cdot) \) are cumulative hazard function and baseline cumulative hazard function, respectively, (Liu et al., 2015)

\[ \hat{\epsilon}_i = \delta_i - \hat{\Lambda}(\tilde{T}_i | Z_i, W_i) \]

\[ = \delta_i - \hat{\Lambda}_0(\tilde{T}_i) \exp\{\hat{\beta}_1(W_i)Z_{1i} + \hat{\beta}_2 Z_{2i} + \hat{\gamma}(W_i)W_i\} \] (2.27)

\[ = \delta_i - \hat{\Lambda}_0(\tilde{T}_i) \exp\{\hat{\xi}(W_i)^T \tilde{R}_i\}. \]

We then resample the pairs \( \{\hat{\epsilon}_i, \delta_i\}_{i=1}^n \), but keep the original covariates matrix \( X \), to generate the new dataset \( \{\hat{\epsilon}_i^*, \delta_i^*, Z_i, W_i\}_{i=1}^n \).

For calculating the bootstrap statistic \( Q^*_1 \), we also need the dependent variable, \( y^* = (\tilde{T}^*, \delta^*) \), where \( \tilde{T}^* = \{\tilde{T}_i^*\}_{i=1}^n \) and \( \delta^* = \{\delta_i^*\}_{i=1}^n \) are the resampled survival time and censoring indicator, respectively. \( X \) will be used together with the bootstrap residual pairs \( \{\hat{\epsilon}_i^*, \delta_i^*\}_{i=1}^n \) to construct the dependent variable pairs \( \{\tilde{T}_i^*, \delta_i^*\}_{i=1}^n \) (Loughin, 1995). Instead of reconstructing the survival time series \( \{\tilde{T}_i^*\}_{i=1}^n \), we can only take the rank of \( \{\tilde{T}_i^*\}_{i=1}^n \) as the bootstrap survival time sample. The reason is that in Cox PH model, all the estimations only depends on the order of time, not the specific time values. Moreover, because the baseline cumulative hazard function \( \Lambda_0(t) \) is a monotone increasing function in time \( \tilde{T}_i \), so their orders are the same. Thus, we could take the order of \( \hat{\Lambda}_0(\tilde{T}_i) \), denoted as \( \hat{\Lambda}_0^*(\tilde{T}_i) \) to be the new survival time sample.
\{T_i^\ast\}_{i=1}^n$, which can be constructed under the null hypothesis $H_0$ from formula (2.27) as
\[
\hat{\Lambda}_0^\ast(T_i) = \frac{\delta_i^\ast - \hat{\epsilon}_i^\ast}{\exp\{\hat{\beta}^T Z_i + \hat{\gamma} W_i\}},
\] (2.28)
where $\hat{\beta}$ and $\hat{\gamma}$ are estimated coefficients under $H_0$ from the regular Cox model (2.15).

Consequently, we obtain the new bootstrap dataset \{\hat{\Lambda}_0^\ast(T_i), \delta_i^\ast, Z_i, W_i\}_{i=1}^n. We will show late in section (3.4) that it is more computational efficiency to calculate the variance estimation when the survival times are sorted from smallest to largest. Therefore, here we sort the bootstrap dataset \{\hat{\Lambda}_0^\ast(T_i), \delta_i^\ast, Z_i, W_i\}_{i=1}^n by \{\hat{\Lambda}_0^\ast(T_i)\}_{i=1}^n and use this sorted data to calculate $Q^\ast_1$.

We do this resampling for $B$ times to obtain $B$ different statistic $Q^\ast_1$, say $Q^\ast_1, Q^\ast_2, \cdots, Q^\ast_B$.

At last, we can evaluate the \textit{p-value} for the hypotheses test as
\[
p = \frac{1}{B} \sum_{b=1}^B I\{Q^b_1 \geq Q_1\},
\] (2.29)
and we reject $H_0$ if \textit{p-value} is small.
Chapter 3

Computational Method and the R package

3.1 Overview

In this chapter, we develop statistical methods for model (1.4) and implement the new model in an R package. Because both the bootstrap procedure and the local partial likelihood method demand lots of computational time, we focus on development of algorithms that are efficient in computation.

In section 3.2, we outline several tools that we used to construct a standard data frame for analysis. In section 3.3, we provide a collection of different type of kernel function to be used in fitting the local partial likelihood model. In section 3.4, we state how to efficiently calculate the estimated variance of \((\hat{\beta}(w_0) - \hat{\beta})\) and the algorithm to compute the first and second partial derivatives \(S_{w_0}^{(k)}\) and \(S^{(k)}\) functions using matrix operation. Then we introduce the function to acquire the test statistic \(Q_1\) in section 3.5 and the fitting of local partial likelihood estimate using Cox model in section 3.6 as well as the bootstrap function in section 3.7. In section 3.8 and 3.9, we describe the constructions and usages of the “lplb” R package, as well as the installation of the package.
3.2 Prepare the Data Set

**Standardize the Biomarker Variable** In real life applications, biomarker can have all kinds of distributions. To simplify the analysis, we consider an algorithm to map them into the interval $(0, 1)$. Based on the fact that if a random variable $W \sim F_W(W)$, then $U = F_W(W) \sim \text{unif}(0, 1)$, we develop an R function `x.cdf()` to standardize the biomarker $W$ into $(0, 1)$ interval, based on the empirical cumulative distribution, $\hat{F}_n(w) = \frac{1}{n} \sum_{i=1}^{n} I\{w_i \leq w\}$. We fit the local partial likelihood with biomarker in the transformed $(0, 1)$ scale. This allows us to estimate the coefficient function $\beta_U(u)$ for the transformed biomarker. We then can use the inverse C.D.F. $w = \hat{F}_W^{-1}(u)$ to transform $\hat{\beta}_U(u)$ back to the original biomarker scale with

$$\hat{\beta}(w) = \hat{\beta}_U(\hat{F}_W^{-1}(u)).$$

**Interaction Terms Functions** The input dataset matrix is $X = (Z_1, Z_2, \cdots, Z_p; W)$ of dimensions $(n \times (p+1))$ as mentioned in section 2.2, where $n, p$ are the number of subjects and the length of vector $Z_i$ as before. However, we need to add the interaction terms into the input matrix $X$ before we fit the local partial likelihood model. So we create an R function `interaction.X.w0()` to change $X$ into $X_R$, a $n \times (p + p_1 + 1)$ matrix with $p_1$ interaction terms of covariates $Z_1, Z_2, \cdots, Z_{p_1}$ and $(W - w_0)$. Moreover, since `interaction.X.w0()` function depends on the value of $w_0$, we need to calculate this matrix for every estimated point $w_0$ we choose. For time saving sake, we will also use a $n \times (p + p_1 + 1)$ matrix, denoted as $X_\phi$, in which the $p_1$ interaction terms are between $Z_1, Z_2, \cdots, Z_{p_1}$ and $W$. Thus $X_\phi$ does not depend on $w_0$, and we create an R function `interaction.X()` to achieve it. See section 3.6 for
3.3 Kernels

Kernel weights serve an important role in local partial likelihood estimation. In Liu et al. (2015)'s simulation models, Epanechnikov kernel was used. In the R package we developed, we provide several different kernel functions that can be used by the users, which include “gaussian”, “epanechnikov”, “rectangular”, “triangular”, “biweight”, “cosine”, “optcosine”, with user specified bandwidth $h$. The kernel functions are as follows.

\[
\begin{align*}
gaussian : & \quad K_h(u) = \frac{1}{\sqrt{2\pi h}} \exp \left( -\frac{u^2}{2h^2} \right), \\
epanechnikov : & \quad K_h(u) = \frac{3}{4h}(1 - \frac{u^2}{h^2})I(|u| \leq h), \\
rectangular : & \quad K_h(u) = \frac{1}{2h}I(|u| \leq h), \\
triangular : & \quad K_h(u) = \frac{1}{h}(1 - \frac{|u|}{h})I(|u| \leq h)), \\
bweight : & \quad K_h(u) = \frac{15}{16h}(1 - \frac{u^2}{h^2})^2I(|u| \leq h), \\
\cosine : & \quad K_h(u) = \frac{1}{2h}(1 + \cos(\frac{\pi u}{h}))I(|u| \leq h), \\
\optcosine : & \quad K_h(u) = \frac{\pi}{4h}\cos(\frac{\pi u}{2h})I(|u| \leq h).
\end{align*}
\]

In our R package, gaussian kernel is used as the default kernel function. We will evaluate the performance of different kernel functions in future research.
3.4 Calculating $\hat{\text{Var}}(\hat{\beta}(w_0) - \hat{\beta})$

**Variance of $(\hat{\beta}(w_0) - \hat{\beta})$.**

As we mentioned in chapter 2, $\hat{\text{Var}}(\hat{\beta}(w_0) - \hat{\beta})$ is calculated from $\hat{\Sigma}_n, \hat{\Gamma}_n$ and $\hat{\Omega}_n$. So we need to compute $\hat{I}_{w_0}(\hat{\phi}(w_0)), \hat{\Pi}_{w_0}(\hat{\phi}(w_0)), I(\hat{\theta}), \Pi(\hat{\theta}),$ and $\Pi^*_{w_0}(\hat{\phi}(w_0), \hat{\theta})$, which are shown in formula (2.13), (2.14), (2.18)~(2.20), respectively.

The above calculation is time consuming if we use “loop” in the R program. To improve the computational speed, we will program this through matrix operation.

Note that we integrate on $dN_i(u)$ for $u$ on $(0, \infty)$, and by definition of $N_i(t) = I(\tilde{T}_i \leq t, \delta_i = 1)$, where $\tilde{T}_i$ is the observed survival time for subject $i$ as mentioned in section 2.2. For observed survival time $\tilde{T}_i = t_i, i = 1, \cdots, n$, $dN_i(u) = 1$ only when $u = t_i$ for an uncensored subject $i$, otherwise $dN_i(u) = 0$. So to improve calculating for the five formula above, we calculate the $\sum_{i=1}^{n} \int_{0}^{\infty} A(u)dN_i(u)$, where $A(u)$ denotes the integral part, by multiplying the status vector from our dataset by $A(u)$, where $A(u)$ has $n$ elements, and each element is calculated by $u = t_i$ for each subject $i$.

**First and Second Order Derivatives.**

Subsequently, we need to calculate $S^{(k)}_{w_0}, S^{(k)}$ function. Recall the definition of $S^{(k)}_{w_0}$ and $S^{(k)}$ formula (2.10) and (2.16),

$$S^{(k)}_{w_0}(\phi, u) = \sum_{j=1}^{n} K_h(W_j - w_0)Y_j(u) \exp(\phi^T R_j)(R_j)^{\otimes k} \quad k = 0, 1, 2,$$

$$S^{(k)}(\theta, u) = \sum_{j=1}^{n} Y_j(u) \exp(\theta^T G_j)(G_j)^{\otimes k}, \quad k = 0, 1, 2.$$
3.4. CALCULATING $\text{VAR}(\hat{\beta}(W_0) - \hat{\beta})$

Here we substitute $i$ with $j$ to distinguish from the index from $\sum_{i=1}^{n} \int_{0}^{\infty} A(u) dN_i(u)$ above.

Note that both the formula has the format of $\sum_{j=1}^{n} B_j Y_j(u)$, where $B_j$ is a function of the covariates for each $j$ from 1 to $n$, and $Y_j(u) = I(\tilde{T}_j \geq u)$, where $u = t_i, i = 1, \cdots, n$. So $Y_j(t_i) = I(\tilde{T}_j \geq t_i)$. Then, $\sum_{j=1}^{n} B_j Y_j(t_i) = \sum_{j=1}^{n} B_j I(\tilde{T}_j \geq t_i)$. If we sort $t_1, \cdots, t_n$, we do not need to loop on $j$. Instead, we can multiply $B$ by an upper triangular matrix of 1 to get all $\sum_{j=1}^{n} B_j Y_j(u)$ for every $u = t_1, \cdots, t_n$. That is,

$$
\begin{pmatrix}
\sum_{j=1}^{n} B_j Y_j(t_1) \\
\vdots \\
\sum_{j=1}^{n} B_j Y_j(t_n)
\end{pmatrix} =
\begin{pmatrix}
\sum_{j=1}^{n} B_j I(\tilde{T}_j \geq t_1) \\
\vdots \\
\sum_{j=1}^{n} B_j I(\tilde{T}_j \geq t_n)
\end{pmatrix} =
\begin{pmatrix}
1 & 1 & 1 & \cdots & 1 \\
0 & 1 & 1 & \cdots & 1 \\
0 & 0 & 1 & \cdots & 1 \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
0 & 0 & 0 & \cdots & 1
\end{pmatrix}
\begin{pmatrix}
B_1 \\
B_2 \\
B_3 \\
\vdots \\
B_n
\end{pmatrix}
$$

The result $(n \times 1)$ vector is the value of $\{ \sum_{j=1}^{n} B_j I(\tilde{T}_j \geq t_i) \}$ for each $i$ from 1 to $n$.

$S_{w_0}^{(k)}$ and $S^{(k)}$ are similar, but their main differences are in two ways: 1) $S_{w_0}^{(k)}$ depends on the value of $w_0$ but $S^{(k)}$ does not; 2) $S^{(k)}$ are based on the $((p+1) \times n)$ matrix of $G$ and $((p+1) \times 1)$ estimated vector of $\theta$, while $S_{w_0}^{(k)}$ are based on the $((p+p_1+1) \times n)$ matrix of $R$ and $((p+p_1+1) \times 1)$ estimated vector of $\phi$. So for $k = 0, 1, 2$, $S^{(k)}$ returns a number, a $((p+1) \times 1)$ vector, and a $((p+1) \times (p+1))$ matrix, while $S_{w_0}^{(k)}$ returns a number, a $((p+p_1+1) \times 1)$ vector, and a $((p+p_1+1) \times (p+p_1+1))$ matrix, respectively.

Since $I(\hat{\theta})$ and $\Pi(\hat{\theta})$ do not depend on $w_0$, we can return them from $S^{(k)}$ function.
We also return the value of \( \left\{ G_i - \frac{S^{(1)}(\hat{\theta}, u)}{S^{(1)}(\theta, w)} \right\} \) for calculating \( \Pi^*_w(\hat{\phi}(w_0), \hat{\theta}) \) in the following \( S^{(k)}_{w_0} \) function.

Since \( \hat{I}_{w_0}(\hat{\phi}(w_0)), \hat{\Pi}_{w_0}(\hat{\phi}(w_0)) \) and \( \Pi^*_w(\hat{\phi}(w_0), \hat{\theta}) \) depend on \( w_0 \), we cannot avoid to use a “loop” on \( w_0 \) to calculating \( S^{(k)}_{w_0} \) for the above three terms, which will be mentioned again in the latter part. For every \( w_0 \), we calculate \( S^{(k)}_{w_0} \) and then obtain \( \hat{I}_{w_0}(\hat{\phi}(w_0)), \hat{\Pi}_{w_0}(\hat{\phi}(w_0)) \) and \( \Pi^*_w(\hat{\phi}(w_0), \hat{\theta}) \).

Thus we have all the five terms to estimate the variance of \( (\hat{\beta}(w_0) - \hat{\beta}) \).

### 3.5 The Test Statistics

The R function \texttt{maxTest()} uses the returned values from \( S^{(k)}_{w_0} \) and \( S^{(k)} \) functions to calculate each statistic for every estimated point of \( w_0 \), then take the maximum value to obtain \( Q_1 \).

Because we have mapped the biomarker \( W \) into \((0, 1)\) interval, we can adopt a pre-specified series of \( w_0 \) in \((0, 1)\) interval correspondingly. Normally, we can setup the \( w_0 \) series to be 0.01 to 0.99 with a fix step \( \Delta w \), for example \( \Delta w = 0.01 \). We can also let \( w_0 \) takes unique values of the biomarker variable. Note that the number of \( w_0 \) is the primary determinant of the computation speed because it is necessary to loop over \( w_0 \). To save some computational time, we can decrease the number of \( w_0 \) series, when regression coefficient \( \beta(W) \) has relative simple shape. On the other hand, a large number of \( w_0 \) may be necessary when \( \beta(W) \) has a complex structure. The shape of \( \beta(W) \) can be viewed using “plot” function in the “lplb” R package. We will discuss more on the plot function later in section 3.8.
3.6 Fit Local Partial Likelihood Estimates (lple)

The R function lple_fit() uses the PH model at each point \( w_0 \) to estimate coefficients of our dataset, and use this result to be one of the input parameters to call the function maxTest(). As a result, this function returns the estimated \( \beta(W) \) and the statistic \( Q_1 \) for the dataset we input.

Here we show how the computational speed can be improved. Originally, we need to fit a model at each \( w_0 \) with interaction terms of \( Z_i(W - w_0) \). Combining the estimated coefficients of \( Z_1, \cdots, Z_p \) gives the estimated vector \( \beta(W) \) at \( W = w_0 \). However, we have to fit \( m \) times for every \( w_0 \), where \( m \) is the length of the \( w_0 \) series, each model with a new covariate matrix. To make the program run faster, we use a common covariate matrix \( X_\phi \) as mentioned before, with the interaction terms of the form \( Z_i \cdot W \). Thus we only need to generate the covariate matrix one time. We can get the same \( \beta(W) \) by adding up the estimator of \( Z_i \) and the estimator of \( Z_i \cdot W \) multiplying \( W \). We illustrate this in the following example, by comparing models 1 and 2 at a given point \( w_0 \).

\[
\text{Model 1: } \log \frac{\lambda(t)}{\lambda_0(t)} = \beta_1 Z_1 + \beta_2 Z_2 + \beta_3 Z_1 W + \beta_4 W \\
= (\beta_1 + \beta_3 W) Z_1 + \beta_2 Z_2 + \beta_4 W \tag{3.1}
\]

\[
\text{Model 2: } \log \frac{\lambda(t)}{\lambda_0(t)} = \beta'_1 Z_1 + \beta'_2 Z_2 + \beta'_3 Z_1(W - w_0) + \beta'_4 (W - w_0) \\
= (\beta'_1 - \beta'_3 w_0) Z_1 + \beta'_2 Z_2 + \beta'_3 Z_1 W + \beta'_4 (W - w_0) \tag{3.3}
\]

\[
= (\beta'_1 + \beta'_3 (W - w_0)) Z_1 + \beta'_2 Z_2 + \beta'_4 (W - w_0) \tag{3.5}
\]
Comparing (3.1) with (3.4), \( \beta_1 = (\beta'_1 - \beta'_3 w_0) \), and \( \beta_3 = \beta'_3 \). (We do not care about \( \beta_2 \) and \( \beta_4 \).) Using these result and comparing (3.2) with (3.5), we can see 
\[
(\beta'_1 + \beta'_3 (W - w_0)) = \beta_1 + \beta'_3 w_0 + \beta_3 (W - w_0) = \beta_1 + \beta_3 W,
\]
coordinating with (3.3). Thus we have the same results for \( \beta(W) \) in the two models at \( W = w_0 \). Note here that the estimator \( \hat{\beta} \) in model 1 still depends on \( w_0 \) because the kernel weight function \( K_h(\frac{w_i - w_0}{h}) \) is used for local partial likelihood estimate.

### 3.7 The Bootstrap Procedure

We develop the R function `bstrp()` to complete the final step of bootstrap. We need to resample the dataset to generate new dataset of \( \mathbf{y}^* \) and \( \mathbf{X}^* \), where \( \mathbf{y}^* = (\mathbf{T}^*, \delta^*) \) represents the resampled survival time and censoring indicator, and \( \mathbf{X}^* \) is the same as \( \mathbf{X} \) except sorted by \{\( \hat{\Lambda}_0^*(\tilde{T}_i) \}\}_{i=1}^n \) mentioned in section (2.5). Then we call `lple_fit()` using the new dataset to get \( Q^*_1 \). We do this process for \( B \) times, and the \( p \)-value will be easily acquired by formula (2.29).

To generate \( \mathbf{y}^* \) and \( \mathbf{X}^* \), we use the method in section (2.5). First we randomly choose \( n \) numbers from 1 to \( n \) with replacement and record the \( n \) number as our index vector. According to the index vector, we then re-order the pairs of the status \( \delta_i \) and martingale residual \( \hat{\epsilon}_i \) to generate \{\( \delta_i^*, \hat{\epsilon}_i^* \}\}_{i=1}^n \). Secondly, we calculate the baseline cumulative hazard function \( \hat{\Lambda}_0(\tilde{T}_i) \) as described in section (2.5). We re-order \{\( \hat{\Lambda}_0(\tilde{T}_i) \)\}_{i=1}^n \) from smallest to largest as the new time vector of \{\( \hat{\Lambda}_0^*(\tilde{T}_i) \)\}_{i=1}^n \), and combine them with \{\( \delta_i^* \)\}_{i=1}^n \) as the new dependent variable \( \mathbf{y}^* \). Finally we re-order the rows of covariate matrix \( \mathbf{X} \) by the sorted \{\( \hat{\Lambda}_0^*(\tilde{T}_i) \)\}_{i=1}^n \) to calculate the bootstrap test statistics. This function returns a series of \{\( Q^*_1, Q^*_2, \ldots, Q^*_B \)\}, for \( B \) bootstrap replications.
3.8 R Package Description

All the above functions are implemented in the lplb package. This package includes the basic function of lplb.default(), a parameter function of lplb.control() that allows to input parameters, a print function of print.lplb() to print a summary of the results, and a plot function of lplb.plot() to plot the relationships between each $\beta_i(W)$ against a series of pre-specified $w_0$ points.

**Basic Function of lplb.default()**

Use $lplb(X, y, control)$ to fit a model without formula. $X$ is the matrix with interactions between biomarker ($W$) and the first $p_1$ terms of the dependent variables, where $p_1$ is included in “control” and $p_1 \leq p$. See “lplb.control()” for details.

Use $lplb(y \sim X_1 + X_2 + ... + X_p + W, data = data, control)$ to fit the same model with formula with the following details.

- $y$ is the response of length $n$ for a “Surv” object as in “coxph” function.

- For the formula call of lplb function, $X_i$ is the $i^{th}$ covariate, and $W$ is the biomarker variable. For the non-formula form, $X$ is the data matrix of dimension $(n \times (p+1))$. Note that $W$ **must be the last dependent variable in the formula**.

- $data$ is an optional data frame, list or environment (or object coercible by ‘as.data.frame’ to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula).

- $control$ is a list of parameters for controlling the fitting process. See “lplb.control” for details.
Parameter Input Function lplb.control() .
This is an auxiliary function for lplb fitting. Typically only used internally by “lplb”,
but may be used to construct a control argument to either function, called by the
following syntax:

\[
\text{lplb.control}(h, \text{kernel} = \textquotesingle \text{gaussian}\textquotesingle, B, w_0, p_1, \text{pctl})
\]

.  h: bandwidth of kernel function. The default value is 0.2.

.  kernel: kernel function types, including “gaussian”, “epanechnikov”, “rectangular”, “triangular”, “biweight”, “cosine”, “optcosine”. The default value is “gaussian”. See section 3.3 for details regarding the available kernel functions.

.  B: number of bootstrap samples. The default value is \( B = 200 \).

.  \( w_0 \): the biomarker points (mapped into the interval of (0,1)), at which \( \beta(W) \) is estimated. The default value is \( \text{seq}(0.05, 0.95, \text{by} = 0.025) \).

.  \( p_1 \): the number of dependent variables that have interactions with the biomarker \( W \). The default value is \( p_1 = 1 \).

.  \text{pctl}: a subset of \( w_0 \) (above) for which \( \beta(W) \) is displayed in the output. The default value is \( \text{seq}(0.2, 0.8, \text{by} = 0.1) \).

Print Function .
The R function \text{print()} is implemented here by “\text{print.lplb()}” to provide a short summary of lplb outputs, which must be an “lplb” class object, including coefficients and
standard error at the chosen biomarker points \( w_0 \), the number of bootstrap samples, kernel type, bandwidth, the statistic \( Q_1 \), \( p\)-value, run time, etc. The function can be called by \( \text{print}(fit) \), where \( fit \) is an output object of the function of \( \text{lplb}() \). See examples in the next chapter.

**Plot Function**

The R function \( \text{plot} \) is implemented here by “\( \text{plot.lplb}() \)” to plot the relationships between \( \beta(W) \) and vector \( W \) for each dependent variable having interactions with the biomarker \( W \), from the \( \text{lplb} \) fit model. The function can be called by syntax \( \text{plot}(fit, scale = 't') \), where \( fit \) is an output object of the function \( \text{lplb}() \), and \( scale \) is an optional parameter to choose the scale of biomarker variable, ‘original’ or ‘original’ for the original biomarker scale, ‘transformed’ or ‘t’ for transformed scale that maps biomarker to interval \((0, 1)\). The default is to plot in the original scale. See examples in the next chapter.

### 3.9 Setup the “lplb” R package

We made the “lplb” R package publicly available online at GitHub (URL: https://github.com/statapps/lplb). The source code can be downloaded by clicking the “Clone or download” button. Please see Figure 3.1 for details.
3.9. SETUP THE “LPLB” R PACKAGE

Figure 3.1: “lplb” R package at GitHub website (https://github.com/statapps/lplb)

The users can also install the “lplb” package to their local machine by simply running the following three lines of commands in R Console:

> library(devtools)
> install_github('statapps/lplb')

assuming that the “devtools” package has already been installed. If the “devtools” package was not previously installed, then users have to run

> install.packages('devtools')

before they run “library(devtools)”. 
Chapter 4

Simulation Study and Application

4.1 Overview

In this chapter, we conduct several numerical simulations to evaluate the finite sample properties of the proposed multivariate test of covariate-biomarker interactions. We compare the fitted $\hat{\beta}(W)$ and the true $\beta(W)$.

In section 4.2, we build up a simulation data and conduct the “lplb” package by different parameters. In section 4.3, we further study the empirical power under the alternative hypothesis and empirical test size under the null hypothesis. Finally in section 4.4, we applied the proposed method in a dataset from a prostate cancer trial with a biomarker.

4.2 Example 1: Simulation Data

We generate a dataset with survival time $T = (T_1, T_2, \cdots, T_n)$ for $n = 300$ subjects, and four dependent covariates with the biomarker, $(Z_1, Z_2, Z_3, Z_4, W)$, in which the coefficients of the first two of them are functions of $W$. Here $Z_1, Z_2, Z_3, Z_4$ are
randomly distributed and \( W \) is uniformly distributed on \( (0, 1) \), all of which are independent. Let \( \gamma(\beta; Z, W) = \exp\{3W^3Z_1 - 2\sin(2\pi W)Z_2 + 0.05Z_3 + 0.5Z_4 + 0.2W\} \). If we could find a hazard function \( \lambda(t, \gamma(\beta; Z, W)) \), where \( t \in T \), and \( T \) has a survival distribution, then our model is built up and the true coefficient of \( Z_1 \) and \( Z_2 \) are \( \beta_1(W) = 3W^3 \) and \( \beta_2(W) = -2\sin(2\pi W) \).

Let \( T \) follow a Weibull distribution in this example. Let \( U \) be uniformly distributed on \( (0, 1) \), then \( -\log(1 - U) \) has an exponential distribution with scale = 1, \( -\log(1 - U)q \) has an exponential distribution with scale = \( q \), and \( [-\log(1 - U)]^{1/k}q \) has a Weibull distribution with scale = \( q \) and shape = \( k \). Let \( k = 4 \) and \( q = \gamma(\beta; Z, W)^{-1/k} \), then we can define \( T = (-\log(1 - U)/\gamma(\beta; Z, W))^{0.25} \) and \( T \) follows a Weibull distribution. Thus,

\[
S(t) = P(T > t)
= P((-\log(1 - U)/\gamma(\beta; Z, W))^{0.25} > t)
= P(-\log(1 - U)/\gamma(\beta; Z, W) > t^{0.25})
= P(-\log(1 - U) > t^{0.25}\gamma(\beta; Z, W))
= \exp(-t^{0.25}\gamma(\beta; Z, W))
\]

Because \( S(t) = \exp(-\Lambda(t)) \), where then \( \Lambda(t) \) is the cumulative hazard function, then \( \Lambda(t) = t^{0.25}\gamma(\beta; Z, W) \), and the hazard function is \( \lambda(t) = 4t^{0.25}\gamma(\beta; Z, W) \), which defines our model.

Then we need to set up the censoring time vector \( C = (C_1, C_2, \ldots, C_n) \) for \( n \) subjects. We choose a point \( c \) on \( (0, 1) \) and \( C \) is uniformly distributed on \( [0, c] \). This point \( c \) leads to \( 30\% \sim 40\% \) of the data are censored.

With the new data we built, we can run, print and plot the lplb function by the
following syntax, including the parameters:

\[
\text{fit} = \text{lplb}(y \sim Z_1 + Z_2 + Z_3 + Z_4 + W, \quad \text{data} = \text{data}, \\
w_0 = \text{seq}(0.01, 0.99, by = 0.04), p_1 = 2, B = 200) \quad (4.2)
\]

\[
\text{print}(\text{fit})
\]

\[
\text{plot}(\text{fit})
\]

The results are shown as below. Although the \( p\)-value is 0 in the output, since we bootstrap 200 times in this test, we can only conclude that \( p\)-value \( \leq 0.005 \). So we reject the hypotheses and the coefficients of \( Z_1 \) and \( Z_2 \) vary substantially as functions of the biomarker \( W \). And the plots shows the estimated \( \hat{\beta}_1(W) \) and \( \hat{\beta}_2(W) \).

Figure 4.1: R output of “lplb” (print), with 25 chosen \( w_0 \) points for estimation of \( \hat{\beta}_1(W) \) and \( \hat{\beta}_2(W) \), \( p_1 = 2 \)
4.2. **EXAMPLE 1: SIMULATION DATA**

Figure 4.2: R output of “lplb” (plot) of $\hat{\beta}_1(W)$ and $\hat{\beta}_2(W)$ at 25 chosen $w_0$ points, $p_1 = 2$

If we add the true value $\beta_1 = 3W^3$ and $\beta_2 = -2sin(2\pi W)$ as the blue dotted line on the figures below, we can see they are very similar, and our model predicts well.

Figure 4.3: R output of “lplb” (plot), compared with true values of $\beta_1(W)$ and $\beta_2(W)$, $p_1 = 2$

Note that this function is very computationally intensive. For example, if we choose 25 points of $w_0$, and bootstrap times 200, as in the above example, the program
4.2. EXAMPLE 1: SIMULATION DATA

runs for 2.79 minutes. If we use 100 \( w_0 \) points instead, the program runs for 11.05 minutes, as approximately four times as long as the previous one. Although the plots look a little smoother, they are still quite similar. The results and plots are as follows.

Figure 4.4: R output of “lplb” (print), with 99 chosen \( w_0 \) points for estimation of \( \beta_1(W) \) and \( \beta_2(W) \), \( p_1 = 2 \)

Figure 4.5: R output of “lplb” (plot) of \( \hat{\beta}_1(W) \) and \( \hat{\beta}_2(W) \) at 99 chosen \( w_0 \) points, \( p_1 = 2 \)

As a result, we prefer to set the number of \( w_0 \) points around 25 to be more efficient.
4.3 Example 2: The Power and Test Size of the LPLB Method

To evaluate the power and test size of the proposed test for multivariate covariates-biomarker interactions, we conducted numerical simulations using the following model

\[ \lambda(t) = \lambda_0(t) \exp\{\beta_1(1-W)^2Z_1 + \beta_2 \sin(2\pi W)Z_2 + 4Z_3 + 3Z_4 + 0 \times (Z_5 + \cdots + Z_9) + 0.2 W \}, \]  

(4.3)

where \( W \) has a uniform distribution on \((0,1)\); \( Z_1 \) has a binomial distribution with probability 0.5; \( Z_2, Z_3, Z_4 \) are normally distributed with mean and standard deviation of \((0, 1), (-2, 0.5), (3, 2)\), respectively; \( Z_5, Z_7 \) are exponentially distributed with rate of 55, 37, respectively; and \( Z_6, Z_8, Z_9 \) have Poisson distributions with parameter \( \lambda \) of \(46, 98, 119\), respectively. All the covariates and \( W \) are independent.

The data were fitted using the following three models: Model 1 was from (Liu et al., 2015) with \( p = 2 \), used to fit covariate-biomarker interactions without adjusting for the effects of other covariates:

\[ \begin{align*}
\text{Model 1:} & \quad \lambda_1(t) = \lambda_0(t) \exp\{\beta_1(W)Z_1 + \beta_2(W)Z_2 + g(W)\}. 
\end{align*} \]  

(4.4)

Model 2 was the method proposed in this report with \( p_1 = 2 \) and \( p = 4 \):

\[ \begin{align*}
\text{Model 2:} & \quad \lambda_2(t) = \lambda_0(t) \exp\{\beta_1(W)Z_1 + \beta_2(W)Z_2 + \beta_3 Z_3 + \beta_4 Z_4 + g(W)\}. 
\end{align*} \]  

(4.5)

This method has the flexibility of fitting a model with covariate-biomarker interactions for selected covariates while adjusting the effects of other covariates that do not interact with biomarker.

One may argue that it is possible to use Liu et al. (2015)’s method and fit a
4.3. EXAMPLE 2: THE POWER AND TEST SIZE OF THE LPLB METHOD

model with covariate-biomarker interactions with all 9 covariates. Here we evaluated
the performance of this method by fitting Model 3 to the data:

\[ \lambda_3(t) = \lambda_0(t) \exp\{\beta_1(W)Z_1 + \beta_2(W)Z_2 + \beta_3(W)Z_3 + \cdots + \beta_9(W)Z_9 + g(W)\}. \] (4.6)

All three models were fitted using tools from “lplb” package with \( B = 100 \) boot-
strap and \( R = 200 \) simulation replications under \( H_1 \). We further increased the repli-
cation \( R = 500 \) under the null hypothesis \( H_0 : \beta_1 = \beta_2 = 0 \) to have a more accurate
estimation of the empirical test size. The results were reported in table 4.1.

To investigate whether the proposed LPLB testing procedure returns a correct
\( p \)-value or not under the null hypothesis \( H_0 \), we use the fact that \( p \)-value follows
an uniform \((0,1)\) distribution when \( H_0 \) is true. Therefore, for any significant level
\( 0 < \alpha < 1 \), if \( H_0 \) is rejected when \( p \)-value < \( \alpha \), we have the probability

\[ Pr\{p\text{-value} \leq \alpha\} = \alpha. \]

If the percentage that \( H_0 \) is rejected is around \( \alpha \times 100\% \) when \( \alpha \) is used as the signif-
icant level, we can conclude empirically that the proposed LPLB testing procedure
provide a correct \( p \)-value under \( H_0 \).

In this simulation, we use alpha = 0.05. We recorded the percentage of the \( p \)-value
\( \leq 0.05 \) from the 500 times’ replications under \( H_0 \).
Table 4.1: Empirical test size under the null hypothesis $H_0$ and empirical test power under the alternative hypotheses $H_1$ for three different models. Results were based on $B = 100$ bootstrap and $R = 200$ simulation replications for $\beta \neq 0$ and $R = 500$ replications for $\beta_1 = \beta_2 = 0$.

Under the null hypothesis $H_0$, we observed that all the three models have the empirical test size around the nominal significant level 5%: 0.042, 0.052 and 0.07 for model 1, model 2 and model 3, respectively. This is because the number of rejection follows a Binomial distribution $Bin(R, \alpha)$. For $\alpha = 0.05$ and $R = 500$, we shall observe the percentage of rejections is around the interval $(0.05 - 1.96 \sqrt{0.05 \times 0.95/500}, 0.05 + 1.96 \sqrt{0.05 \times 0.95/500})$ (that is $(0.03, 0.07)$). From the table, we observed approximately 5% of the chances that $H_0$ is rejected when $H_0$ is true. Thus we can say the percentage of $p$-value $\leq 0.05$ is around the nominal significant level, which means the proposed test method controls the type I error well.

Under the alternative hypotheses $H_1$, we found that Model 2 has higher empirical power to detect the covariate-biomarker interactions than Model 1 and Model 3. For example, when $\beta_1 = 0.3$ and $\beta_2 = -0.4$, Model 2 provides an empirical power of 0.67 while Model 1 and Model 3 only have empirical power of 0.10 and 0.13, respectively. Similar results were observed for $\beta_1 = 0.5$ and $\beta_2 = -0.9$ and $\beta_1 = 1.5$ and $\beta_2 = 2.0$, respectively.
4.4. APPLICATION: PROSTATE CANCER DATA

Model 1 is under power because it ignores two covariates $Z_3$ and $Z_4$, which have large effect on the survival outcome. While model 3 is not necessary misspecified, it includes too many interaction terms that are actually have regression coefficients equal to 0. That is why it does not perform as well as model 2.

4.4 Application: Prostate Cancer Data

In this section, we apply our model to the prostate cancer data from the second Veterans Administration Cooperative Urologic Research Group clinical trial (Byar and Corle, 1977). This is a double-blind clinical trial with 506 prostate cancer patients randomly allocated into different treatments. The biomarker is the serum prostatic acid phosphatase ($ap$). It is originally measured in King-Armstrong units and has a range from 1 to 5960, and it was mapped to (0, 1) interval in the analysis. $trt$, representing “treatment”, is the main effect that has interaction with the biomarker $ap$, which has been shown in Liu et al. (2015). $age$, $weight$ and $pf$ are three other covariates, representing “age”, “weight” and “performance status” of the patients. The effects of these three variables are not shown in the previous research.

We fit three models using the following syntaxes:

\[
\begin{align*}
Model 1: & \quad fit = lplb(y \sim trt + ap, p1 = 1) \\
Model 2: & \quad fit = lplb(y \sim trt + age + weight + pf + ap, p1 = 1) \\
Model 3: & \quad fit = lplb(y \sim age + weight + pf + trt + ap, p1 = 3)
\end{align*}
\] (4.7)

Note here that “$ap$” should be placed in the last position of all the dependent variables.

Model 1 is the model of Liu et al. (2015), which only contains the main effect of
treatment and the biomarker \( ap \). The \( p\text{-value} \) in model 1 is 0.032, which certifies the coefficient of \( trt \) varies as a function of the biomarker \( ap \). The estimated treatment effect at different levels of the biomarker \( ap \), mapped into the interval of (0,1), is shown in Figure 4.6.

In model 2, we test the interactions between treatment and the biomarker adjusting for three other variable. The \( p\text{-value} \) in model 2 is 0.012, which is also significant. Moreover, the \( p\text{-value} \) is smaller than the one in model 1. The effect of the biomarker on treatment is more clear after adjusting for other covariates. The estimated treatment effect of model 2 is shown in Figure 4.7. We can see it is very similar to the figure of model 1.
4.4. APPLICATION: PROSTATE CANCER DATA

Figure 4.7: Model 2 estimated treatment effects as functions of biomarker \( ap \) on transformed scale. Estimation is based on 25 chosen \( w_0 \) points.

In model 3, we test whether there are interactions between any of the three covariates, \( age, weight \) and \( pf \), and the biomarker \( ap \), adjusting for \( trt \). As a result, we obtain a large \( p-value \) which equals 0.366. So we reject the null hypothesis and we conclude that there is no interactions between any of the three covariates and the biomarker \( ap \).
Chapter 5

Summary and Future Work

5.1 Summary

In this project, we develop a new method to estimate and test covariates and biomarker interactions using local partial likelihood bootstrap method. The proposed method can adjust for other covariates that do not interact with the biomarker variable. This extends the methods of Liu et al. (2015) that requires all covariates interacting with the biomarker variable.

We further develop an R package to test whether there is a relationship between estimated coefficients of any number of chosen covariates and the biomarker $W$, with the plot and $p$-value as the output. We use lots of vector and matrix calculation in the R code to improve the computation speed. In the R package, users can choose different types of kernel function and bandwidth, the number of interaction terms and bootstrap replication number, as well as the series of points $w_0$ and output key points for the estimation of $\hat{\beta}_1(W)$.

When using this R package, users shall pay attention to the number of $w_0$ we choose, otherwise it might take much time to run the function. And we need to put
the biomarker $W$ at the last column of the covariates and as the last covariate in formula.

We conduct numerical simulation to show that under the null hypothesis, the proposed $lplb$ test procedure provides an empirical test size that is close to the nominal significance level $\alpha = 0.05$. This suggests that the proposed bootstrap test controls the type I error.

Under the alternative hypothesis, our method has higher statistical power to detect the covariates-biomarker interactions compared to the method of Liu et al. (2015). The reason is that when some covariates have interactions with biomarker variable and others do not, the model can not be correctly specified using methods of Liu et al. (2015), while the methods proposed in this project are flexible enough to deal with these different types of model.

## 5.2 Future Work

There are several research directions that can be considered in the future.

1. The power of the proposed test depends on the selection of $w_0$, the bandwidth $h$, and the different type of kernel function. The relationship among these factors will be explored in the future.

2. We can further study model selection methods based on the proposed model.

3. In the proposed method, we treat covariates that do not interact with the biomarker and $g(W)$ as nuisance parameter and do not provide any statistical inference for these parameters. As a future research direction, we will explore on methods to estimate these parameters and provide the corresponding confidence interval.

4. We will also study the estimation of the baseline hazard function $\Lambda_0(t)$ in the
future.

(5) It is possible to write the function in C or Fortran language and called by R to reduce computation time.


Appendix A

LPLB R package Help File
Package ‘lplb’

August 18, 2016

Type Package

Title Local Partial Likelihood Bootstrap (LPLB) test

Version 0.1

Date 2016-07-29

Depends survival

Description Local Partial Likelihood Bootstrap (LPLB) is a set of tools to test multivariate covariates-biomarker interactions for survival data

License GPL-2

LazyData TRUE

Author Siwei Zhang and Bingshu Chen [aut, cre]

Maintainer Siwei Zhang and Bingshu Chen <bingshu.chen@queensu.ca>

R topics documented:

lplb-package .............................................................. 1
control ................................................................. 2
lplb ................................................................. 3
plot.lplb ........................................................... 5
print.lplb ........................................................... 6

Index 8

lplb-package Local Partial Likelihood Bootstrap test

Description

This package fits a multivariable local partial likelihood model for covariate-biomarker interaction with survival data.
Details
"lplb" is a R package for multivariate covariate-biomarker interaction using local partial likelihood method.

Please use the following steps to install 'lplb' package:
1. First, you need to install the 'devtools' package. You can skip this step if you have 'devtools' installed in your R. Invoke R and then type
   
   install.packages("devtools")

2. Load the devtools package.
   
   library(devtools)

3. Install "lplb" package with R command
   
   install_github("statapps/lplb")

"lplb" uses local partial likelihood to estimate covariate-biomarker interactions and bootstrap method to test the significance of the interactions.

Author(s)
Siwei Zhang and Bingshu E. Chen
Maintainer: Bingshu E. Chen <bingshu.chen@queensu.ca>

References

See Also
coxph, bhm, survival

Examples

# fit = lplb(y~biomarker+treatment)

---

control

Auxiliary function for lplb fitting

Description

Auxiliary function for lplb fitting. Typically only used internally by 'lplb', but may be used to construct a control argument to either function.

Usage

# lplb.control(h, kernel = 'gaussian', B, w0, p1, pcti)
Arguments

- **h**: bandwidth of kernel function. The default value is \( h = 0.2 \)
- **kernel**: kernel function types, including "gaussian", "epanechnikov", "rectangular", "triangular", "biweight", "cosine", "optcosine". The default value is 'gaussian'
- **B**: number of bootstrap times. The default value is 200
- **w0**: the estimated points in the interval of (0,1), select arbitrarily. The default value is seq(0.05, 0.95, 0.025)
- **p1**: the number of dependend variables that make interactions with the biomarker \( W \). The default value is 1
- **pctl**: the estimated points that want to be shown in the output. The default value is seq(0.2, 0.8, 0.1)

Details

- Control is used in model fitting of lplb.
- Value
  - This function checks the internal consistency and returns a list of value as inputed to control model fit of lplb.

Author(s)

- Siwei Zhang and Bingshu E. Chen

See Also

- lplb

Examples

```r
## The default control values are: \( h = 0.2 \), kernel = 'gaussian', \( B = 200 \), \( w0 = \text{seq}(0.05, 0.95, 0.025) \)
## To fit the lplb model with some control variables changed,

w0=seq(0.03,0.97,by=2/100)
ctl = lplb.control(w0=w0, h=0.3, p1=2, B=100)

## then fit the following model

#fit = lplb(x, y, control = ctl)
```

lplb

**Local partial likelihood bootstrap (LPLB) method to fit biomarker Models**

Description

{lplb} is a R package for local partial likelihood estimation (LPLE) (Fan et al., 2006) of the coefficients of covariates with interactions of the biomarker \( W \), and hypothesis test of whether the relationships between covariates and \( W \) are significant, by using bootstrap method.
Usage

```r
## Default S3 method:
lplb(x, y, control, ...)
## S3 method for class 'formula'
lplb(formula, data=list(...), control = list(...), ...)
```

# use
# lplb(y ~ X1+X2+...+Xp+w, data=data, control)
# to fit a model with interactions between biomarker (w) with the first p1
# terms of dependent variables.
# p1 is included in 'control'. p1<p. See 'lplb.control' for details
# use
# lplb(x, y, control)
# to fit a model without the formula
# Biomarker w should be the 'LAST' dependend variable

Arguments

- **formula**: an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
- **data**: an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula).
- **x, y**: For `lplb.default`, x is a design matrix of dimension n \( (p+1) \) and y is a vector of observations of length n for a "Surv" object for "coxph".
- **control**: a list of parameters for controlling the fitting process. See 'lplb.control' for details
- **...**: additional arguments to be passed to the low level regression fitting functions

Details

Here 'w' is a Biomarker variable. This variable is required and shall be the last dependent variable in the formula.

'x.cdf' is a function that maps biomarker values to interval (0, 1) using its empirical cumulative distribution function.

Value

`lplb` returns an object of class inheriting from "lplb" which inherits from the class 'coxph'. See later in this section.

The function "print" (i.e., "print.lplb") can be used to obtain or print a summary of the results.

An object of class "lplb" is a list containing at least the following components:

- **beta_w**: a matrix of m * p1, the estimated coefficients at each of the m estimated points, for the first p1 dependent variables with interactions of the biomarker w
The test statistic of the data
mTstar a vector of the test statistics from B times’ bootstrap
pvalue the p-value of the hypothesis that beta_w is a constant

Note
This package was build on code developed by Yicong Liu for simple treatment-biomarker interaction model.

Author(s)
Siwei Zhang and Bingshu E. Chen (bingshu.chen@queensu.ca)

References

See Also
bhm, coxph, lplb.control print.lplb plot.lplb

Examples
## fit1 = lplb(Surv(time, status)~z1 + z2 + w, data = dat, B = B, p1 = 2)
## print(fit1)

plot.lplb 5
Q1 the test statistic of the data
mTstar a vector of the test statistics from B times’ bootstrap
pvalue the p-value of the hypothesis that beta_w is a constant

Note
This package was build on code developed by Yicong Liu for simple treatment-biomarker interaction model.

Author(s)
Siwei Zhang and Bingshu E. Chen (bingshu.chen@queensu.ca)

References

See Also
bhm, coxph, lplb.control print.lplb plot.lplb

Examples
## fit1 = lplb(Surv(time, status)~z1 + z2 + w, data = dat, B = B, p1 = 2)
## print(fit1)

plot.lplb 5
Q1 the test statistic of the data
mTstar a vector of the test statistics from B times’ bootstrap
pvalue the p-value of the hypothesis that beta_w is a constant

Note
This package was build on code developed by Yicong Liu for simple treatment-biomarker interaction model.

Author(s)
Siwei Zhang and Bingshu E. Chen (bingshu.chen@queensu.ca)

References

See Also
bhm, coxph, lplb.control print.lplb plot.lplb

Examples
## fit1 = lplb(Surv(time, status)~z1 + z2 + w, data = dat, B = B, p1 = 2)
## print(fit1)

plot.lplb 5
Q1 the test statistic of the data
mTstar a vector of the test statistics from B times’ bootstrap
pvalue the p-value of the hypothesis that beta_w is a constant

Note
This package was build on code developed by Yicong Liu for simple treatment-biomarker interaction model.

Author(s)
Siwei Zhang and Bingshu E. Chen (bingshu.chen@queensu.ca)

References

See Also
bhm, coxph, lplb.control print.lplb plot.lplb

Examples
## fit1 = lplb(Surv(time, status)~z1 + z2 + w, data = dat, B = B, p1 = 2)
## print(fit1)

plot.lplb 5
Q1 the test statistic of the data
mTstar a vector of the test statistics from B times’ bootstrap
pvalue the p-value of the hypothesis that beta_w is a constant

Note
This package was build on code developed by Yicong Liu for simple treatment-biomarker interaction model.

Author(s)
Siwei Zhang and Bingshu E. Chen (bingshu.chen@queensu.ca)

References

See Also
bhm, coxph, lplb.control print.lplb plot.lplb

Examples
## fit1 = lplb(Surv(time, status)~z1 + z2 + w, data = dat, B = B, p1 = 2)
## print(fit1)

plot.lplb 5
Q1 the test statistic of the data
mTstar a vector of the test statistics from B times’ bootstrap
pvalue the p-value of the hypothesis that beta_w is a constant

Note
This package was build on code developed by Yicong Liu for simple treatment-biomarker interaction model.

Author(s)
Siwei Zhang and Bingshu E. Chen (bingshu.chen@queensu.ca)

References

See Also
bhm, coxph, lplb.control print.lplb plot.lplb

Examples
## fit1 = lplb(Surv(time, status)~z1 + z2 + w, data = dat, B = B, p1 = 2)
## print(fit1)
Details
plot.lplb is called to plot the relationships between beta_w and w_est for each dependent variable with interactions with the biomarker w, from the lplb fit model.
The number of interaction terms can be set in lplb.control.
The default method, print.default has its own help page. Use methods("print") to get all the methods for the print generic.

Author(s)
Bingshu E. Chen and Siwei Zhang

See Also
lplb, lplb.control, print.lplb

Examples

```r
# plot.lplb(x)
```

print.lplb

print a lplb object

Description
print are used to provide a short summary of lplb outputs.

Usage

```r
## S3 method for class 'lplb'
print(x, ...)
```

Arguments

- `x` a lplb class returned from lplb fit
- `...` other options used in print()

Details
print.lplb is called to print object or summary of object from the lplb model lplb.
The default method, print.default has its own help page. Use methods("print") to get all the methods for the print generic.

Author(s)
Siwei Zhand and Bingshu E. Chen

See Also
The default method for print print.default, lplb
print.lplb

Examples

#
# print(fit)
#
Appendix B

LPLB R code
R script: lplb.R

#######clean worspace and set file path
#rm(list=ls())
#getwd()

########Load library and functions
library(survival)
library(MASS)

#source('lplb_basicFunctions.R')

lplb <- function(x, ...) UseMethod("lplb")

lplb.default <- function(x, y, control, ...)
{
  t0 = Sys.time()
  X = as.matrix(x)
  ## data needs to be ordered by time
  st = sort(y[, 1], index.return = TRUE)
  idx = st$ix
  y = y[idx, ]
  X = X[idx, ]
  ## transform w into interval (0,1)
  p=ncol(X)-1
  w = X[, p+1]
  X[,p+1]=x.cdf(X[,p+1])
  X = as.matrix(X)
  fit = lple_fit(X, y, control)
  Q1 = fit$maxT
  sd = fit$sd
  cat('Q1 = ', Q1, '
      sd = ', sd, '
      ',
  fit$mTstar = bstrp(X, y, control)
  B = control$B
  pvalue = (sum(fit$mTstar>=Q1)+0.5)/(B+1)
  cat('p-value = ', pvalue, '

  t1 = Sys.time()
  runningtime=t1-t0
  fit$w = w
  fit$Q1 = Q1
  fit$B = B
  fit$pvalue = pvalue
  fit$control = control
  fit$call <- match.call()
  fit$kernel = control$kernel
  fit$h = control$h
fit$runningtime = runningtime
class(fit) <- "lplb"

return(fit)
}
lplb.control = function(h = 0.2, kernel = 'gaussian', B = 200, w0 = seq(0.05, 0.95, 0.025), p1 = 1, pctl = seq(0.2, 0.8, 0.1)) {
  if (!is.numeric(B) || B <= 0)
    stop("value of 'B' must be > 0")
  if (!is.numeric(h) || h <= 0 || h >= 1)
    stop("value of 'h' must be in (0, 1)"
  if (!is.numeric(p1) || p1 < 1)
    stop("value of 'p1' must be > or = 1")
  if (!is.numeric(pctl) || max(pctl) >= 1 || min(pctl)<=0)
    stop("value of 'pctl' must be in (0, 1)"
  return(list(h = h, B = B, w_est = w0, p1 = p1, pctl = pctl, kernel = kernel))
}

lplb.formula <- function(formula, data=list(...), control = list(...), ...)
{
  mf <- model.frame(formula=formula, data=data)
  x <- model.matrix(attr(mf, "terms"), data=mf)
  #remove intercept
  x = x[, -1]
  y <- model.response(mf)
  control = do.call("lplb.control", control)
  fit <- lplb.default(x, y, control)
  fit$call <- match.call()
  fit$formula <- formula
  return(fit)
}

print.lplb <- function(x, ...)
{
  cat("Call:\n")
  print(x$call)
  p1 = ncol(x$beta_w)
  control = x$control
c_names = as.character(rep(0,p1))  # to save col names
w_q = quantile(x$w_est, probs = control$pctl, type=3)

opmtrx = matrix(0,p1*2, length(control$pctl))
for (i in 1:p1){
  opmtrx[(i*2-1),] = x$beta_w[match(w_q,x$w_est),i]
  c_names[(i*2-1)] = colnames(x$beta_w)[i]
  opmtrx[(i*2),] = x$sd[match(w_q,x$w_est),i]
  c_names[(i*2)] = paste0(colnames(x$beta_w)[i],'(sd)')
}

opmtrx = rbind(w_q, opmtrx)
rownames(opmtrx) = c('w0',c_names)
cat("\nCoefficients(w_est quantile):\n")
print(opmtrx)
print(x$runningtime)
cat('p1 =', p1, '; Bootstrap times =', x$B, 'n')
cat('Kernel type:',x$kernel, '; Bandwidth (h) = ',x$h, 'n')
cat('Statistic Q1 =', x$Q1, '; p_value =', x$pvalue, 'n')
}

## plot function.
plot.lplb = function(x, scale = c('original', 'transformed'), ...) {
  scale = match.arg(scale)
  bw = x$beta_w
  p1 = ncol(bw)
  control = x$control
  w_q = quantile(x$w, probs = x$w_est)
  w = switch(scale, original = w_q, transformed = x$w_est)

  if(p1 == 1) {
    par(mfrow = c(1, 1))
    plot(w, bw, xlab = 'w', ylab = 'beta(w)', main = 'x1', type = 'l')
  } else if (p1 == 2) {
    par(mfrow = c(1, 2))
  } else if ((p1 <= 4) & (p1 > 2)) {
    par(mfrow = c(2, 2))
  } else if ((p1 <= 6) & (p1 > 4)) {
    par(mfrow = c(2, 3))
  } else {
    p2 = ceiling(sqrt(p1))
    par(mfrow = c(p2, p2))
  }

  if(p1 > 1) {
    bw_names = colnames(bw)  # keep the col names
    for (i in 1:p1) {
      plot(w, bw[, i], xlab = 'w', ylab = 'beta(w)', main = bw_names[i], type = 'l')
    }
  }
}
R script: lplb_basicFunctions.R

### 01. basic functions

## 01_1) transform w into interval (0,1)
x.cdf = function(x) {
  n = length(x)
  p = rep(0, n)
  for (i in 1:n)
    p[i] = sum(x<=x[i])
  p = p/(n+1)
  return(p)
}

## 01_2) Kernel function
K_func<-
function(w, u, h, kernel = c("gaussian", "epanechnikov", "rectangular", "triangular",
"biweight", "cosine", "optcosine")){
  kernel = match.arg(kernel)
  x = w-u
  ax = abs(x)
  esp = 1e-40
  kh = switch(kernel, gaussian = dnorm(x, sd = h/2), # will set the default for guassian
    rectangular = ifelse(ax < h, 0.5/h, esp),
    triangular = ifelse(ax < h, (1 - ax/h)/h, esp),
    epanechnikov = ifelse(ax < h, 3/4 * (1 - (ax/h)^2)/h, esp),
    biweight = ifelse(ax < h, 15/16 * (1 - (ax/h)^2)^2/h, esp),
    cosine = ifelse(ax < h, (1 + cos(pi * x/h))/(2*h), esp),
    optcosine = ifelse(ax < h, pi/4 * cos(pi * x/(2*h))/h, esp)
  )
  return(kh)
}

## 01_3) transform matrix X into new matrix with p1 interaction (with w) terms
interaction_X=function(X, p1){
  p = ncol(X)-1
  n = nrow(X)
  X2 = matrix(0, n, p+p1+1)
  w = X[, p+1]
  X1 = X[, 1:p]
  # Assign value to X2
  X2[, 1:p] = X[, 1:p]
  X2[, (p+1):(p+p1)] = diag(w)%*%X1
  X2[, p+p1+1] = w
  return(X2)
}
### 01) Transform matrix X into new matrix with p1 interaction with (w-w0) terms

```r
interaction_X_w0=function(X, p1, w0){
  p=ncol(X)-1
  n=nrow(X)
  ## build interaction terms with (w-w0) in X
  X2 = matrix(0, n, p+p1+1)
  w = X[,p+1]
  X1 =X[,1:p1]
  X2[,1:p] = X[,1:p]
  X2[, (p+1):(p+p1)] = diag(w-w0)%*%X1 # (n*p1)
  X2[, p+p1+1] = w
  return(X2)
}
```

### 02. Calculate S(k)_theta and S(k)_fai

# Since St0 to St2 do not change w0, so do I_theta and pi_theta, we can separate Sk2 into two parts: Sk2t and Sk2f

#### 02_1 S(k)_theta and I_theta, pi_theta

```r
Sk2t = function(X, y, theta) {
  status = y[, 2] # y[,1] is time; y[,2] is status
  p_th = ncol(X)
  n = nrow(X)
  Gx = t(X)
  ## Sm is a upper triangular matrix of 1
  Sm = matrix(1, n, n)
  Sm[lower.tri(Sm)] = 0
  ezb_th = as.vector(exp(t(Gx)%*%theta)) # (n*1)
  St0 = Sm%*%ezb_th # (n*1)
  St1 = Sm%*%(t(Gx)*ezb_th) # (n*p_th)
  GSt1 = t(Gx) - St1/c(St0) # (n*p_th)
  # Gi - S1/S0, we also need this to calculate pi_cov
  # Gi_2= apply(GSt1, 1, function(x){return(x%*%t(x))}) # ((p_th*p_th)*n)
  GSt2 = aperm(array(GSt1_2, c(p_th, p_th, n)), c(3, 1, 2))
  # ap perm changes a p1*p2*p3 array to a p3*p1*p2 array with c(3, 1, 2)
  # GSt2 is a n*p*p array with ith pxp matrix = (Gi-S1i/S0i)%*%t(Gi-S1i/S0i)
  pi_theta = colSums(status*GSt2, dims = 1)
  Zt_th = apply(t(Gx), 1, function(x){return(x%*%t(x))})
  Z2_th = array(Zt_th, c(p_th, p_th, n))
  # multiply each Z2(p, p, i) with ezb[i], by change ezb to a p*p*n array with each of i th pxp matrix = ezb[i]
  Z2ezb_th = Z2_th * array(rep(ezb_th, each = p_th*p_th), c(p_th, p_th, n))
  ## calculate S2, a n*p*p array
  St2 = apply(Z2ezb_th, c(1, 2), function(x, y){return(y%*%x)}, Sm)
  # ap perm changes a p1*p2*p3 array to a p3*p1*p2 array with c(3, 1, 2)
```

St1_2 = aperm(array(apply(St1,1,function(x){return(x%*%t(x))}),c(p_th,p_th,n)),c(3,1,2))
I_theta = colSums(status*(St2/c(St0)-St1_2/c(St0)^2), dims = 1)
# need GST1= Gi-S1i/S0i for pi_cov
return(list(I_theta = I_theta, pi_theta=pi_theta, GSt1=GSt1))

## 02_2 S(k)_fai and I_fai, pi_fai, pi_cov
Sk2f = function(X, y, control, bw0, w0, GSt1) {
  kernel = control$kernel
  ## NOTE!!!!!: input X should be ordered by time already!!!!!!!!!!!!!
  ## setup parameters
  status = y[, 2] #y[,1] is time; y[,2] is status
  h = control$h
  w_est = control$w_est
  p1 = control$p1
  X_R = interaction_X_w0(X,p1,w0)
  p = ncol(X)-1
  p_th = ncol(X)
  p_fai = p+p1+1
  n = nrow(X)
  w = X[,ncol(X)]
  xi = as.matrix(bw0) # class: numeric (a row/col of a matrix)
  R = t(X_R) # (p_fai * n); depend on w0
  H = diag(rep(c(1,h,h),c(p,p1,1)),p_fai,p_fai) # (p_fai * p_fai)
  fai = H%*%xi # (p_fai * 1)
  Tx = solve(H)%*%R # (p_fai * n); depend on w0
  # Tx = (T1, T2, ..., Tn), each Ti is a column vector of (p_fai*1)
  ## Sm is a upper triangular matrix of 1
  Sm = matrix(1, n, n)
  Sm[lower.tri(Sm)] = 0
  ## setup Kernel weight vector: kh
  kh=K_func(w, w0, h, kernel)
  ezb_fai = as.vector(exp(t(Tx)%*%fai)) * kh # (n*1)
  Sf0 = Sm%*%ezb_fai # vector: (n*1)
  Sf1 = Sm%*%(t(Tx)*ezb_fai) # matrix: (n*p_fai)
  TSf1 = t(Tx)-Sf1/c(Sf0) # (n*p_fai)
  TSf1_2= apply(TSf1, 1, function(x){return(x%*%t(x))}) # ((p_fai*p_fai)*n)
  TSf2 = aperm(array(TSf1_2, c(p_fai, p_fai, n)), c(3, 1, 2))
  # TSf2 is a n*p*p array with ith pxp matrix = (Ti-S1i/S0i)%*%(Ti-S1i/S0i)
  pi_fai = colSums(status*(TSf2*kh^2), dims = 1)
  Zt_fai = apply(t(Tx), 1, function(x){return(x%*%t(x))})
  Z2_fai = array(Zt_fai, c(p_fai, p_fai, n))
Z2ezb_fai = Z2_fai * array(rep(ezb_fai, each = p_fai*p_fai), c(p_fai, p_fai, n))
## calculate Sf2, a n*p*p array
Sf2 = apply(Z2ezb_fai, c(1, 2), function(x, y){return(y%*%x)}, Sm)
Sf1_2 = aperm(array(apply(Sf1,1,function(x){return(x%*%t(x))}),c(p_fai,p_fai,n)),c(3,1,2))
I_fai = colSums(status*(kh*(Sf2/c(Sf0)-Sf1_2/c(Sf0)^2)), dims = 1)

## calculate pi_cov
# Note: TSf1 = t(Tx)-Sf1/c(Sf0) # (n*p_fai)
# Note: GST1 = t(Gx)-St1/c(St0) # (n*p_th)
pi_cov = t(TSf1)%*%diag(status*kh)%*%GST1 #p_fai*p_th
return(list(I_fai = I_fai, pi_fai=pi_fai, pi_cov=pi_cov))

### 03. Calculate maximum of Q1 (for 1:m), m is the length of w_est
maxTest = function(X,y,control,theta, betaw){
## NOTE: input X, status should be ordered by time already
h = control$h
w_est = control$w_est
m = length(w_est)
p1 = control$p1
n=nrow(X)
p=ncol(X)-1
w=X[,p+1]

## build matrix of beta(w)-beta0, col number: p1
beta_hat = matrix(rep(theta[1:p1],time=m),nrow=m,ncol=p1,byrow=T)
betaw_hat = betaw[, 1:p1]
diff_est = betaw_hat

## build var(beta(w)-beta)
## I_theta, pi_theta, sigma do not change with w0, so do St0, St1, St2, which should be outside the loop m
skt=Sk2t(X, y, theta)
l_theta = skt$I_theta
pi_theta = skt$pi_theta
GST1 = skt$GST1 # for calculating pi_cov in Sk2f()
sigma=solve(l_theta)%*%pi_theta%*%solve(l_theta)
## constuct return matrixs: sigma11
mtrx1=matrix(0,nrow=p1,ncol=p+1)
for (i in 1:p1) mtrx1[i,i]=1
sigma11=mtrx1%*%sigma%*%t(mtrx1)

mtrx2=matrix(0,nrow=p1,ncol=p+p+1)
for (i in 1:p1) mtrx2[i,i]=1
# for later use in loop to constrect gamma11 and omiga11

## initial
Q1 = rep(0,time=m)
sd.err = matrix(0,nrow=m,ncol=p1) # to save var, then calculate std.err
## loop on w_est
for (i in 1:m){
  w0 = w_est[i]
  bw0 = betaw[i, ]
  skf=Sk2f(X, y, control, bw0, w0, GSt1)
  l_fai = skf$l_fai
  pi_fai = skf$pi_fai
  pi_cov = skf$pi_cov
  gamma=solve(l_fai)%*%pi_fai%*%solve(l_fai)
  omiga=solve(l_fai)%*%pi_cov%*%solve(l_theta)
  ## constuct return matrixs: gamma11, omiga11
  gamma11=mtrx2%*%gamma%*%t(mtrx2)
  omiga11=mtrx2%*%omiga%*%t(mtrx1)
  ## save diag(gamma11) to calculate sd.err of beta_w
  sd.err[i, ] = sqrt(diag(gamma11))
  var_hat = sigma11 + gamma11 - 2*omiga11
  ## calculate Q1 at w0=w_est[i]
  diff_hat = diff_est[i, ] # class: numeric
  Q1[i] = as.numeric( t(diff_hat) %*% solve(var_hat) %*% as.matrix(diff_hat))
}
maxQ1 = max(abs(Q1))
return(list(maxQ1=maxQ1,sd.err=sd.err))
}

### 04. estimate beta(w) (Under H1) and theta (Under H0)
lple_fit = function(X, y, control) {
  h = control$h
  kernel = control$kernel
  w_est = control$w_est
  x_names = colnames(X)
  p1 = control$p1 # number of interaction terms between Zi and w
  p = length(X[1, ])-1 # number of Zi's
  n = length(X[ ,1]) # number of observations
  m = length(w_est) # number of estimate points; choose from interval (0,1) arbitrarily
  X = as.matrix(X) # coxph() need X to be class matrix

  ## X_fai is the data of the interaction model (H1), estimator is beta(w)
  X_fai=interaction_X(X,p1)
  ## X is the data of the no-interaction model (H0), estimator is theta
  fitH0=coxph(y ~ X)
  theta=fitH0$coef

  ## estimate beta(w)
## set matrix "betaw" to save the coef of Z1 to Zp, interaction terms (zi with (w-w0)) and (w-w0) in every row, each row for a different w0. dim: (m* (p+p1+1))

\[
\text{betaw} = \text{matrix}(0, \text{nrow}=m, \text{ncol}=p+p1+1)
\]

## set matrix "beta_w" to save only the z1 to zp1, by which we can plot the relationship of beta(w) vs. w_est. dim: (m*p1)

\[
\text{beta_w} = \text{matrix}(0, m, p1)
\]

for (i in 1:m) {
  \[
  \text{wg} = \text{K_func}(\text{X_fai}[,p+p1+1], \text{w_est}[i], h, \text{kernel})
  \]
  \[
  \text{fit} = \text{coxph}(y \sim X_{fai}, \text{subset}= (\text{wg}>0), \text{weights}=\text{wg})
  \]
  \[
  \text{betaw}[i, ] = \text{fit}\$\text{coef}
  \]
}

\[
\text{beta_w}[, 1:p1] = \text{betaw}[,(1:p1)]+\text{betaw}[,(p+1):(p+p1)] \times \text{w_est}
\]

\[
\text{betaw}[, 1:p1]= \text{beta_w}
\]

## return value

\[
\text{maxreturn} = \text{maxTest}(X, y, \text{control}, \text{theta}, \text{betaw})
\]

\[
\text{maxT} = \text{maxreturn$\text{maxQ1}}
\]

\[
\text{sd} = \text{maxreturn$\text{sd.err}}
\]

\[
\text{colnames(\text{beta_w}) = x_names[1:p1]}
\]

\[
\text{fit} = \text{list(\text{w_est = w_est, beta_w = beta_w, betaw = betaw, maxT = maxT, sd=sd}})
\]

\[
\text{return(fit)}
\]

### 05. bootstrap

\[
\text{bstrp = function(X, y, control)}{
\]
  \[
  X = \text{as.matrix(X)}
  \]
  \[
  \text{kernel = control$\text{kernel}}
  \]
  \[
  h = \text{control$h}
  \]
  \[
  p1 = \text{control$p1}
  \]
  \[
  B = \text{control$B}
  \]
  \[
  \text{## Fit model under H0}
  \]
  \[
  \text{fitH0=coxph(y \sim X)}
  \]
  \[
  \text{exb} = \exp(X\%\%\text{fitH0$\text{coef}} \text{#to be used for residual bootstrap}
  \]

\[
X_{fai} = \text{interaction}_X(X, p1)
\]

\[
w = X[, \text{ncol(X)}]
\]

\[
n=nrow(X)
\]

\[
\text{status = y[, 2]} \text{#y[, 1] is time; y[, 2]is status, to be the initial value}
\]

## build Martingale residuals

\[
\text{resid_1 = rep(0, n)}
\]

for (i in 1:n) {
  \[
  \text{wg = K_func(w, w[i], h, kernel)} \text{#do not use w_est here, but use w[i], length(w) = n}
  \]
  \[
  \text{fit = coxph(y \sim X_{fai}, \text{subset}= (\text{wg}>0), \text{weights}=\text{wg})}
  \]
  \[
  \text{resid_1[i]<-residuals(fit, "martingale")}[i]
  \]
}
# Bootstraping
mTstar = rep(0, B)
i = 1
cat('Bootstraping')
Bn = floor(B/10) + 1
while (i <= B) {
sample_index = sample(n, size=n,replace=T)
status_sample = status[sample_index]
resid_sample = status_sample - resid_1[sample_index]
# status:n*1, resid_1:n*1
resid_sample2 = resid_sample/exb
time_order = order(resid_sample2)
status_star = status_sample[time_order]
X_star = X[time_order, ]
time_star = 1:n
y_star = Surv(time_star, status_star)
g = try(ileave_fit(X_star, y_star, control))
# use another set of bootstrap if coxph does not converge within 100 iter
if(class(g) == "try-error") next
# next halts the processing of the current iteration and advances the looping index.
mTstar[i] = g$maxT
#cat('i = ', i, mTstar[i], 'n') # for inspection
#cat(' ', i, ',', sep = "") # for simplified inspection
if((i %% Bn)==0) cat('.') # %% indicates x mod y
i = i + 1
}
cat('DONE\n')
return(mTstar)