ESTIMATING FAILURE PROBABILITIES AND TESTING FOR TREATMENT EFFECTS IN THE PRESENCE OF COMPETING RISKS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the Graduate

School of The Ohio State University

By

Kevin P. Tordoff, M.S.

* * * * *

The Ohio State University
2007

Dissertation Committee:

Professor Melvin L. Moeschberger, Advisor

Professor Mei-Ling Ting Lee

Professor Zhangsheng Yu

Professor Janet M. Box-Steffensmeier

Approved by

_____

Advisor
College of Public Health

ABSTRACT

In the analysis of survival data, one may encounter the problem of competing risks where each subject may fail due to one of K causes, referred to as competing risks. In the competing risk problem, an experimental unit is observed until a particular event occurs in the presence of several events. The occurrence of any one event sometimes precludes us from observing the other events of interest. The cumulative incidence function, $C_i(t)$, gives the probability of experiencing the i^{th} competing cause of failure prior to time t and before any of the other competing causes of failure. Estimating the cumulative incidence function is of primary interest in most clinical studies. Therefore, we compare several methods of estimating the cumulative incidence function using nonparametric and semi-parametric methods described in the literature, as well as those implemented in popular software packages.

Often we are also interested in comparing the risk of failure from a particular cause over two or more treatment groups. Some authors have recently pointed out that under certain circumstances, when testing for a treatment effect in the presence of competing risks, the popular cumulative incidence based approach may be problematic. Four methods commonly used when testing for a treatment effect in the presence of competing risks are: (1) the test based on Cox's proportional hazards

model, (2) the log-rank test, (3) Gray's test, and (4) a test due to Fine and Gray. We investigate the level of significance and power associated with these four methods of testing for failure specific treatment effects in the presence of competing risks via a simulation study.

Dedicated to my family and friends

ACKNOWLEDGMENTS

I would like to extend a special thank you to Dr. Melvin L. Moeschberger for agreeing to be my mentor. His guidance and support over the past three years have been invaluable.

I would also like to thank Dr. Mei-Ling Lee, Dr. Zhangsheng Yu, and Dr. Janet M. Box-Steffensmeier for their insight and thoughtful suggestions throughout the completion of this manuscript. Many thanks also go to Leslie Davidson and Judy Dawson for always taking the time to help.

To my parents and sister, I thank you for your patience, your encouragement, and your unwavering conviction. I also thank you for your wit and your help in maintaining perspective throughout.

VITA

May 02, 1977……Born – Wheeling, WV

1999……..B.S. Mathematics Education,
Clarion University of Pennsylvania

2001……..M.S. Statistics,
West Virginia University

2003……..M.S. Statistics,
The Ohio State University

2003-present…….Graduate Teaching and Research Associate,
College of Public Health,
The Ohio State University

PUBLICATIONS

Research Publication

1. Melvin L. Moeschberger, Kevin P. Tordoff and Nidhi Kochar. "A Review of Statistical Analyses for Competing Risks" in C.R. Rao, J.P. Miller, and D.C. Rao, eds., Handbook of Statistics, Elsevier, Amsterdam, **27**, 321-341, (2008).

FIELDS OF STUDY

Major Field: Public Health

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 COMPETING RISKS

In the analysis of survival data, one may encounter the problem of competing risks. These situations are often encountered in clinical research settings. Here each subject may fail due to one of K $(K \geq 2)$ causes, referred to as the competing risks. In the competing risk problem, an individual or an experimental unit (referred to in this discussion as the subject) is observed until a particular event occurs in the presence of several events. The occurrence of one of these events precludes us from observing any of the other events of interest. The key departure from standard survival analysis is that these event times may be dependent upon each other or several of the same risk factors, thereby violating the typical assumption of independence. Therefore another application of competing risks analysis would be situations involving typical univariate survival data in the presence of right censoring where the censoring mechanism is not independent of the failure time. A variety of disciplines may encounter problems involving competing risks. These include cancer research, reliability of physical equipment, economics, social sciences, insurance assessments, and outcomes of preventive behaviors, amongst many other areas.

Assessing the importance of these risks over a period of time has long been a concern of biostatisticians, demographers, vital statisticians, and actuaries.

The problem of assessing the effect of what the mortality pattern of a population would be if smallpox could be eradicated dates back to Daniel Bernoulli (1700 – 1782), one of the great scientists of the 18th century. Just a few years before Jenner inoculated an 8 year old boy named James Phipps with cow pox; Bernoulli wrote a mathematical analysis of the problem and also encouraged the universal inoculation against small pox. His analysis was first presented at the Royal Academy of Sciences in Paris during 1760, and later published in Bernoulli (1766).

As noted earlier, the subject under consideration is often exposed to several risks, but the eventual failure of the subject is attributed to only one of the risks, usually referred to as the *cause of failure*. The available data for competing risks is in the form of time until event occurrence where T is the time from some suitable starting point until some cause of failure for each individual who fails, and δ_i is an indicator variable equal to 1 if failure is due to the i^{th} cause, 0 otherwise. In the latent failure time approach, one assumes that there are K potential failure times, $X_1, X_2, ..., X_K$, associated with each risk. T is then the $\min(X_1, ..., X_K)$ and δ_i is an indicator variable equal to 1 if failure is due to the ith cause, 0 otherwise. References for the latent failure time model may be found in David and Moeschberger (1978) or in Klein and Moeschberger (2003).

Gichangi and Vach (2005) present a guided tour to the analysis of competing risk data. They point out that "there is still a great deal of uncertainty in the medical research and biostatistical community about how to approach this type of data."

In their opinion, there are three main reasons for this uncertainty:

1) "The analysis of competing risks data does often not allow one to answer all research questions of interest, as we are unable to analyze associations or relationships among different risks. This general research methodological problem is often confused with a lack of adequate statistical methodology."

2) "Some standard methods of survival analysis like the log rank test and the Cox Model can be used in analyzing competing risks while other standard methods, especially the Kaplan-Meier estimator, have limited use. This is very confusing from a pedagogocal point of view."

3) "Most papers on competing risks are written on a rather formal mathematical level. This is a natural consequence of the fact that the basic problem in analyzing competing risks is the proper definition of the quantities of interest. However, the formal mathematical level makes many papers difficult to understand for many researchers."

This paper is certainly worth a close reading since the authors seem to have an appreciation for the difficulty and practical understanding of the issues inherent in sorting out this very real problem.

When modeling competing risks data, we are often required to make assumptions about the dependence structure among the potential causes of failure. However when we are only able to observe the failure time and cause of failure, and

not the potential failure times, these assumptions are not testable. This is what's known as the *identifiability dilemma*.

The direction of the statistical analyses of competing risks studies changed dramatically after the identifiability problem for marginal distributions of the latent failure time model was pointed out by Tsiatis (1975), Prentice et al. (1978), and many others. At that time, interest centered on estimating identifiable competing risk probabilities. Such probabilities are represented by the crude (or cause-specific) hazard rate for cause i in the latent failure time model by

$$h_i(t) = \lim_{\Delta t \to 0} \frac{P(t \le X_i < t + \Delta t \mid T \ge t)}{\Delta t},$$

which is the conditional rate of occurrence for the i^{th} cause of failure in the presence of all possible causes of failure, or by the cumulative incidence function

$$C_i(t) = P(T \le t, \delta_i = 1) = \int_0^t h_i(u) \exp\left\{ -\int_0^u \sum_{i=1}^{K} h_i(v) dv \right\} du .$$

Here $h_i(t)$ tells us the rate at which subjects who have yet to experience any of the competing risks are experiencing the i^{th} competing cause of failure. The cumulative incidence function, $C_i(t)$, gives the probability of both experiencing the i^{th} competing cause of failure prior to time t and that it occurs before any of the other competing causes of failure. Note that the value of $C_i(t)$ depends not only on the rate at which the i^{th} competing cause occurs, but it also depends on the rate at which all the other competing risks occur. Also note that the cumulative incidence function is not a true distribution function since $C_i(\infty) = P(\delta_i = 1)$, which is not necessarily equal

to one. It has the property that it is non-decreasing with $C_i(0) = 0$ and $C_i(t) < 1$. Such a function is often called a "sub-distribution" function.

1.2 CUMULATIVE INCIDENCE

Estimating the cumulative incidence function is of primary interest in most clinical studies. The Kaplan-Meier method can be used to obtain a nonparametric estimate of the cumulative incidence when the data consists of subjects who experience an event and the censoring mechanism is assumed to be non-informative (a special case of which is if the time at which a subject experiences an event is assumed to be independent of a mechanism that would cause the patient to be censored).

Refinements and extensions of the cumulative incidence approach have been considered by various authors. We present a brief discussion of these methods.

Several authors have discussed the topic of competing risks and the estimation of the cumulative incidence of an event. The theoretical concepts underlying the estimation of the cumulative incidence of an event using a variety of models is reviewed by Gail (1975) and Benichou and Gail (1990). Prentice and colleagues (1978) have discussed the likelihood inference approach to examine the effect of prognostic factors on the event of interest in the presence of competing risk events. A variety of probability models for summarizing competing risk data are described by Pepe and Mori (1993). A method to estimate the cumulative incidence of a specific event based on an extension of Cox proportional hazards regression model has been developed by Tai et al. (2001). Their findings suggest that the estimates obtained

using the Kaplan-Meier approach are numerically larger than those accounting for competing risk events.

An alternative approach that accounts for informative censoring is given in the paper by Satagopan and colleagues (2004). They discuss cumulative incidence estimation in the presence of competing risk events. This approach is based on work done by Kalbfleisch and Prentice (2002) and Marubini and Valsecchi (1995). They outline a two-step process in which the first step involves calculating the Kaplan-Meier estimate of the overall survival from any event (where both the event of interest and the competing risk event are considered 'events'), while the second step involves calculating the conditional probabilities of experiencing the event of interest. The cumulative incidences are then estimated using these probabilities.

Gooley et al. (1999) discuss the appropriateness of the cumulative incidence (CI) function in competing risks analysis as opposed to routinely using the Kaplan-Meier (KM) estimator. The authors point out that 1-KM is a function of the hazard for failures due to the cause of interest only and do not depend on the hazard for failures due to the competing risks. Therefore 1-KM is not interpretable as an estimate of the probability of failure due to the cause of interest when competing risks are present. However cumulative incidence is a function of both hazards. The result being that it is interpretable as an estimate for the probability of failure due to the cause of interest when competing risks are present (or absent). The authors provide a representation of both the cumulative incidence and Kaplan-Meier functions utilizing the concept of censored observations being 'redistributed to the right' and show that 1-KM is a biased estimator for the probability of failure due to the cause of interest,

with 1-KM ≥ CI. The cumulative incidence function eliminates all patients who experience the competing risk as their first event from the risk set associated with the event of interest. However, the Kaplan-Meier estimator treats patients who experience the competing risk prior to the event of interest as censored. Therefore they are 'redistributed to the right' which incorrectly assumes that the cause of interest is still possible beyond the time at which the censoring occurred. The authors claim that these interpretations "allow a more intuitive understanding of each estimate and therefore an appreciation of why the Kaplan-Meier method is inappropriate for estimation purposes in the presence of competing risks, while the cumulative incidence estimate is appropriate."

There are several software packages available that estimate the cumulative incidence functions for an event of interest. Some of these include R (The R Foundation; http://www.r-project.org/), S-plus (Insightful Corp.; http://www.insightful.com), Stata (StataCorp LP; http://www.stata.com), and SAS (SAS Institute Inc.; http://www.sas.com). A popular software technique that has experienced widespread usage has been made available in R by Gray (1988, 2004; http://biowww.dfci.harvard.edu/~gray/). This software, along with others, will be implemented for selected examples.

Several authors have suggested extensions of the above software. Rosthoj et al. (2004) suggest SAS macros for estimation of the cumulative incidence functions based on a Cox regression model for competing risks survival data. They describe how to estimate the parameters in this model when some of the covariates may, or may not, have exactly the same effect on several causes of failure. In their paper, two

SAS macros are presented. The first macro named 'CumInc' is for estimation of the cumulative incidences, and a second macro named 'CumIncV' is for estimation of the cumulative incidences as well as the variances of those estimated cumulative incidences.

1.3 TESTING FOR A TREATMENT EFFECT

Often we are interested in comparing the risk of failure from a particular cause over two or more treatment groups. The cumulative incidences in the various groups can be calculated by a variety of methods. If we are calculating incidences based on the Kaplan-Meier approach when the risks are noninformative, then the log-rank test is appropriate for comparison of the treatment groups. However, when calculating incidences in the presence of competing risks, a modified test based on Gray's paper (1988) may be more appropriate. Both these methods are nonparametric and not based on any specific model.

Model based approaches such as the Cox proportional hazards model (1972), or a refinement of this model, are also often used for the analysis of competing risks data. These models are particularly attractive in situations where one observes additional covariates which may be related to the event of interest.

Under certain regularity conditions, Heckman and Honore (1989) show that for both proportional hazards and accelerated failure time models, if there is an explanatory variable whose support is the real line then the joint distribution of the competing risk times may be identifiable under certain conditions.

Abbring and van den Berg (2003) prove identification of dependent competing risks models in which each risk has a mixed proportional hazard specification with regressors, and the risks are dependent by way of the unobserved heterogeneity, or frailty, components. The authors also show that the conditions for identification given by Heckman and Honore, discussed above, can be relaxed.

Lunn and McNeil (1995) present two methods for the joint estimation of parameters in survival analysis models with competing risks. They demonstrate that it is possible to analyze survival data with competing risks using existing programs for fitting Cox's proportional hazards regression model with censored data. Two vectors of regression coefficients may be defined depending on the type of failure. The first procedure discussed runs a Cox regression stratified by the type of the failure. The second procedure uses unstratified Cox regression assuming that the hazard functions associated with the two types of failure have a constant ratio, which is an assumption that is often too stringent.

Another approach for the comparison of treatment groups that is a modification of the Cox proportional hazards model has been developed by Fine and Gray (1999). This model directly assesses the effect of covariates on the cumulative incidence function, or subdistribution function, of a particular type of failure in the presence of competing risks. The basic model assumes that the subdistribution with covariates is a constant shift on the complementary log log scale from some baseline subdistribution function.

Fine (2001) proposes a semi-parametric transformation model for the crude failure probabilities of a competing risk, conditional on covariates. This model is

developed as an extension of the standard approach to survival data with independent right censoring. The procedures are useful for subgroup analyses of cumulative incidence functions which may be complex under a cause-specific hazards formulation. Estimation of the regression coefficients is achieved using a rank-based least squares criterion. The new estimating equations are computationally simpler than the partial likelihood procedure in Fine and Gray (1999), and simulations show that the procedure works well with practical sample sizes.

Klein and Andersen (2005) argue that the estimates from regression models for competing risks outcomes based on proportional hazards models for the crude hazard rates do not agree with impressions drawn from plots of cumulative incidence functions for each level of a risk factor. They present a technique which models the cumulative incidence function directly by using pseudovalues from a jackknife statistic constructed from the cumulative incidence curve. They then study the properties of this estimator and apply the technique to a study dealing with the effect of alternative donors on relapse for leukemia patients that were given a bone marrow transplant.

Another alternative approach is to assume an additive hazard model presented by Aalen (1989), McKeague (1988), and more recently explored by Klein (2006). The latter author argues that additive models for either the hazard rates or the cumulative incidence functions are more natural and that these models properly partition the effect of a covariate on treatment failure into its component parts. The use and interpretation of such models is explored in detail in a study of the efficacy of two preparative regimes for hematopoietic stem cell transplantation.

Sun et al. (2004) also present an additive hazards model for competing risks analyses of the case-cohort design. This design may be applicable when the proportional hazards model does not provide a good fit for the observed survival data. Methods are presented for estimating regression parameters and cumulative baseline hazard functions, as well as cumulative incidence functions. The proposed estimators are shown to be consistent and asymptotically normal using the martingale central limit theory. The simulation studies conducted suggest that the proposed methods perform well.

Escarela and Carriere (2003) propose fitting a fully parametric model for competing risks with an assumed copula. Using the assumed copula, the authors show how the dependence structure between the cause-specific survival times can be modeled. Features include: identifiability of the problem, accessible understanding of the dependence structure, and flexibility in choosing marginal survival functions. The model is constructed in such a way that it allows us to adjust for concomitant variables and a dependence parameter in order to assess the effect that these have on each marginal survival model and on the relationships between the causes of death.

Jewell et al. (2003) provide an analysis with current status data (an extreme form of censoring which arises where the only information on studied individuals is their current survival status at a single monitoring time) in the context of competing risks. They look at simple parametric models and compare the results to nonparametric estimation. In addition to simulation results, the authors establish asymptotic efficiency of smooth functionals of the sub-distribution functions.

Gilbert et al. (2004) provide tests for comparing mark-specific hazards and cumulative incidence functions. The authors develop nonparametric tests for the problem of determining whether there is a relationship between a hazard rate function or a cumulative incidence function, and a mark variable which is only observed at uncensored failure times. They consider the case where the mark variable is continuous.

Pepe (1991) presents an alternative approach to estimating the cumulative incidence functions in a competing risk situation. The author develops inferential procedures for functions which are not simply survival, cumulative incidence, or cumulative hazard functions, but that can be expressed as simple functions of several of them. This paper's strong contribution is that it develops the asymptotic distribution theory for estimators of these constituent functions when there is a dependence structure among the multiple time-to-event endpoints.

Two interesting papers by Andersen et al. (2002, 2003) show how the competing risks model may be viewed as a special case of a multi-state model. The properties of this model are reviewed and contrasted to the so-called latent failure time approach. The relation between the competing risks model and right-censoring is discussed, and a regression analysis of the cumulative incidence function is also reviewed.

Steele et al. (2004) propose a general discrete time model for multilevel event history data. This model is developed for the analysis of longitudinal repeated episodes within individuals where there are multiple states and multiple types of events (competing risks) which may vary across states. The different transitions are

modeled jointly to allow for correlation across transitions in unobserved individual risk factors. This model is then applied to the analysis of contraceptive use dynamics in Indonesia where transitions from two states, contraceptive use and nonuse, are of interest.

The following two papers deal with sample sizes for competing risks. First, Pintilie (2002) presents a method to calculate the sample size for testing the effect of a covariate on an outcome in the presence of competing risks. Secondly, Latouche et al. (2004) present approximate sample size formulas for the proportional hazards modeling of competing risk subdistributions, considering either independent or correlated covariates. The validity of these approximate formulas is investigated through numerical simulations.

CHAPTER 2

RESEARCH OBJECTIVE

Prior to Gray (1988), much of the work for determining treatment effects on competing risks had focused on the cause-specific hazard for failures of each type within the treatment groups under a proportional hazards assumption. However, Gray pointed out that the effect of a factor on the cause-specific hazard for a particular type of failure may be quite different than its effect on the cumulative incidence of that type of failure. Therefore, the hypothesis of equal cause-specific hazard functions is not necessarily equivalent to the hypothesis of equal cumulative incidence functions.

Gray and others have proposed popular cumulative incidence based methods to test for covariate (or treatment) effects in the presence of competing risks. However some recent literature has suggested that when testing for treatment effects in the presence of competing risks, the popular cumulative incidence based approach may be problematic.

Gooley et al. (1999) point out that when our interest lies in evaluating the effect of a covariate on the hazard of failure from the cause of interest, the use of cumulative incidence estimates and tests based on them may be misleading if the treatment also affects the hazard of the competing risk. The authors claim that in

situations such as these, the log-rank test is appropriate for inference since it is solely a function of the hazard of failure from the cause of interest and failures from the competing risk are treated as censored observations. They also note that if we are interested in comparing the probabilities of failure between two groups, then tests based on the cumulative incidence are appropriate since the cumulative incidence estimates depend on the hazards of both types of failure.

Freidlin and Korn (2005) have also pointed out that under certain circumstances when testing for treatment effects in the presence of competing risks, the popular cumulative incidence based approach may be problematic. The authors consider comparing treatments A and B in the presence of two causes of failure denoted by $i = 1,2$. In their paper, they consider three common approaches to the analysis of competing risks data. The first method is based on the distribution of the time until first failure. For this method, they consider the null hypothesis, $S^A(t) \equiv S^B(t)$, where $S^A(t)$ and $S^B(t)$ denote the overall (time to first event) survival functions in the two treatment groups. This hypothesis is commonly tested via the log-rank test T^o using the time until first event data. The second method is based on the cause-specific hazard rates $h_i^A(t)$ and $h_i^B(t)$ where the null hypothesis $h_i^A(t) \equiv h_i^B(t)$ is typically tested via the log rank test T^{h_i} using the times to the specific type of failure. The third method is based on the cumulative incidence functions denoted by $I_i^A(t)$ and $I_i^B(t)$. For this method, the null hypothesis $I_i^A(t) \equiv I_i^B(t)$ is often examined using the nonparametric test described by Gray (1988) which the authors denote by T^{I_i}. Based on their results, Freidlin and Korn

conclude that the cause-specific hazard approach to comparing the marginal latent failure distributions is generally more powerful and better at preserving the nominal alpha level than the cumulative incidence-based approach. The authors provide results from a special case of general copula models for generating bivariate survival data, using an exponential marginal distribution and a sample size of 200. However, the authors neglect to consider situations where independent right censoring may occur. Therefore, it would seem that further investigation is required to ascertain when and under what circumstances the cumulative incidence based approach is preferred over the log-rank test.

Other literature published as recently as this year provides a viewpoint contrary to the claims made by Gooley, and Freidlin and Korn. Among others, Kim (2007) discusses competing risks data analysis including methods to calculate the cumulative incidence of an event of interest in the presence of competing risks, methods to compare cumulative incidence curves in the presence of competing risks, and methods to perform competing risks regression analysis. The author explains that if there is more than one type of event, and if these events are dependent, then the widely used Kaplan-Meier method is a biased estimate of cumulative incidence. In these situations, Kim points out that it is inappropriate to use the log-rank test for comparing the cumulative incidence rates among different groups, and that Gray's (1988) test is more applicable. Finally, the author provides reasoning for the use of direct regression modeling of the effect of covariates on the cumulative incidence function for competing risks data, as opposed to the use of a standard Cox

proportional hazards model. The author concludes the article by stating that future research should involve power studies in the presence of competing risks data.

Due to the discrepancies seen in the recent literature, we believe this topic requires further investigation. This investigation should include, but not be limited to, the power studies suggested by Kim (2007).

CHAPTER 3

METHODS

3.1 ESTIMATING CUMULATIVE INCIDENCE

As a portion of this dissertation, I will review the various techniques for estimating the cumulative incidence functions in the presence of competing risks. These will include both nonparametric methods and methods based on semi-parametric models such as the Cox proportional hazards model. Some of these methods have already been implemented in popular software packages while others will require programming of the methods described in the literature. Specifically, I will be using R, Stata, and SAS software to compare methods employed by those software packages, as well as those developed by Gray, Fine and Gray, and Rosthoj. Additionally, I will review a nonparametric method for estimating cumulative incidence which has been reported by Klein and Moeschberger.

3.2 POWER STUDY

I will also review the methodology commonly used for testing failure specific treatment effects in the presence of competing risks. As stated previously, the

literature on this topic has been inconsistent; some recent literature suggesting that when testing for treatment effects in the presence of competing risks, the popular cumulative incidence based approach may be problematic. Via a simulation study, I will investigate the level of significance and power associated with the methods commonly used when testing for failure specific treatment effects in the presence of competing risks. The simulations will be performed for bivariate survival data under a variety of conditions.

I will begin by considering two therapy groups (treatment vs. control), as well as two competing causes of failure in the presence of censoring. The times to failure for the competing causes will be generated according to specific bivariate distributions. These will include the independent bivariate exponential distribution, the bivariate exponential distribution due to Sarmanov, the bivariate exponential distribution due to Marshall and Olkin (1967), the bivariate exponential distribution due to Cai and Prentice (1995), and the bivariate exponential distribution using a gamma frailty. Furthermore, I will also consider the bivariate Weibull distributions corresponding to each of the previous bivariate exponential distributions via the appropriate transformation. Each of these exponential and Weibull distributions will result in hazards for the treatment and control group that are proportional to each other. To investigate the non-proportional hazards setting, I will also generate failure times according to the bivariate normal and bivariate log-normal distributions. I will consider a variety of treatment effects (none, small, and large) and other parameters for both causes of failure, various correlations between the competing causes of failure (0, 0.20, 0.50, and 0.80), and various sample sizes within each of the treatment

groups (50, 100, and 200). Additionally, I will also allow for a variety of censoring probabilities within each of the therapy groups (0, 0.20, 0.50, and 0.80).

I will then compare the level of significance and power associated with four methods for analyzing survival data in the presence of competing risks. The four methods to be compared are (1) the Cox proportional hazards model treating competing risk events as censored, (2) the log-rank test, (3) Gray's test, and (4) a semi-parametric test due to Fine and Gray. The test based on Cox's proportional hazards model and the log-rank test are based on the cause-specific hazards, while Gray's test and the test due to Fine and Gray are based on the cumulative incidence functions.

The Cox proportional hazards model is often used for the analysis of competing risks data. Assuming just one covariate, this model is given by

$$\lambda(t \mid Z) = \lambda_o(t)\exp(\beta Z)$$

where $\lambda(t \mid Z)$ denotes the hazard rate at time t for an individual with covariate Z, $\lambda_o(t)$ is an arbitrary baseline hazard rate, and β is the parameter associated with covariate Z. This model is often used to test that the hazard rate is independent of the covariate Z. Therefore the null and alternative hypotheses can be written as

$$H_o : \beta = 0$$

and

$$H_a : \beta \neq 0 .$$

We are then able to test this hypothesis by fitting a model that includes the covariate Z and comparing it's fit to that of the reduced model excluding the covariate Z, via

a Likelihood Ratio Test. Note that when being used for competing risks survival data, the Cox proportional hazards model treats all deaths due to causes other than the event of interest as censored. Thus it does not account for the dependence that may often be present between the competing causes of failure.

The log-rank test (Mantel, 1966) is a popular test for comparing the hazard rates of two or more populations in the typical survival analysis setting where independent right censoring may occur. I will focus on the case where we wish to compare only two populations. The null hypothesis is that the hazards for both populations are the same for all times $t \le \tau$ where τ is the largest time at which both groups have at least one subject at risk. The alternative is that they are different for some time $t \le \tau$. Thus we have

$$H_o : h_1(t) = h_2(t) \quad \forall t \le \tau$$

$$H_a : h_1(t) \ne h_2(t) \quad \text{for some } t \le \tau .$$

For only two populations, the log-rank test statistic can be written as:

$$Z = \frac{\sum_{i=1}^{D}\left[d_{i1} - Y_{i1}\left(\frac{d_i}{Y_i}\right)\right]}{\sqrt{\sum_{i=1}^{D}\frac{Y_{i1}}{Y_i}\left(1-\frac{Y_{i1}}{Y_i}\right)\left(\frac{Y_i - d_i}{Y_i - 1}\right)d_i}}$$

where D is the total number of deaths in the pooled sample, d_{i1} is the number of deaths in population 1 at time t_i, d_i is the number of deaths at time t_i in the pooled sample, Y_{i1} is the number at risk in population 1 at time t_i, and Y_i is the number at risk at time t_i in the pooled sample. Essentially, the test statistic Z is comparing the

number of deaths observed in the first population to the number of deaths we would have expected to see in the first population if the two hazards were in fact equal. Under the null hypothesis, Z follows a standard normal distribution. Therefore, we reject the null hypothesis if $|Z| > z_{\alpha/2}$. This test has optimum power to detect alternatives when the hazard rates in the populations are proportional. Some authors have also claimed that this is the preferred test for comparing treatments in the presence of competing risks, however it appears that further investigation is needed.

The third method of analysis being considered is described in Gray (1988). The author defines the cumulative incidence function, or subdistribution function, for failures of type j in group k by

$$F_{jk}(t) = P(T_j \le t, \delta_{jk} = 1) \text{ for } j = 1,...,J,\ k = 1,...,K$$

where J is the number of competing risks being considered and K represents the number of treatment groups. The purpose of the paper is to develop a test for the hypothesis

$$H_o : F_{1k} = F_1^o \text{ for } k = 1,...,K$$

where F_1^o is an unspecified cumulative incidence function and the cause of interest is taken to be type 1. For simplicity, the author assumes the $F_{jk}(t)$ are continuous with subdensities $f_{jk}(t)$. For $J = 2$ causes of failure and $K = 2$ treatment groups, Gray proposed that the *K*-sample test be based on a score of the form

$$z = \int_0^\tau W(t)\left\{\left[1 - \hat{F}_{11}(t^-)\right]^{-1} d\hat{F}_{11}(t) - \left[1 - \hat{F}_{12}(t^-)\right]^{-1} d\hat{F}_{12}(t)\right\}$$

where $W(t)$ is a suitably chosen weight function and $\hat{F}_{1k}$ is an estimate of the cumulative incidence function $F_{1k}(t)$. This statistic essentially compares weighted averages of the nonparametric maximum likelihood estimators of the subdistribution hazards for each treatment group. By selecting $W(t)$ to be of a certain form, an appropriate *K*-sample test statistic can be formed such that it will asymptotically follow a chi-square distribution with $K-1$ degrees of freedom under the null hypothesis. Gray's method can be performed using the 'cmprsk' package found in the R 2.1.0 statistical software package. Specifically, the 'cuminc' function provides statistical tests, estimates of the cumulative incidences, and variance estimates for the different environment groups. The 'plot.cuminc' function can then used to plot the cumulative incidence functions for the different failure types within each of the treatment groups.

Another test available in R 2.1.0 is due to Fine and Gray (1999). The authors point out that prior to Gray (1988), the standard analysis for competing risks in the presence of covariates involved modeling the cause-specific hazard functions of the different failure types under a proportional hazards assumption. However, many authors have claimed that the effect of a covariate on a particular failure type's cause-specific hazard function may be very different from its effect on the corresponding cumulative incidence function (Gray 1988, Pepe 1991). The result is that testing for covariate effects on the subdistribution is not possible under the cause-specific hazard formulation. Therefore the authors' goal is to develop a semi-parametric model for the crude incidence, similar to the Cox model commonly used for univariate failure

time data. For simplicity, I will consider the case with only one covariate, although the results can be easily extended to allow for multiple covariates. Interest is on modeling the cumulative incidence function for failure due to the cause of interest (cause 1) conditional on the covariate Z,

$$F_1(t;Z) = P(T \le t, \delta_1 = 1 \mid Z).$$

Consider the class of semi-parametric models used for univariate survival data (Cox 1972). Assume that for some known increasing function $g(\cdot)$,

$$g\{F_1(t;Z)\} = h_o(t) + \beta_o Z$$

where $h_o(t)$ is a completely unspecified, invertible, and monotone increasing function and β_o is the parameter associated with covariate Z. Note that for two individuals having covariates Z_1 and Z_2, the conditional cumulative incidence functions will satisfy a vertical shift model after the transformation. In other words

$$g\{F_1(t;Z_1)\} - g\{F_1(t;Z_2)\} = (Z_1 - Z_2)\beta_o$$

for all values of t. Selecting $g = \log\{-\log(1-u)\}$ corresponds to the proportional hazards model. The authors go on to derive an estimate of β_o, denoted by $\hat{\beta}$, and show that this estimate is consistent for β_o. The distribution of $\sqrt{n}(\hat{\beta} - \beta_o)$ can be approximated by a normal distribution with known variance. Therefore, this limiting result allows us to make inferences about the covariate effect of Z on the cumulative incidence function. The null and alternative hypotheses are given by

$$H_o : \beta_o = 0$$

and

$$H_a : \beta_o \neq 0 .$$

In R 2.1.0, we are able to apply the ‘crr’ function to fit the above proportional subdistribution hazards regression model described in Fine and Gray (1999).

CHAPTER 4

HOEL EXAMPLE

To illustrate some of the methods for estimating cumulative incidence in the presence of competing risks, we considered the mouse carcinogenicity data set published by Hoel (1972). The data were obtained from a laboratory experiment on two groups of RFM strain male mice which had received a radiation dose of 300r at an age of 5-6 weeks. The mice were randomly assigned to either a conventional laboratory environment or a germ-free environment, and the number of days until death was recorded for each mouse. Two major causes of death were considered, namely Thymic Lymphoma and Reticulum Cell Sarcoma, while all other causes of death were combined into a single group. Thus, the data consisted of three variables: number of days until death, cause of death, and type of environment.

A total of n=181 mice were randomly assigned to either a conventional laboratory environment or a germ-free environment. Of the 99 mice that were randomly assigned to the conventional laboratory environment, 22 died as a result of Thymic Lymphoma while 38 died as a result of Reticulum Cell Sarcoma. The remaining 39 mice assigned to the conventional laboratory environment experienced deaths attributed to other causes. Of the 82 mice that were randomly assigned to the

germ-free environment, 29 died as a result of Thymic Lymphoma while 15 died as a result of Reticulum Cell Sarcoma. The remaining 38 mice assigned to the germ-free environment experienced deaths attributed to other causes.

4.1 DATA ANALYSIS

We begin by estimating the cumulative incidence functions using nonparametric methods that are implemented in widely available software packages such as R 2.1.0 and Stata/SE 9.2. We will then consider other nonparametric methods of estimation described in the literature and compare their results to those given by the software packages. Lastly, we will apply estimation techniques using two semi-parametric methods also described in recent literature, and again compare their results to those previously obtained.

We first analyzed the data using the method described in Gray (1988). Gray's method can be performed using the 'cmprsk' package found in the R 2.1.0 statistical software package. Specifically, the 'cuminc' function provides statistical tests, estimates of the cumulative incidences, and variance estimates for the different environment groups. The 'plot.cuminc' function was then used to plot the cumulative incidence functions for the three failure types within each of the two environment groups. The output from the 'cuminc' function is given in Tables 4.1 - 4.3 (Section 4.2) while the plots are displayed in Figures 4.1 – 4.3 (Section 4.3). Although so many significant digits would not typically be recorded in the Tables, we have included the actual R output strictly for the benefit of the reader. From the p-values given in Table 4.1, we see that there are significant differences (using $\alpha = 0.05$)

between the environment groups in terms of death due to Reticulum Cell Sarcoma and death due to Other Causes. By examining Figures 4.2 and 4.3, we see that in both cases, the germ-free environment appears to improve survival time over the conventional environment. However, there is not enough evidence to suggest any significant differences between the environment groups in terms of death due to Thymic Lymphoma. In fact, Figure 4.1 appears to suggest that the conventional environment may even be preferred when considering death due to Thymic Lymphoma.

Again using the Hoel data, we were able to estimate the cumulative incidence functions in the presence of competing risks by downloading the 'st0059' package available in the Stata/SE 9.2 software program. Specifically, the 'stcompet' function provides estimates of the cumulative incidence functions and their standard errors, as well as upper and lower confidence bounds for the cumulative incidences based on a log(-log) transformation. The resulting estimated cumulative incidence functions are displayed in Figures 4.4 – 4.6. Note that these plots are in agreement with those produced using Gray's method. The lone difference being the convention with which the two software packages determine the largest time on study to be used in the extension of the plots. Whereas Gray's method in R 2.1.0 extends the plots out until the largest time on study within each treatment group regardless of cause, Stata/SE 9.2 uses the largest *cause-specific* time on study within each of the treatment groups.

We then analyzed the same data set using the cumulative incidence estimator described in Klein and Moeschberger (2003). This method produced the same estimates as those obtained using Gray's method. The resulting cumulative incidence

plots due to Klein and Moeschberger are given in Figures 4.7 – 4.9. Klein and Moeschberger also provide a SAS macro for computing cumulative incidence on their web-site (http://www.biostat.mcw.edu/homepgs/klein/book.html), however this macro is written for the special case of only one competing risk in addition to the cause of interest. Therefore as an example, we considered only the deaths due to Thymic Lymphoma or Reticulum Cell Sarcoma and again performed the analysis using Gray's method in R 2.1.0 as well as the SAS macro provided by Klein and Moeschberger. Again, we found the estimates to be identical. The cumulative incidence plots obtained from Gray's method when considering only Thymic Lymphoma and Reticulum Cell Sarcoma are given in Figures 4.10 and 4.11, while those obtained from Klein and Moeschberger's SAS macro are displayed in Figures 4.12 and 4.13.

Once again using the complete data set given in Hoel (1972), we applied the 'crr' function in R 2.1.0 to fit the proportional subdistribution hazards regression model described in Fine and Gray (1999). The output from the 'crr' function is given in Tables 4.4 – 4.6 while the estimated cumulative incidence plots are displayed in Figures 4.14 – 4.16. From the p-values given in Table 4.4, we see that there are marginal differences (using $\alpha = 0.05$) between the environment groups in terms of death due to Thymic Lymphoma and significant differences for death due to Reticulum Cell Sarcoma. By examining Figure 4.14, we see that in the case of Thymic Lymphoma, the conventional environment appears to improve survival time over the germ-free environment. However, Figure 4.15 suggests that the germ-free environment is preferred in the case of Reticulum Cell Sarcoma. The p-value for the

effect of environment and the cumulative incidence plot associated with death due to Other Causes are also displayed in Table 4.4 and Figure 4.16, respectively. Examination of these results does not suggest any significant differences between the environment groups in terms of death due to Other Causes. Note that for Thymic Lymphoma and Reticulum Cell Sarcoma, the cumulative incidence estimates and conclusions based on this semi-parametric method due to Fine and Gray are quite similar, though not identical, to those drawn from the non-parametric method developed by Gray (1988). On the other hand, for deaths due to Other Causes, Gray's nonparametric method suggested a significant effect due to environment while the semi-parametric method due to Fine and Gray does not. Also note that the plots produced using Fine and Gray's method are extended out until the largest cause-specific time on study regardless of treatment group.

We again estimated the cumulative incidence functions for each cause of failure using a method proposed by Rosthoj et al. (2004). In their paper, the authors provide two SAS macros which can be used for estimating the cumulative incidence functions based on a Cox regression model for competing risks survival data. The 'CumInc' macro was used in SAS to obtain the cumulative incidence estimates for each cause of death within both the conventional and germ-free environments. The resulting plots are very similar to those produced using Gray's method (1988) and are displayed in Figures 4.17 – 4.19. Figure 4.17 suggests that for death due to Thymic Lymphoma, the conventional environment may improve survival time over the germ-free environment. However, Figures 4.18 and 4.19 suggest that when considering death due to Reticulum Cell Sarcoma or Other Causes, the germ-free environment

may actually be preferred. One noticeable difference between the plots produced using the techniques described by Rosthoj and Gray's method is the crossing of the cumulative incidence functions for Other Causes in Figure 4.19. This is explained by the convention employed by Rosthoj, which extends the cumulative incidence functions out until the largest time on study regardless of the specific cause or treatment group.

As a final note, we point out that the cumulative incidence plots published in Hoel (1972) may be reproduced by considering separate data sets for each specific cause of death (Thymic Lymphoma, Reticulum Cell Sarcoma, and Other Causes) and performing Gray's Method. This technique, however, fails to estimate the cumulative incidence functions in a simultaneous fashion.

At this time, it is our recommendation that the technique described in Gray (1988) be the preferred method of analysis for estimating cumulative incidence in the presence of competing risks. Although several of the methods described above produced similar results, the recommendation to use Gray's method is based primarily on the fact that it is nonparametric in nature. Therefore no assumption of proportional hazards is required.

In addition to estimating the cumulative incidences, we will also compare four methods used for testing failure specific treatment effects in the presence of competing risks. The four methods to be compared are (1) the test based on Cox's proportional hazards model treating competing risk events as censored, (2) the log-rank test, (3) Gray's test, and (4) a semi-parametric test due to Fine and Gray.

The results of Gray's test have already been displayed in Table 4.1, and the results of the semi-parametric test due to Fine and Gray are given in Table 4.4. We will now apply the 'coxph' command in R to test for a treatment effect using the Cox proportional hazards model treating competing risk events as censored. The resulting coefficients, standard errors, and Likelihood ratio test p-values for each cause of failure are given in Table 4.5. Finally, we will apply the 'survdiff' command in R to test for a treatment effect using the log-rank test. The resulting test statistics, degrees of freedom, and p-values for each cause of failure are given in Table 4.6. The results of the test based on Cox's proportional hazards model and the log-rank test are in close agreement with those based on Gray's test.

Computer code for the Tables and Figures presented in Sections 4.3 and 4.4 is given in Appendix A.

4.2 TABLES

Tests:

	stat	pv	df
1	2.895483	0.0888281369	1
2	13.885342	0.0001943080	1
3	6.640314	0.0099696340	1

Table 4.1: Gray's test statistics and p-values for comparing the environment groups. Test 1 corresponds to death due to Thymic Lymphoma. Test 2 corresponds to death due to Reticulum Cell Sarcoma. Test 3 corresponds to death due to Other Causes.

		200	400	600	800	1000
1	1	0.05050505	0.18181818	0.22222222	NA	NA
2	1	0.06097561	0.23170732	0.31707317	0.3536585	0.3536585
1	2	0.00000000	0.03030303	0.15151515	NA	NA
2	2	0.00000000	0.00000000	0.02439024	0.1585366	0.1829268
1	3	0.06060606	0.17171717	0.30303030	NA	NA
2	3	0.01219512	0.04878049	0.07317073	0.2682927	0.4390244

Table 4.2: Gray's Cumulative Incidence estimates. The first column indicates that the estimates are for the two environment groups (1 corresponds to conventional, 2 corresponds to germ-free). The second column indicates that the estimates are for the three causes of death (1 corresponds to Thymic Lymphoma, 2 corresponds to Reticulum Cell Sarcoma, 3 corresponds to Other Causes). Also, the first row of each table indicates the time points for which the estimates are given.

		200	400	600	800	1000
1	1	0.0004897504	0.0015219666	0.0017700850	NA	NA
2	1	0.0007072238	0.0022024528	0.0026829906	0.002840750	0.002840750
1	2	0.0000000000	0.0003012409	0.0013277692	NA	NA
2	2	0.0000000000	0.0000000000	0.0002957942	0.001674860	0.001933082
1	3	0.0005811311	0.0014549986	0.0021697718	NA	NA
2	3	0.0001487210	0.0005743169	0.0008405426	0.002460062	0.003147561

Table 4.3: Gray's estimated variances of the Cumulative Incidence estimates. The first column indicates that the estimates are for the two environment groups (1 corresponds to conventional, 2 corresponds to germ-free). The second column indicates that the estimates are for the three causes of death (1 corresponds to Thymic Lymphoma, 2 corresponds to Reticulum Cell Sarcoma, 3 corresponds to Other Causes). Also, the first row of each table indicates the time points for which the estimates are given.

Tests:			
	coefficient	standard error	2-sided p-value
1	0.5192	0.2820	0.066
2	-0.9336	0.2966	0.002
3	0.0045	0.2270	0.980

Table 4.4: Fine and Gray's estimated regression coefficients, standard errors, and p-values for comparing the environment groups. Test 1 corresponds to death due to Thymic Lymphoma. Test 2 corresponds to death due to Reticulum Cell Sarcoma. Test 3 corresponds to death due to Other Causes.

Tests:				
	coefficient	standard error	Likelihood ratio test	p-value
1	0.307	0.287		0.282
2	-2.030	0.345		< 0.001
3	-1.180	0.298		< 0.001

Table 4.5: Cox proportional hazards coefficient, standard error, and p-value for comparing the environment groups. Test 1 corresponds to death due to Thymic Lymphoma. Test 2 corresponds to death due to Reticulum Cell Sarcoma. Test 3 corresponds to death due to Other Causes.

Tests:			
	Chisq	df	p-value
1	1.2	1	0.282
2	45.0	1	< 0.001
3	17.1	1	< 0.001

Table 4.6: Log Rank test statistic, degrees of freedom, and p-value for comparing the environment groups. Test 1 corresponds to death due to Thymic Lymphoma. Test 2 corresponds to death due to Reticulum Cell Sarcoma. Test 3 corresponds to death due to Other Causes.

4.3 FIGURES

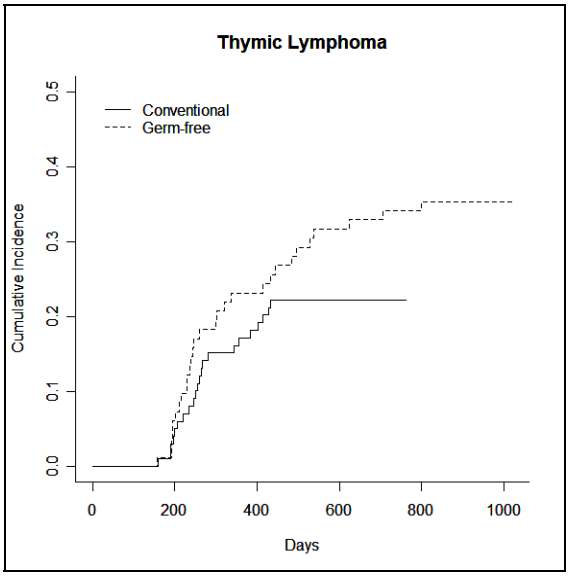

Figure 4.1: Cumulative Incidence estimates for the Thymic Lymphoma failure type (Gray).

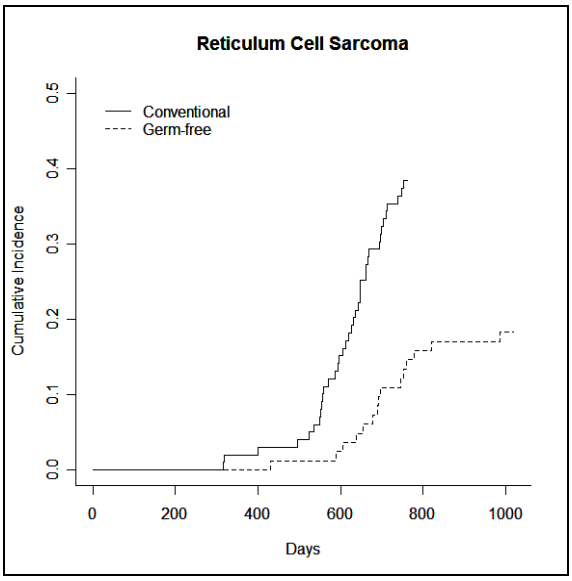

Figure 4.2: Cumulative Incidence estimates for the Reticulum Cell Sarcoma failure type (Gray).

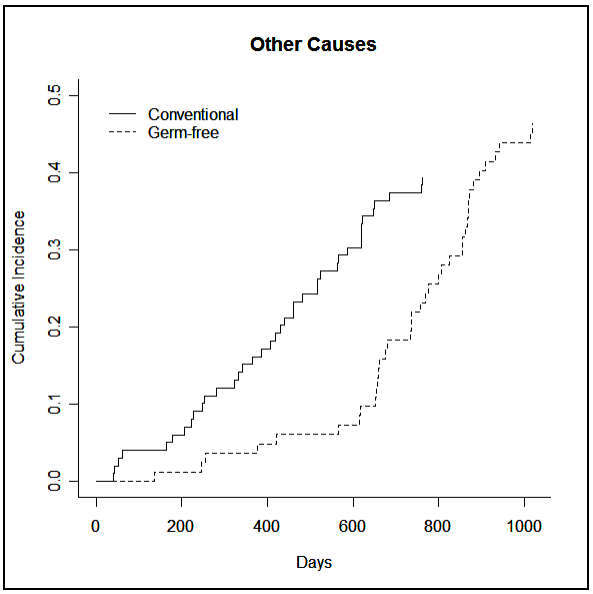

Figure 4.3: Cumulative Incidence estimates for the Other Causes failure type (Gray).

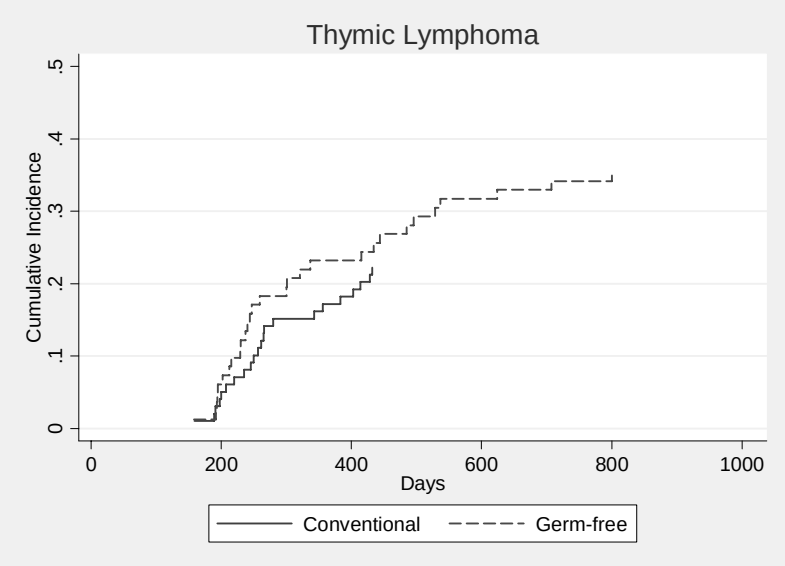

Figure 4.4: Cumulative Incidence estimates for the Thymic Lymphoma failure type (Stata/SE 9.2).

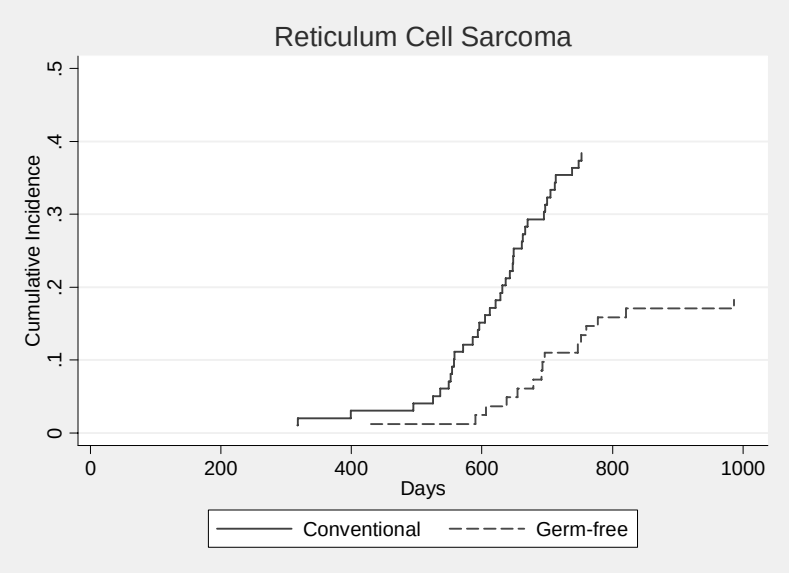

Figure 4.5: Cumulative Incidence estimates for the Reticulum Cell Sarcoma failure type (Stata/SE 9.2).

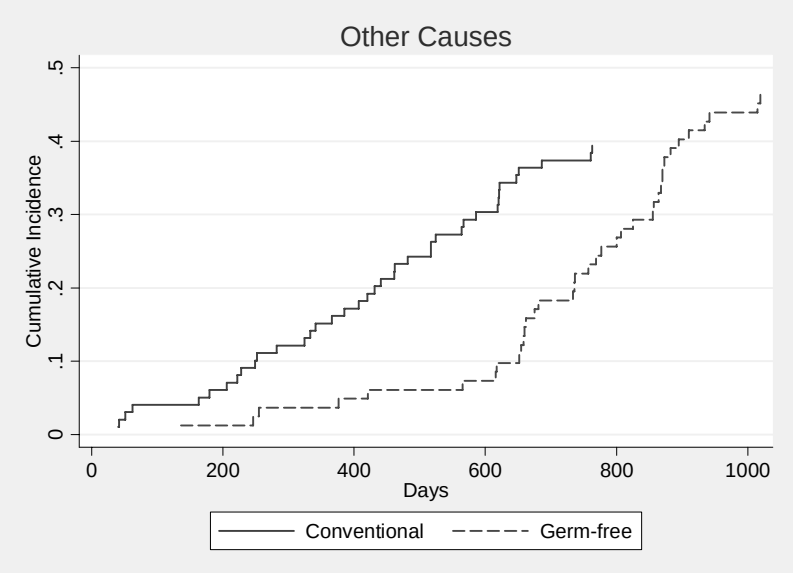

Figure 4.6: Cumulative Incidence estimates for the Other Causes failure type (Stata/SE 9.2).

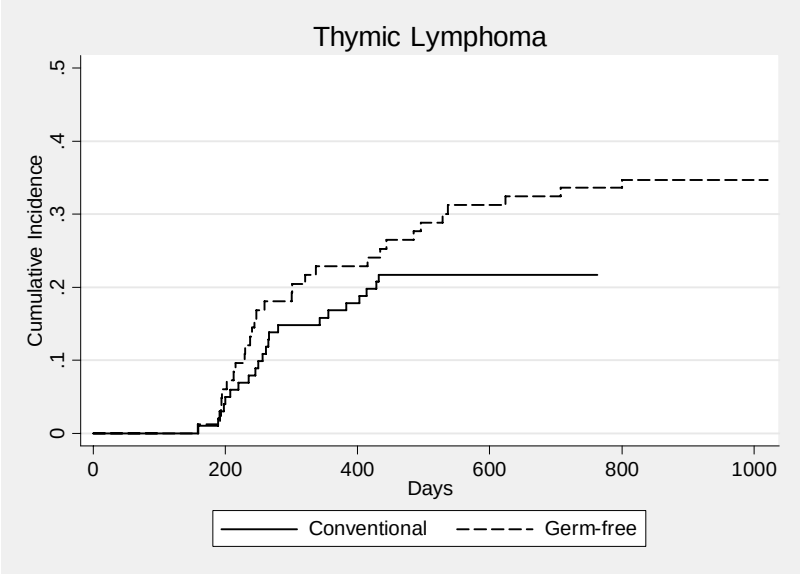

Figure 4.7: Cumulative Incidence estimates for the Thymic Lymphoma failure type (Klein and Moeschberger).

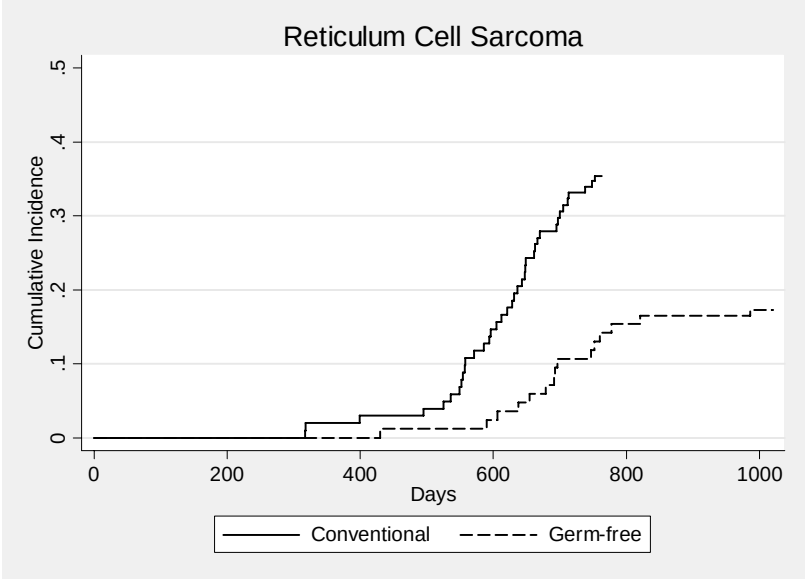

Figure 4.8: Cumulative Incidence estimates for the Reticulum Cell Sarcoma failure type (Klein and Moeschberger).

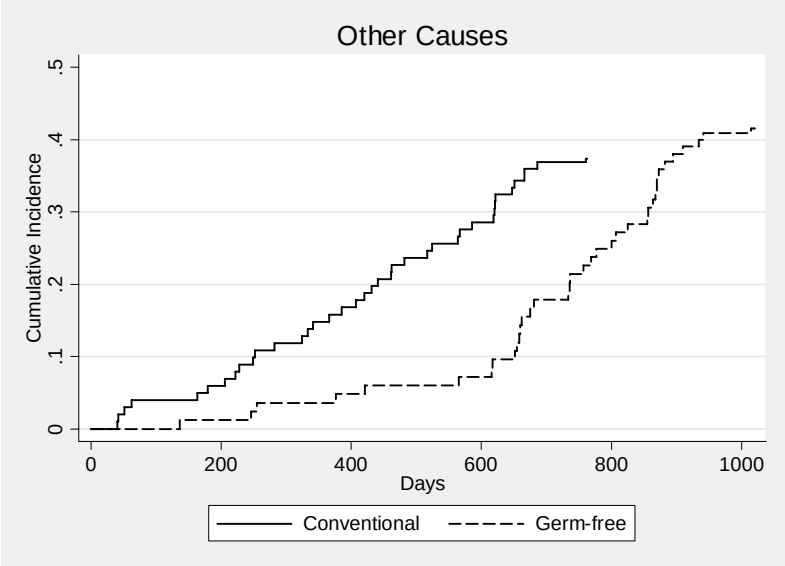

Figure 4.9: Cumulative Incidence estimates for the Other Causes failure type (Klein and Moeschberger).

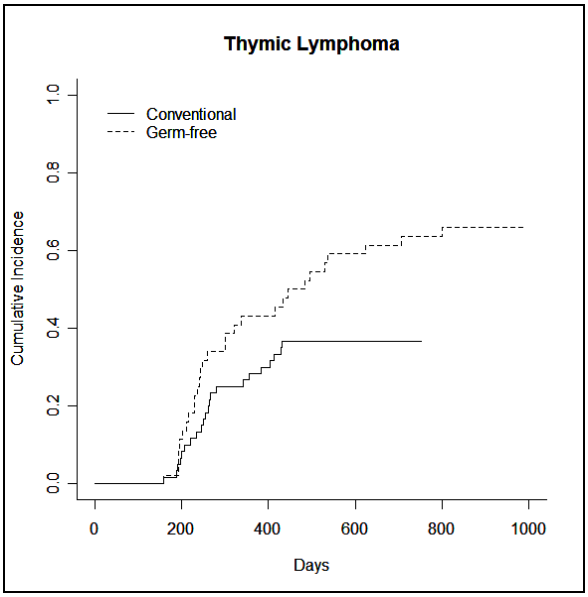

Figure 4.10: Cumulative Incidence estimates for the Thymic Lymphoma failure type when Other Causes are excluded (Gray).

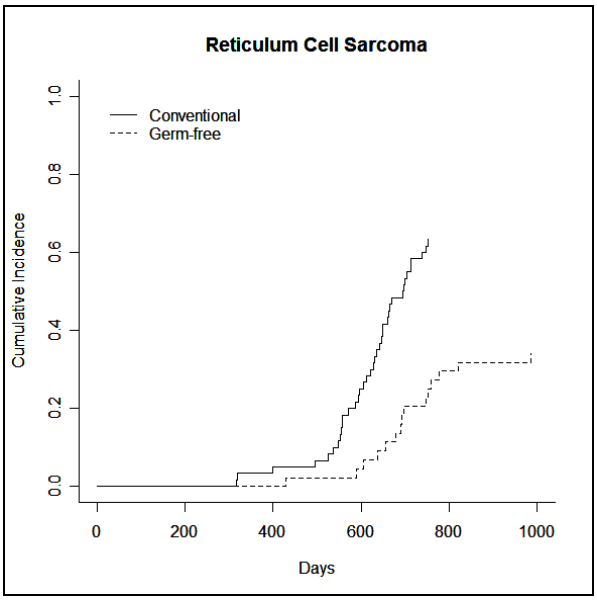

Figure 4.11: Cumulative Incidence estimates for the Reticulum Cell Sarcoma failure type when Other Causes are excluded (Gray).

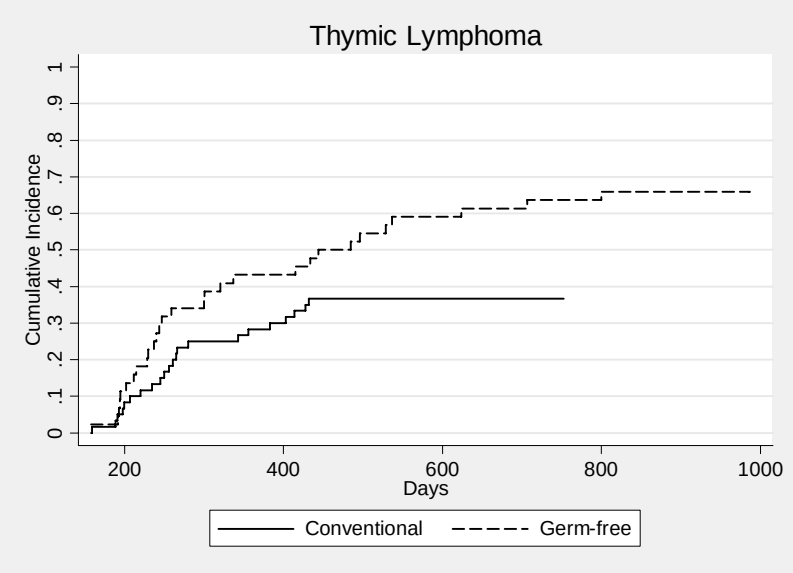

Figure 4.12: Cumulative Incidence estimates for the Thymic Lymphoma failure type when Other Causes are excluded (Klein and Moeschberger SAS macro).

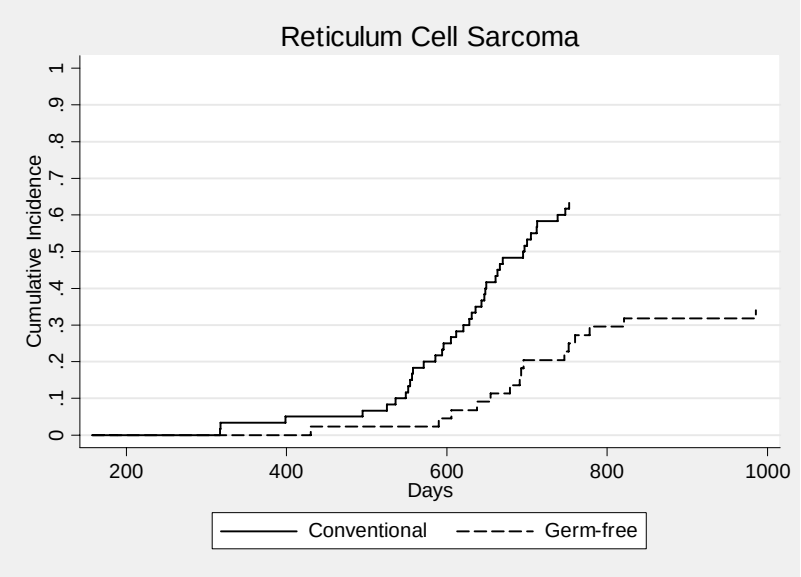

Figure 4.13: Cumulative Incidence estimates for the Reticulum Cell Sarcoma failure type when Other Causes are excluded (Klein and Moeschberger SAS macro).

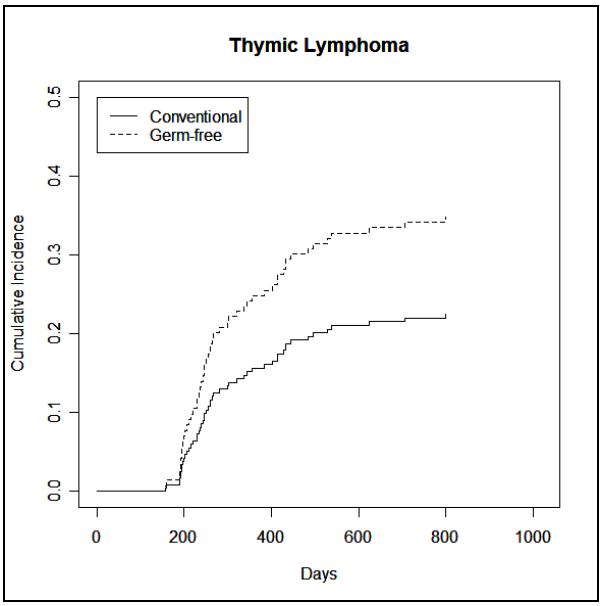

Figure 4.14: Cumulative Incidence estimates for the Thymic Lymphoma failure type (Fine and Gray).

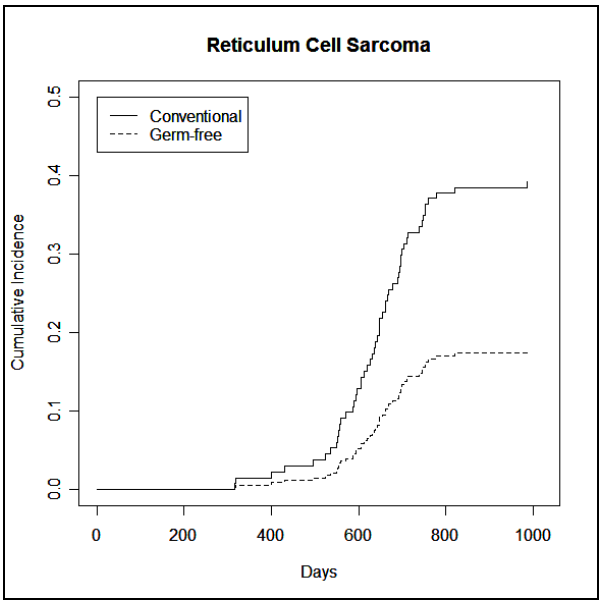

Figure 4.15: Cumulative Incidence estimates for the Reticulum Cell Sarcoma failure type (Fine and Gray).

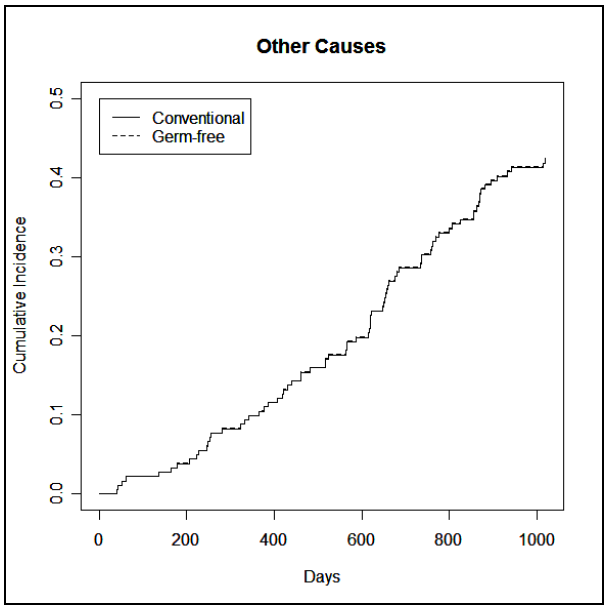

Figure 4.16: Cumulative Incidence estimates for the Other Causes failure type (Fine and Gray).

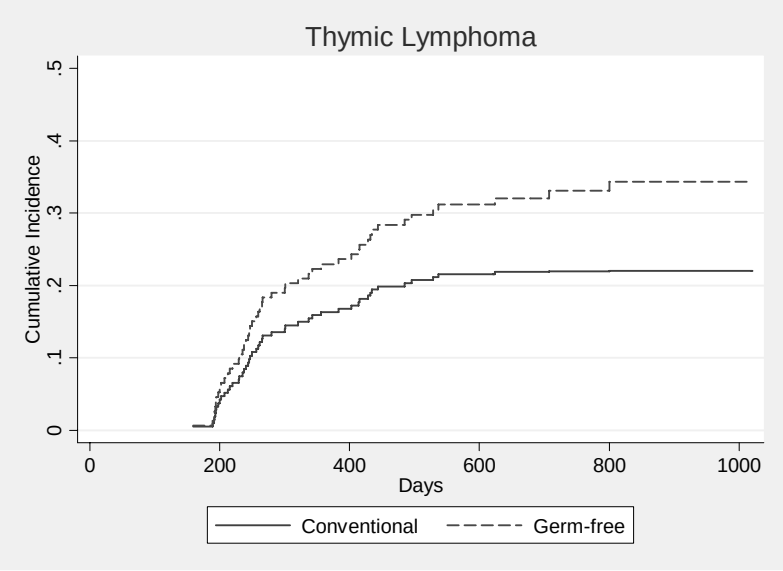

Figure 4.17: Cumulative Incidence estimates for the Thymic Lymphoma failure type (Rosthoj).

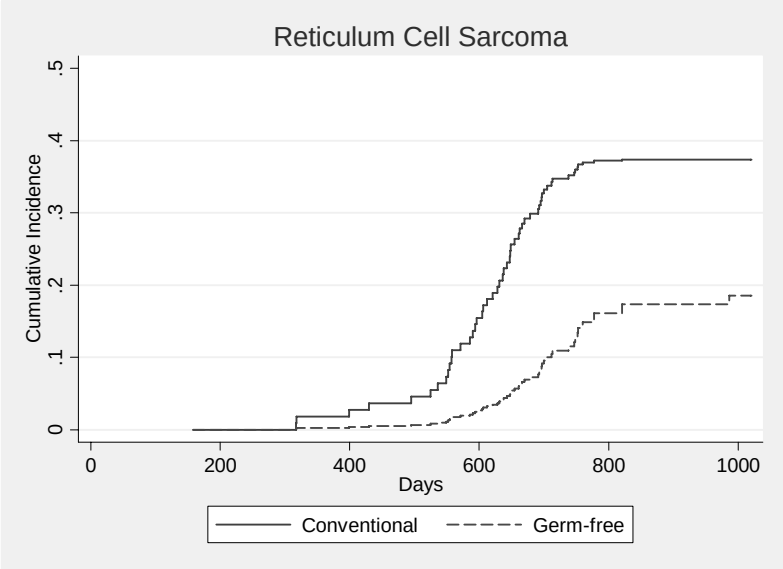

Figure 4.18: Cumulative Incidence estimates for the Reticulum Cell Sarcoma failure type (Rosthoj).

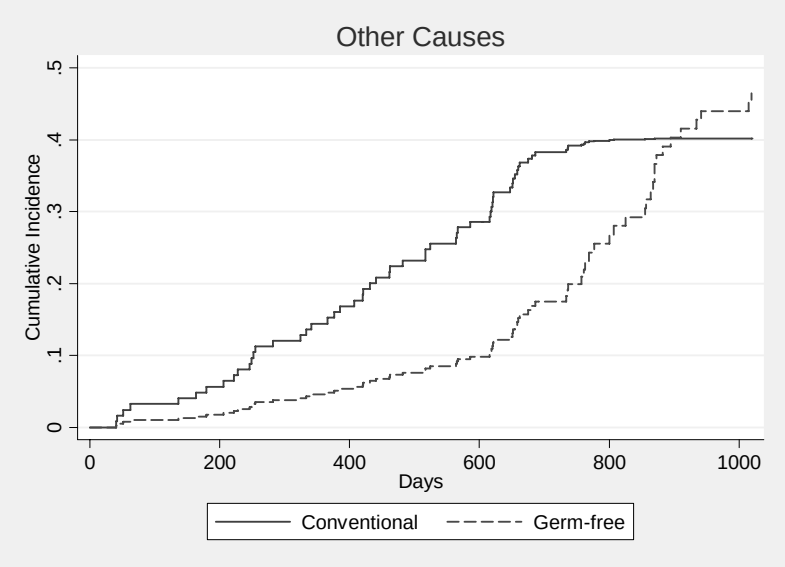

Figure 4.19: Cumulative Incidence estimates for the Other Causes failure type (Rosthoj).

CHAPTER 5

AML EXAMPLE

As another example, we considered an acute myelocytic leukemia (AML) data set obtained from the Department of Internal Medicine and Statistics at The Ohio State University. We are interested in comparing the leukemia free survival of those patients in their first remission to that of patients having two or more remissions. The number of years until AML relapse was recorded for each patient, with death being the competing risk. In addition to those patients experiencing a relapse or death, there was a third group of patients that were still alive and also yet to experience a relapse at the conclusion of the study. Thus, the data consisted of three variables: number of years on study, AML status (relapse, death, alive/no relapse), and disease stage (1^{st} remission, 2+ remissions).

A total of n=334 patients were included in the AML study. Of the 168 patients in their first remission, 35 experienced an AML relapse while 49 died. The remaining 84 patients in their first remission were alive without relapse at the conclusion of the study. Of the 166 patients having two or more remissions, 76 experienced an AML relapse while 51 died. The remaining 39 patients having two or more remissions were alive without relapse at the conclusion of the study.

5.1 DATA ANALYSIS

We will first analyze the AML data using the method described in Gray (1988). The results of Gray's significance tests are given in Table 5.1 (Section 5.2) while the cumulative incidence plots are displayed in Figures 5.1 – 5.3 (Section 5.3). From the p-values given in Table 5.1, we see that there are significant differences (using $\alpha = 0.05$) between the disease stages for those patients experiencing an AML relapse and also for those who were still alive without experiencing a relapse. However, there is not enough evidence to suggest any significant differences between the disease stages in terms of death. By examining Figure 5.1, we see that patients having two or more remissions experience shorter relapse-free survival than those having only one remission.

The significance tests and corresponding cumulative incidence plots due to Fine and Gray (1999) are given in Table 5.2 and Figures 5.4 - 5.6, respectively. Note that these results are in agreement with those produced using Gray's method.

Tables 5.3 and 5.4 contain the results of tests based on Cox's proportional hazards model and the log-rank test, respectively. While they yield the same conclusion as Gray's test with respect to AML relapse, they also show marginally significant results for differences between the disease stages in terms of death.

Computer code for the Tables and Figures presented in Sections 5.2 and 5.3 is given in Appendix B.

Tests:			
	stat	pv	df
1	28.789	< 0.001	1
2	0.159	0.690	1
3	13.335	< 0.001	1

Table 5.1: Gray's test statistics and p-values for comparing the disease stages. Test 1 corresponds to patients experiencing AML relapse. Test 2 corresponds to patients experiencing death. Test 3 corresponds to patients still alive/no relapse.

Tests:			
	coefficient	standard error	2-sided p-value
1	1.055	0.198	< 0.001
2	0.080	0.199	0.690
3	-0.909	0.196	< 0.001

Table 5.2: Fine and Gray's estimated regression coefficients, standard errors, and p-values for comparing the disease stages. Test 1 corresponds to patients experiencing AML relapse. Test 2 corresponds to patients experiencing death. Test 3 corresponds to patients still alive/no relapse.

Tests:				
	Coefficient	standard error	Likelihood ratio test	p-value
1	1.230	0.205		< 0.001
2	0.389	0.202		0.054
3	0.117	0.198		0.556

Table 5.3: Cox proportional hazards coefficient, standard error, and p-value for comparing the disease stages. Test 1 corresponds to patients experiencing AML relapse. Test 2 corresponds to patients experiencing death. Test 3 corresponds to patients still alive/no relapse.

Tests:			
	Chisq	df	p-value
1	40.6	1	< 0.001
2	3.7	1	0.053
3	0.4	1	0.545

Table 5.4: Log Rank test statistic, degrees of freedom, and p-value for comparing the disease stages. Test 1 corresponds to patients experiencing AML relapse. Test 2 corresponds to patients experiencing death. Test 3 corresponds to patients still alive/no relapse.

5.3 FIGURES

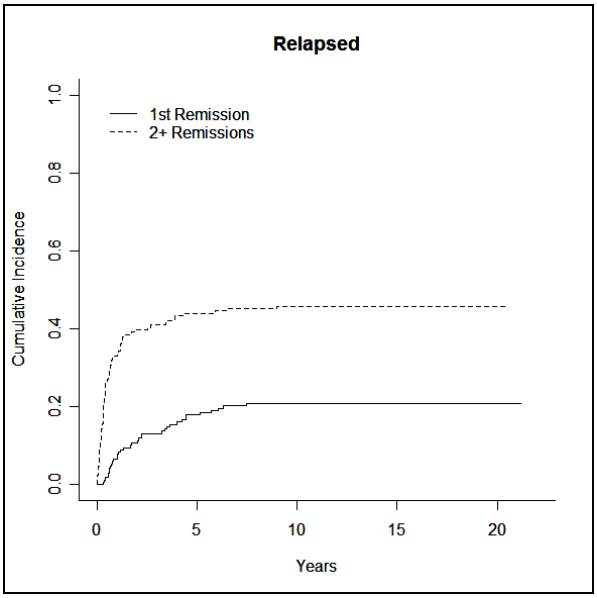

Figure 5.1: Cumulative Incidence estimates for patients experiencing AML relapse (Gray).

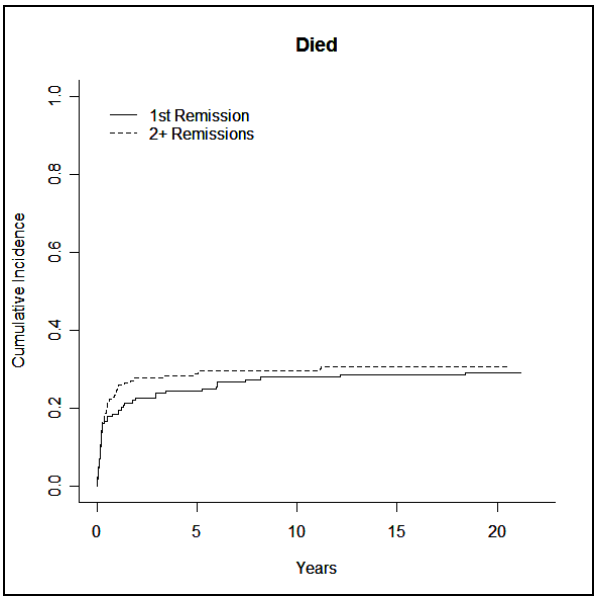

Figure 5.2: Cumulative Incidence estimates for patients experiencing death (Gray).

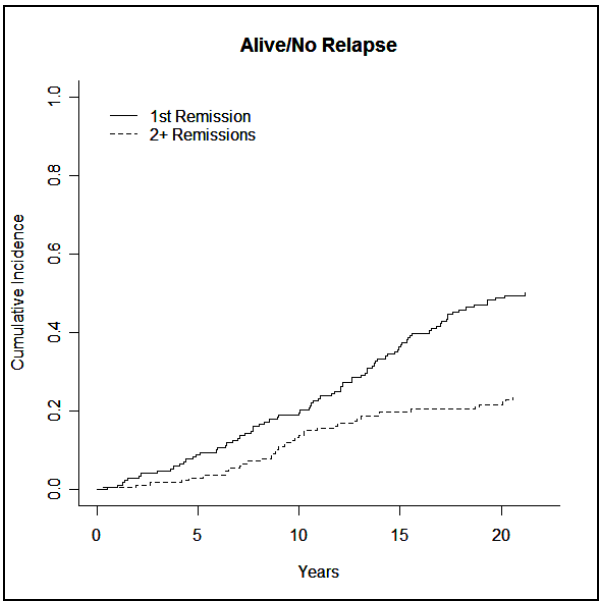

Figure 5.3: Cumulative Incidence estimates for patients still alive/no relapse (Gray).

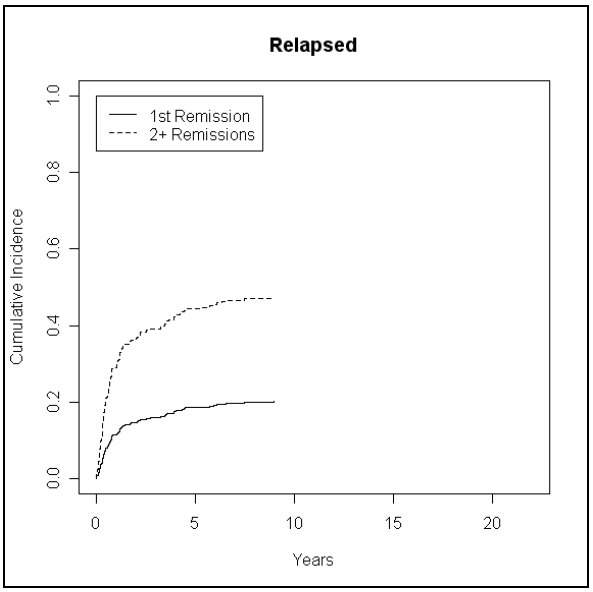

Figure 5.4: Cumulative Incidence estimates for patients experiencing AML relapse (Fine and Gray).

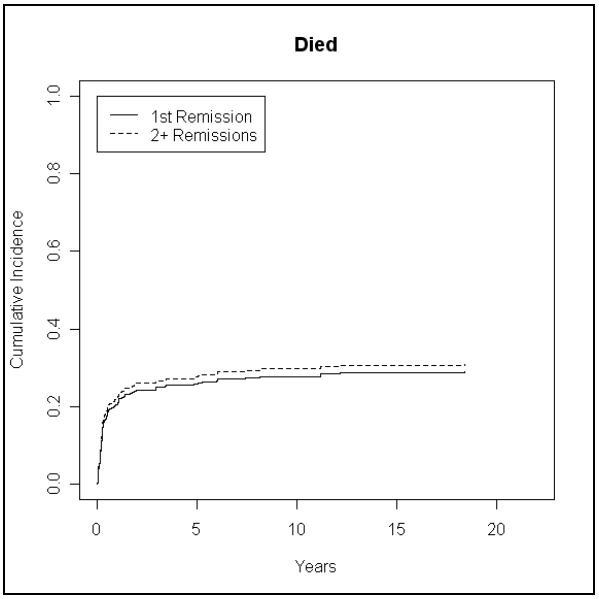

Figure 5.5: Cumulative Incidence estimates for patients experiencing death (Fine and Gray).

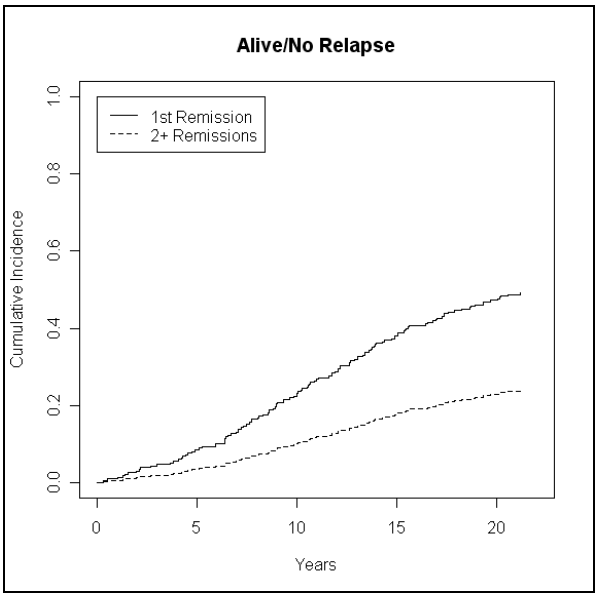

Figure 5.6: Cumulative Incidence estimates for patients still alive/no relapse (Fine and Gray).

CHAPTER 6

SIMULATION

We will begin by generating competing risks data to test for a treatment effect in the presence of independent censoring. We will generate bivariate data for two competing causes of failure from a variety of bivariate distributions. To investigate the proportional hazards setting, we will consider the following distributions: independent exponential, Sarmanov exponential, Marshall-Olkin exponential, Cai-Prentice exponential, exponential with a gamma frailty, independent Weibull, Sarmanov Weibull, Marshall-Olkin Weibull, Cai-Prentice Weibull, and the Weibull with a gamma frailty. To investigate the non-proportional hazards setting, we will consider both the normal and log-normal distributions. We will also consider a variety of treatment effect sizes, differing parameter values for the causes of failure, and various amounts of censoring. Examples to verify the data generation for each distribution are given in Appendix F.

6.1 SIMULATION DESIGN

We will consider a trial comparing a new therapy (treatment) to a standard therapy (control) in the presence of two competing causes of failure where failure due

to cause 1 is of primary interest. We conducted the simulation study to investigate the level of significance and power associated with the four methods commonly used when testing for failure specific treatment effects in the presence of competing risks under a variety of conditions. The test results corresponding to the method based on Cox's proportional hazards model will be denoted by T^{CPH}, while the test results corresponding to the log-rank test will be denoted by T^{LR}. Similary, the test results based on Gray's test and the test developed by Fine and Gray will be denoted by T^{G} and T^{FG}, respectively. Simulations were performed using 1000 replications generated from the bivariate models previously described with a variety of parameters and censoring probabilities. The Tables present the empirical rejection probabilities (at the 0.05 two-sided significance level) under seven settings corresponding to different combinations of the relative treatment effects (treatment vs. control) on the two competing causes of failure (type 1 and type 2).

(1) treatment has no effect on type 1, and a small benefit on type 2;

(2) treatment has no effect on type 1, and a large benefit on type 2.

(3) treatment has a large benefit on type 1, and no effect on type 2;

(4) treatment has a large benefit on type 1, and a small benefit on type 2;

(5) treatment has a large benefit on type 1, and a large benefit on type 2.

(6) treatment has no effect on type 1, and a small adverse effect on type 2;

(7) treatment has a large benefit on type 1, and a small adverse effect on type 2.

For example, in terms of the underlying latent failure time model, the first panel of Table 7.2 contains results corresponding to a simulated independent bivariate exponential distribution having a treatment mean of 16 and a control mean of 10 for

failure due to cause 1, while having both treatment and control means of 10 for failure due to cause 2. The specific parameters considered for each distribution are given in Appendix D.

6.2 HYPOTHESES AND EXPECTATIONS

All of the tests being considered (T^{CPH}, T^{LR}, T^{G}, and T^{FG}) are applicable under models where (1) the proportional hazards assumption is met, (2) the event times due to the two competing causes of failure are independent, and (3) either no censoring occurs or the censoring times are independent of latent failure times of the competing risks. The test based on Cox's proportional hazards model and the log-rank test have optimal power when the hazard rates are proportional and make the additional assumption of independence. However, Gray's test does not require that either of these assumptions be made, while the test due to Fine and Gray also makes no such assumption of independence. Therfore, in situations satisfying all three of the above conditions, we would expect the test based on Cox's proportional hazards model and the log-rank test to perform better than Gray's test and the test due to Fine and Gray.

When considering bivariate distributions where the correlation between the event times for the competing risks is non-zero or the censoring times are not independent of the event times, the assumptions for the test based on Cox's proportional hazards model and the log-rank test are no longer met. However, Gray's test and the test due to Fine and Gray make no such independence assumptions. Therefore, we would expect these tests to perform well in situations where the independence assumption is violated.

Furthermore, when considering any bivariate distribution with non-proportional hazards, the proportional hazards assumptions for the test based on Cox's proportional hazards model and the test due to Fine and Gray will not be satisfied, while the log-rank test will no longer have optimal power. Gray's test, however, makes no such assumption regarding proportional hazards. For this reason, some authors have suggested that Gray's test be the preferred method when testing for treatment effects in the presence of competing risks data. We will now investigate the performance of all four tests under a variety of situations when the assumptions for each test may, or may not be met.

The computer code for generating the competing risks data sets and computing the power associated with each hypothesis test is given in Appendix C.

CHAPTER 7

MULTIVARIATE EXPONENTIAL DISTRIBUTIONS

We will first considering the situation where the time until each cause of failure follows an exponential distribution. A random variable (RV) X is said to follow an exponential distribution with parameter $\lambda > 0$ if its probability density function (pdf) $f_X(x)$ is given by

$$f_X(x) = \begin{cases} \lambda \exp(-\lambda x), & x \geq 0 \\ 0 & , \text{ otherwise.} \end{cases}$$

We denote this by $X \sim Exponential(\lambda)$. Thus, the cumulative distribution function (cdf) $F_X(x)$, the survival function $S_X(x)$, and the hazard function $h_X(x)$ associated with X are given by

$$F_X(x) = \begin{cases} 1 - \exp(-\lambda x), & x \geq 0 \\ 0 & , \text{ otherwise} \end{cases}$$

$$S_X(x) = \begin{cases} \exp(-\lambda x), & x \geq 0 \\ 1 & , \text{ otherwise,} \end{cases}$$

and

$$h_X(x) = \frac{f_X(x)}{S_X(x)} = \begin{cases} \lambda & , x \geq 0 \\ 0 & , \text{ otherwise} \end{cases}$$

respectively. The expected value and variance of X are given by

$$E(X) = \frac{1}{\lambda}$$

and

$$Var(X) = \frac{1}{\lambda^2},$$

respectively.

To generate a random variable $X \sim Exponential(\lambda)$, we first consider the Probability Integral Transformation (PIT).

Theorem (PIT): Let X have continuous cdf $F_X(x)$ and define the random variable Y as $Y = F_X(X)$. Then Y is uniformly distributed on (0,1), that is, $P(Y \le y) = y, 0 < y < 1$. (Casella and Berger, 2002)

Thus the inverse transformation may be used to generate a random variable for which there exists a closed-form function $g(u)$ such that the transformed variable $Y = g(U)$ has the desired distribution when $U \sim Uniform(0,1)$. Applying the PIT, we have that

$$U = F_X(X) = 1 - \exp(-\lambda X) \sim Uniform(0,1)$$

and the inverse transformation yields

$$X = F_X^{-1}(U) = \frac{-1}{\lambda}\ln(1-U) \sim Exp(\lambda).$$

7.1 INDEPENDENT MULTIVARIATE EXPONENTIAL

7.1.1 DATA GENERATION

We will first use the method described above to generate competing risks data when the times until each cause of failure are independent and identically distributed random variables following an exponential distribution. If we consider

$$U_1, U_2, ..., U_K \overset{iid}{\sim} Uniform(0,1)$$

and we define

$$X_k = \frac{-1}{\lambda}\ln(1-U_k) \text{ for } k = 1,...,K,$$

then we have that $X_1, X_2, ..., X_K \overset{iid}{\sim} Exponential(\lambda)$.

We will now extend this idea to allow for independent, but not identically distributed random variables. Furthermore, we will incorporate a treatment indicator covariate, *Z*, to allow for a treatment effect. For each type of competing risks data, we will assume that the individuals in the study are independent of each other. Therefore, we are able to develop the theory while focusing on a single multivariate observation $\boldsymbol{X_i} = (X_{i1}, ..., X_{iK})'$ and extend the results to the situation involving $i = 1,...,n$ individuals. For simplicity, the subscript *i* will be dropped during all derivations.

Again consider $U_1, U_2, ..., U_K \overset{iid}{\sim} Uniform(0,1)$ and let

$$X_k \mid Z = \frac{-1}{\lambda_k}\exp(-\beta_k Z)\ln(1-U_k) \text{ for } k = 1,...,K.$$

Therefore we have

$$X_k \mid Z \overset{ind}{\sim} Exponential\left(\lambda_k \exp\left(\beta_k Z\right)\right) \text{ for } k = 1,...,K .$$

By incorporating the λ_k parameter, we are allowing the control group for each cause of failure to have its own mean survival time. Additionally, we have included the β_k term to allow for differing treatment effects within each of the *K* causes of failure. Thus, the expected survival time and variance associated with the k^{th} cause of failure for those in the treatment group are given by

$$E\left[X_k \mid Z = 1\right] = \frac{1}{\lambda_k \exp\left(\beta_k\right)} \text{ for } k = 1,...,K$$

and

$$Var\left[X_k \mid Z = 1\right] = \left[\frac{1}{\lambda_k \exp\left(\beta_k\right)}\right]^2 \text{ for } k = 1,...,K ,$$

respectively. While the expected survival time and variance associated with the k^{th} cause of failure for those in the control group are given by

$$E\left[X_k \mid Z = 0\right] = \frac{1}{\lambda_k} \text{ for } k = 1,...,K$$

and

$$Var\left[X_k \mid Z = 0\right] = \left[\frac{1}{\lambda_k}\right]^2 \text{ for } k = 1,...,K ,$$

respectively.

If we assume no censoring in the competing risks setting, the observed survival time will be $X_{(1)} = \min\left(X_1,...,X_K\right)$. Thus it is of use to determine the

distribution of $X_{(1)}$ given Z, or $X_{(1)} \mid Z$. We will consider $S_{(1)}(x \mid Z)$, the survival function associated with $X_{(1)} \mid Z$.

$$\begin{aligned} S_{(1)}(x \mid Z) &= P(X_{(1)} > x \mid Z) \\ &= P(X_1 > x,\ X_2 > x,\ ...,\ X_K > x \mid Z) \\ &= \prod_{k=1}^{K} P(X_k > x \mid Z) \qquad \textit{by independence} \\ &= \prod_{k=1}^{K} \exp\{-\lambda_k \exp(\beta_k Z)x\} \\ &= \exp\left\{-x\left(\sum_{k=1}^{K} \lambda_k \exp(\beta_k Z)\right)\right\} \end{aligned}$$

Therefore

$$X_{(1)} \mid Z \sim \textit{Exponential}(\lambda_m)$$

where

$$\lambda_m = \sum_{k=1}^{K} \lambda_k \exp(\beta_k Z).$$

We now wish to consider the case where independent right censoring may occur. We will generate a random censoring time *C*, independent of the *K* failure times $(X_1, X_2, ..., X_K)$ given Z. Thus *C* will be independent of $X_{(1)} \mid Z$. The censoring times will follow an exponential distribution with parameter λ_c. Note that the censoring should not depend on the patient being assigned to the treatment or control group. Therefore the distribution of *C* will not depend on *Z*. We have $X_{(1)} \mid Z \sim \textit{Exponential}(\lambda_m)$ independent of $C \sim \textit{Exponential}(\lambda_c)$. Thus the joint distribution of $X_{(1)} \mid Z$ and *C* is given by

$$g_{X_{(1)},C}(x, c \mid z) = f_{(1)}(x \mid z) f_C(c)$$

where $f_{(1)}(x \mid z)$ represents the pdf associated with $X_{(1)} \mid Z$ and $f_C(c)$ represents the pdf associated with *C*. The censoring parameter λ_c will be chosen such that it results in an overall probability of censoring $p = P(\textit{censoring}) = P(X_{(1)} > C \mid Z)$. We can compute

$$
\begin{aligned}
p &= \iint_{c<x} g_{X_{(1)},C}(x,c \mid z)\,dc\,dx \\
&= \int_0^\infty \int_0^x \lambda_m \exp(-\lambda_m x)\lambda_c \exp(-\lambda_c c)\,dc\,dx \\
&= \int_0^\infty \lambda_m \exp(-\lambda_m x)\left\{\int_0^x \lambda_c \exp(-\lambda_c c)\,dc\right\}dx \\
&= \int_0^\infty \lambda_m \exp(-\lambda_m x)(1-\exp(-\lambda_c c))\,dx \\
&= \int_0^\infty \lambda_m \exp(-\lambda_m x)\,dx - \int_0^\infty \lambda_m \exp(-x(\lambda_m+\lambda_c))\,dx \\
&= 1-\left(\frac{\lambda_m}{\lambda_m+\lambda_c}\right) \\
&= \frac{\lambda_c}{\lambda_m+\lambda_c}.
\end{aligned}
$$

Solving this for λ_c, we have that the selection of $\lambda_c = \dfrac{p(\lambda_m)}{(1-p)}$ will yield a censoring probability equal to *p*. Thus we will consider $U_c \sim \textit{Uniform}(0,1)$ and let $C = \dfrac{-1}{\lambda_c}\ln(1-U_c)$ in order to achieve the desired censoring mechanism.

7.1.2 DETERMINING SIMULATION PARAMETERS

We will now describe the method used for determining the parameters required in order to simulate data from a desired bivariate exponential distribution where the two causes of failure are independent of one another. Recall that the means, conditional on Z, are given by

$$E[X_k \mid Z] = \frac{1}{\lambda_k \exp(\beta_k Z)} \text{ for } k = 1,2 .$$

Suppose we wish to generate data having control means

$$E[X_k \mid Z = 0] = \mu_{0_k} \text{ for } k = 1,2 .$$

Therefore we have that

$$E[X_k \mid Z = 0] = \mu_{0_k}$$

and

$$E[X_k \mid Z = 0] = \frac{1}{\lambda_k} \text{ for } k = 1,2 .$$

Thus we are able to generate data with the desired means by setting

$$\lambda_k = \frac{1}{\mu_{0_k}} \text{ for } k = 1,2 .$$

Now suppose we wish to generate data for the treatment groups such that they result in treatment means having a specified percentage increase (or decrease) over the control means. We will denote the treatment means by

$$E[X_k \mid Z = 1] = \mu_{1_k} \text{ for } k = 1,2 .$$

The desired percentage increase over the control means will be denoted by r_k for $k = 1,2$. Therefore we have that

$$E[X_k \mid Z=1] = \mu_{1_k} = \mu_{0_k}(1+r_k)$$

and

$$E[X_k \mid Z=1] = \mu_{1_k} = \frac{1}{\lambda_k \exp(\beta_k)} \text{ for } k=1,2.$$

Thus we are able to generate data with the desired means by setting

$$\beta_k = \log\left(\frac{1}{\mu_{0_k}(1+r_k)\lambda_k}\right) \text{ for } k=1,2.$$

This method similarly holds for determining the parameters required to generate data from the Sarmanov bivariate exponential distribution, the Cai-Prentice bivariate exponential distribution, and the bivariate exponential distribution having a gamma frailty.

The computer code for determining the parameters to be used in the simulation study is given in Appendix D. Examples verifying the results of these simulations are given in Appendix F.

7.1.3 SIMULATION SUMMARY

For data having exponential marginals, consider the ratio of the treatment hazard and the control hazard:

$$\frac{h_{X_k}(x \mid Z=1)}{h_{X_k}(x \mid Z=0)} = \frac{\lambda_k \exp(\beta_k)}{\lambda_k} = \exp(\beta_k).$$

Note that this ratio does not depend on the value of *x*, indicating that the hazards are proportional for the exponential distribution.

For the independent bivariate exponential setting, the proportional hazards assumption is met, the event times due to the two competing causes of failure are independent, and the censoring times are independent of latent failure times of the competing risks. Therefore, we would expect the test based on Cox's proportional hazards model (T^{CPH}) and the log-rank test (T^{LR}) to perform better than Gray's test (T^{G}) and the test due to Fine and Gray (T^{FG}). We see in Tables 7.1 – 7.3 that the pair of tests, T^{CPH} and T^{LR}, perform similarly well while the tests T^{G} and T^{FG} also provide similar results to each other.

When the treatment has no effect on type 1 failure and some beneficial effect on type 2 failure (Table 7.1), we would expect patients receiving the treatment to experience type 1 failure (as the first event) more often than those patients receiving the control. Therefore, the incidence of type 1 failures in the treatment group will likely be greater than the incidence of the type 1 failures in the control group. In these cases, it appears that T^{CPH} and T^{LR} are consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring, regardless of whether the beneficial treatment effect on type 2 failure is small or large. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small.

When the treatment has a large beneficial effect on type 1 failure and no effect on type 2 failure (panel 1 of Table 7.2), we would expect patients receiving the control to experience type 1 failure (as the first event) more often than those patients

receiving the treatment. Therefore, the incidence of type 1 failures in the treatment group will likely be less than the incidence of the type 1 failures in the control group. In this case, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. When the effect on type 2 failure is beneficial (panels 2 and 3 of Table 7.2), we see that the power results for T^{CPH} and T^{LR} remain similar to those found when there was no treatment effect on type 2 failure, however the power for T^{G} and T^{FG} drops considerably as the treatment's beneficial effect on type 2 gets larger.

When the treatment has no effect on type 1 failure and an adverse effect on type 2 failure (panel 1 of Table 7.3), we would expect patients receiving the treatment to experience type 2 failure (as the first event) more often than those patients receiving the control. Therefore, the incidence of type 1 failures in the treatment group will likely be less than than the incidence of the type 1 failures in the control group. In these cases, it appears that T^{CPH} and T^{LR} are again consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small and perform quite well when the amount of censoring is large.

When the treatment has a large beneficial effect on type 1 failure and an adverse effect on type 2 failure (panel 2 of Table 7.3), we would expect patients receiving the control to experience type 1 failure (as the first event) more often than those patients receiving the treatment. Therefore, the incidence of type 1 failures in

the treatment group will likely be much smaller than the incidence of the type 1 failures in the control group. In this case, it appears that T^{G} and T^{FG} are now generally the more powerful tests when compared to T^{CPH} and T^{LR}, although T^{CPH} and T^{LR} overtake T^{G} and T^{FG} when the amount of censoring is very high. Again, all tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases.

7.1.4 SIMULATION RESULTS

The computer code for producing the Power Tables is given in Appendix E.

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.049	.051	.074	.072
	cens = 0.2	.042	.043	.055	.060
	cens = 0.5	.040	.040	.038	.051
	cens = 0.8	.057	.048	.044	.036
n = 100	cens = 0	.055	.057	.072	.075
	cens = 0.2	.056	.056	.058	.062
	cens = 0.5	.045	.044	.060	.062
	cens = 0.8	.047	.043	.041	.038
n = 200	cens = 0	.045	.045	.112	.125
	cens = 0.2	.041	.041	.081	.085
	cens = 0.5	.047	.047	.064	.068
	cens = 0.8	.050	.050	.064	.065
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.067	.068	.131	.169
	cens = 0.2	.041	.040	.094	.117
	cens = 0.5	.049	.044	.072	.085
	cens = 0.8	.058	.052	.051	.033
n = 100	cens = 0	.053	.054	.228	.264
	cens = 0.2	.043	.043	.169	.196
	cens = 0.5	.045	.047	.119	.128
	cens = 0.8	.041	.037	.060	.056
n = 200	cens = 0	.046	.048	.435	.483
	cens = 0.2	.047	.046	.322	.351
	cens = 0.5	.057	.057	.201	.228
	cens = 0.8	.051	.051	.093	.095

Table 7.1: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Independent Exponential).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.334	.336	.276	.248
	cens = 0.2	.280	.280	.183	.178
	cens = 0.5	.171	.172	.097	.083
	cens = 0.8	.127	.116	.061	.040
n = 100	cens = 0	.570	.568	.413	.409
	cens = 0.2	.474	.477	.318	.305
	cens = 0.5	.331	.333	.180	.169
	cens = 0.8	.162	.161	.087	.073
n = 200	cens = 0	.883	.883	.716	.701
	cens = 0.2	.787	.788	.552	.538
	cens = 0.5	.596	.599	.321	.315
	cens = 0.8	.274	.276	.124	.116
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.335	.338	.168	.148
	cens = 0.2	.286	.289	.142	.111
	cens = 0.5	.214	.217	.092	.075
	cens = 0.8	.121	.110	.057	.044
n = 100	cens = 0	.593	.595	.272	.245
	cens = 0.2	.532	.534	.202	.169
	cens = 0.5	.343	.345	.108	.089
	cens = 0.8	.163	.164	.073	.061
n = 200	cens = 0	.889	.890	.468	.429
	cens = 0.2	.816	.817	.332	.307
	cens = 0.5	.626	.626	.172	.161
	cens = 0.8	.281	.287	.081	.069
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.353	.365	.083	.051
	cens = 0.2	.301	.307	.063	.044
	cens = 0.5	.221	.233	.057	.049
	cens = 0.8	.113	.112	.051	.043
n = 100	cens = 0	.637	.645	.099	.074
	cens = 0.2	.521	.530	.067	.057
	cens = 0.5	.389	.399	.063	.065
	cens = 0.8	.188	.191	.056	.044
n = 200	cens = 0	.906	.908	.128	.086
	cens = 0.2	.826	.827	.085	.064
	cens = 0.5	.635	.641	.052	.044
	cens = 0.8	.293	.298	.049	.047

Table 7.2: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Independent Exponential).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.046	.046	.069	.068
	cens = 0.2	.042	.043	.062	.067
	cens = 0.5	.042	.042	.056	.056
	cens = 0.8	.051	.046	.040	.035
n = 100	cens = 0	.063	.062	.092	.093
	cens = 0.2	.046	.047	.080	.083
	cens = 0.5	.038	.038	.060	.066
	cens = 0.8	.044	.041	.054	.044
n = 200	cens = 0	.052	.052	.149	.154
	cens = 0.2	.056	.057	.129	.139
	cens = 0.5	.044	.044	.090	.091
	cens = 0.8	.041	.039	.057	.060
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.302	.301	.373	.383
	cens = 0.2	.249	.246	.265	.274
	cens = 0.5	.145	.142	.146	.140
	cens = 0.8	.119	.097	.086	.070
n = 100	cens = 0	.564	.560	.673	.675
	cens = 0.2	.473	.468	.521	.527
	cens = 0.5	.312	.309	.308	.304
	cens = 0.8	.151	.139	.129	.113
n = 200	cens = 0	.848	.847	.922	.924
	cens = 0.2	.761	.760	.810	.810
	cens = 0.5	.536	.535	.539	.539
	cens = 0.8	.290	.287	.240	.235

Table 7.3: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Independent Exponential).

7.2 SARMANOV BIVARIATE EXPONENTIAL

7.2.1 DATA GENERATION

Sarmanov (1966) described a family of bivariate densities. Suppose that $f_1(x_1)$ and $f_2(x_2)$ are univariate probability density functions with supports defined on $A_1 \subseteq R$ and $A_2 \subseteq R$, respectively. Let $\phi_i(t)$ be bounded nonconstant functions such that $\int_{-\infty}^{\infty} \phi_i(t) f_i(t) dt = 0$, for $i = 1,2$. These will be referred to as the "mixing" functions. The function defined by

$$h(x_1, x_2) = f_1(x_1) f_2(x_2)\{1 + \omega \phi_1(x_1) \phi_2(x_2)\}$$

is then a bivariate joint density with marginals specified by $f_1(x_1)$ and $f_2(x_2)$, provided that ω is a real number satisfying the condition that $1 + \omega \phi_1(x_1) \phi_2(x_2) \geq 0$ for all x_1 and x_2.

In the following Theorems, given in Lee (1996), define $\upsilon_i = \int_{-\infty}^{\infty} t \phi_i(t) f_i(t) dt$, $\mu_i = \int_{-\infty}^{\infty} t f_i(t) dt$, and $\sigma_i^2 = \int_{-\infty}^{\infty} (t - \mu_i)^2 f_i(t) dt$, if they exist, for $i = 1,2$.

Theorem 1: Assume that the vector (X_1, X_2) has a joint pdf $h(x_1, x_2)$ as defined above. The conditional distribution function of X_2 given $X_1 = x_1$ is

$$P(X_2 \leq x_2 \mid X_1 = x_1) = F_2(x_2) - \omega \phi_1(x_1) \int_{x_2}^{\infty} f_2(t) \phi_2(t) dt$$

where $F_2(x_2) = P(X_2 \leq x_2)$ is the marginal distribution function of X_2.

Theorem 2: Let (X_1, X_2) be a bivariate random vector with joint pdf $h(x_1, x_2)$ as defined above.

(a) The Pearson correlation coefficient of X_1 and X_2, if it exists, is given by

$$\rho = Corr(X_1, X_2) = \frac{\omega \upsilon_1 \upsilon_2}{\sigma_1 \sigma_2}.$$

Thus X_1 and X_2 are independent if $\omega = 0$.

(b) The correlation coefficient ρ of X_1 and X_2 is bounded by

$$|\rho| = |\omega| \sqrt{E[\phi_1^2(X_1)] E[\phi_2^2(X_2)]}$$

Corollary: Assume that $f_i(x_i)$ are defined on [0,∞). Let $L_i(t) = \int_0^\infty \exp(-tx_i) f_i(x_i) dx_i$ denote the Laplace transform of f_i for $i = 1,2$. Define $\phi_i(x_i) = \exp(-x_i) - L_i(1)$ for $x_i \geq 0$. Then the function $h(x_1, x_2)$ is a bivariate density with designated marginals $f_i(x_i)$, $i = 1,2$. The correlation coefficient of X_1 and X_2, if it exists, is given by $\rho = \frac{\omega[-L_1'(1) - L_1(1)\mu_1][-L_2'(1) - L_2(1)\mu_2]}{\sigma_1 \sigma_2}$, where L_i' is the first derivative of the function L_i and ω is a real number which is bounded above by $1/\max\{L_1(1)[1 - L_2(1)], L_2(1)[1 - L_1(1)]\}$.

We now wish to form a bivariate density having exponential marginals given by

$$f_i(x_i) = \begin{cases} \lambda_i \exp(-\lambda_i x_i), & x_i \geq 0 \\ 0 & , \textit{otherwise.} \end{cases}$$

The Laplace transform of f_i has the form $L_i(t) = \left(1 + \frac{t}{\lambda_i}\right)^{-1} = \frac{\lambda_i}{t + \lambda_i}$. Since $f_i(x_i)$ are defined on [0,∞), we will apply the Corollary to find the mixing functions given by $\phi_i(x_i) = \exp(-x_i) - L_i(1) = \exp(-x_i) - \frac{\lambda_i}{1+\lambda_i}$. Thus we can construct bivariate exponential data having a density given by

$$h(x_1, x_2) = \lambda_1 \exp(-\lambda_1 x_1)\lambda_2 \exp(-\lambda_2 x_2) \left\{1 + \omega\left(\exp(-x_1) - \frac{\lambda_1}{1+\lambda_1}\right)\left(\exp(-x_2) - \frac{\lambda_2}{1+\lambda_2}\right)\right\}.$$

For this distribution, we have that

$$\mu_i = \sigma_i = \frac{1}{\lambda_i}$$

and

$$\begin{aligned}
\upsilon_i &= \int_{-\infty}^{\infty} t\phi_i(t) f_i(t) dt \\
&= \int_0^{\infty} t\left(\exp(-t) - \frac{\lambda_i}{1+\lambda_i}\right)\lambda_i \exp(-\lambda_i t) dt \\
&= \int_0^{\infty} t\lambda_i \exp\{-t(1+\lambda_i)\} dt - \int_0^{\infty} t\frac{\lambda_i^2}{1+\lambda_i}\exp(-\lambda_i t) dt \\
&= \frac{\lambda_i}{1+\lambda_i}\int_0^{\infty} t(1+\lambda_i)\exp\{-t(1+\lambda_i)\} dt - \frac{\lambda_i}{1+\lambda_i}\int_0^{\infty} t\lambda_i \exp\{-t\lambda_i\} dt \\
&= \frac{\lambda_i}{1+\lambda_i}\left(\frac{1}{1+\lambda_i}\right) - \frac{\lambda_i}{1+\lambda_i}\left(\frac{1}{\lambda_i}\right) \\
&= \frac{-1}{(1+\lambda_i)^2}.
\end{aligned}$$

Thus $\rho = Corr(X_1, X_2) = \dfrac{\omega \upsilon_1 \upsilon_2}{\sigma_1 \sigma_2} = \dfrac{\omega \lambda_1 \lambda_2}{(1+\lambda_1)^2 (1+\lambda_2)^2}$ where ω is a real number

which is bounded above by $1/\max\left\{ \dfrac{\lambda_1}{(1+\lambda_1)(1+\lambda_2)}, \dfrac{\lambda_2}{(1+\lambda_1)(1+\lambda_2)} \right\}$.

We will now generate data from the bivariate exponential density given by $h(x_1, x_2)$. To allow for a treatment effect, we will generate failure times due to cause 1 by considering $U_1 \sim Uniform(0,1)$ and letting

$$X_1 \mid Z = \frac{-1}{\lambda_1} \exp(-\beta_1 Z) \ln(1 - U_1).$$

Therefore we have

$$X_1 \mid Z \sim Exponential(\lambda_1 \exp(\beta_1 Z)).$$

To obtain the failure times due to cause 2, we will first let $\lambda_i^* = \lambda_i \exp(\beta_i Z)$ for $i = 1,2$ and consider the conditional distribution function of X_2 given $X_1 = x_1$ in Theorem 1,

$$\begin{aligned}
F_{X_2|X_1}(x_2 \mid x_1) &= P(X_2 \le x_2 \mid X_1 = x_1) \\
&= F_2(x_2) - \omega \phi_1(x_1) \int_{x_2}^{\infty} f_2(t) \phi_2(t) dt \\
&= \left[1 - \exp(-\lambda_2^* x_2)\right] \\
&\quad - \omega \left[\exp(-x_1) - \frac{\lambda_1^*}{1+\lambda_1^*} \right] \int_{x_2}^{\infty} \lambda_2^* \exp(-\lambda_2^* t) \left[\exp(-t) - \frac{\lambda_2^*}{1+\lambda_2^*} \right] dt \\
&= \left[1 - \exp(-\lambda_2^* x_2)\right] \\
&\quad - \omega \left[\exp(-x_1) - \frac{\lambda_1^*}{1+\lambda_1^*} \right] \left[\exp(-x_2) - 1\right] \frac{\lambda_2^* \exp(-\lambda_2^* x_2)}{1+\lambda_2^*}.
\end{aligned}$$

Thus we will generate $U_2 \sim Uniform(0,1)$ and solve $U_2 = F_{X_2|X_1}(X_2 \mid x_1)$ for X_2 to produce failure times associated with cause 2. The solution to the equation

$U_2 = F_{X_2|X_1}(X_2 \mid x_1)$ will be obtained using an iterative method described in Brent (1973). Thus the marginal distribution for the failure times associated with cause 2 will be given by

$$X_2 \mid Z \sim Exponential(\lambda_2 \exp(\beta_2 Z))$$

and the pair of failure times (X_1, X_2) will follow the bivariate exponential density given by $h(x_1, x_2)$ with parameters λ_1^*, λ_2^*, and ω.

We will then generate censoring variables (C_1, C_2) independently of one another and independently of $(X_1, X_2) \mid Z$ where C_k is the censoring time associated with failure time X_k for $k = 1,2$. The censoring variables will follow exponential distributions with $C_k \sim Exp(\lambda_{c_k})$ for $k = 1,2$ where the λ_{c_k} are chosen in such a way that they result in censoring probabilities

$$p_1 = P(C_1 < X_1 \mid Z)$$

and

$$p_2 = P(C_2 < X_2 \mid Z).$$

Since $X_k \mid Z$ is independent of C_k for $k = 1,2$, the censoring parameters λ_{c_1} and λ_{c_2} will be chosen in a manner similar to the one previously described for the independent bivariate exponential distribution. Defining $\lambda_{c_1} = \frac{p_1(\lambda_1^*)}{(1-p_1)}$ and $\lambda_{c_2} = \frac{p_2(\lambda_2^*)}{(1-p_2)}$ will result in the desired censoring probabilities of p_1 and p_2, respectively.

7.2.2 SIMULATION SUMMARY

Not surprisingly, the results for the Sarmanov bivariate exponential having a correlation coefficient of zero (Tables 7.4 - 7.6) are very similar to those of the independent bivariate exponential distribution given above. However, we will also consider the case where the competing risks are positively correlated. Thus violating the independence assumption inherent in the test based on Cox's proportional hazards model and the log-rank test.

In Table 7.4, it appears that T^{CPH} and T^{LR} are consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring, regardless of whether the beneficial treatment effect on type 2 failure is small or large. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small. When we allowed for the competing risks to be positively correlated (Tables 7.7, 7.10, and 7.13), we again saw similar trends at each level of correlation. Furthermore the results did not appear to vary greatly across the correlations being considered.

In Table 7.5, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. When the effect on type 2 failure is beneficial (panels 2 and 3 of Table 7.5), we see that the power results for T^{CPH} and T^{LR} remain similar to those found when there was no treatment effect on type 2 failure, however the power for T^{G} and T^{FG} drop considerably as the treatment's beneficial effect on

type 2 failure gets larger. When we allowed for the competing risks to be positively correlated (Tables 7.8, 7.11, and 7.14), we again saw similar trends at each level of correlation. All tests also appeared to be more powerful when the competing risks were positively correlated than when they were independent.

In panel 1 of Table 7.6, it appears that T^{CPH} and T^{LR} are again consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small and perform quite well when the amount of censoring is large. When we allowed for the competing risks to be positively correlated (panel 1 of Tables 7.9, 7.12, and 7.15), we again saw similar trends at each level of correlation. Furthermore the results did not appear to vary greatly across the correlations being considered.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 7.6, it appears that T^{G} and T^{FG} are now generally the more powerful tests when compared to T^{CPH} and T^{LR}, although T^{CPH} and T^{LR} overtake T^{G} and T^{FG} when the amount of censoring is high. Again, all tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. When we allowed for the competing risks to be positively correlated (panel 2 of Tables 7.9, 7.12, and 7.15), we again saw similar trends at each level of correlation. All tests also appeared to be more powerful when the competing risks were positively correlated than when they were independent.

7.1.3 SIMULATION RESULTS

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.047	.049	.062	.060
	cens = 0.2	.055	.057	.055	.058
	cens = 0.5	.046	.047	.043	.046
	cens = 0.8	.040	.032	.030	.028
n = 100	cens = 0	.053	.054	.073	.091
	cens = 0.2	.046	.046	.056	.065
	cens = 0.5	.049	.049	.057	.056
	cens = 0.8	.065	.058	.059	.054
n = 200	cens = 0	.052	.052	.120	.132
	cens = 0.2	.055	.053	.095	.099
	cens = 0.5	.046	.046	.080	.083
	cens = 0.8	.056	.055	.053	.049
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.042	.044	.108	.160
	cens = 0.2	.051	.053	.096	.120
	cens = 0.5	.058	.052	.067	.088
	cens = 0.8	.064	.056	.062	.044
n = 100	cens = 0	.055	.056	.239	.276
	cens = 0.2	.051	.051	.161	.199
	cens = 0.5	.045	.043	.107	.116
	cens = 0.8	.052	.048	.075	.075
n = 200	cens = 0	.056	.056	.449	.494
	cens = 0.2	.044	.044	.301	.344
	cens = 0.5	.042	.041	.192	.213
	cens = 0.8	.046	.045	.105	.105

Table 7.4: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Sarmanov Exponential, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.322	.327	.254	.230
	cens = 0.2	.246	.245	.160	.147
	cens = 0.5	.192	.193	.112	.099
	cens = 0.8	.106	.099	.050	.039
n = 100	cens = 0	.590	.591	.459	.428
	cens = 0.2	.495	.496	.336	.324
	cens = 0.5	.310	.311	.166	.143
	cens = 0.8	.178	.177	.102	.081
n = 200	cens = 0	.878	.879	.714	.699
	cens = 0.2	.798	.800	.556	.543
	cens = 0.5	.579	.579	.317	.307
	cens = 0.8	.305	.310	.151	.136
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.360	.370	.174	.152
	cens = 0.2	.273	.277	.143	.120
	cens = 0.5	.170	.170	.075	.060
	cens = 0.8	.110	.104	.051	.035
n = 100	cens = 0	.601	.603	.268	.234
	cens = 0.2	.530	.535	.188	.165
	cens = 0.5	.335	.334	.120	.106
	cens = 0.8	.170	.173	.077	.062
n = 200	cens = 0	.883	.883	.462	.433
	cens = 0.2	.791	.793	.300	.269
	cens = 0.5	.594	.603	.182	.168
	cens = 0.8	.280	.286	.086	.075
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.374	.380	.083	.057
	cens = 0.2	.289	.305	.072	.053
	cens = 0.5	.201	.209	.044	.035
	cens = 0.8	.136	.137	.048	.039
n = 100	cens = 0	.605	.608	.097	.071
	cens = 0.2	.525	.531	.058	.039
	cens = 0.5	.361	.369	.062	.052
	cens = 0.8	.163	.169	.044	.043
n = 200	cens = 0	.902	.903	.127	.094
	cens = 0.2	.813	.816	.085	.060
	cens = 0.5	.624	.631	.070	.056
	cens = 0.8	.296	.303	.050	.047

Table 7.5: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Sarmanov Exponential, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.056	.057	.074	.088
	cens = 0.2	.054	.054	.054	.061
	cens = 0.5	.045	.043	.047	.051
	cens = 0.8	.069	.055	.054	.038
n = 100	cens = 0	.043	.044	.091	.105
	cens = 0.2	.056	.054	.090	.105
	cens = 0.5	.047	.047	.052	.048
	cens = 0.8	.057	.050	.055	.045
n = 200	cens = 0	.053	.053	.130	.152
	cens = 0.2	.051	.053	.117	.124
	cens = 0.5	.037	.036	.087	.096
	cens = 0.8	.055	.057	.073	.072
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.310	.310	.383	.400
	cens = 0.2	.282	.280	.306	.307
	cens = 0.5	.193	.188	.182	.164
	cens = 0.8	.109	.095	.076	.063
n = 100	cens = 0	.555	.553	.680	.689
	cens = 0.2	.458	.460	.506	.513
	cens = 0.5	.322	.319	.309	.308
	cens = 0.8	.156	.151	.120	.105
n = 200	cens = 0	.848	.847	.923	.925
	cens = 0.2	.754	.753	.793	.791
	cens = 0.5	.557	.555	.532	.533
	cens = 0.8	.269	.266	.220	.212

Table 7.6: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Sarmanov Exponential, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.062	.063	.066	.074
	cens = 0.2	.054	.054	.064	.061
	cens = 0.5	.056	.056	.061	.059
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.055	.055	.097	.108
	cens = 0.2	.054	.053	.069	.072
	cens = 0.5	.040	.040	.057	.066
	cens = 0.8	.061	.051	.055	.045
n = 200	cens = 0	.059	.059	.138	.144
	cens = 0.2	.047	.047	.120	.128
	cens = 0.5	.049	.049	.085	.088
	cens = 0.8	.057	.054	.057	.052
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.059	.059	.168	.223
	cens = 0.2	.053	.054	.135	.165
	cens = 0.5	.052	.049	.087	.106
	cens = 0.8	.058	.042	.052	.058
n = 100	cens = 0	.047	.046	.315	.373
	cens = 0.2	.040	.041	.233	.273
	cens = 0.5	.051	.050	.163	.183
	cens = 0.8	.065	.057	.080	.070
n = 200	cens = 0	.050	.048	.592	.635
	cens = 0.2	.051	.050	.444	.492
	cens = 0.5	.056	.055	.285	.310
	cens = 0.8	.045	.044	.104	.107

* Programs did not converge due to sparse data.

Table 7.7: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Sarmanov Exponential, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.424	.425	.353	.327
	cens = 0.2	.359	.365	.260	.231
	cens = 0.5	.221	.216	.121	.109
	cens = 0.8	.108	.088	.071	.059
n = 100	cens = 0	.706	.708	.575	.553
	cens = 0.2	.596	.601	.399	.381
	cens = 0.5	.402	.403	.218	.209
	cens = 0.8	.098	.093	.065	.046
n = 200	cens = 0	.944	.944	.859	.845
	cens = 0.2	.896	.899	.667	.648
	cens = 0.5	.650	.652	.373	.365
	cens = 0.8	.156	.155	.089	.075
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.452	.457	.221	.198
	cens = 0.2	.377	.381	.153	.131
	cens = 0.5	.285	.294	.103	.077
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.752	.755	.386	.357
	cens = 0.2	.666	.667	.259	.218
	cens = 0.5	.496	.501	.153	.127
	cens = 0.8	.256	.256	.110	.077
n = 200	cens = 0	.973	.973	.599	.564
	cens = 0.2	.923	.924	.452	.412
	cens = 0.5	.797	.799	.265	.230
	cens = 0.8	.453	.462	.176	.147
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.444	.449	.094	.071
	cens = 0.2	.384	.393	.080	.053
	cens = 0.5	.240	.252	.062	.053
	cens = 0.8	.180	.162	.059	.080
n = 100	cens = 0	.742	.748	.109	.070
	cens = 0.2	.637	.650	.077	.058
	cens = 0.5	.478	.490	.069	.043
	cens = 0.8	.253	.265	.069	.043
n = 200	cens = 0	.968	.969	.151	.110
	cens = 0.2	.898	.902	.093	.063
	cens = 0.5	.781	.788	.070	.057
	cens = 0.8	.442	.458	.067	.048

* Programs did not converge due to sparse data.

Table 7.8: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Sarmanov Exponential, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.051	.052	.080	.086
	cens = 0.2	.051	.053	.061	.073
	cens = 0.5	.052	.048	.060	.065
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.045	.046	.122	.125
	cens = 0.2	.051	.054	.107	.124
	cens = 0.5	.045	.044	.068	.075
	cens = 0.8	.056	.049	.049	.034
n = 200	cens = 0	.046	.048	.205	.219
	cens = 0.2	.045	.045	.149	.167
	cens = 0.5	.041	.041	.111	.124
	cens = 0.8	.041	.040	.053	.050
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.424	.424	.537	.543
	cens = 0.2	.363	.363	.417	.420
	cens = 0.5	.294	.287	.257	.230
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.696	.695	.822	.831
	cens = 0.2	.670	.665	.707	.711
	cens = 0.5	.490	.484	.459	.457
	cens = 0.8	.270	.247	.205	.161
n = 200	cens = 0	.935	.933	.986	.986
	cens = 0.2	.919	.918	.947	.951
	cens = 0.5	.775	.775	.766	.765
	cens = 0.8	.489	.486	.423	.384

* Programs did not converge due to sparse data.

Table 7.9: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Sarmanov Exponential, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.059	.061	.068	.076
	cens = 0.2	.058	.060	.059	.065
	cens = 0.5	.056	.052	.040	.041
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.048	.048	.110	.113
	cens = 0.2	.047	.047	.076	.089
	cens = 0.5	.061	.059	.071	.072
	cens = 0.8	.063	.045	.041	.071
n = 200	cens = 0	.054	.054	.168	.179
	cens = 0.2	.046	.047	.134	.147
	cens = 0.5	.053	.051	.088	.099
	cens = 0.8	.050	.047	.050	.039
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.033	.033	.161	.206
	cens = 0.2	.051	.051	.128	.164
	cens = 0.5	.055	.053	.079	.093
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.063	.063	.310	.369
	cens = 0.2	.053	.052	.241	.295
	cens = 0.5	.056	.057	.152	.177
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.052	.052	.606	.654
	cens = 0.2	.044	.045	.469	.522
	cens = 0.5	.053	.052	.277	.306
	cens = 0.8	.064	.055	.084	.098

* Programs did not converge due to sparse data.

Table 7.10: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Sarmanov Exponential, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.447	.452	.324	.297
	cens = 0.2	.301	.302	.210	.181
	cens = 0.5	.093	.093	.067	.060
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.721	.721	.572	.546
	cens = 0.2	.560	.559	.371	.351
	cens = 0.5	.189	.189	.108	.089
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.950	.949	.849	.840
	cens = 0.2	.855	.856	.593	.572
	cens = 0.5	.319	.319	.182	.164
	cens = 0.8	.051	.050	.049	.046
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.437	.444	.176	.142
	cens = 0.2	.331	.338	.112	.099
	cens = 0.5	.215	.228	.070	.053
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.767	.769	.273	.232
	cens = 0.2	.637	.646	.186	.162
	cens = 0.5	.416	.428	.077	.063
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.963	.963	.478	.456
	cens = 0.2	.887	.893	.249	.216
	cens = 0.5	.653	.658	.106	.090
	cens = 0.8	.270	.284	.058	.043
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.445	.453	.071	.054
	cens = 0.2	.366	.380	.048	.046
	cens = 0.5	.215	.233	.044	.048
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.734	.738	.070	.048
	cens = 0.2	.660	.668	.051	.042
	cens = 0.5	.377	.398	.038	.052
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.970	.971	.064	.047
	cens = 0.2	.885	.889	.053	.049
	cens = 0.5	.676	.688	.046	.072
	cens = 0.8	.277	.308	.039	.036

* Programs did not converge due to sparse data.

Table 7.11: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Sarmanov Exponential, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.060	.060	.107	.118
	cens = 0.2	.057	.055	.093	.106
	cens = 0.5	.050	.049	.060	.057
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.052	.054	.171	.185
	cens = 0.2	.040	.040	.112	.124
	cens = 0.5	.046	.046	.092	.098
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.060	.060	.275	.299
	cens = 0.2	.054	.055	.204	.223
	cens = 0.5	.043	.040	.120	.128
	cens = 0.8	.056	.049	.047	.039
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.460	.456	.539	.542
	cens = 0.2	.376	.370	.356	.363
	cens = 0.5	.247	.240	.182	.168
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.732	.729	.829	.833
	cens = 0.2	.651	.649	.621	.635
	cens = 0.5	.422	.420	.337	.334
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.968	.968	.987	.988
	cens = 0.2	.913	.913	.907	.903
	cens = 0.5	.735	.735	.600	.587
	cens = 0.8	.317	.296	.200	.161

* Programs did not converge due to sparse data.

Table 7.12: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Sarmanov Exponential, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.058	.060	.079	.081
	cens = 0.2	.051	.050	.065	.066
	cens = 0.5	.058	.056	.043	.047
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.045	.046	.073	.087
	cens = 0.2	.048	.049	.083	.086
	cens = 0.5	.052	.052	.057	.059
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.047	.047	.114	.126
	cens = 0.2	.058	.058	.108	.119
	cens = 0.5	.041	.041	.071	.074
	cens = 0.8	.055	.047	.048	.040
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.065	.065	.130	.157
	cens = 0.2	.043	.043	.096	.129
	cens = 0.5	.060	.054	.080	.090
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.041	.043	.231	.264
	cens = 0.2	.037	.037	.181	.224
	cens = 0.5	.049	.049	.109	.135
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.054	.054	.442	.503
	cens = 0.2	.053	.053	.353	.411
	cens = 0.5	.043	.042	.192	.221
	cens = 0.8	.067	.062	.103	.097

* Programs did not converge due to sparse data.

Table 7.13: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Sarmanov Exponential, corr = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.400	.399	.275	.242
	cens = 0.2	.240	.238	.157	.132
	cens = 0.5	.072	.070	.053	.044
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.656	.657	.434	.400
	cens = 0.2	.426	.426	.247	.212
	cens = 0.5	.107	.104	.077	.062
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.938	.938	.765	.738
	cens = 0.2	.720	.721	.442	.414
	cens = 0.5	.160	.159	.108	.096
	cens = 0.8	.057	.057	.050	.050
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.471	.476	.199	.162
	cens = 0.2	.393	.402	.105	.078
	cens = 0.5	.198	.211	.055	.045
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.788	.790	.268	.244
	cens = 0.2	.649	.654	.136	.116
	cens = 0.5	.414	.422	.062	.051
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.970	.970	.442	.410
	cens = 0.2	.918	.921	.214	.190
	cens = 0.5	.678	.686	.087	.079
	cens = 0.8	.235	.249	.050	.038
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.454	.459	.093	.052
	cens = 0.2	.368	.377	.055	.051
	cens = 0.5	.215	.229	.045	.056
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.774	.777	.091	.067
	cens = 0.2	.650	.657	.046	.048
	cens = 0.5	.418	.445	.037	.052
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.964	.966	.082	.060
	cens = 0.2	.902	.905	.047	.053
	cens = 0.5	.688	.699	.059	.084
	cens = 0.8	.231	.263	.031	.030

* Programs did not converge due to sparse data.

Table 7.14: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Sarmanov Exponential, corr = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.045	.046	.066	.087
	cens = 0.2	.035	.035	.063	.072
	cens = 0.5	.053	.050	.069	.062
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.043	.043	.115	.129
	cens = 0.2	.057	.059	.088	.102
	cens = 0.5	.053	.051	.079	.087
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.059	.059	.199	.217
	cens = 0.2	.059	.060	.154	.179
	cens = 0.5	.059	.060	.109	.112
	cens = 0.8	.057	.042	.044	.041
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.443	.443	.507	.512
	cens = 0.2	.370	.370	.314	.328
	cens = 0.5	.242	.236	.138	.112
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.775	.776	.823	.827
	cens = 0.2	.625	.624	.573	.577
	cens = 0.5	.422	.416	.277	.265
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.975	.975	.977	.977
	cens = 0.2	.916	.916	.870	.872
	cens = 0.5	.673	.672	.473	.467
	cens = 0.8	.244	.220	.128	.129

* Programs did not converge due to sparse data.

Table 7.15: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Sarmanov Exponential, corr = 0.8).

7.3 MARSHALL-OLKIN BIVARIATE EXPONENTIAL

7.3.1 DATA GENERATION

To further investigate the situation where the *K* causes of failure are not independent, I will now consider the Marshall-Olkin Bivariate Exponential (MOBVE) distribution, first introduced by Marshall and Olkin (1967). A pair of random variables (X_1, X_2) are said to follow the MOBVE distribution if their joint survival function is given by

$$S_{MO}(x_1, x_2) = \exp[-\lambda_1 x_1 - \lambda_2 x_2 - \lambda_{12} \max(x_1, x_2)],$$

where $x_1, x_2 > 0$, $\lambda_1, \lambda_2 > 0$, and $\lambda_{12} \geq 0$. Note that X_1 and X_2 will be independent only when $\lambda_{12} = 0$, and dependent otherwise. It is also of interest to note that the MOBVE distribution allows for the case where $X_1 = X_2$. In fact,

$$P(X_1 = X_2) = \frac{\lambda_{12}}{\lambda}$$

where

$$\lambda = \lambda_1 + \lambda_2 + \lambda_{12}.$$

In the competing risks setting, the case where X_1 and X_2 are simultaneously observed will be treated as the 3rd cause of failure. When $(X_1, X_2) \sim MOBVE(\lambda_1, \lambda_2, \lambda_{12})$, both X_1 and X_2 will have exponential marginal distributions given by $X_1 \sim Exponential(\lambda_1 + \lambda_{12})$ and $X_2 \sim Exponential(\lambda_2 + \lambda_{12})$. When generating data, Barlow and Proschan (1981) note a useful property inherent in

the MOBVE distribution: If $(X_1, X_2) \sim MOBVE(\lambda_1, \lambda_2, \lambda_{12})$, then there exist independent exponential random variables Y_1, Y_2, and Y_{12} such that $X_1 = \min(Y_1, Y_{12})$ and $X_2 = \min(Y_2, Y_{12})$.

Therefore to simulate MOBVE random variables, we will begin by generating $U_1, U_2, U_{12} \overset{iid}{\sim} Uniform(0,1)$ and letting

$$Y_1 = \frac{-1}{\lambda_1}\ln(1-U_1),$$

$$Y_2 = \frac{-1}{\lambda_2}\ln(1-U_2),$$

and

$$Y_{12} = \frac{-1}{\lambda_{12}}\ln(1-U_{12}).$$

Thus Y_1, Y_2, and Y_{12} will be independent exponential random variables with parameters λ_1, λ_2, and λ_{12}, respectively. We then obtain dependent random variables X_1 and X_2 by letting

$$X_1 = \min(Y_1, Y_{12})$$

and

$$X_2 = \min(Y_2, Y_{12})$$

such that $(X_1, X_2) \sim MOBVE(\lambda_1, \lambda_2, \lambda_{12})$.

If we again assume no censoring, the observed survival time will be

$$X_{(1)} = \min(X_1, .X_2) = \min(Y_1, Y_2, Y_{12}).$$

Since Y_1, Y_2, and Y_{12} are independent exponential random variables, we can use the result from the section dealing with independent exponential failure times to show that $X_{(1)}$ again follows an exponential distribution with parameter $\lambda = \lambda_1 + \lambda_2 + \lambda_{12}$.

As before, we will extend these ideas to include different treatment effects for each cause of failure as well as an independent censoring variable. We will generate $U_1, U_2, U_{12} \overset{iid}{\sim} Uniform(0,1)$ and let

$$Y_1 = \frac{-1}{\lambda_1}\exp(-\beta_1 Z)\ln(1-U_1),$$

$$Y_2 = \frac{-1}{\lambda_2}\exp(-\beta_2 Z)\ln(1-U_2),$$

and

$$Y_{12} = \frac{-1}{\lambda_{12}}\exp(-\beta_{12} Z)\ln(1-U_{12}).$$

Thus Y_1, Y_2, and Y_{12} will be conditionally independent exponential random variables given Z such that

$$Y_1 \mid Z \sim Exponential(\lambda_1 \exp(\beta_1 Z)),$$

$$Y_2 \mid Z \sim Exponential(\lambda_2 \exp(\beta_2 Z)),$$

and

$$Y_{12} \mid Z \sim Exponential(\lambda_{12} \exp(\beta_{12} Z)).$$

We obtain dependent random variables X_1 and X_2 by letting

$$X_1 = \min(Y_1, Y_{12})$$

and

$$X_2 = \min(Y_2, Y_{12}).$$

Thus we have

$$X_1 \mid Z \sim \textit{Exponential}\left(\lambda_1 \exp(\beta_1 Z) + \lambda_{12} \exp(\beta_{12} Z)\right),$$

$$X_2 \mid Z \sim \textit{Exponential}\left(\lambda_2 \exp(\beta_2 Z) + \lambda_{12} \exp(\beta_{12} Z)\right),$$

and

$$(X_1, X_2) \mid Z \sim \textit{MOBVE}\left(\lambda_1 \exp(\beta_1 Z), \lambda_2 \exp(\beta_2 Z), \lambda_{12} \exp(\beta_{12} Z)\right).$$

Assuming no censoring, the observed survival time will be $X_{(1)} = \min(X_1, .X_2) = \min(Y_1, Y_2, Y_{12})$. Since Y_1, Y_2, and Y_{12} are conditionally independent exponential random variables given Z, we have that $X_{(1)} \mid Z$ follows an exponential distribution with parameter $\lambda_m = \lambda_1 \exp(\beta_1 Z) + \lambda_2 \exp(\beta_2 Z) + \lambda_{12} \exp(\beta_{12} Z)$.

We again wish to consider the case where independent censoring may occur. To do so, we will generate a random censoring time *C*, independent of $X_1 \mid Z$ and $X_2 \mid Z$. Thus *C* will be independent of $X_{(1)} \mid Z$. Again the censoring times will follow an exponential distribution with parameter λ_c. We have that $X_{(1)} \mid Z \sim \textit{Exponential}(\lambda_m)$ independent of $C \sim \textit{Exponential}(\lambda_c)$. Therefore, as in the case of independent exponential failure times, we can determine the censoring parameter in order to achieve an overall censoring probability equal to *p* by letting $\lambda_c = \frac{p(\lambda_m)}{(1-p)}$. Thus we will consider $U_c \sim \textit{Uniform}(0,1)$ and let $C = \frac{-1}{\lambda_c} \ln(1 - U_c)$.

7.3.2 DETERMINING SIMULATION PARAMETERS

We will now describe the method used for determining the parameters required in order to simulate data from a desired Marshall-Olkin bivariate exponential distribution. Recall that the means, conditional on Z, are given by

$$E[X_k \mid Z] = \frac{1}{\lambda_k \exp(\beta_k Z) + \lambda_{12} \exp(\beta_{12} Z)} \text{ for } k = 1,2 .$$

Additionally, the Marshall-Olkin bivariate exponential distribution allows for the situation where both causes of failure occur simultaneously. The probability of this event occurring is given by

$$P(X_1 = X_2 \mid Z) = \frac{\lambda_{12} \exp(\beta_{12} Z)}{\lambda_1 \exp(\beta_1 Z) + \lambda_2 \exp(\beta_2 Z) + \lambda_{12} \exp(\beta_{12} Z)} .$$

Suppose we wish to generate data having control means

$$E[X_k \mid Z = 0] = \mu_{0_k} \text{ for } k = 1,2$$

and a probability of simultaneous failure given by

$$P(X_1 = X_2 \mid Z = 0) = p_{12} .$$

Therefore we have that

$$E[X_k \mid Z = 0] = \mu_{0_k} = \frac{1}{\lambda_k + \lambda_{12}} \text{ for } k = 1,2$$

and

$$P(X_1 = X_2 \mid Z = 0) = p_{12} = \frac{\lambda_{12}}{\lambda_1 + \lambda_2 + \lambda_{12}} .$$

We now have three equations which can be solved for the three unknown parameters (λ_1, λ_2, and λ_{12}). Thus we are able to generate data with the desired control means and probability of simultaneous failure by setting

$$\lambda_{12} = \left(\frac{1}{\mu_{0_1}} + \frac{1}{\mu_{0_2}} \right) \left(\frac{p_{12}}{1 + p_{12}} \right)$$

and

$$\lambda_k = \frac{1}{\mu_{0_k}} - \lambda_{12} \text{ for } k = 1,2 .$$

As before, now suppose we wish to generate data for the treatment groups such that they result in treatment means having a specified percentage increase (or decrease) over the control means. Again the treatment means will be denoted by

$$E[X_k \mid Z = 1] = \mu_{1_k} \text{ for } k = 1,2$$

and the desired percentage increase over the control means will be given by r_k for $k = 1,2$. We will desire the same probability of simultaneous failure that we had in the control group, which is again given by

$$P(X_1 = X_2 \mid Z = 1) = p_{12} .$$

Therefore we have that

$$E[X_k \mid Z = 1] = \mu_{1_k} = \mu_{0_k}(1 + r_k) = \frac{1}{\lambda_k \exp(\beta_k) + \lambda_{12} \exp(\beta_{12})} \text{ for } k = 1,2$$

and

$$P(X_1 = X_2 \mid Z = 1) = p_{12} = \frac{\lambda_{12} \exp(\beta_{12})}{\lambda_1 \exp(\beta_1) + \lambda_2 \exp(\beta_2) + \lambda_{12} \exp(\beta_{12})} .$$

We again have three equations which can be solved for the three unknown parameters (β_1, β_2, and β_{12}). Thus we are able to generate data with the desired treatment means and probability of simultaneous failure by setting

$$\beta_{12} = \log\left\{\frac{p_{12}}{(1+p_{12})\lambda_{12}}\left(\frac{1}{\mu_{0_1}(1+r_1)}+\frac{1}{\mu_{0_2}(1+r_2)}\right)\right\}$$

and

$$\beta_k = \log\left\{\frac{1}{\mu_{0_k}(1+r_k)\lambda_k}-\frac{\lambda_{12}}{\lambda_k}\exp(\beta_{12})\right\} \text{ for } k=1,2.$$

7.3.3 SIMULATION SUMMARY

The following results are for competing risks data that follow a Marshall-Olkin bivariate exponential distribution with a 20% probabililty of simultaneous failure. Therefore the competing risks will not be independent of one another and that assumption will be violated for the test based on Cox's proportional hazards model and the log-rank test.

In Table 7.16, it appears that T^{CPH} and T^{LR} are consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring when the beneficial treatment effect on type 2 failure is small. When the beneficial treatment effect on type 2 failure is large, T^{CPH} and T^{LR} appear to best preserve the nominal alpha level for smaller sample sizes or when the amount of censoring is high. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is

small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small.

In Table 7.17, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. We also see that the power for each test diminishes as the treatment's beneficial effect on type 2 failure gets larger.

In panel 1 of Table 7.18, it appears that T^{CPH} and T^{LR} are again consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to be better at preserving the nominal alpha level for smaller sample sizes, while their performance does not seem influenced by the amount of censoring that occurs. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small and perform quite well when the amount of censoring is large.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 7.18, it appears that T^{G} and T^{FG} are now generally the more powerful tests when compared to T^{CPH} and T^{LR}, although T^{CPH} and T^{LR} overtake T^{G} and T^{FG} when the amount of censoring is high. Again, all tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases.

7.3.4 SIMULATION RESULTS

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.056	.056	.067	.076
	cens = 0.2	.060	.061	.062	.068
	cens = 0.5	.050	.048	.047	.053
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.050	.050	.099	.106
	cens = 0.2	.045	.046	.091	.097
	cens = 0.5	.056	.055	.091	.092
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.050	.050	.146	.156
	cens = 0.2	.058	.057	.122	.134
	cens = 0.5	.048	.048	.081	.080
	cens = 0.8	.058	.056	.068	.066
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.043	.044	.168	.231
	cens = 0.2	.054	.051	.142	.179
	cens = 0.5	.069	.067	.116	.131
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.067	.069	.383	.437
	cens = 0.2	.063	.063	.268	.312
	cens = 0.5	.066	.066	.177	.201
	cens = 0.8	.055	.049	.081	.077
n = 200	cens = 0	.087	.087	.662	.696
	cens = 0.2	.091	.090	.531	.576
	cens = 0.5	.064	.062	.330	.354
	cens = 0.8	.059	.057	.139	.139

* Programs did not converge due to sparse data.

Table 7.16: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Marshall-Olkin Exponential, $P(X_1=X_2)=0.20$).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.416	.412	.344	.325
	cens = 0.2	.349	.347	.266	.258
	cens = 0.5	.241	.235	.150	.125
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.730	.733	.592	.573
	cens = 0.2	.642	.642	.482	.458
	cens = 0.5	.437	.439	.275	.259
	cens = 0.8	.200	.193	.117	.093
n = 200	cens = 0	.951	.950	.879	.873
	cens = 0.2	.896	.896	.754	.749
	cens = 0.5	.719	.719	.482	.472
	cens = 0.8	.344	.343	.197	.178
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.349	.352	.183	.161
	cens = 0.2	.304	.305	.157	.138
	cens = 0.5	.209	.209	.079	.071
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.649	.654	.324	.295
	cens = 0.2	.559	.563	.246	.216
	cens = 0.5	.358	.364	.136	.116
	cens = 0.8	.195	.195	.102	.075
n = 200	cens = 0	.915	.917	.552	.526
	cens = 0.2	.816	.816	.406	.389
	cens = 0.5	.642	.647	.229	.215
	cens = 0.8	.324	.325	.114	.096
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.306	.315	.083	.055
	cens = 0.2	.252	.257	.061	.046
	cens = 0.5	.196	.201	.058	.041
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.530	.533	.085	.067
	cens = 0.2	.469	.478	.059	.049
	cens = 0.5	.300	.312	.046	.049
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.806	.814	.068	.059
	cens = 0.2	.743	.746	.062	.056
	cens = 0.5	.527	.534	.055	.043
	cens = 0.8	.247	.254	.047	.043

* Programs did not converge due to sparse data.

Table 7.17: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Marshall-Olkin Exponential, $P(X_1=X_2)=0.20$).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.054	.055	.090	.098
	cens = 0.2	.044	.043	.063	.074
	cens = 0.5	.056	.054	.059	.059
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.067	.068	.156	.170
	cens = 0.2	.069	.069	.129	.138
	cens = 0.5	.061	.061	.090	.093
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.063	.063	.246	.259
	cens = 0.2	.071	.070	.184	.193
	cens = 0.5	.071	.070	.125	.130
	cens = 0.8	.061	.059	.070	.070
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.512	.505	.579	.584
	cens = 0.2	.476	.472	.490	.481
	cens = 0.5	.272	.263	.268	.253
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.812	.809	.872	.882
	cens = 0.2	.715	.712	.760	.760
	cens = 0.5	.549	.538	.533	.523
	cens = 0.8	.253	.236	.218	.177
n = 200	cens = 0	.983	.982	.992	.992
	cens = 0.2	.954	.954	.960	.963
	cens = 0.5	.814	.812	.823	.825
	cens = 0.8	.418	.412	.389	.372

* Programs did not converge due to sparse data.

Table 7.18: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Marshall-Olkin Exponential, $P(X_1=X_2)=0.20$).

7.4 CAI-PRENTICE BIVARIATE EXPONENTIAL

7.4.1 DATA GENERATION

The next method used to generate bivariate exponential data is an extension of that employed by Cai and Prentice (1995). The technique was originally described by Clayton (1978) and Clayton and Cuzick (1985), and was later utilized in papers by Prentice and Cai (1992) and Cai and Prentice (1995). The method described by Cai and Prentice allows us to control the degree of correlation between two causes of failure through the use of an association parameter θ where maximal positive dependence is achieved as $\theta \to 0$ and approaches independence as $\theta \to \infty$. The θ later proposed by Cai and Prentice is not identical to the one originally proposed by Clayton. Therefore in order to distinguish between the two, we will denote the θ described in Clayton (1978) by θ_C. In fact, the relationship between the two association parameters is given by

$$\theta = \frac{1}{\theta_C - 1} \ .$$

Furthermore, Klein and Moeschberger (1988) show that θ_C is related to Kendall's coefficient of concordance τ by

$$\tau = \frac{\theta_C - 1}{\theta_C + 1}.$$

Therefore, we have that the relationship between the association parameter used by Cai and Prentice (1995) and Kendall's tau is given by

$$\theta = \frac{1 - \tau}{2\tau}.$$

Using this fact, we can determine θ for desired levels of τ. Note that for a pair of continuous variables X_1 and X_2,

$$\hat{\tau} = \frac{C - D}{C + D}$$

$$= \frac{C - D}{\left[\frac{(n-1)n}{2}\right]}$$

where C = the number of concordant pairs, and D = the number of discordant pairs in a sample of n subjects (Agresti, 1990). Kendall's $\tau > 0$ implies there is a tendency for X_2 to be higher at higher levels of X_1. Independence of X_1 and X_2 implies $\tau = 0$, however $\tau = 0$ does not necessarily imply independence of X_1 and X_2.

We will let $U_1, U_2 \overset{iid}{\sim} Uniform(0,1)$ and generate

$$X_2 = -\frac{1}{\lambda_2}\log(1 - U_2)\exp(-\beta_2 Z)$$

and

$$X_1 = \frac{1}{\lambda_1}\theta\log\left[(1-a) + a(1-U_1)^{-(1+\theta)^{-1}}\right]\exp(-\beta_1 Z)$$

where $a = (1 - U_2)^{-\theta^{-1}}$ and Z is the treatment indicator. Thus

$$X_k \mid Z \sim Exponential(\lambda_k^*)$$

with

$$E[X_k \mid Z] = \frac{1}{\lambda_k^*}$$

where

$$\lambda_k^* = \lambda_k \exp(\beta_k Z) \text{ for } k = 1,2 .$$

By incorporating the parameters λ_1 and λ_2, we are allowing the control group for each cause of failure to have its own mean survival time. Additionally, we have included the β_1 and β_2 terms to allow for differing treatment effects within each cause of failure.

We will then generate censoring variables (C_1, C_2) independently of each other and of $(X_1, X_2) | Z$ where C_k is the censoring time associated with failure time X_k for $k = 1,2$. The censoring variables will follow exponential distributions with $C_k \sim Exp(\lambda_{c_k})$ for $k = 1,2$ where the λ_{c_k} are chosen in such a way that they result in censoring probabilities

$$p_1 = P(C_1 < X_1 \mid Z)$$

and

$$p_2 = P(C_2 < X_2 \mid Z).$$

Since $X_k \mid Z$ is independent of C_k for $k = 1,2$, the censoring parameters λ_{c_1} and λ_{c_2} will be chosen in a manner similar to the one previously described. Defining $\lambda_{c_1} = \frac{p_1(\lambda_1^*)}{(1-p_1)}$ and $\lambda_{c_2} = \frac{p_2(\lambda_2^*)}{(1-p_2)}$ will result in censoring probabilities of p_1 and p_2, respectively.

7.4.2 SIMULATION SUMMARY

Like those of the Sarmanov bivariate exponential distribution with no correlation, the results for the Cai-Prentice bivariate exponential having a Kendall's tau coefficient of zero (Tables 19 - 21) are also very similar to those of the independent bivariate exponential distribution given above. However, we will also consider the case where the competing risks have a positive Kendall's tau coefficient. Thus violating the independence assumption inherent in the test based on Cox's proportional hazards model and the log-rank test.

In Table 19, it appears that T^{CPH} and T^{LR} are consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring, regardless of whether the beneficial treatment effect on type 2 failure is small or large. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small. When we allowed for the competing risks to be positively correlated (Tables 22, 25, and 28), we saw that all tests were better at preserving the nominal alpha level when the sample size was small, when the amount of censoring was large, and when the beneficial treatment effect on type 2 failure was small. Furthermore, each test's ability to preserve the nominal alpha level appeared to worsen as the positive correlation between the two competing risks increased.

In Table 20, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. When the effect on type 2 failure is beneficial

(panels 2 and 3 of Table 20), we see that the power results for T^{CPH} and T^{LR} remain similar to those found when there was no treatment effect on type 2 failure, however T^{G} and T^{FG} lose considerable power as the treatment's beneficial effect on type 2 failure gets larger. When we allowed for the competing risks to be positively correlated (Tables 23, 26, and 29), we again saw that all tests appeared to gain power for larger sample sizes and lose power as the amount of censoring increased. However when the correlation was introduced, all tests now displayed considerable losses in power as the treatment's beneficial effect on type 2 got larger.

In panel 1 of Table 21, it appears that T^{CPH} and T^{LR} are again consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small and perform quite well when the amount of censoring is large. When we allowed for the competing risks to be positively correlated (panel 1 of Tables 24, 27, and 30), we saw that all tests were better at preserving the nominal alpha level when the sample size was small and when the amount of censoring was large. Furthermore, each test's ability to preserve the nominal alpha level appeared to worsen as the positive correlation between the two competing risks increased.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 21, it appears that T^{G} and T^{FG} are now generally the more powerful tests when compared to T^{CPH} and T^{LR}, although T^{CPH} and T^{LR} overtake T^{G} and T^{FG} when the amount of censoring is high. Again, all tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. When we allowed

for the competing risks to be positively correlated (panel 2 of Tables 24, 27, and 30), we again saw that all tests appeared to gain power for larger sample sizes and lose power as the amount of censoring increased. Also, the power for each test rose as the correlation between the competing risks increased.

7.4.3 SIMULATION RESULTS

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.049	.054	.062	.072
	cens = 0.2	.034	.037	.050	.061
	cens = 0.5	.056	.057	.057	.048
	cens = 0.8	.057	.051	.045	.036
n = 100	cens = 0	.055	.056	.082	.087
	cens = 0.2	.050	.050	.070	.077
	cens = 0.5	.052	.051	.058	.060
	cens = 0.8	.050	.048	.049	.052
n = 200	cens = 0	.040	.041	.108	.115
	cens = 0.2	.052	.052	.106	.105
	cens = 0.5	.050	.049	.070	.083
	cens = 0.8	.055	.054	.060	.060
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.051	.053	.117	.165
	cens = 0.2	.052	.052	.098	.116
	cens = 0.5	.059	.059	.067	.081
	cens = 0.8	.051	.045	.053	.043
n = 100	cens = 0	.052	.053	.259	.289
	cens = 0.2	.056	.058	.181	.226
	cens = 0.5	.052	.052	.117	.131
	cens = 0.8	.039	.037	.058	.059
n = 200	cens = 0	.053	.053	.458	.495
	cens = 0.2	.043	.044	.327	.367
	cens = 0.5	.041	.043	.194	.220
	cens = 0.8	.054	.053	.095	.095

Table 7.19: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Cai-Prentice Exponential, tau = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.330	.335	.256	.240
	cens = 0.2	.268	.272	.170	.157
	cens = 0.5	.190	.190	.124	.115
	cens = 0.8	.136	.120	.075	.055
n = 100	cens = 0	.589	.589	.456	.443
	cens = 0.2	.507	.508	.334	.313
	cens = 0.5	.324	.331	.160	.148
	cens = 0.8	.164	.165	.075	.057
n = 200	cens = 0	.869	.868	.712	.707
	cens = 0.2	.812	.812	.562	.546
	cens = 0.5	.586	.586	.303	.290
	cens = 0.8	.255	.261	.123	.114
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.341	.345	.151	.125
	cens = 0.2	.291	.296	.116	.099
	cens = 0.5	.167	.168	.078	.058
	cens = 0.8	.111	.103	.053	.035
n = 100	cens = 0	.605	.610	.289	.258
	cens = 0.2	.490	.496	.197	.179
	cens = 0.5	.348	.349	.113	.101
	cens = 0.8	.159	.162	.062	.051
n = 200	cens = 0	.875	.875	.451	.421
	cens = 0.2	.793	.794	.320	.285
	cens = 0.5	.583	.584	.165	.148
	cens = 0.8	.304	.307	.103	.092
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.367	.376	.085	.048
	cens = 0.2	.293	.301	.054	.038
	cens = 0.5	.209	.214	.055	.056
	cens = 0.8	.119	.123	.042	.032
n = 100	cens = 0	.603	.615	.101	.077
	cens = 0.2	.524	.531	.065	.056
	cens = 0.5	.371	.376	.047	.046
	cens = 0.8	.163	.168	.035	.030
n = 200	cens = 0	.891	.892	.119	.091
	cens = 0.2	.828	.835	.076	.056
	cens = 0.5	.596	.599	.057	.045
	cens = 0.8	.291	.298	.044	.041

Table 7.20: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Cai-Prentice Exponential, tau = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.045	.045	.055	.072
	cens = 0.2	.045	.044	.060	.083
	cens = 0.5	.044	.045	.054	.051
	cens = 0.8	.060	.053	.040	.034
n = 100	cens = 0	.044	.043	.102	.120
	cens = 0.2	.047	.049	.094	.098
	cens = 0.5	.054	.055	.078	.073
	cens = 0.8	.061	.055	.061	.054
n = 200	cens = 0	.049	.049	.162	.176
	cens = 0.2	.045	.046	.116	.127
	cens = 0.5	.049	.049	.087	.101
	cens = 0.8	.044	.040	.059	.060
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.310	.305	.399	.396
	cens = 0.2	.278	.279	.300	.298
	cens = 0.5	.165	.161	.151	.145
	cens = 0.8	.104	.087	.077	.066
n = 100	cens = 0	.589	.588	.682	.688
	cens = 0.2	.475	.473	.512	.511
	cens = 0.5	.326	.325	.318	.315
	cens = 0.8	.138	.130	.122	.099
n = 200	cens = 0	.858	.857	.924	.927
	cens = 0.2	.768	.767	.823	.827
	cens = 0.5	.547	.545	.551	.548
	cens = 0.8	.262	.259	.216	.206

Table 7.21: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Cai-Prentice Exponential, tau = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.048	.047	.069	.087
	cens = 0.2	.050	.053	.046	.058
	cens = 0.5	.059	.058	.054	.056
	cens = 0.8	.044	.037	.030	.018
n = 100	cens = 0	.070	.069	.106	.118
	cens = 0.2	.045	.046	.080	.087
	cens = 0.5	.051	.051	.064	.068
	cens = 0.8	.054	.053	.058	.050
n = 200	cens = 0	.057	.058	.143	.148
	cens = 0.2	.062	.063	.113	.121
	cens = 0.5	.048	.048	.075	.079
	cens = 0.8	.062	.060	.057	.055
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.060	.061	.197	.241
	cens = 0.2	.056	.055	.120	.154
	cens = 0.5	.058	.056	.086	.096
	cens = 0.8	.051	.045	.051	.043
n = 100	cens = 0	.070	.071	.383	.428
	cens = 0.2	.056	.057	.222	.262
	cens = 0.5	.060	.064	.145	.161
	cens = 0.8	.050	.049	.069	.069
n = 200	cens = 0	.103	.103	.670	.702
	cens = 0.2	.078	.078	.461	.501
	cens = 0.5	.062	.061	.238	.264
	cens = 0.8	.052	.050	.105	.103

Table 7.22: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Cai-Prentice Exponential, tau = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.351	.353	.320	.314
	cens = 0.2	.279	.278	.218	.217
	cens = 0.5	.192	.193	.108	.099
	cens = 0.8	.121	.112	.080	.065
n = 100	cens = 0	.648	.649	.595	.592
	cens = 0.2	.502	.505	.379	.369
	cens = 0.5	.322	.323	.191	.181
	cens = 0.8	.160	.153	.085	.074
n = 200	cens = 0	.913	.913	.860	.856
	cens = 0.2	.792	.792	.637	.631
	cens = 0.5	.591	.595	.355	.345
	cens = 0.8	.281	.280	.150	.140
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.333	.335	.199	.185
	cens = 0.2	.271	.275	.132	.124
	cens = 0.5	.184	.190	.087	.072
	cens = 0.8	.116	.112	.060	.043
n = 100	cens = 0	.563	.567	.314	.298
	cens = 0.2	.463	.470	.203	.184
	cens = 0.5	.321	.324	.119	.107
	cens = 0.8	.159	.164	.067	.056
n = 200	cens = 0	.879	.883	.545	.531
	cens = 0.2	.757	.759	.344	.331
	cens = 0.5	.568	.571	.180	.161
	cens = 0.8	.282	.287	.076	.067
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.317	.325	.077	.056
	cens = 0.2	.255	.266	.075	.056
	cens = 0.5	.168	.171	.044	.042
	cens = 0.8	.112	.106	.046	.036
n = 100	cens = 0	.568	.575	.095	.063
	cens = 0.2	.435	.445	.071	.067
	cens = 0.5	.327	.334	.058	.052
	cens = 0.8	.164	.171	.050	.037
n = 200	cens = 0	.816	.817	.102	.087
	cens = 0.2	.713	.715	.091	.069
	cens = 0.5	.546	.551	.055	.055
	cens = 0.8	.268	.278	.044	.037

Table 7.23: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Cai-Prentice Exponential, tau = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.052	.053	.075	.084
	cens = 0.2	.043	.041	.057	.068
	cens = 0.5	.046	.043	.047	.047
	cens = 0.8	.052	.045	.043	.036
n = 100	cens = 0	.059	.059	.115	.133
	cens = 0.2	.056	.056	.093	.097
	cens = 0.5	.053	.055	.063	.067
	cens = 0.8	.048	.047	.050	.043
n = 200	cens = 0	.055	.054	.225	.241
	cens = 0.2	.055	.055	.156	.162
	cens = 0.5	.051	.051	.078	.082
	cens = 0.8	.052	.051	.060	.056
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.375	.369	.484	.523
	cens = 0.2	.338	.333	.361	.377
	cens = 0.5	.210	.203	.185	.180
	cens = 0.8	.112	.094	.080	.076
n = 100	cens = 0	.690	.690	.820	.828
	cens = 0.2	.549	.546	.616	.620
	cens = 0.5	.322	.318	.319	.317
	cens = 0.8	.148	.143	.122	.101
n = 200	cens = 0	.932	.931	.987	.989
	cens = 0.2	.825	.823	.877	.887
	cens = 0.5	.602	.600	.597	.601
	cens = 0.8	.251	.249	.215	.203

Table 7.24: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Cai-Prentice Exponential, tau = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.071	.069	.134	.159
	cens = 0.2	.057	.056	.076	.087
	cens = 0.5	.046	.047	.052	.052
	cens = 0.8	.068	.056	.060	.039
n = 100	cens = 0	.109	.109	.224	.254
	cens = 0.2	.093	.091	.141	.151
	cens = 0.5	.051	.052	.071	.073
	cens = 0.8	.057	.054	.053	.045
n = 200	cens = 0	.146	.145	.447	.474
	cens = 0.2	.095	.094	.226	.242
	cens = 0.5	.068	.066	.092	.104
	cens = 0.8	.046	.045	.053	.050
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.142	.139	.440	.530
	cens = 0.2	.090	.089	.238	.297
	cens = 0.5	.069	.069	.100	.113
	cens = 0.8	.052	.044	.047	.033
n = 100	cens = 0	.266	.262	.798	.833
	cens = 0.2	.149	.147	.474	.520
	cens = 0.5	.087	.084	.201	.221
	cens = 0.8	.053	.050	.083	.078
n = 200	cens = 0	.470	.462	.976	.980
	cens = 0.2	.241	.238	.748	.776
	cens = 0.5	.119	.115	.361	.392
	cens = 0.8	.060	.059	.119	.124

Table 7.25: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Cai-Prentice Exponential, tau = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.580	.576	.601	.620
	cens = 0.2	.383	.378	.341	.335
	cens = 0.5	.210	.207	.147	.127
	cens = 0.8	.105	.091	.064	.053
n = 100	cens = 0	.883	.883	.917	.926
	cens = 0.2	.654	.655	.622	.619
	cens = 0.5	.348	.348	.236	.226
	cens = 0.8	.168	.164	.106	.081
n = 200	cens = 0	.997	.997	.997	.998
	cens = 0.2	.923	.923	.872	.876
	cens = 0.5	.640	.641	.462	.457
	cens = 0.8	.256	.257	.145	.139
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.471	.472	.354	.367
	cens = 0.2	.326	.332	.193	.183
	cens = 0.5	.177	.179	.093	.077
	cens = 0.8	.098	.090	.046	.039
n = 100	cens = 0	.784	.783	.636	.642
	cens = 0.2	.557	.556	.319	.308
	cens = 0.5	.331	.337	.146	.131
	cens = 0.8	.150	.149	.067	.052
n = 200	cens = 0	.977	.977	.898	.901
	cens = 0.2	.855	.854	.565	.555
	cens = 0.5	.542	.543	.219	.213
	cens = 0.8	.254	.256	.083	.074
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.267	.277	.072	.055
	cens = 0.2	.225	.227	.064	.059
	cens = 0.5	.151	.152	.051	.047
	cens = 0.8	.103	.102	.045	.034
n = 100	cens = 0	.479	.488	.080	.063
	cens = 0.2	.388	.396	.081	.067
	cens = 0.5	.253	.264	.058	.051
	cens = 0.8	.139	.144	.040	.032
n = 200	cens = 0	.789	.791	.101	.076
	cens = 0.2	.656	.659	.073	.063
	cens = 0.5	.453	.462	.043	.040
	cens = 0.8	.243	.250	.048	.042

Table 7.26: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Cai-Prentice Exponential, tau = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.093	.092	.166	.195
	cens = 0.2	.064	.064	.102	.118
	cens = 0.5	.053	.053	.061	.058
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.139	.137	.327	.366
	cens = 0.2	.083	.081	.163	.179
	cens = 0.5	.074	.071	.098	.101
	cens = 0.8	.057	.050	.064	.053
n = 200	cens = 0	.230	.227	.595	.622
	cens = 0.2	.122	.120	.313	.329
	cens = 0.5	.069	.070	.123	.126
	cens = 0.8	.050	.051	.071	.068
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.670	.659	.814	.853
	cens = 0.2	.464	.452	.544	.559
	cens = 0.5	.246	.232	.225	.210
	cens = 0.8	.110	.096	.082	.083
n = 100	cens = 0	.922	.920	.985	.991
	cens = 0.2	.790	.786	.868	.877
	cens = 0.5	.417	.409	.427	.418
	cens = 0.8	.176	.171	.150	.117
n = 200	cens = 0	.999	.999	1.00	1.00
	cens = 0.2	.976	.974	.989	.991
	cens = 0.5	.720	.717	.725	.730
	cens = 0.8	.286	.281	.237	.231

* Programs did not converge due to sparse data.

Table 7.27: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Cai-Prentice Exponential, tau = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.287	.280	.560	.619
	cens = 0.2	.134	.131	.223	.244
	cens = 0.5	.067	.063	.084	.086
	cens = 0.8	.055	.045	.049	.046
n = 100	cens = 0	.527	.523	.858	.878
	cens = 0.2	.240	.239	.441	.469
	cens = 0.5	.087	.085	.138	.144
	cens = 0.8	.058	.052	.057	.045
n = 200	cens = 0	.800	.798	.991	.994
	cens = 0.2	.447	.445	.748	.775
	cens = 0.5	.118	.117	.228	.240
	cens = 0.8	.052	.049	.071	.068
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.484	.477	.911	.948
	cens = 0.2	.314	.311	.644	.699
	cens = 0.5	.127	.121	.221	.237
	cens = 0.8	.055	.044	.057	.056
n = 100	cens = 0	.816	.813	.999	1.00
	cens = 0.2	.532	.526	.909	.923
	cens = 0.5	.200	.191	.420	.456
	cens = 0.8	.071	.067	.109	.107
n = 200	cens = 0	.985	.985	1.00	1.00
	cens = 0.2	.818	.815	.995	.995
	cens = 0.5	.330	.328	.720	.745
	cens = 0.8	.063	.060	.164	.166

Table 7.28: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Cai-Prentice Exponential, tau = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.938	.934	.972	.976
	cens = 0.2	.753	.743	.762	.756
	cens = 0.5	.351	.341	.287	.248
	cens = 0.8	.150	.121	.091	.113
n = 100	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	.952	.952	.953	.954
	cens = 0.5	.600	.598	.520	.512
	cens = 0.8	.171	.163	.095	.079
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.875	.871	.811	.805
	cens = 0.8	.305	.304	.189	.178
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.887	.886	.889	.887
	cens = 0.2	.612	.610	.525	.524
	cens = 0.5	.254	.246	.153	.125
	cens = 0.8	.111	.090	.050	.066
n = 100	cens = 0	.998	.998	.995	.995
	cens = 0.2	.902	.901	.823	.823
	cens = 0.5	.467	.466	.288	.274
	cens = 0.8	.160	.159	.075	.053
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	.993	.993
	cens = 0.5	.752	.754	.530	.526
	cens = 0.8	.230	.231	.109	.101
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.341	.346	.094	.054
	cens = 0.2	.244	.248	.067	.058
	cens = 0.5	.138	.142	.051	.046
	cens = 0.8	.097	.090	.046	.047
n = 100	cens = 0	.569	.572	.099	.069
	cens = 0.2	.399	.405	.063	.044
	cens = 0.5	.233	.245	.059	.052
	cens = 0.8	.142	.145	.052	.040
n = 200	cens = 0	.855	.857	.118	.090
	cens = 0.2	.687	.693	.079	.060
	cens = 0.5	.379	.386	.051	.049
	cens = 0.8	.179	.185	.054	.046

Table 7.29: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Cai-Prentice Exponential, tau = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.473	.454	.714	.750
	cens = 0.2	.224	.217	.334	.355
	cens = 0.5	.063	.061	.084	.081
	cens = 0.8	.063	.037	.042	.059
n = 100	cens = 0	.770	.767	.951	.964
	cens = 0.2	.414	.409	.620	.641
	cens = 0.5	.097	.091	.160	.162
	cens = 0.8	.062	.053	.059	.051
n = 200	cens = 0	.970	.970	.999	.999
	cens = 0.2	.699	.694	.905	.917
	cens = 0.5	.195	.189	.355	.363
	cens = 0.8	.063	.059	.079	.079
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.961	.958	.993	.997
	cens = 0.2	.815	.800	.883	.881
	cens = 0.5	.434	.411	.422	.381
	cens = 0.8	*	*	*	*
n = 100	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	.988	.987	.999	.999
	cens = 0.5	.736	.730	.760	.750
	cens = 0.8	.200	.188	.175	.142
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.951	.950	.966	.964
	cens = 0.8	.378	.369	.355	.328

* Programs did not converge due to sparse data.

Table 7.30: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Cai-Prentice Exponential, tau = 0.8).

7.5 BIVARIATE EXPONENTIAL WITH A GAMMA FRAILTY

7.5.1 FRAILTY MODELS

We are also able to model the association between survival times within subgroups by using what's referred to as a *frailty model*. A frailty is an unobservable random effect shared by those subjects within a certain subgroup. For our frailty model, we will generate a random effect that acts multiplicatively on the hazard rates of all members within a specific subgroup. Therefore, subgroups with a large frailty value for the random effect will experience the event sooner than subgroups with smaller frailty values.

The most common model for a frailty is the so-called shared frailty model extension of the proportional hazards regression model (Klein and Moeschberger, 2003). To apply the shared frailty model to the competing risks setting we have been discussing, we will assume that each subject is his or her own subgroup and that he or she may experience one of two competing risks. Thus, there are $K = 2$ competing risks for each of the n subjects. The shared frailty model then assumes that the hazard rate for the k^{th} subject in the i^{th} subgroup, given the frailty, is of the form

$$h_{ik}(t) = h_o(t)\exp(\alpha_k + \beta_k Z_i + \sigma a_i) \text{ for } i = 1,...,n \text{ and } k = 1,2$$

where $h_o(t)$ is an arbitrary baseline hazard rate, α_k is the parameter associated with the hazard of the control group for cause k, Z_i is the treatment indicator for subject i, β_k is the parameter associated with the treatment indicator for cause k, and $a_1,...,a_n$ are the frailties drawn independently from some distribution with mean 0 and variance 1. We will further assume $h_o(t) = 1$ and rewrite the model as

$$h_{ik}(t) = w_i \lambda_k \exp(\beta_k Z_i) \text{ for } i = 1,...,n \text{ and } k = 1,2$$

where the $w_i's$ are frailties obtained via an independent and identically distributed sample from a distribution with mean 1 and some unknown variance.

A common model for the random effect is the one-parameter gamma distribution (Clayton, 1978 and Houggard, 2000). The pdf for the gamma distribution with a shape parameter δ and an inverse scale parameter θ, denoted $Gamma(\delta, \theta)$, is given by

$$f(w) = \begin{cases} \dfrac{\theta^{\delta} w^{\delta-1} \exp(-\theta w)}{\Gamma(\delta)}, & w \geq 0 \\ 0 \qquad\qquad\qquad\quad , & o.w. \end{cases}$$

The expected value and variance of the gamma distribution are given by $E(W) = \dfrac{\delta}{\theta}$ and $Var(W) = \dfrac{\delta}{\theta^2}$, respectively. Note that if $\delta = 1$, then the gamma distribution reduces to the exponential distribution with parameter θ. The one-parameter gamma distribution commonly used in frailty models makes the restriction that $\theta = \delta$, yielding an expected value of 1 for the frailty parameter.

7.5.2 DATA GENERATION

To generate data from the frailty model described above, first consider $U_{ki} \overset{iid}{\sim} Uniform(0,1)$ and let

$$Y_{ki} \mid Z_i = \frac{-1}{\lambda_k} \exp(-\beta_k Z_i) \ln(1 - U_{ki}) \text{ for } k = 1,2 \text{ and } i = 1,...,n.$$

Therefore we have

$$Y_{ki} \mid Z_i \overset{ind}{\sim} Exponential\left(\lambda_k \exp\left(\beta_k Z_i\right)\right) \text{ for } k = 1,2 \text{ and } i = 1,...,n.$$

We will then generate the frailty associated with each individual in the sample independent of $Y_{ki} \mid Z_i$. First restrict θ to be an integer and consider $U_{ji} \overset{iid}{\sim} Uniform(0,1)$ for $j = 1,...,\theta$ and $i = 1,...,n$. We will then let

$$V_{ji} = \frac{-1}{\theta}\ln\left(1 - U_{ji}\right) \text{ for } j = 1,...,\theta \text{ and } i = 1,...,n.$$

Therefore $V_{ji} \overset{ind}{\sim} Exponential(\theta)$ for $j = 1,...,\theta$ and $i = 1,...,n$. Finally we consider $w_i = \sum_{j=1}^{\theta} V_{ji}$ for $i = 1,...,n$ to obtain the frailty associated with the i^{th} individual in the sample. Note that the exponential and gamma distributions are related in such a way that $w_i \overset{ind}{\sim} gamma(\theta,\theta)$ for $i = 1,...,n$. Note that $E(w_i) = \frac{\theta}{\theta} = 1$. The survival times for each individual in the sample are then obtained by letting

$$X_{ki} \mid Z_i = w_i \times Y_{ki} \mid Z_i \text{ for } k = 1,2 \text{ and } i = 1,...,n.$$

Therefore, the i^{th} individual has a conditional expected survival time and variance for cause *k* which are given by

$$\begin{aligned} E\left(X_{ki} \mid Z_i\right) &= E\left(w_i \times Y_{ki} \mid Z_i\right) \\ &= E\left(w_i\right)E\left(Y_{ki} \mid Z_i\right) \qquad \textit{by independence} \\ &= E\left(Y_{ki} \mid Z_i\right) \\ &= \frac{1}{\lambda_k \exp\left(\beta_k Z_i\right)} \end{aligned}$$

and

$$
\begin{aligned}
Var(X_{ki} \mid Z_i) &= Var(w_i \times Y_{ki} \mid Z_i) \\
&= E(w_i^2)E(Y_{ki}^2 \mid Z_i) - [E(w_i)]^2 [E(Y_{ki} \mid Z_i)]^2 \qquad \textit{by independence} \\
&= \left(\frac{1}{\theta} + 1\right)\left(\left\{\frac{1}{\lambda_k \exp(\beta_k Z_i)}\right\}^2 + \left\{\frac{1}{\lambda_k \exp(\beta_k Z_i)}\right\}^2\right) - \left\{\frac{1}{\lambda_k \exp(\beta_k Z_i)}\right\}^2,
\end{aligned}
$$

respectively.

We again wish to consider the case where independent censoring may occur. We will generate random censoring times C_i independent of $X_{1i} \mid Z_i, w_i$ and $X_{2i} \mid Z_i, w_i$ for $i = 1,...,n$. Thus C_i will be independent of $X_{(1)i} \mid Z_i, w_i$. The censoring times for the the i^{th} individual will follow an exponential distribution with parameter λ_{c_i}. We have that $X_{(1)i} \mid Z_i, w_i \sim Exponential(\lambda_{m_i})$ is independent of $C_i \sim Exponential(\lambda_{c_i})$ where $\lambda_{m_i} = \frac{1}{w_i}\sum_{k=1}^{2} \lambda_k \exp(\beta_k Z_i)$ for $i = 1,...,n$. Therefore we will select $\lambda_{c_i} = \frac{p(\lambda_{m_i})}{(1-p)}$ in order to achieve an overall censoring probability equal to *p*. Thus, in order to generate the censoring times, we will consider $U_{c_i} \sim Uniform(0,1)$ and let $C_i = \frac{-1}{\lambda_{c_i}} \ln(1 - U_{c_i})$ for $i = 1,...,n$.

7.5.3 SIMULATION SUMMARY

By introducing a frailty parameter for each subject, we are again inducing a positive correlation between the competing risks. Thus we are again violating the independence assumption inherent in the test based on Cox's proportional hazards model and the log-rank test.

In Table 31, it appears that T^{CPH} and T^{LR} are generally better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring regardless of whether the beneficial treatment effect on type 2 failure is small or large. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small.

In Table 32, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. We also see that the power for each test diminishes as the treatment's beneficial effect on type 2 failure gets larger.

In panel 1 of Table 33, it appears that T^{CPH} and T^{LR} are again consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small and the amount of censoring is large.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 33, it appears that T^{G} and T^{FG} are now generally the more powerful tests when compared to T^{CPH} and T^{LR}, although T^{CPH} and T^{LR} overtake T^{G} and T^{FG} when the amount of censoring is high. Again, all tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases.

7.5.4 SIMULATION RESULTS

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.061	.061	.066	.074
	cens = 0.2	.045	.046	.050	.050
	cens = 0.5	.055	.053	.056	.055
	cens = 0.8	.063	.053	.040	.031
n = 100	cens = 0	.050	.050	.091	.106
	cens = 0.2	.048	.047	.089	.090
	cens = 0.5	.057	.056	.060	.064
	cens = 0.8	.053	.052	.050	.042
n = 200	cens = 0	.053	.053	.139	.143
	cens = 0.2	.059	.059	.095	.098
	cens = 0.5	.056	.056	.075	.077
	cens = 0.8	.052	.051	.054	.055
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.053	.053	.126	.165
	cens = 0.2	.049	.051	.099	.116
	cens = 0.5	.050	.051	.072	.075
	cens = 0.8	.058	.048	.051	.041
n = 100	cens = 0	.067	.068	.251	.294
	cens = 0.2	.064	.064	.204	.225
	cens = 0.5	.070	.070	.136	.145
	cens = 0.8	.053	.048	.071	.068
n = 200	cens = 0	.064	.065	.508	.542
	cens = 0.2	.050	.050	.345	.379
	cens = 0.5	.065	.064	.206	.221
	cens = 0.8	.055	.054	.105	.104

Table 7.31: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Exponential with Gamma frailty).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.278	.277	.248	.250
	cens = 0.2	.215	.217	.171	.151
	cens = 0.5	.152	.153	.102	.101
	cens = 0.8	.097	.084	.057	.043
n = 100	cens = 0	.506	.507	.434	.435
	cens = 0.2	.401	.401	.296	.295
	cens = 0.5	.262	.263	.159	.155
	cens = 0.8	.133	.131	.080	.071
n = 200	cens = 0	.796	.796	.709	.700
	cens = 0.2	.698	.697	.544	.531
	cens = 0.5	.487	.489	.311	.305
	cens = 0.8	.232	.233	.129	.116
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.267	.272	.133	.121
	cens = 0.2	.208	.215	.095	.082
	cens = 0.5	.163	.164	.072	.069
	cens = 0.8	.094	.087	.056	.036
n = 100	cens = 0	.485	.486	.242	.213
	cens = 0.2	.390	.390	.184	.173
	cens = 0.5	.258	.261	.102	.090
	cens = 0.8	.138	.138	.066	.058
n = 200	cens = 0	.759	.762	.397	.389
	cens = 0.2	.653	.653	.291	.271
	cens = 0.5	.461	.462	.170	.157
	cens = 0.8	.194	.197	.065	.062
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.222	.222	.055	.048
	cens = 0.2	.205	.214	.060	.048
	cens = 0.5	.134	.137	.066	.068
	cens = 0.8	.096	.089	.050	.040
n = 100	cens = 0	.391	.395	.078	.068
	cens = 0.2	.319	.328	.072	.063
	cens = 0.5	.220	.224	.043	.042
	cens = 0.8	.115	.118	.047	.041
n = 200	cens = 0	.684	.687	.090	.076
	cens = 0.2	.589	.599	.062	.054
	cens = 0.5	.409	.412	.047	.047
	cens = 0.8	.192	.196	.059	.054

Table 7.32: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Exponential with Gamma frailty).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.051	.052	.073	.085
	cens = 0.2	.061	.061	.068	.074
	cens = 0.5	.061	.060	.047	.045
	cens = 0.8	.052	.040	.046	.030
n = 100	cens = 0	.043	.043	.091	.094
	cens = 0.2	.065	.064	.095	.107
	cens = 0.5	.064	.064	.059	.068
	cens = 0.8	.053	.050	.059	.049
n = 200	cens = 0	.063	.062	.145	.154
	cens = 0.2	.049	.049	.106	.110
	cens = 0.5	.052	.052	.086	.086
	cens = 0.8	.060	.057	.078	.076
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.293	.290	.361	.382
	cens = 0.2	.249	.245	.282	.282
	cens = 0.5	.181	.179	.169	.167
	cens = 0.8	.116	.091	.085	.067
n = 100	cens = 0	.535	.532	.656	.673
	cens = 0.2	.457	.458	.522	.523
	cens = 0.5	.310	.303	.322	.308
	cens = 0.8	.156	.150	.145	.121
n = 200	cens = 0	.814	.812	.904	.909
	cens = 0.2	.740	.739	.814	.818
	cens = 0.5	.546	.542	.560	.565
	cens = 0.8	.245	.237	.213	.208

Table 7.33: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Exponential with Gamma frailty).

CHAPTER 8

MULTIVARIATE WEIBULL DISTRIBUTIONS

The Weibull distribution is often used in the field of lifetime data analysis due to its flexibility. It can mimic the behavior of other statistical distributions such as the normal and exponential distributions. A random variable (RV) X is said to follow a Weibull distribution with parameters $\alpha, \lambda > 0$ if its probability density function (pdf) $f_X(x)$ is given by

$$f_X(x) = \begin{cases} \alpha\lambda x^{\alpha-1} \exp\left(-\lambda x^{\alpha}\right), & x \geq 0 \\ 0 & , \text{ otherwise.} \end{cases}$$

We denote this by $X \sim Weibull(\alpha, \lambda)$. Thus, the cumulative distribution function (cdf) $F_X(x)$ and the survival function $S_X(x)$ associated with X are given by

$$F_X(x) = \begin{cases} 1 - \exp\left(-\lambda x^{\alpha}\right), & x \geq 0 \\ 0 & , \text{ otherwise} \end{cases}$$

and

$$S_X(x) = \begin{cases} \exp\left(-\lambda x^{\alpha}\right), & x \geq 0 \\ 1 & , \text{ otherwise,} \end{cases}$$

respectively. The scale parameter is given by λ while the shape parameter is represented by α. The hazard function associated with the Weibull distribution has a

fairly flexible form given by

$$h_X(x) = \begin{cases} \alpha\lambda x^{\alpha-1}, & x \geq 0 \\ 0 & , \text{ otherwise.} \end{cases}$$

The result is that the Weibull distribution is able to accommodate increasing $(\alpha > 1)$, decreasing $(\alpha < 1)$, or constant hazard rates $(\alpha = 1)$. In the case of $\alpha = 1$, the Weibull distribution reduces to the exponential distribution discussed previously. The expected value and variance of *X* when $X \sim Weibull(\alpha, \lambda)$ are given by

$$E(X) = \left[\Gamma\left(1 + \frac{1}{\alpha}\right)\right]\lambda^{-1/\alpha}$$

and

$$Var(X) = \left[\Gamma\left(1 + \frac{2}{\alpha}\right) - \left\{\Gamma\left(1 + \frac{1}{\alpha}\right)\right\}^2\right]\lambda^{-2/\alpha},$$

respectively. Here $\Gamma(\alpha) = \int_0^\infty u^{\alpha-1}\exp(-u)du$ represents the well-known gamma function. When α is an integer, $\Gamma(\alpha) = (\alpha-1)! = (\alpha-1)(\alpha-2)\ldots(1)$.

Since the univariate Weibull distribution can be obtained from an exponential distribution via a power transformation, a multivariate Weibull distribution can also be obtained from a multivariate exponential distribution via similar power transformations. Suppose $Y_1, Y_2, ..., Y_K$ have a multivariate distribution with exponential marginals such that $Y_k \sim Exponential(\lambda_k)$ for $k = 1, ..., K$. Taking $X_k = Y_k^{1/\alpha_k}$ will produce a multivariate distribution having Weibull marginals such that $X_k \sim Weibull(\alpha_k, \lambda_k)$ for $k = 1, ..., K$ (Kotz et al. 2000).

8.1 INDEPENDENT BIVARIATE WEIBULL

8.1.1 DATA GENERATION

Consider $U_1, U_2, ..., U_K \overset{iid}{\sim} Uniform(0,1)$ and let

$$X_k \mid Z = \left[\frac{-1}{\lambda_k} \exp(-\beta_k Z) \ln(1 - U_k) \right]^{-1/\alpha} \text{ for } k = 1, ..., K .$$

Therefore we have

$$X_k \mid Z \overset{ind}{\sim} Weibull(\alpha, \lambda_k \exp(\beta_k Z)) \text{ for } k = 1, ..., K .$$

Thus, the expected survival time and variance associated with the k^{th} cause of failure for those in the treatment group are given by

$$E[X_k \mid Z = 1] = \left[\Gamma\left(1 + \frac{1}{\alpha}\right) \right] [\lambda_k \exp(\beta_k)]^{-1/\alpha} \text{ for } k = 1, ..., K$$

and

$$Var[X_k \mid Z = 1] = \left[\Gamma\left(1 + \frac{2}{\alpha}\right) - \left\{ \Gamma\left(1 + \frac{1}{\alpha}\right) \right\}^2 \right] [\lambda_k \exp(\beta_k)]^{-2/\alpha} \text{ for } k = 1, ..., K ,$$

respectively. While the expected survival time and variance associated with the k^{th} cause of failure for those in the control group are given by

$$E[X_k \mid Z = 0] = \left[\Gamma\left(1 + \frac{1}{\alpha}\right) \right] [\lambda_k]^{-1/\alpha} \text{ for } k = 1, ..., K$$

and

$$Var[X_k \mid Z = 0] = \left[\Gamma\left(1 + \frac{2}{\alpha}\right) - \left\{ \Gamma\left(1 + \frac{1}{\alpha}\right) \right\}^2 \right] [\lambda_k]^{-2/\alpha} \text{ for } k = 1, ..., K ,$$

respectively.

We will then generate an independent censoring time, *C*, by considering $U_c \sim Uniform(0,1)$ and letting

$$C = \left[\frac{-1}{\lambda_c} \ln(1 - U_c) \right]^{-1/\alpha}$$

where $\lambda_c = \dfrac{p \sum_{k=1}^{2} \lambda_k \exp(\beta_k Z)}{1-p}$. Thus $C \sim Weibull(\alpha, \lambda_c)$ will produce an overall probability of censoring equal to *p*.

8.1.2 DETERMINING SIMULATION PARAMETERS

We will now describe the method used for determining the parameters required in order to simulate data from a desired bivariate Weibull distribution where the two causes of failure are independent of one another. We will first fix $\alpha > 1$ to indicate an increasing hazard and recall that the means, conditional on Z, are given by

$$E[X_k \mid Z] = \Gamma\left(1 + \frac{1}{\alpha}\right)(\lambda_k \exp(\beta_k Z))^{-1/\alpha} \text{ for } k = 1,2.$$

Suppose we wish to generate data having control means

$$E[X_k \mid Z = 0] = \mu_{0_k} \text{ for } k = 1,2.$$

Therefore we have that

$$E[X_k \mid Z = 0] = \mu_{0_k}$$

and

$$E\left[X_k \mid Z=0\right]=\Gamma\left(1+\frac{1}{\alpha}\right)\left(\lambda_k\right)^{-1/\alpha} \text{ for } k=1,2.$$

Thus we are able to generate data with the desired means by setting

$$\lambda_k=\left[\frac{\mu_{0_k}}{\Gamma\left(1+\frac{1}{\alpha}\right)}\right]^{-\alpha} \text{ for } k=1,2.$$

Now suppose we wish to generate data for the treatment groups such that they result in treatment means having a specified percentage increase (or decrease) over the control means. We will denote the treatment means by

$$E\left[X_k \mid Z=1\right]=\mu_{1_k} \text{ for } k=1,2.$$

The desired percentage increase over the control means will be denoted by r_k for $k=1,2$. Therefore we have that

$$E\left[X_k \mid Z=1\right]=\mu_{1_k}=\mu_{0_k}\left(1+r_k\right)$$

and

$$E\left[X_k \mid Z=1\right]=\mu_{1_k}=\Gamma\left(1+\frac{1}{\alpha}\right)\left(\lambda_k \exp\left(\beta_k\right)\right)^{-1/\alpha} \text{ for } k=1,2.$$

Thus we are able to generate data with the desired means by setting

$$\beta_k=\log\left(\frac{1}{\lambda_k}\left[\frac{\mu_{0_k}\left(1+r_k\right)}{\Gamma\left(1+\frac{1}{\alpha}\right)}\right]^{-\alpha}\right) \text{ for } k=1,2.$$

This method similarly holds for determining the parameters required to generate data from the Sarmanov bivariate Weibull distribution, the Cai-Prentice

bivariate Weibull distribution, and the bivariate Weibull distribution having a gamma frailty.

8.1.3 SIMULATION SUMMARY

For our Weibull marginals, consider the ratio of the treatment hazard and the control hazard:

$$\frac{h_{X_k}(x \mid Z=1)}{h_{X_k}(x \mid Z=0)} = \frac{\alpha\lambda_k \exp(\beta_k) x^{\alpha-1}}{\alpha\lambda_k x^{\alpha-1}} = \exp(\beta_k).$$

Note that this ratio does not depend on the value of *x*, indicating that the hazards are proportional for the Weibull distribution.

Therefore in this setting the proportional hazards assumption is again met, the event times due to the two competing causes of failure are independent, and the censoring times are independent of latent failure times of the competing risks. Therefore, we would again expect the test based on Cox's proportional hazards model (T^{CPH}) and the log-rank test (T^{LR}) to perform better than Gray's test (T^{G}) and the test due to Fine and Gray (T^{FG}).

In Table 8.1, it appears that T^{CPH} and T^{LR} are consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring, regardless of whether the beneficial treatment effect on type 2 failure is small or large. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small.

In Table 8.2, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. When the effect on type 2 failure is beneficial (panels 2 and 3 of Table 8.2), we see that the power results for T^{CPH} and T^{LR} remain similar to those found when there was no treatment effect on type 2 failure, however the power for T^{G} and T^{FG} drop considerably as the treatment's beneficial effect on type 2 gets larger.

In panel 1 of Table 8.3, it appears that T^{CPH} and T^{LR} are again consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small and when the amount of censoring is large.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 8.3, it appears that T^{G} and T^{FG} are now the more powerful tests when compared to T^{CPH} and T^{LR}. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases.

8.1.4 SIMULATION RESULTS

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.060	.062	.091	.109
	cens = 0.2	.047	.048	.079	.087
	cens = 0.5	.053	.052	.058	.064
	cens = 0.8	.051	.042	.055	.040
n = 100	cens = 0	.041	.042	.171	.194
	cens = 0.2	.050	.050	.122	.153
	cens = 0.5	.048	.048	.072	.080
	cens = 0.8	.046	.045	.061	.057
n = 200	cens = 0	.051	.053	.309	.351
	cens = 0.2	.046	.046	.234	.261
	cens = 0.5	.050	.052	.148	.155
	cens = 0.8	.053	.052	.070	.071
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.047	.047	.296	.433
	cens = 0.2	.050	.049	.216	.321
	cens = 0.5	.050	.051	.147	.205
	cens = 0.8	.048	.041	.064	.065
n = 100	cens = 0	.059	.060	.592	.736
	cens = 0.2	.051	.052	.465	.560
	cens = 0.5	.053	.054	.270	.337
	cens = 0.8	.058	.058	.127	.146
n = 200	cens = 0	.044	.045	.914	.963
	cens = 0.2	.049	.049	.778	.849
	cens = 0.5	.069	.069	.515	.585
	cens = 0.8	.061	.062	.233	.255

Table 8.1: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Independent Weibull).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.809	.809	.687	.650
	cens = 0.2	.717	.719	.538	.497
	cens = 0.5	.485	.494	.292	.255
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.970	.970	.927	.916
	cens = 0.2	.952	.952	.818	.801
	cens = 0.5	.796	.803	.533	.496
	cens = 0.8	.459	.461	.251	.202
n = 200	cens = 0	.999	.999	.997	.998
	cens = 0.2	1.00	1.00	.983	.979
	cens = 0.5	.983	.983	.850	.832
	cens = 0.8	.720	.726	.413	.379
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.846	.849	.497	.394
	cens = 0.2	.742	.745	.359	.275
	cens = 0.5	.530	.540	.214	.158
	cens = 0.8	.273	.276	.089	.059
n = 100	cens = 0	.995	.995	.737	.648
	cens = 0.2	.971	.971	.584	.486
	cens = 0.5	.843	.851	.321	.249
	cens = 0.8	.475	.489	.137	.089
n = 200	cens = 0	1.00	1.00	.942	.919
	cens = 0.2	1.00	1.00	.816	.744
	cens = 0.5	.988	.990	.538	.478
	cens = 0.8	.751	.760	.219	.188
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.869	.875	.243	.093
	cens = 0.2	.790	.797	.156	.068
	cens = 0.5	.586	.610	.108	.058
	cens = 0.8	.284	.302	.067	.038
n = 100	cens = 0	.993	.993	.335	.150
	cens = 0.2	.974	.976	.212	.083
	cens = 0.5	.843	.853	.115	.051
	cens = 0.8	.457	.475	.063	.037
n = 200	cens = 0	1.00	1.00	.421	.196
	cens = 0.2	.999	.999	.277	.100
	cens = 0.5	.990	.990	.138	.052
	cens = 0.8	.779	.790	.070	.046

* Programs did not converge due to sparse data.

Table 8.2: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Independent Weibull).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.047	.048	.116	.155
	cens = 0.2	.047	.046	.094	.126
	cens = 0.5	.052	.052	.077	.093
	cens = 0.8	.058	.050	.044	.046
n = 100	cens = 0	.057	.057	.230	.290
	cens = 0.2	.045	.045	.197	.233
	cens = 0.5	.058	.059	.119	.133
	cens = 0.8	.065	.058	.070	.068
n = 200	cens = 0	.058	.058	.441	.491
	cens = 0.2	.049	.048	.337	.375
	cens = 0.5	.051	.050	.214	.232
	cens = 0.8	.049	.047	.098	.107
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.732	.723	.875	.887
	cens = 0.2	.635	.623	.737	.743
	cens = 0.5	.430	.416	.464	.454
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.963	.960	.991	.993
	cens = 0.2	.905	.903	.971	.974
	cens = 0.5	.745	.738	.780	.782
	cens = 0.8	.391	.367	.378	.325
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	.997	.997	1.00	1.00
	cens = 0.5	.962	.962	.983	.984
	cens = 0.8	.666	.656	.670	.657

* Programs did not converge due to sparse data.

Table 8.3: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Independent Weibull).

8.2 SARMANOV BIVARIATE WEIBULL

8.2.1 DATA GENERATION

We will now consider failure times having Weibull marginals within the Sarmanov family of bivariate densities. We will first generate exponential failure times using the method previously described for the Sarmanov Bivariate Exponential case. Thus we will have

$$Y_1 \mid Z \sim \textit{Exponential}\left(\lambda_1 \exp\left(\beta_1 Z\right)\right)$$

and

$$Y_2 \mid Z \sim \textit{Exponential}\left(\lambda_2 \exp\left(\beta_2 Z\right)\right).$$

Letting $X_1 = Y_1^{1/\alpha}$ and $X_2 = Y_2^{1/\alpha}$ will produce a bivariate distribution having Weibull marginals such that

$$X_1 \mid Z \sim \textit{Weibull}\left(\alpha, \lambda_1 \exp\left(\beta_1 Z\right)\right)$$

and

$$X_2 \mid Z \sim \textit{Weibull}\left(\alpha, \lambda_2 \exp\left(\beta_2 Z\right)\right).$$

We will then generate censoring variables $\left(C_1, C_2\right)$ independently of each other and of $\left(X_1, X_2\right) \mid Z$ where C_k is the censoring time associated with failure time X_k for $k = 1,2$. The censoring variables will follow Weibull distributions with $C_k \sim \textit{Weibull}\left(\alpha, \lambda_{c_k}\right)$ for $k = 1,2$ where the λ_{c_k} are chosen in such a way that they result in censoring probabilities

$$p_1 = P\left(C_1 < X_1 \mid Z\right)$$

and

$$p_2 = P(C_2 < X_2 \mid Z).$$

Defining $\lambda_{c_1} = \dfrac{p_1(\lambda_1^*)}{(1-p_1)}$ and $\lambda_{c_2} = \dfrac{p_2(\lambda_2^*)}{(1-p_2)}$ where $\lambda_k^* = \lambda_k \exp(\beta_k Z)$ for $k = 1,2$ will result in censoring probabilities of p_1 and p_2, respectively. Thus we will consider $U_{c_k} \sim Uniform(0,1)$ for $k = 1,2$ and let

$$C_k = \left[\frac{-1}{\lambda_{c_k}} \ln(1 - U_{c_k})\right]^{-1/\alpha}$$

so that $C_k \sim Weibull(\alpha, \lambda_{c_k})$ will produce censoring probabilities equal to p_k for $k = 1,2$.

8.2.2 SIMULATION SUMMARY

Not surprisingly, the results for the Sarmanov bivariate Weibull having a correlation coefficient of zero (Tables 8.4 – 8.6) are very similar to those of the independent bivariate Weibull distribution given above. However, we will also consider the case where the competing risks are positively correlated. Thus violating the independence assumption inherent in the test based on Cox's proportional hazards model and the log-rank test.

In Table 8.4, it appears that T^{CPH} and T^{LR} are consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring, regardless of whether the beneficial treatment effect on type 2 failure is small or large. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample

size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small. When we allowed for the competing risks to be positively correlated (Tables 8.7, 8.10, and 8.13), we again saw similar trends at each level of correlation. Furthermore the results did not appear to vary greatly across the correlations being considered.

In Table 8.5, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. When the effect on type 2 failure is beneficial (panels 2 and 3 of Table 8.5), we see that the power results for T^{CPH} and T^{LR} remain similar to those found when there was no treatment effect on type 2 failure, however the power for T^{G} and T^{FG} drop considerably as the treatment's beneficial effect on type 2 failure gets larger. When we allowed for the competing risks to be positively correlated (Tables 8.8, 8.11, and 8.14), we again saw similar trends at each level of correlation. All tests also appeared to gain power as the level of correlation between the competing risks increased. Although T^{CPH} and T^{LR} were consistently more powerful than T^{G} and T^{FG} at all correlation levels, T^{G} and T^{FG} did gain ground as the level of correlation was increased.

In panel 1 of Table 8.6, it appears that T^{CPH} and T^{LR} are again consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small and when the amount of censoring is large. When we allowed for the competing risks to be positively correlated (panel 1 of Tables 8.9, 8.12, and

8.15), we again saw similar trends at each level of correlation. Furthermore the results did not appear to vary greatly across the correlations being considered.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 8.6, it appears that T^{G} and T^{FG} are now generally the more powerful tests when compared to T^{CPH} and T^{LR}. Again, all tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. When we allowed for the competing risks to be positively correlated (panel 2 of Tables 8.9, 8.12, and 8.15), we again saw similar trends at each level of correlation. All tests also appeared to gain power as the level of correlation between the competing risks increased.

8.2.3 SIMULATION RESULTS

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.064	.064	.111	.133
	cens = 0.2	.054	.057	.073	.083
	cens = 0.5	.055	.055	.075	.081
	cens = 0.8	.057	.053	.051	.043
n = 100	cens = 0	.047	.047	.160	.189
	cens = 0.2	.053	.051	.130	.155
	cens = 0.5	.042	.042	.087	.098
	cens = 0.8	.048	.047	.064	.059
n = 200	cens = 0	.050	.049	.289	.330
	cens = 0.2	.041	.040	.228	.256
	cens = 0.5	.048	.047	.147	.164
	cens = 0.8	.054	.052	.091	.088
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.055	.053	.314	.440
	cens = 0.2	.053	.057	.224	.302
	cens = 0.5	.048	.048	.115	.165
	cens = 0.8	.048	.040	.058	.054
n = 100	cens = 0	.053	.054	.604	.718
	cens = 0.2	.045	.044	.459	.560
	cens = 0.5	.056	.055	.279	.336
	cens = 0.8	.047	.045	.132	.140
n = 200	cens = 0	.050	.051	.902	.943
	cens = 0.2	.048	.050	.786	.854
	cens = 0.5	.047	.046	.525	.603
	cens = 0.8	.043	.044	.241	.270

Table 8.4: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Sarmanov Weibull, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.815	.816	.710	.680
	cens = 0.2	.721	.722	.544	.500
	cens = 0.5	.542	.544	.342	.295
	cens = 0.8	.251	.242	.139	.102
n = 100	cens = 0	.982	.982	.932	.923
	cens = 0.2	.951	.952	.822	.791
	cens = 0.5	.790	.791	.547	.505
	cens = 0.8	.429	.432	.214	.168
n = 200	cens = 0	1.00	1.00	.995	.994
	cens = 0.2	.999	.999	.979	.979
	cens = 0.5	.984	.985	.834	.815
	cens = 0.8	.726	.729	.411	.382
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.838	.840	.475	.388
	cens = 0.2	.745	.750	.350	.272
	cens = 0.5	.547	.561	.199	.144
	cens = 0.8	.246	.250	.064	.037
n = 100	cens = 0	.989	.989	.727	.645
	cens = 0.2	.957	.958	.550	.465
	cens = 0.5	.841	.846	.329	.261
	cens = 0.8	.459	.470	.143	.098
n = 200	cens = 0	1.00	1.00	.952	.932
	cens = 0.2	1.00	1.00	.822	.768
	cens = 0.5	.990	.990	.584	.501
	cens = 0.8	.739	.751	.192	.148
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.870	.879	.212	.091
	cens = 0.2	.750	.764	.182	.075
	cens = 0.5	.592	.606	.107	.050
	cens = 0.8	.272	.289	.054	.035
n = 100	cens = 0	.997	.998	.306	.123
	cens = 0.2	.967	.968	.203	.073
	cens = 0.5	.875	.884	.111	.038
	cens = 0.8	.501	.530	.070	.054
n = 200	cens = 0	1.00	1.00	.381	.176
	cens = 0.2	1.00	1.00	.255	.114
	cens = 0.5	.994	.994	.134	.059
	cens = 0.8	.788	.804	.074	.039

Table 8.5: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Sarmanov Weibull, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.048	.050	.108	.149
	cens = 0.2	.051	.050	.094	.129
	cens = 0.5	.056	.054	.085	.087
	cens = 0.8	.074	.056	.057	.045
n = 100	cens = 0	.070	.070	.221	.273
	cens = 0.2	.058	.059	.146	.192
	cens = 0.5	.055	.058	.121	.137
	cens = 0.8	.056	.049	.073	.068
n = 200	cens = 0	.046	.046	.441	.502
	cens = 0.2	.058	.059	.336	.372
	cens = 0.5	.048	.049	.197	.213
	cens = 0.8	.053	.053	.097	.093
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.749	.744	.882	.892
	cens = 0.2	.647	.635	.751	.765
	cens = 0.5	.447	.429	.470	.451
	cens = 0.8	.259	.215	.210	.198
n = 100	cens = 0	.953	.950	.990	.989
	cens = 0.2	.921	.918	.970	.970
	cens = 0.5	.763	.752	.802	.806
	cens = 0.8	.408	.389	.386	.352
n = 200	cens = 0	.998	.998	1.00	1.00
	cens = 0.2	.995	.995	1.00	1.00
	cens = 0.5	.959	.958	.980	.979
	cens = 0.8	.666	.656	.667	.646

Table 8.6: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Sarmanov Weibull, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.063	.064	.143	.163
	cens = 0.2	.052	.052	.097	.113
	cens = 0.5	.056	.054	.062	.067
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.054	.054	.188	.226
	cens = 0.2	.055	.055	.132	.148
	cens = 0.5	.049	.048	.082	.095
	cens = 0.8	.053	.049	.066	.064
n = 200	cens = 0	.046	.046	.348	.376
	cens = 0.2	.042	.041	.259	.279
	cens = 0.5	.042	.043	.161	.170
	cens = 0.8	.055	.055	.078	.081
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.051	.053	.343	.505
	cens = 0.2	.047	.048	.249	.324
	cens = 0.5	.059	.058	.152	.203
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.048	.050	.680	.819
	cens = 0.2	.042	.042	.478	.589
	cens = 0.5	.050	.051	.273	.344
	cens = 0.8	.066	.065	.135	.153
n = 200	cens = 0	.054	.056	.940	.974
	cens = 0.2	.056	.056	.805	.868
	cens = 0.5	.048	.051	.544	.609
	cens = 0.8	.050	.050	.217	.251

* Programs did not converge due to sparse data.

Table 8.7: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Sarmanov Weibull, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.925	.926	.818	.787
	cens = 0.2	.843	.844	.637	.584
	cens = 0.5	.604	.607	.363	.268
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.998	.998	.983	.977
	cens = 0.2	.989	.989	.922	.906
	cens = 0.5	.881	.882	.590	.535
	cens = 0.8	.190	.172	.084	.070
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	.998	.996
	cens = 0.5	.994	.995	.881	.857
	cens = 0.8	.287	.283	.153	.103
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.891	.896	.516	.430
	cens = 0.2	.776	.784	.365	.264
	cens = 0.5	.568	.582	.175	.122
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.993	.994	.792	.723
	cens = 0.2	.984	.984	.619	.533
	cens = 0.5	.851	.858	.303	.242
	cens = 0.8	.461	.476	.091	.059
n = 200	cens = 0	1.00	1.00	.966	.944
	cens = 0.2	1.00	1.00	.870	.817
	cens = 0.5	.993	.993	.485	.414
	cens = 0.8	.728	.749	.125	.093
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.897	.901	.264	.100
	cens = 0.2	.821	.831	.168	.066
	cens = 0.5	.591	.608	.090	.046
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.995	.995	.321	.127
	cens = 0.2	.986	.986	.200	.070
	cens = 0.5	.888	.897	.098	.051
	cens = 0.8	.482	.510	.055	.048
n = 200	cens = 0	1.00	1.00	.422	.199
	cens = 0.2	1.00	1.00	.249	.091
	cens = 0.5	.993	.994	.106	.052
	cens = 0.8	.742	.755	.047	.060

* Programs did not converge due to sparse data.

Table 8.8: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Sarmanov Weibull, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.064	.065	.150	.191
	cens = 0.2	.053	.056	.103	.137
	cens = 0.5	.056	.052	.077	.088
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.062	.062	.281	.332
	cens = 0.2	.059	.059	.218	.253
	cens = 0.5	.051	.050	.124	.151
	cens = 0.8	.059	.054	.085	.081
n = 200	cens = 0	.045	.045	.516	.568
	cens = 0.2	.052	.052	.416	.474
	cens = 0.5	.057	.056	.226	.260
	cens = 0.8	.048	.045	.125	.125
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.803	.800	.931	.946
	cens = 0.2	.685	.675	.801	.814
	cens = 0.5	.504	.487	.533	.533
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.981	.980	.998	.998
	cens = 0.2	.955	.953	.981	.983
	cens = 0.5	.788	.784	.829	.831
	cens = 0.8	.373	.355	.320	.277
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	.999	.999	1.00	1.00
	cens = 0.5	.974	.973	.984	.984
	cens = 0.8	.632	.626	.587	.569

* Programs did not converge due to sparse data.

Table 8.9: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Sarmanov Weibull, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.048	.048	.108	.127
	cens = 0.2	.053	.054	.087	.101
	cens = 0.5	.050	.051	.072	.075
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.038	.038	.189	.217
	cens = 0.2	.047	.047	.126	.154
	cens = 0.5	.049	.049	.083	.091
	cens = 0.8	.061	.061	.068	.060
n = 200	cens = 0	.053	.053	.316	.346
	cens = 0.2	.053	.054	.212	.229
	cens = 0.5	.061	.060	.136	.144
	cens = 0.8	.042	.040	.062	.066
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.052	.054	.298	.467
	cens = 0.2	.049	.049	.225	.319
	cens = 0.5	.049	.050	.111	.148
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.044	.044	.609	.773
	cens = 0.2	.059	.059	.410	.531
	cens = 0.5	.041	.039	.229	.291
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.043	.042	.911	.959
	cens = 0.2	.065	.064	.716	.793
	cens = 0.5	.043	.044	.457	.538
	cens = 0.8	.051	.052	.199	.230

* Programs did not converge due to sparse data.

Table 8.10: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Sarmanov Weibull, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.963	.963	.863	.825
	cens = 0.2	.842	.847	.676	.595
	cens = 0.5	.420	.398	.197	.157
	cens = 0.8	*	*	*	*
n = 100	cens = 0	1.00	1.00	.990	.985
	cens = 0.2	.988	.988	.905	.885
	cens = 0.5	.700	.696	.440	.339
	cens = 0.8	*	*	*	*
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	.997	.994
	cens = 0.5	.913	.914	.716	.635
	cens = 0.8	.068	.059	.048	.033
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.922	.926	.626	.533
	cens = 0.2	.844	.849	.412	.305
	cens = 0.5	.621	.628	.187	.142
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.997	.997	.877	.830
	cens = 0.2	.985	.985	.666	.587
	cens = 0.5	.886	.890	.299	.231
	cens = 0.8	*	*	*	*
n = 200	cens = 0	1.00	1.00	.982	.978
	cens = 0.2	1.00	1.00	.903	.881
	cens = 0.5	.994	.994	.498	.418
	cens = 0.8	.720	.739	.123	.101
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.932	.935	.324	.129
	cens = 0.2	.858	.864	.207	.072
	cens = 0.5	.647	.669	.108	.053
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.999	.999	.445	.219
	cens = 0.2	.989	.991	.278	.104
	cens = 0.5	.900	.910	.118	.061
	cens = 0.8	*	*	*	*
n = 200	cens = 0	1.00	1.00	.623	.412
	cens = 0.2	1.00	1.00	.337	.151
	cens = 0.5	.994	.995	.106	.048
	cens = 0.8	.744	.772	.047	.057

* Programs did not converge due to sparse data.

Table 8.11: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Sarmanov Weibull, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.052	.053	.134	.167
	cens = 0.2	.048	.049	.105	.127
	cens = 0.5	.040	.040	.082	.095
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.048	.051	.284	.331
	cens = 0.2	.064	.063	.219	.258
	cens = 0.5	.054	.055	.125	.143
	cens = 0.8	.050	.047	.074	.064
n = 200	cens = 0	.048	.047	.548	.590
	cens = 0.2	.056	.057	.434	.468
	cens = 0.5	.056	.058	.227	.259
	cens = 0.8	.061	.053	.113	.114
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.854	.848	.953	.958
	cens = 0.2	.763	.758	.850	.863
	cens = 0.5	.514	.503	.502	.496
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.989	.989	1.00	1.00
	cens = 0.2	.971	.971	.991	.993
	cens = 0.5	.828	.825	.820	.826
	cens = 0.8	*	*	*	*
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.983	.983	.982	.982
	cens = 0.8	.643	.635	.576	.564

* Programs did not converge due to sparse data.

Table 8.12: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Sarmanov Weibull, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.046	.047	.102	.126
	cens = 0.2	.048	.047	.074	.096
	cens = 0.5	.040	.040	.053	.057
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.048	.048	.151	.171
	cens = 0.2	.058	.061	.104	.121
	cens = 0.5	.059	.058	.085	.090
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.051	.052	.325	.354
	cens = 0.2	.059	.060	.201	.224
	cens = 0.5	.042	.043	.136	.146
	cens = 0.8	.053	.051	.071	.071
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.058	.059	.246	.385
	cens = 0.2	.040	.044	.147	.228
	cens = 0.5	*	*	*	*
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.035	.037	.466	.667
	cens = 0.2	.052	.053	.304	.406
	cens = 0.5	.060	.058	.199	.255
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.058	.060	.814	.924
	cens = 0.2	.053	.053	.617	.720
	cens = 0.5	.065	.063	.390	.467
	cens = 0.8	.062	.065	.231	.259

* Programs did not converge due to sparse data.

Table 8.13: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Sarmanov Weibull, corr = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.958	.957	.844	.775
	cens = 0.2	.818	.812	.625	.460
	cens = 0.5	*	*	*	*
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.999	.999	.984	.979
	cens = 0.2	.979	.979	.863	.807
	cens = 0.5	.370	.365	.214	.140
	cens = 0.8	*	*	*	*
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	.994	.989
	cens = 0.5	.643	.643	.413	.336
	cens = 0.8	.059	.055	.054	.039
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.960	.960	.709	.628
	cens = 0.2	.882	.888	.454	.376
	cens = 0.5	.632	.647	.183	.119
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.999	1.00	.936	.905
	cens = 0.2	.996	.996	.710	.618
	cens = 0.5	.903	.908	.294	.217
	cens = 0.8	*	*	*	*
n = 200	cens = 0	1.00	1.00	.997	.997
	cens = 0.2	1.00	1.00	.911	.877
	cens = 0.5	.995	.995	.458	.374
	cens = 0.8	.723	.739	.105	.079
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.951	.953	.429	.186
	cens = 0.2	.883	.889	.234	.082
	cens = 0.5	*	*	*	*
	cens = 0.8	*	*	*	*
n = 100	cens = 0	1.00	1.00	.611	.322
	cens = 0.2	.993	.993	.307	.128
	cens = 0.5	.929	.931	.124	.048
	cens = 0.8	*	*	*	*
n = 200	cens = 0	1.00	1.00	.734	.558
	cens = 0.2	1.00	1.00	.410	.194
	cens = 0.5	.997	.997	.142	.051
	cens = 0.8	.737	.753	.051	.069

* Programs did not converge due to sparse data.

Table 8.14: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Sarmanov Weibull, corr = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.061	.064	.160	.209
	cens = 0.2	.046	.047	.110	.129
	cens = 0.5	.047	.046	.089	.099
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.061	.061	.283	.343
	cens = 0.2	.064	.065	.214	.246
	cens = 0.5	.043	.043	.113	.132
	cens = 0.8	.051	.044	.078	.068
n = 200	cens = 0	.054	.054	.538	.585
	cens = 0.2	.039	.040	.381	.428
	cens = 0.5	.056	.056	.232	.249
	cens = 0.8	.044	.043	.096	.099
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.899	.895	.979	.984
	cens = 0.2	.783	.781	.863	.872
	cens = 0.5	.567	.563	.514	.515
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.995	.993	1.00	1.00
	cens = 0.2	.978	.977	.991	.991
	cens = 0.5	.839	.837	.819	.826
	cens = 0.8	*	*	*	*
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.990	.990	.983	.984
	cens = 0.8	.641	.632	.545	.527

* Programs did not converge due to sparse data.

Table 8.15: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Sarmanov Weibull, corr = 0.8).

8.3 MARSHALL-OLKIN BIVARIATE WEIBULL

8.3.1 DATA GENERATION

We now discuss another bivariate Weibull distribution that was first introduced by Marshall and Olkin (1967) and later discussed by Moeschberger (1974). We will generate $U_1, U_2, U_{12} \overset{iid}{\sim} Uniform(0,1)$ and let

$$Y_1 = \left[\frac{-1}{\lambda_1}\exp(-\beta_1 Z)\ln(1-U_1)\right]^{-1/\alpha},$$

$$Y_2 = \left[\frac{-1}{\lambda_2}\exp(-\beta_2 Z)\ln(1-U_2)\right]^{-1/\alpha},$$

and

$$Y_{12} = \left[\frac{-1}{\lambda_{12}}\exp(-\beta_{12} Z)\ln(1-U_{12})\right]^{-1/\alpha}.$$

Thus Y_1, Y_2, and Y_{12} will be conditionally independent Weibull random variables given Z such that

$$Y_1 \mid Z \sim Weibull(\alpha, \lambda_1 \exp(\beta_1 Z)),$$

$$Y_2 \mid Z \sim Weibull(\alpha, \lambda_2 \exp(\beta_2 Z)),$$

and

$$Y_{12} \mid Z \sim Weibull(\alpha, \lambda_{12} \exp(\beta_{12} Z)).$$

As before, we obtain dependent random variables X_1 and X_2 by letting

$$X_1 = \min(Y_1, Y_{12})$$

and

$$X_2 = \min(Y_2, Y_{12}).$$

Thus we have

$$X_1 \mid Z \sim Weibull\left(\alpha, \lambda_1 \exp\left(\beta_1 Z\right) + \lambda_{12} \exp\left(\beta_{12} Z\right)\right),$$

$$X_2 \mid Z \sim Weibull\left(\alpha, \lambda_2 \exp\left(\beta_2 Z\right) + \lambda_{12} \exp\left(\beta_{12} Z\right)\right),$$

and

$$\left(X_1, X_2\right) \mid Z \sim MOBVW\left(\alpha, \lambda_1 \exp\left(\beta_1 Z\right), \lambda_2 \exp\left(\beta_2 Z\right), \lambda_{12} \exp\left(\beta_{12} Z\right)\right).$$

We will again generate a censoring time, *C*, that is conditionally independent of $X_1 \mid Z$ and $X_2 \mid Z$ by considering $U_c \sim Uniform(0,1)$ and letting

$$C = \left[\frac{-1}{\lambda_c} \ln\left(1 - U_c\right)\right]^{-1/\alpha}$$

where $\lambda_c = \frac{p\lambda_m}{1-p}$ and $\lambda_m = \lambda_1 \exp\left(\beta_1 Z\right) + \lambda_2 \exp\left(\beta_2 Z\right) + \lambda_{12} \exp\left(\beta_{12} Z\right)$. Thus $C \sim Weibull\left(\alpha, \lambda_c\right)$ will produce an overall probability of censoring equal to *p*.

8.3.2 DETERMINING SIMULATION PARAMETERS

We will now describe the method used for determining the parameters required in order to simulate data from a desired Marshall-Olkin bivariate Weibull distribution. We will first fix $\alpha > 1$ to indicate an increasing hazard and recall that the means, conditional on Z, are given by

$$E\left[X_k \mid Z\right] = \Gamma\left(1 + \frac{1}{\alpha}\right)\left[\lambda_k \exp\left(\beta_k Z\right) + \lambda_{12} \exp\left(\beta_{12} Z\right)\right]^{-1/\alpha} \text{ for } k = 1,2.$$

Additionally, the Marshall-Olkin bivariate Weibull distribution allows for the situation where both causes of failure occur simultaneously. The probability of this event occurring is given by

$$P(X_1 = X_2 \mid Z) = \frac{\lambda_{12} \exp(\beta_{12} Z)}{\lambda_1 \exp(\beta_1 Z) + \lambda_2 \exp(\beta_2 Z) + \lambda_{12} \exp(\beta_{12} Z)}.$$

Suppose we wish to generate data having control means

$$E[X_k \mid Z = 0] = \mu_{0_k} \text{ for } k = 1,2$$

and a probability of simultaneous failure given by

$$P(X_1 = X_2 \mid Z = 0) = p_{12}.$$

Therefore we have that

$$E[X_k \mid Z = 0] = \mu_{0_k} = \Gamma\left(1 + \frac{1}{\alpha}\right)[\lambda_k + \lambda_{12}]^{-1/\alpha} \text{ for } k = 1,2$$

and

$$P(X_1 = X_2 \mid Z = 0) = p_{12} = \frac{\lambda_{12}}{\lambda_1 + \lambda_2 + \lambda_{12}}.$$

We now have three equations which can be solved for the three unknown parameters (λ_1, λ_2, and λ_{12}). Thus we are able to generate data with the desired control means and probability of simultaneous failure by setting

$$\lambda_{12} = \left[\left(\frac{\mu_{0_1}}{\Gamma\left(1 + \frac{1}{\alpha}\right)} \right)^{-\alpha} + \left(\frac{\mu_{0_2}}{\Gamma\left(1 + \frac{1}{\alpha}\right)} \right)^{-\alpha} \right] \left(\frac{p_{12}}{1 + p_{12}} \right)$$

and

$$\lambda_k = \left(\frac{\mu_{0_k}}{\Gamma\left(1 + \frac{1}{\alpha}\right)} \right)^{-\alpha} - \lambda_{12} \text{ for } k = 1,2.$$

As before, we now wish to generate data for the treatment groups such that they result in treatment means having a specified percentage increase (or decrease) over the control means. Again the treatment means will be denoted by

$$E\left[X_k \mid Z=1\right]=\mu_{1_k} \quad \text{for } k=1,2$$

and the desired percentage increase over the control means will be given by r_k for $k=1,2$. We will desire the same probability of simultaneous failure that we had in the control group, which is again given by

$$P\left(X_1=X_2 \mid Z=1\right)=p_{12}.$$

Therefore we have that

$$E\left[X_k \mid Z=1\right]=\mu_{1_k}=\mu_{0_k}\left(1+r_k\right)=\Gamma\left(1+\frac{1}{\alpha}\right)\left[\lambda_k \exp\left(\beta_k\right)+\lambda_{12} \exp\left(\beta_{12}\right)\right]^{-1/\alpha} \quad \text{for } k=1,2$$

and

$$P\left(X_1=X_2 \mid Z=1\right)=p_{12}=\frac{\lambda_{12} \exp\left(\beta_{12}\right)}{\lambda_1 \exp\left(\beta_1\right)+\lambda_2 \exp\left(\beta_2\right)+\lambda_{12} \exp\left(\beta_{12}\right)}.$$

We now have three equations which can be solved for the three unknown parameters (β_1, β_2, and β_{12}). Thus we are able to generate data with the desired treatment means and probability of simultaneous failure by setting

$$\beta_{12}=\log\left\{\frac{p_{12}}{\left(1+p_{12}\right)\lambda_{12}}\left(\left[\frac{\mu_{0_1}\left(1+r_1\right)}{\Gamma\left(1+\frac{1}{\alpha}\right)}\right]^{-\alpha}+\left[\frac{\mu_{0_2}\left(1+r_2\right)}{\Gamma\left(1+\frac{1}{\alpha}\right)}\right]^{-\alpha}\right)\right\}$$

and

$$\beta_k = \log\left\{\frac{1}{\lambda_k}\left[\frac{\mu_{0_k}(1+r_k)}{\Gamma\left(1+\frac{1}{\alpha}\right)}\right]^{-\alpha} - \frac{\lambda_{12}}{\lambda_k}\exp(\beta_{12})\right\} \text{ for } k = 1,2.$$

8.3.3 SIMULATION SUMMARY

The following results are for competing risks data that follow a Marshall-Olkin bivariate Weibull distribution with a 20% probabililty of simultaneous failure. Therefore the competing risks will not be independent of one another and that assumption will be violated for the test based on Cox's proportional hazards model and the log-rank test.

In Table 8.16, it appears that T^{CPH} and T^{LR} are consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring when the beneficial treatment effect on type 2 failure is small. When the beneficial treatment effect on type 2 failure is large, T^{CPH} and T^{LR} appear to best preserve the nominal alpha level for smaller sample sizes or when the amount of censoring is high. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small.

In Table 8.17, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. We also see that the power for each test diminishes as the treatment's beneficial effect on type 2 failure gets larger.

In panel 1 of Table 8.18, it appears that T^{CPH} and T^{LR} are again consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. All tests appear to be better at preserving the nominal alpha level for smaller sample sizes and when the amount of censoring is large.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 8.18, it appears that T^{G} and T^{FG} are now generally the more powerful tests when compared to T^{CPH} and T^{LR}. Again, all tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases.

8.3.4 SIMULATION RESULTS

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.060	.060	.133	.163
	cens = 0.2	.037	.035	.090	.108
	cens = 0.5	.084	.083	.103	.105
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.062	.060	.251	.290
	cens = 0.2	.056	.055	.193	.214
	cens = 0.5	.058	.059	.114	.124
	cens = 0.8	*	*	*	*
n = 200	cens = 0	.080	.081	.476	.517
	cens = 0.2	.080	.080	.351	.379
	cens = 0.5	.067	.066	.211	.235
	cens = 0.8	.058	.056	.111	.108
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.092	.086	.501	.656
	cens = 0.2	.075	.073	.360	.493
	cens = 0.5	.063	.060	.220	.280
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.116	.114	.856	.911
	cens = 0.2	.096	.094	.664	.767
	cens = 0.5	.089	.083	.448	.504
	cens = 0.8	.063	.057	.167	.193
n = 200	cens = 0	.164	.154	.994	.999
	cens = 0.2	.125	.123	.953	.975
	cens = 0.5	.107	.103	.779	.834
	cens = 0.8	.077	.071	.326	.356

* Programs did not converge due to sparse data.

Table 8.16: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Marshall-Olkin Weibull, $P(X_1=X_2)=0.20$).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.948	.948	.888	.870
	cens = 0.2	.879	.873	.776	.749
	cens = 0.5	.685	.679	.526	.456
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.998	.998	.991	.989
	cens = 0.2	.994	.994	.968	.959
	cens = 0.5	.919	.918	.797	.769
	cens = 0.8	.592	.588	.373	.309
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.999	.999	.976	.974
	cens = 0.8	.869	.871	.669	.640
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.864	.869	.576	.512
	cens = 0.2	.783	.787	.462	.390
	cens = 0.5	.574	.580	.252	.195
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.998	.998	.838	.791
	cens = 0.2	.976	.978	.683	.613
	cens = 0.5	.871	.876	.444	.398
	cens = 0.8	.470	.488	.174	.133
n = 200	cens = 0	1.00	1.00	.980	.974
	cens = 0.2	1.00	1.00	.937	.911
	cens = 0.5	.992	.993	.722	.684
	cens = 0.8	.789	.810	.319	.276
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.801	.809	.180	.067
	cens = 0.2	.692	.709	.120	.053
	cens = 0.5	.486	.508	.080	.043
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.976	.979	.203	.076
	cens = 0.2	.920	.927	.158	.076
	cens = 0.5	.768	.781	.084	.043
	cens = 0.8	.406	.427	.066	.042
n = 200	cens = 0	1.00	1.00	.277	.112
	cens = 0.2	1.00	1.00	.190	.073
	cens = 0.5	.965	.965	.097	.046
	cens = 0.8	.666	.682	.060	.038

* Programs did not converge due to sparse data.

Table 8.17: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Marshall-Olkin Weibull, $P(X_1=X_2)=0.20$).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.075	.070	.198	.251
	cens = 0.2	.064	.060	.154	.193
	cens = 0.5	.068	.063	.117	.132
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.095	.091	.400	.443
	cens = 0.2	.075	.074	.314	.343
	cens = 0.5	.071	.067	.204	.225
	cens = 0.8	.062	.056	.092	.086
n = 200	cens = 0	.134	.133	.717	.749
	cens = 0.2	.115	.111	.573	.612
	cens = 0.5	.100	.097	.375	.398
	cens = 0.8	.074	.069	.156	.153
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.990	.988	.999	.999
	cens = 0.2	.978	.973	.986	.981
	cens = 0.5	.873	.840	.882	.851
	cens = 0.8	*	*	*	*
n = 100	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.990	.990	.994	.994
	cens = 0.8	.811	.764	.761	.713
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	.974	.970	.977	.965

* Programs did not converge due to sparse data.

Table 8.18: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Marshall-Olkin Weibull, $P(X_1=X_2)=0.20$).

8.4 CAI-PRENTICE BIVARIATE WEIBULL

8.4.1 DATA GENERATION

We will let $U_1, U_2 \overset{iid}{\sim} Uniform(0,1)$ and generate

$$X_2 = \left[-\frac{1}{\lambda_2} \log(1 - U_2) \exp(-\beta_2 Z) \right]^{-1/\alpha}$$

and

$$X_1 = \left[\frac{1}{\lambda_1} \theta \log\left[(1-a) + a(1-U_1)^{-(1+\theta)^{-1}} \right] \exp(-\beta_1 Z) \right]^{-1/\alpha}$$

where $a = (1 - U_2)^{-\theta^{-1}}$ and Z is the treatment indicator. Thus

$$X_k \mid Z \sim Weibull(\alpha, \lambda_k^*)$$

with

$$E(X_k \mid Z) = \left[\Gamma\left(1 + \frac{1}{\alpha}\right) \right] (\lambda_k^*)^{-1/\alpha}$$

where

$$\lambda_k^* = \lambda_k \exp(\beta_k Z) \text{ for } k = 1,2 .$$

We will then generate censoring variables (C_1, C_2) independently of each other and of $(X_1, X_2) \mid Z$ where C_k is the censoring time associated with failure time X_k for $k = 1,2$. The censoring variables will follow Weibull distributions with $C_k \sim Weibull(\alpha, \lambda_{c_k})$ for $k = 1,2$ where the λ_{c_k} are chosen in such a way that they result in censoring probabilities

$$p_1 = P(C_1 < X_1 \mid Z)$$

and

$$p_2 = P(C_2 < X_2 \mid Z).$$

Defining $\lambda_{c_1} = \frac{p_1(\lambda_1^*)}{(1-p_1)}$ and $\lambda_{c_2} = \frac{p_2(\lambda_2^*)}{(1-p_2)}$ will result in censoring probabilities of p_1 and p_2, respectively. Thus we will consider $U_{c_k} \sim Uniform(0,1)$ for $k = 1,2$ and let

$$C_k = \left[\frac{-1}{\lambda_{c_k}} \ln(1 - U_{c_k})\right]^{-1/\alpha}$$

so that $C_k \sim Weibull(\alpha, \lambda_{c_k})$ will produce censoring probabilities equal to p_k for $k = 1,2$.

8.4.2 SIMULATION SUMMARY

Like those of the Sarmanov bivariate Weibull distribution with no correlation, the results for the Cai-Prentice bivariate Weibull having a Kendall's tau coefficient of zero (Tables 8.19 – 8.21) are also very similar to those of the independent bivariate Weibull distribution given above. However, we will also consider the case where the competing risks have a positive Kendall's tau coefficient. Thus violating the independence assumption inherent in the test based on Cox's proportional hazards model and the log-rank test.

In Table 8.19, it appears that T^{CPH} and T^{LR} are consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring,

regardless of whether the beneficial treatment effect on type 2 failure is small or large. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small. When we allowed for the competing risks to be positively correlated (Tables 8.22, 8.25, and 8.28), we saw that all tests were better at preserving the nominal alpha level when the sample size was small, when the amount of censoring was large, and when the beneficial treatment effect on type 2 failure was small. Furthermore, each test's ability to preserve the nominal alpha level appeared to worsen as the positive correlation between the two competing risks increased.

In Table 8.20, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. When the effect on type 2 failure is beneficial (panels 2 and 3 of Table 8.20), we see that the power results for T^{CPH} and T^{LR} remain similar to those found when there was no treatment effect on type 2 failure, however T^{G} and T^{FG} lose considerable power as the treatment's beneficial effect on type 2 failure gets larger. When we allowed for the competing risks to be positively correlated (Tables 8.23, 8.26, and 8.29), we again saw that all tests appeared to gain power for larger sample sizes and lose power as the amount of censoring increased. However when the correlation was introduced, all tests now displayed a loss of power as the treatment's beneficial effect on type 2 got larger.

In panel 1 of Table 8.21, it appears that T^{CPH} and T^{LR} are again consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and

T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small and perform quite well when the amount of censoring is large. When we allowed for the competing risks to be positively correlated (panel 1 of Tables 8.24, 8.27, and 8.30), we saw that all tests were better at preserving the nominal alpha level for small sample sizes and when the amount of censoring was large. Furthermore, each test's ability to preserve the nominal alpha level appeared to worsen as the positive correlation between the two competing risks increased.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 8.21, it appears that T^{G} and T^{FG} are now generally the more powerful tests when compared to T^{CPH} and T^{LR}. Again, all tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. When we allowed for the competing risks to be positively correlated (panel 2 of Tables 8.24, 8.27, and 8.30), we again saw that all tests appeared to gain power for larger sample sizes and lose power as the amount of censoring increased. Also, the power for each test rose as the correlation between the competing risks increased.

8.4.3 SIMULATION RESULTS

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.052	.052	.085	.109
	cens = 0.2	.058	.059	.075	.087
	cens = 0.5	.051	.051	.067	.080
	cens = 0.8	.066	.061	.049	.042
n = 100	cens = 0	.043	.042	.150	.179
	cens = 0.2	.052	.053	.118	.142
	cens = 0.5	.045	.048	.094	.102
	cens = 0.8	.050	.047	.065	.065
n = 200	cens = 0	.055	.054	.290	.326
	cens = 0.2	.052	.054	.235	.262
	cens = 0.5	.046	.046	.134	.144
	cens = 0.8	.051	.048	.086	.089
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.053	.051	.296	.431
	cens = 0.2	.052	.051	.233	.320
	cens = 0.5	.052	.050	.152	.191
	cens = 0.8	.065	.058	.099	.085
n = 100	cens = 0	.042	.040	.627	.750
	cens = 0.2	.057	.054	.471	.566
	cens = 0.5	.044	.044	.264	.325
	cens = 0.8	.058	.059	.118	.128
n = 200	cens = 0	.063	.062	.914	.951
	cens = 0.2	.042	.042	.808	.868
	cens = 0.5	.053	.054	.511	.574
	cens = 0.8	.049	.047	.231	.257

Table 8.19: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Cai-Prentice Weibull, tau = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.807	.808	.677	.663
	cens = 0.2	.738	.740	.531	.486
	cens = 0.5	.529	.534	.315	.280
	cens = 0.8	.243	.228	.115	.097
n = 100	cens = 0	.971	.971	.925	.919
	cens = 0.2	.959	.959	.829	.815
	cens = 0.5	.815	.816	.536	.513
	cens = 0.8	.427	.432	.229	.183
n = 200	cens = 0	1.00	1.00	.996	.995
	cens = 0.2	.997	.998	.983	.980
	cens = 0.5	.974	.976	.831	.813
	cens = 0.8	.707	.712	.394	.364
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.835	.840	.508	.396
	cens = 0.2	.729	.735	.362	.286
	cens = 0.5	.527	.543	.192	.136
	cens = 0.8	.286	.282	.091	.061
n = 100	cens = 0	.986	.986	.756	.686
	cens = 0.2	.966	.968	.551	.458
	cens = 0.5	.831	.835	.322	.264
	cens = 0.8	.472	.484	.146	.092
n = 200	cens = 0	1.00	1.00	.952	.927
	cens = 0.2	.999	.999	.822	.777
	cens = 0.5	.988	.989	.530	.459
	cens = 0.8	.751	.759	.205	.175
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.866	.872	.229	.086
	cens = 0.2	.776	.785	.171	.068
	cens = 0.5	.571	.586	.095	.039
	cens = 0.8	.264	.286	.058	.033
n = 100	cens = 0	.992	.993	.315	.129
	cens = 0.2	.975	.976	.203	.104
	cens = 0.5	.868	.874	.111	.061
	cens = 0.8	.501	.512	.052	.028
n = 200	cens = 0	1.00	1.00	.423	.213
	cens = 0.2	1.00	1.00	.280	.119
	cens = 0.5	.991	.991	.130	.065
	cens = 0.8	.776	.786	.069	.048

Table 8.20: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Cai-Prentice Weibull, tau = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.048	.049	.112	.146
	cens = 0.2	.044	.044	.100	.135
	cens = 0.5	.054	.052	.081	.096
	cens = 0.8	.063	.048	.064	.062
n = 100	cens = 0	.044	.044	.221	.264
	cens = 0.2	.056	.056	.175	.213
	cens = 0.5	.048	.047	.128	.146
	cens = 0.8	.051	.050	.057	.061
n = 200	cens = 0	.048	.051	.443	.490
	cens = 0.2	.045	.045	.338	.384
	cens = 0.5	.052	.051	.215	.242
	cens = 0.8	.063	.059	.101	.102
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.728	.721	.867	.882
	cens = 0.2	.667	.656	.752	.759
	cens = 0.5	.452	.434	.487	.482
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.958	.954	.996	.997
	cens = 0.2	.931	.929	.975	.976
	cens = 0.5	.735	.730	.783	.787
	cens = 0.8	.392	.368	.372	.326
n = 200	cens = 0	.998	.998	1.00	1.00
	cens = 0.2	.998	.998	1.00	1.00
	cens = 0.5	.972	.970	.988	.988
	cens = 0.8	.679	.671	.670	.649

* Programs did not converge due to sparse data.

Table 8.21: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Cai-Prentice Weibull, tau = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.059	.061	.133	.155
	cens = 0.2	.062	.063	.103	.123
	cens = 0.5	.051	.053	.059	.068
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.070	.070	.246	.289
	cens = 0.2	.053	.053	.157	.175
	cens = 0.5	.054	.052	.107	.107
	cens = 0.8	.045	.044	.062	.058
n = 200	cens = 0	.068	.067	.458	.486
	cens = 0.2	.061	.059	.299	.328
	cens = 0.5	.055	.054	.177	.186
	cens = 0.8	.036	.036	.074	.073
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.078	.078	.486	.609
	cens = 0.2	.071	.071	.322	.418
	cens = 0.5	.049	.049	.154	.205
	cens = 0.8	.043	.040	.081	.075
n = 100	cens = 0	.124	.124	.814	.881
	cens = 0.2	.083	.083	.588	.670
	cens = 0.5	.070	.067	.352	.405
	cens = 0.8	.055	.049	.125	.131
n = 200	cens = 0	.171	.169	.991	.996
	cens = 0.2	.121	.119	.910	.938
	cens = 0.5	.058	.057	.592	.641
	cens = 0.8	.042	.041	.236	.260

* Programs did not converge due to sparse data.

Table 8.22: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Cai-Prentice Weibull, tau = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.833	.829	.784	.779
	cens = 0.2	.696	.696	.578	.572
	cens = 0.5	.504	.506	.323	.294
	cens = 0.8	.255	.249	.119	.097
n = 100	cens = 0	.984	.985	.971	.971
	cens = 0.2	.943	.944	.847	.846
	cens = 0.5	.799	.800	.583	.548
	cens = 0.8	.418	.421	.216	.178
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	.995	.994
	cens = 0.5	.980	.980	.839	.833
	cens = 0.8	.719	.722	.391	.359
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.837	.841	.529	.488
	cens = 0.2	.725	.729	.382	.336
	cens = 0.5	.479	.491	.186	.141
	cens = 0.8	.243	.242	.089	.060
n = 100	cens = 0	.977	.978	.792	.771
	cens = 0.2	.951	.951	.637	.600
	cens = 0.5	.792	.800	.344	.297
	cens = 0.8	.441	.450	.136	.098
n = 200	cens = 0	1.00	1.00	.982	.979
	cens = 0.2	.999	.999	.895	.874
	cens = 0.5	.982	.984	.541	.489
	cens = 0.8	.706	.716	.190	.160
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.760	.774	.155	.075
	cens = 0.2	.693	.703	.128	.061
	cens = 0.5	.505	.520	.071	.046
	cens = 0.8	.265	.284	.053	.029
n = 100	cens = 0	.974	.975	.212	.104
	cens = 0.2	.935	.939	.157	.089
	cens = 0.5	.780	.795	.110	.072
	cens = 0.8	.461	.481	.066	.048
n = 200	cens = 0	1.00	1.00	.304	.175
	cens = 0.2	1.00	1.00	.196	.099
	cens = 0.5	.970	.972	.129	.062
	cens = 0.8	.730	.741	.068	.045

Table 8.23: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Cai-Prentice Weibull, tau = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.065	.066	.177	.225
	cens = 0.2	.054	.053	.145	.167
	cens = 0.5	.057	.054	.071	.071
	cens = 0.8	.051	.039	.049	.044
n = 100	cens = 0	.077	.075	.358	.412
	cens = 0.2	.062	.062	.266	.298
	cens = 0.5	.045	.045	.117	.137
	cens = 0.8	.054	.050	.081	.066
n = 200	cens = 0	.112	.112	.669	.712
	cens = 0.2	.067	.065	.476	.499
	cens = 0.5	.047	.047	.228	.255
	cens = 0.8	*	*	*	*
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.833	.822	.941	.949
	cens = 0.2	.688	.682	.803	.816
	cens = 0.5	.484	.470	.517	.507
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.980	.977	.998	.999
	cens = 0.2	.943	.939	.986	.987
	cens = 0.5	.756	.750	.819	.824
	cens = 0.8	.380	.356	.365	.321
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	.999	.999	1.00	1.00
	cens = 0.5	.977	.977	.986	.985
	cens = 0.8	.653	.644	.661	.646

* Programs did not converge due to sparse data.

Table 8.24: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Cai-Prentice Weibull, tau = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.114	.116	.325	.384
	cens = 0.2	.068	.069	.164	.200
	cens = 0.5	.062	.057	.090	.100
	cens = 0.8	.070	.063	.070	.052
n = 100	cens = 0	.211	.210	.630	.685
	cens = 0.2	.109	.109	.317	.365
	cens = 0.5	.073	.071	.131	.139
	cens = 0.8	.044	.040	.056	.050
n = 200	cens = 0	.353	.353	.897	.924
	cens = 0.2	.172	.169	.591	.626
	cens = 0.5	.084	.084	.236	.248
	cens = 0.8	.049	.048	.084	.084
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.285	.278	.857	.917
	cens = 0.2	.159	.151	.541	.648
	cens = 0.5	.069	.062	.219	.271
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.500	.491	.993	.997
	cens = 0.2	.286	.279	.859	.900
	cens = 0.5	.118	.107	.473	.529
	cens = 0.8	.066	.059	.147	.160
n = 200	cens = 0	.777	.775	1.00	1.00
	cens = 0.2	.475	.470	.997	.998
	cens = 0.5	.152	.150	.770	.811
	cens = 0.8	.053	.048	.271	.294

* Programs did not converge due to sparse data.

Table 8.25: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Cai-Prentice Weibull, tau = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.942	.942	.964	.967
	cens = 0.2	.827	.821	.768	.766
	cens = 0.5	.517	.517	.384	.350
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.998	.998	.999	.999
	cens = 0.2	.984	.983	.966	.968
	cens = 0.5	.819	.816	.672	.662
	cens = 0.8	.408	.408	.224	.176
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.988	.989	.929	.928
	cens = 0.8	.689	.692	.412	.384
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.903	.904	.783	.773
	cens = 0.2	.734	.737	.488	.471
	cens = 0.5	.509	.514	.239	.200
	cens = 0.8	.219	.210	.083	.070
n = 100	cens = 0	.998	.998	.976	.976
	cens = 0.2	.967	.967	.781	.774
	cens = 0.5	.755	.760	.379	.365
	cens = 0.8	.406	.412	.116	.082
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	.979	.977
	cens = 0.5	.973	.973	.625	.603
	cens = 0.8	.648	.662	.213	.188
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.754	.764	.197	.071
	cens = 0.2	.588	.605	.103	.049
	cens = 0.5	.420	.439	.077	.048
	cens = 0.8	.236	.253	.053	.028
n = 100	cens = 0	.964	.966	.224	.104
	cens = 0.2	.903	.904	.133	.070
	cens = 0.5	.689	.704	.088	.065
	cens = 0.8	.383	.398	.060	.041
n = 200	cens = 0	.999	.999	.323	.170
	cens = 0.2	.995	.995	.201	.100
	cens = 0.5	.947	.948	.077	.049
	cens = 0.8	.653	.665	.071	.053

* Programs did not converge due to sparse data.

Table 8.26: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Cai-Prentice Weibull, tau = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.184	.178	.434	.524
	cens = 0.2	.113	.109	.226	.274
	cens = 0.5	.065	.060	.114	.118
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.335	.328	.790	.845
	cens = 0.2	.157	.156	.442	.488
	cens = 0.5	.079	.076	.197	.204
	cens = 0.8	.050	.045	.071	.060
n = 200	cens = 0	.595	.587	.983	.994
	cens = 0.2	.285	.282	.780	.805
	cens = 0.5	.104	.100	.339	.354
	cens = 0.8	.054	.054	.131	.128
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.932	.923	.992	.995
	cens = 0.2	.818	.795	.912	.921
	cens = 0.5	.548	.512	.583	.559
	cens = 0.8	*	*	*	*
n = 100	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	.982	.981	.999	.999
	cens = 0.5	.825	.818	.882	.884
	cens = 0.8	.417	.391	.402	.354
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.992	.992	.996	.996
	cens = 0.8	.690	.678	.691	.672

* Programs did not converge due to sparse data.

Table 8.27: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Cai-Prentice Weibull, tau = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.460	.456	.868	.921
	cens = 0.2	.245	.239	.483	.533
	cens = 0.5	.104	.099	.162	.177
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.734	.729	.993	.995
	cens = 0.2	.439	.434	.803	.836
	cens = 0.5	.147	.144	.324	.344
	cens = 0.8	.046	.041	.076	.074
n = 200	cens = 0	.972	.970	1.00	1.00
	cens = 0.2	.753	.749	.988	.990
	cens = 0.5	.289	.285	.570	.602
	cens = 0.8	.073	.070	.142	.144
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.594	.587	.981	.997
	cens = 0.2	.426	.413	.842	.899
	cens = 0.5	.179	.171	.413	.485
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.898	.892	1.00	1.00
	cens = 0.2	.704	.693	.991	.992
	cens = 0.5	.336	.321	.784	.817
	cens = 0.8	.074	.067	.194	.213
n = 200	cens = 0	.993	.993	1.00	1.00
	cens = 0.2	.951	.948	1.00	1.00
	cens = 0.5	.573	.560	.975	.981
	cens = 0.8	.116	.110	.422	.449

* Programs did not converge due to sparse data.

Table 8.28: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Cai-Prentice Weibull, tau = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.996	.995	.999	.999
	cens = 0.2	.967	.959	.970	.967
	cens = 0.5	.646	.636	.559	.497
	cens = 0.8	*	*	*	*
n = 100	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.935	.932	.889	.876
	cens = 0.8	.431	.420	.253	.210
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.998	.998	.996	.996
	cens = 0.8	.715	.716	.518	.484
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.997	.996	.992	.991
	cens = 0.2	.942	.941	.862	.852
	cens = 0.5	.595	.594	.366	.323
	cens = 0.8	*	*	*	*
n = 100	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	.997	.995
	cens = 0.5	.872	.872	.662	.633
	cens = 0.8	.408	.411	.180	.137
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.993	.993	.913	.913
	cens = 0.8	.636	.646	.287	.267
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.804	.808	.209	.080
	cens = 0.2	.607	.621	.119	.051
	cens = 0.5	.363	.381	.072	.047
	cens = 0.8	.191	.204	.044	.051
n = 100	cens = 0	.983	.984	.242	.100
	cens = 0.2	.920	.927	.152	.065
	cens = 0.5	.614	.625	.081	.058
	cens = 0.8	.312	.328	.049	.038
n = 200	cens = 0	1.00	1.00	.372	.173
	cens = 0.2	1.00	1.00	.181	.076
	cens = 0.5	.892	.902	.091	.048
	cens = 0.8	.497	.517	.054	.054

* Programs did not converge due to sparse data.

Table 8.29: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Cai-Prentice Weibull, tau = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.708	.683	.917	.956
	cens = 0.2	.390	.359	.642	.705
	cens = 0.5	.150	.130	.228	.229
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.940	.936	1.00	1.00
	cens = 0.2	.711	.701	.944	.952
	cens = 0.5	.240	.225	.443	.451
	cens = 0.8	.072	.063	.106	.087
n = 200	cens = 0	.998	.998	1.00	1.00
	cens = 0.2	.955	.951	1.00	1.00
	cens = 0.5	.438	.422	.754	.771
	cens = 0.8	.080	.077	.154	.148
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.994	.991	1.00	1.00
	cens = 0.2	.945	.937	.983	.983
	cens = 0.5	.721	.669	.746	.708
	cens = 0.8	*	*	*	*
n = 100	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	.998	.998	1.00	1.00
	cens = 0.5	.954	.949	.970	.971
	cens = 0.8	.517	.458	.461	.412
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	.777	.760	.784	.751

* Programs did not converge due to sparse data.

Table 8.30: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Cai-Prentice Weibull, tau = 0.8).

8.5 BIVARIATE WEIBULL WITH A GAMMA FRAILTY

8.5.1 DATA GENERATION

To generate data from this frailty model, consider $U_{ki} \overset{iid}{\sim} Uniform(0,1)$ and let

$$Y_{ki} \mid Z_i = \left[\frac{-1}{\lambda_k} \exp(-\beta_k Z_i) \ln(1 - U_{ki}) \right]^{-1/\alpha} \text{ for } k = 1,2 \text{ and } i = 1,...,n.$$

Therefore we have

$$Y_{ki} \mid Z_i \overset{ind}{\sim} Weibull(\alpha, \lambda_k \exp(\beta_k Z_i)) \text{ for } k = 1,2 \text{ and } i = 1,...,n.$$

We will then generate the frailty associated with each individual in the sample independent of $Y_{ki} \mid Z_i$. Again restrict θ to be an integer and consider $U_{ji} \overset{iid}{\sim} Uniform(0,1)$ for $j = 1,...,\theta$ and $i = 1,...,n$. We will then let

$$V_{ji} = \frac{-1}{\theta} \ln(1 - U_{ji}) \text{ for } j = 1,...,\theta \text{ and } i = 1,...,n.$$

Therefore $V_{ji} \overset{ind}{\sim} Exponential(\theta)$ for $j = 1,...,\theta$ and $i = 1,...,n$. Finally we consider $w_i = \sum_{j=1}^{\theta} V_{ji}$ for $i = 1,...,n$ to obtain the frailty associated with the i^{th} individual in the sample. Note that the exponential and gamma distributions are related in such a way that $w_i \overset{ind}{\sim} gamma(\theta, \theta)$ for $i = 1,...,n$. Note that $E(w_i) = \frac{\theta}{\theta} = 1$. The survival times for each individual in the sample are then obtained by letting

$$X_{ki} \mid Z_i = w_i \times Y_{ki} \mid Z_i \text{ for } k = 1,2 \text{ and } i = 1,...,n.$$

Therefore, the i^{th} individual has a conditional expected survival time and variance for cause *k* which are given by

$$
\begin{aligned}
E(X_{ki} \mid Z_i) &= E(w_i \times Y_{ki} \mid Z_i) \\
&= E(w_i) E(Y_{ki} \mid Z_i) \qquad \textit{by independence} \\
&= E(Y_{ki} \mid Z_i) \\
&= \left[\Gamma\left(1 + \frac{1}{\alpha}\right) \right] \left[\lambda_k \exp(\beta_k Z_i)\right]^{-1/\alpha}
\end{aligned}
$$

and

$$
\begin{aligned}
Var(X_{ki} \mid Z_i) &= Var(w_i \times Y_{ki} \mid Z_i) \\
&= E(w_i^2) E(Y_{ki}^2 \mid Z_i) - \left[E(w_i)\right]^2 \left[E(Y_{ki} \mid Z_i)\right]^2 \qquad \textit{by independence} \\
&= \left(\frac{1}{\theta} + 1\right) \left(\left[\Gamma\left(1 + \frac{2}{\alpha}\right) - \left\{ \Gamma\left(1 + \frac{1}{\alpha}\right) \right\}^2 \right] \left[\lambda_k \exp(\beta_k Z_i)\right]^{-2/\alpha} \right. \\
&\quad \left. + \left\{ \left[\Gamma\left(1 + \frac{1}{\alpha}\right) \right] \left[\lambda_k \exp(\beta_k Z_i)\right]^{-1/\alpha} \right\}^2 \right) \\
&\quad - \left\{ \left[\Gamma\left(1 + \frac{1}{\alpha}\right) \right] \left[\lambda_k \exp(\beta_k Z_i)\right]^{-1/\alpha} \right\}^2,
\end{aligned}
$$

respectively.

We again wish to consider the case where independent censoring may occur. We will generate random censoring times C_i independent of $X_{1i} \mid Z_i, w_i$ and $X_{2i} \mid Z_i, w_i$ for $i = 1,...,n$. Thus C_i will be independent of $X_{(1)i} \mid Z_i, w_i$. The censoring times for the the i^{th} individual will follow a Weibull distribution with parameters α and λ_{c_i}. In order to achieve an overall censoring probability equal to p, we will select $\lambda_{c_i} = \dfrac{p(\lambda_{m_i})}{(1-p)}$ where $\lambda_{m_i} = \dfrac{1}{w_i} \sum_{k=1}^{2} \lambda_k \exp(\beta_k Z_i)$ for $i = 1,...,n$. Thus,

to generate the censoring times, we will consider $U_{c_i} \sim Uniform(0,1)$ and let

$$C_i = \left[\frac{-1}{\lambda_{c_i}} \ln\left(1 - U_{c_i}\right) \right]^{-1/\alpha} \text{ for } i = 1,...,n.$$

8.5.2 SIMULATION SUMMARY

By introducing a frailty parameter for each subject, we are again inducing a positive correlation between the competing risks. Thus we are again violating the independence assumption inherent in the test based on Cox's proportional hazards model and the log-rank test.

In Table 8.31, it appears that T^{CPH} and T^{LR} are generally better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. All tests appear to be better at preserving the nominal alpha level when the sample size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small.

In Table 8.32, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG} except when there is no censoring and no treatment effect on type 2 failure. All tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases. We also see that the power for each test diminishes as the treatment's beneficial effect on type 2 failure gets larger.

In panel 1 of Table 8.33, it appears that T^{CPH} and T^{LR} are again consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. Although not always the case, all tests typically appear to be better at preserving the nominal alpha level when the sample size is small and the amount of censoring is large.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 8.33, it appears that T^{G} and T^{FG} are now much more powerful than T^{CPH} and T^{LR}. Again, all tests appear to gain power for larger sample sizes and lose power as the amount of censoring increases.

8.5.3 SIMULATION RESULTS

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.071	.072	.117	.127
	cens = 0.2	.076	.076	.116	.124
	cens = 0.5	.044	.042	.064	.071
	cens = 0.8	.055	.051	.059	.051
n = 100	cens = 0	.078	.078	.227	.241
	cens = 0.2	.060	.060	.152	.170
	cens = 0.5	.065	.066	.107	.107
	cens = 0.8	.069	.066	.089	.089
n = 200	cens = 0	.101	.102	.367	.375
	cens = 0.2	.070	.069	.251	.267
	cens = 0.5	.063	.061	.160	.162
	cens = 0.8	.051	.050	.101	.101
No effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.099	.097	.426	.501
	cens = 0.2	.114	.108	.329	.393
	cens = 0.5	.084	.080	.198	.217
	cens = 0.8	.062	.060	.101	.097
n = 100	cens = 0	.144	.143	.735	.785
	cens = 0.2	.141	.141	.573	.609
	cens = 0.5	.102	.101	.370	.387
	cens = 0.8	.088	.084	.184	.189
n = 200	cens = 0	.260	.258	.972	.978
	cens = 0.2	.218	.217	.887	.903
	cens = 0.5	.175	.174	.682	.701
	cens = 0.8	.111	.107	.323	.324

Table 8.31: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Weibull with Gamma frailty).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
60% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.621	.616	.639	.641
	cens = 0.2	.570	.564	.541	.534
	cens = 0.5	.412	.407	.318	.307
	cens = 0.8	.234	.222	.159	.126
n = 100	cens = 0	.913	.913	.910	.910
	cens = 0.2	.860	.858	.816	.806
	cens = 0.5	.694	.695	.559	.542
	cens = 0.8	.400	.397	.270	.244
n = 200	cens = 0	.994	.994	.998	.996
	cens = 0.2	.986	.986	.978	.980
	cens = 0.5	.929	.928	.852	.849
	cens = 0.8	.678	.681	.520	.501
60% effect on type 1 failure, 20% effect on type 2 failure					
n = 50	cens = 0	.531	.532	.371	.344
	cens = 0.2	.457	.461	.254	.237
	cens = 0.5	.336	.337	.182	.156
	cens = 0.8	.189	.183	.095	.061
n = 100	cens = 0	.834	.837	.628	.595
	cens = 0.2	.763	.764	.491	.468
	cens = 0.5	.590	.596	.288	.266
	cens = 0.8	.311	.319	.133	.121
n = 200	cens = 0	.987	.987	.875	.859
	cens = 0.2	.963	.964	.761	.738
	cens = 0.5	.867	.868	.508	.487
	cens = 0.8	.570	.576	.246	.233
60% effect on type 1 failure, 60% effect on type 2 failure					
n = 50	cens = 0	.413	.419	.087	.063
	cens = 0.2	.329	.342	.076	.060
	cens = 0.5	.258	.268	.064	.054
	cens = 0.8	.156	.163	.053	.033
n = 100	cens = 0	.697	.706	.104	.074
	cens = 0.2	.583	.591	.079	.062
	cens = 0.5	.408	.415	.058	.054
	cens = 0.8	.228	.236	.058	.051
n = 200	cens = 0	.930	.930	.135	.101
	cens = 0.2	.871	.873	.093	.076
	cens = 0.5	.704	.712	.065	.055
	cens = 0.8	.404	.417	.042	.045

Table 8.32: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Weibull with Gamma frailty).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.072	.073	.152	.187
	cens = 0.2	.063	.064	.128	.134
	cens = 0.5	.057	.057	.077	.078
	cens = 0.8	.058	.049	.062	.047
n = 100	cens = 0	.079	.077	.278	.306
	cens = 0.2	.072	.071	.202	.223
	cens = 0.5	.060	.061	.139	.146
	cens = 0.8	.064	.059	.089	.082
n = 200	cens = 0	.121	.121	.503	.518
	cens = 0.2	.117	.114	.372	.387
	cens = 0.5	.076	.075	.232	.241
	cens = 0.8	.075	.072	.138	.133
60% effect on type 1 failure, -20% effect on type 2 failure					
n = 50	cens = 0	.729	.724	.871	.882
	cens = 0.2	.658	.651	.776	.789
	cens = 0.5	.484	.473	.548	.544
	cens = 0.8	.275	.255	.247	.198
n = 100	cens = 0	.945	.944	.991	.991
	cens = 0.2	.921	.917	.971	.974
	cens = 0.5	.780	.774	.840	.839
	cens = 0.8	.478	.463	.499	.471
n = 200	cens = 0	1.00	.999	1.00	1.00
	cens = 0.2	.999	.999	1.00	1.00
	cens = 0.5	.979	.978	.990	.991
	cens = 0.8	.797	.796	.829	.822

Table 8.33: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Weibull with Gamma frailty).

CHAPTER 9

BIVARIATE NORMAL DISTRIBUTION

9.1 DATA GENERATION

The bivariate normal failure times will be generated using the conditional distribution method described in Devroye (1986). The conditional distribution method allows us to reduce the K-dimensional multivariate generation problem into K univariate generation problems. Note that this method can only be used when all the conditional densities are known (or equivalently when all marginal distributions are known).

We say that (X_1, X_2) follow a bivariate normal distribution with means μ_1 and μ_2, variances σ_1^2 and σ_2^2, and correlation ρ if their joint pdf is given by

$$f_{X_1,X_2}(x_1,x_2) = \left(2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}\right)^{-1} \exp\left\{\frac{-1}{2(1-\rho^2)}\left[\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2 - 2\rho\left(\frac{x_1-\mu_1}{\sigma_1}\right)\left(\frac{x_2-\mu_2}{\sigma_2}\right) + \left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right]\right\}$$

for $-\infty < x_1 < \infty$ and $-\infty < x_2 < \infty$. This will be denoted by $(X_1, X_2)' \sim BVN(\mu, \Sigma)$ where $\mu = (\mu_1, \mu_2)'$ and $\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}$ with $\sigma_{12} = Cov(X_1, X_2) = \rho\sqrt{\sigma_1^2 + \sigma_2^2}$. In

the case of the bivariate normal distribution, we have that $(X_1, X_2)' \sim BVN(\mu, \Sigma)$ if and only if

$$X_1 \sim N(\mu_1, \sigma_1^2)$$

and

$$X_2 \mid X_1 = x_1 \sim N\left(\mu_2 + \rho\sqrt{\frac{\sigma_2^2}{\sigma_1^2}}(x_1 - \mu_1), \sigma_2^2(1-\rho^2)\right).$$

Therefore we can use the conditional method to generate bivariate normal data by univariately generating data from the distributions of X_1 and $X_2 \mid X_1 = x_1$. An alternative method for generating bivariate normal data is available in the statistical software package R 2.1.0 using the 'mvrnorm' command found in the MASS package.

We can now generalize the above result to generate bivariate normal competing risks data in the presence of treatment effects. Therefore, in order to generate $(X_1, X_2 \mid Z)' \sim BVN(\mu^*, \Sigma^*)$ where

$$\begin{aligned}\mu^* &= (\mu_1^*, \mu_2^*)' \\ &= (\mu_1 \exp(\beta_1 Z), \mu_2 \exp(\beta_2 Z))'\end{aligned}$$

and

$$\Sigma^* = \begin{bmatrix} \sigma_1^2 \exp(\lambda_1 Z) & \rho\sqrt{\sigma_1^2 \exp(\lambda_1 Z)\sigma_2^2 \exp(\lambda_2 Z)} \\ \rho\sqrt{\sigma_1^2 \exp(\lambda_1 Z)\sigma_2^2 \exp(\lambda_2 Z)} & \sigma_2^2 \exp(\lambda_2 Z) \end{bmatrix},$$

we will generate

$$X_1 \mid Z \sim N(\mu_1 \exp(\beta_1 Z), \sigma_1^2 \exp(\lambda_1 Z))$$

and $X_2 \mid X_1 = x_1, Z$ will be distributed as

$$N\left(\mu_2 \exp(\beta_2 Z) + \rho\sqrt{\frac{\sigma_2^2 \exp(\lambda_2 Z)}{\sigma_1^2 \exp(\lambda_1 Z)}}(x_1 - \mu_1 \exp(\beta_1 Z)), \sigma_2^2 \exp(\lambda_2 Z)(1-\rho^2)\right).$$

We are again incorporating additional parameters β_1 and β_2 to allow for different treatment effects on the mean survival time within each cause of failure, and also allowing for different variances within the treatments groups for each cause of failure by incorporating parameters λ_1 and λ_2 into the variance terms. Since we are dealing with survival times and the normal distribution allows for negative values, those observations that result in negative survival times will be treated as zeros.

Within each treatment group, we will generate fixed censoring times (c_{1Z}, c_{2Z}) where c_{kZ} is the treatment specific censoring time associated with failure time X_k for $k = 1,2$. These censoring times will be chosen in such a way that they result in censoring probabilities

$$p_1 = P(c_{1Z} < X_1 \mid Z)$$

and

$$p_2 = P(c_{2Z} < X_2 \mid Z).$$

Therefore

$$c_{1Z} = \mu_1^* - \Phi^{-1}(p_1)\sqrt{\sigma_1^2 \exp(\lambda_1 Z)}$$

and

$$c_{2Z} = \mu_2^* - \Phi^{-1}(p_2)\sqrt{\sigma_2^2 \exp(\lambda_2 Z)}$$

where $\Phi^{-1}(x)$ denotes the inverse standard normal cumulative distribution function evaluated at x.

9.2 DETERMINING SIMULATION PARAMETERS

We will now describe the method used for determining the parameters required in order to simulate data from a desired bivariate normal distribution. Recall that the means, conditional on Z, are given by

$$E[X_k \mid Z] = \mu_k \exp(\beta_k Z) \text{ for } k = 1,2 .$$

Therefore the control means are simply given by

$$E[X_k \mid Z = 0] = \mu_k \text{ for } k = 1,2 .$$

Now suppose we wish to generate data for the treatment groups such that they result in treatment means having a specified percentage increase (or decrease) over the control means. The desired percentage increase over the control means will be denoted by r_k for $k = 1,2$. Therefore we have that

$$E[X_k \mid Z = 1] = \mu_k (1 + r_k)$$

and

$$E[X_k \mid Z = 1] = \mu_k \exp(\beta_k) \text{ for } k = 1,2 .$$

Thus we are able to generate data with the desired means by setting

$$\beta_k = \log(1 + r_k) \text{ for } k = 1,2 .$$

Furthermore, we will assume that the variances do not depend on the treatment groups. Therefore, $\lambda_1 = \lambda_2 = 0$ and the variances will be given by

$$Var(X_k) = \sigma_k^2 \text{ for } k = 1,2 .$$

9.3 SIMULATION SUMMARY

In this situation, the proportional hazards assumption is not met and the event times due to the two competing causes of failure are independent only when the correlation is set to zero. Therefore we might suspect that the test based on Cox's proportional hazards model (T^{CPH}) and the log-rank test (T^{LR}) may not perform as well as they did in the previous situations when compared to Gray's test (T^{G}) and the test due to Fine and Gray (T^{FG}).

In Table 9.1, it appears that T^{CPH} and T^{LR} are consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring, regardless of whether the beneficial treatment effect on type 2 failure is small or large. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small. When we allowed the competing risks to have a small positive correlation in Table 9.4 or when we allowed for a moderate correlation and a small treatment effect on type 2 failure (panel 1 of Table 9.7), we again saw similar results. However as the correlation became larger (Tables 9.7 and 9.10), we saw that all tests were better at preserving the nominal alpha level when the sample size was small, when the amount of censoring was large, and when the beneficial treatment effect on type 2 failure was small. Furthermore, each test's ability to preserve the nominal alpha level appeared to worsen as the positive correlation between the two competing risks increased.

In Table 9.2, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes. When the effect on type 2 failure is beneficial (panels 2 and 3 of Table 9.2), we see that the power results for T^{CPH} and T^{LR} remain similar to those found when there was no treatment effect on type 2 failure, however T^{G} and T^{FG} lose considerable power as the treatment's beneficial effect on type 2 failure gets larger. When we allowed for the competing risks to be positively correlated (Tables 9.5, 9.8, and 9.11), we again saw that all tests appeared to gain power for larger sample sizes and lose power as the beneficial treatment effect on type 2 failure increased. Although T^{CPH} and T^{LR} are typically the more powerful tests, we see that T^{G} and T^{FG} are more powerful for highly correlated competing risks when there is no treatment effect on type 2 failure. When there is little or no beneficial treatment effect on type 2 failure, all tests appear to gain power as the correlation between the competing risks increased. However when the beneficial treatment effect on type 2 failure is large, all tests appear to lose power as the correlation between the competing risks increased.

In panel 1 of Table 9.3, it appears that T^{CPH} and T^{LR} are typically better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring. T^{G} is better at preserving the nominal alpha level when the sample size is small and performs quite well when the amount of censoring is large. T^{FG} is better at preserving the nominal alpha level for smaller sample sizes and the results are similar regardless of the amount of censoring. When we allowed for the competing risks to be positively correlated (panel 1 of Tables 9.6, 9.9, and 9.12), we saw that all tests

were better at preserving the nominal alpha level for small sample sizes and when the amount of censoring was large. Furthermore, each test's ability to preserve the nominal alpha level appeared to worsen as the positive correlation between the two competing risks increased.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 9.3, we see that T^{G} and T^{FG} are now the more powerful tests when compared to T^{CPH} and T^{LR}. Again, all tests appear to gain power for larger sample sizes. When we allowed for the competing risks to be positively correlated (panel 2 of Tables 9.6, 9.9, and 9.12), we again saw that all tests appeared to gain power for larger sample sizes. Also, the power for each test rose as the correlation between the competing risks increased.

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.044	.044	.054	.060
	cens = 0.2	.053	.054	.058	.059
	cens = 0.5	.044	.044	.047	.049
	cens = 0.8	.045	.042	.043	.033
n = 100	cens = 0	.054	.054	.068	.078
	cens = 0.2	.045	.045	.074	.075
	cens = 0.5	.050	.050	.046	.045
	cens = 0.8	.042	.042	.044	.038
n = 200	cens = 0	.050	.051	.115	.129
	cens = 0.2	.049	.050	.119	.120
	cens = 0.5	.054	.054	.065	.064
	cens = 0.8	.046	.046	.053	.052
No effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.061	.060	.162	.204
	cens = 0.2	.048	.053	.150	.157
	cens = 0.5	.052	.054	.076	.075
	cens = 0.8	.066	.063	.067	.056
n = 100	cens = 0	.048	.046	.309	.364
	cens = 0.2	.053	.055	.261	.263
	cens = 0.5	.062	.061	.135	.132
	cens = 0.8	.049	.048	.059	.052
n = 200	cens = 0	.047	.047	.583	.620
	cens = 0.2	.059	.059	.483	.487
	cens = 0.5	.063	.063	.203	.201
	cens = 0.8	.049	.048	.059	.059

Table 9.1: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Normal, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
15% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.468	.476	.349	.332
	cens = 0.2	.416	.417	.318	.325
	cens = 0.5	.406	.405	.373	.368
	cens = 0.8	.302	.290	.289	.236
n = 100	cens = 0	.737	.738	.603	.582
	cens = 0.2	.729	.730	.607	.613
	cens = 0.5	.701	.699	.672	.674
	cens = 0.8	.510	.503	.494	.480
n = 200	cens = 0	.944	.946	.852	.848
	cens = 0.2	.951	.951	.881	.883
	cens = 0.5	.954	.953	.930	.931
	cens = 0.8	.850	.846	.853	.845
15% effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.465	.470	.242	.233
	cens = 0.2	.434	.439	.220	.201
	cens = 0.5	.426	.433	.322	.228
	cens = 0.8	.318	.310	.283	.157
n = 100	cens = 0	.742	.744	.405	.371
	cens = 0.2	.744	.748	.468	.418
	cens = 0.5	.716	.719	.573	.394
	cens = 0.8	.553	.547	.523	.331
n = 200	cens = 0	.957	.957	.641	.624
	cens = 0.2	.961	.961	.725	.657
	cens = 0.5	.942	.942	.859	.709
	cens = 0.8	.857	.857	.828	.549
15% effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.469	.482	.082	.059
	cens = 0.2	.465	.476	.114	.074
	cens = 0.5	.453	.455	.257	.057
	cens = 0.8	.318	.307	.261	.046
n = 100	cens = 0	.737	.742	.106	.081
	cens = 0.2	.769	.771	.169	.082
	cens = 0.5	.738	.742	.441	.081
	cens = 0.8	.529	.524	.448	.045
n = 200	cens = 0	.950	.952	.139	.112
	cens = 0.2	.961	.962	.261	.103
	cens = 0.5	.955	.955	.716	.109
	cens = 0.8	.841	.840	.769	.066

Table 9.2: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Normal, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.057	.056	.064	.066
	cens = 0.2	.047	.047	.054	.061
	cens = 0.5	.051	.051	.055	.074
	cens = 0.8	.050	.046	.038	.049
n = 100	cens = 0	.043	.044	.070	.083
	cens = 0.2	.059	.059	.086	.098
	cens = 0.5	.064	.064	.068	.097
	cens = 0.8	.049	.045	.045	.099
n = 200	cens = 0	.056	.055	.108	.121
	cens = 0.2	.052	.053	.105	.135
	cens = 0.5	.057	.058	.061	.154
	cens = 0.8	.057	.057	.058	.154
15% effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.450	.449	.457	.451
	cens = 0.2	.466	.465	.494	.544
	cens = 0.5	.397	.392	.426	.531
	cens = 0.8	.294	.274	.297	.400
n = 100	cens = 0	.741	.738	.782	.777
	cens = 0.2	.719	.718	.779	.812
	cens = 0.5	.676	.672	.725	.856
	cens = 0.8	.506	.491	.509	.729
n = 200	cens = 0	.933	.933	.964	.963
	cens = 0.2	.948	.948	.979	.983
	cens = 0.5	.931	.931	.958	.985
	cens = 0.8	.792	.785	.817	.957

Table 9.3: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Normal, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.056	.057	.062	.071
	cens = 0.2	.051	.053	.058	.061
	cens = 0.5	.044	.044	.049	.049
	cens = 0.8	.043	.036	.042	.034
n = 100	cens = 0	.046	.047	.092	.095
	cens = 0.2	.057	.058	.074	.076
	cens = 0.5	.041	.041	.067	.067
	cens = 0.8	.053	.050	.059	.057
n = 200	cens = 0	.065	.066	.147	.150
	cens = 0.2	.061	.061	.117	.118
	cens = 0.5	.040	.040	.056	.057
	cens = 0.8	.051	.050	.051	.051
No effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.049	.049	.195	.238
	cens = 0.2	.051	.050	.185	.186
	cens = 0.5	.051	.052	.092	.093
	cens = 0.8	.044	.042	.038	.033
n = 100	cens = 0	.049	.049	.389	.434
	cens = 0.2	.054	.054	.347	.350
	cens = 0.5	.046	.044	.132	.130
	cens = 0.8	.055	.052	.065	.065
n = 200	cens = 0	.049	.050	.694	.727
	cens = 0.2	.061	.061	.558	.559
	cens = 0.5	.065	.065	.228	.228
	cens = 0.8	.048	.048	.070	.068

Table 9.4: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Normal, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
15% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.461	.461	.379	.367
	cens = 0.2	.455	.453	.408	.417
	cens = 0.5	.427	.427	.423	.416
	cens = 0.8	.312	.296	.295	.229
n = 100	cens = 0	.774	.774	.659	.647
	cens = 0.2	.759	.759	.696	.696
	cens = 0.5	.698	.698	.671	.671
	cens = 0.8	.546	.534	.539	.519
n = 200	cens = 0	.964	.964	.925	.919
	cens = 0.2	.961	.960	.930	.930
	cens = 0.5	.947	.947	.943	.943
	cens = 0.8	.830	.825	.829	.818
15% effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.449	.453	.251	.230
	cens = 0.2	.410	.415	.256	.237
	cens = 0.5	.439	.438	.361	.257
	cens = 0.8	.298	.287	.267	.143
n = 100	cens = 0	.737	.736	.441	.418
	cens = 0.2	.729	.732	.512	.444
	cens = 0.5	.698	.699	.582	.422
	cens = 0.8	.527	.524	.491	.265
n = 200	cens = 0	.975	.975	.703	.683
	cens = 0.2	.955	.956	.798	.730
	cens = 0.5	.937	.938	.869	.698
	cens = 0.8	.828	.826	.788	.509
15% effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.438	.445	.085	.069
	cens = 0.2	.407	.418	.101	.063
	cens = 0.5	.421	.424	.222	.060
	cens = 0.8	.289	.284	.246	.035
n = 100	cens = 0	.696	.701	.090	.070
	cens = 0.2	.718	.726	.190	.090
	cens = 0.5	.690	.694	.403	.059
	cens = 0.8	.515	.506	.431	.051
n = 200	cens = 0	.958	.959	.134	.114
	cens = 0.2	.949	.953	.308	.109
	cens = 0.5	.932	.933	.676	.095
	cens = 0.8	.813	.813	.738	.054

Table 9.5: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Normal, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.049	.051	.068	.088
	cens = 0.2	.045	.045	.056	.071
	cens = 0.5	.065	.066	.062	.083
	cens = 0.8	.051	.045	.048	.060
n = 100	cens = 0	.071	.072	.100	.113
	cens = 0.2	.062	.062	.082	.104
	cens = 0.5	.061	.061	.075	.134
	cens = 0.8	.051	.048	.050	.107
n = 200	cens = 0	.047	.047	.141	.145
	cens = 0.2	.059	.059	.143	.180
	cens = 0.5	.054	.054	.087	.194
	cens = 0.8	.032	.032	.036	.148
15% effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.487	.481	.541	.551
	cens = 0.2	.459	.455	.531	.585
	cens = 0.5	.441	.435	.482	.611
	cens = 0.8	.266	.237	.259	.327
n = 100	cens = 0	.758	.756	.805	.811
	cens = 0.2	.746	.745	.826	.861
	cens = 0.5	.693	.684	.741	.865
	cens = 0.8	.490	.479	.506	.701
n = 200	cens = 0	.967	.967	.985	.986
	cens = 0.2	.969	.969	.989	.991
	cens = 0.5	.942	.941	.969	.993
	cens = 0.8	.789	.785	.812	.962

Table 9.6: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Normal, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.048	.049	.096	.098
	cens = 0.2	.048	.047	.064	.064
	cens = 0.5	.049	.049	.050	.048
	cens = 0.8	.069	.066	.063	.055
n = 100	cens = 0	.063	.063	.108	.121
	cens = 0.2	.062	.062	.102	.102
	cens = 0.5	.054	.053	.072	.071
	cens = 0.8	.052	.048	.052	.047
n = 200	cens = 0	.052	.052	.181	.197
	cens = 0.2	.062	.062	.154	.154
	cens = 0.5	.062	.062	.098	.098
	cens = 0.8	.042	.042	.041	.040
No effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.080	.080	.322	.371
	cens = 0.2	.056	.057	.258	.263
	cens = 0.5	.063	.063	.123	.123
	cens = 0.8	.052	.048	.056	.046
n = 100	cens = 0	.091	.091	.580	.636
	cens = 0.2	.101	.098	.466	.468
	cens = 0.5	.087	.088	.219	.219
	cens = 0.8	.062	.059	.065	.064
n = 200	cens = 0	.129	.129	.874	.900
	cens = 0.2	.132	.132	.766	.767
	cens = 0.5	.112	.111	.356	.354
	cens = 0.8	.071	.070	.093	.091

Table 9.7: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Normal, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
15% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.529	.525	.506	.510
	cens = 0.2	.532	.528	.511	.514
	cens = 0.5	.499	.493	.483	.473
	cens = 0.8	.345	.319	.335	.259
n = 100	cens = 0	.855	.855	.817	.809
	cens = 0.2	.845	.846	.833	.835
	cens = 0.5	.776	.772	.786	.782
	cens = 0.8	.550	.538	.543	.522
n = 200	cens = 0	.982	.982	.985	.985
	cens = 0.2	.989	.989	.989	.989
	cens = 0.5	.973	.972	.971	.971
	cens = 0.8	.845	.841	.841	.834
15% effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.477	.483	.326	.317
	cens = 0.2	.491	.491	.371	.328
	cens = 0.5	.413	.411	.343	.247
	cens = 0.8	.279	.265	.259	.121
n = 100	cens = 0	.811	.811	.566	.546
	cens = 0.2	.777	.777	.627	.560
	cens = 0.5	.722	.722	.645	.495
	cens = 0.8	.517	.511	.501	.329
n = 200	cens = 0	.967	.967	.836	.825
	cens = 0.2	.972	.972	.894	.835
	cens = 0.5	.948	.948	.910	.787
	cens = 0.8	.826	.824	.800	.542
15% effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.366	.374	.085	.072
	cens = 0.2	.377	.382	.116	.056
	cens = 0.5	.364	.365	.231	.060
	cens = 0.8	.266	.256	.221	.029
n = 100	cens = 0	.652	.657	.084	.063
	cens = 0.2	.650	.653	.199	.082
	cens = 0.5	.616	.618	.385	.067
	cens = 0.8	.471	.465	.410	.042
n = 200	cens = 0	.907	.910	.130	.110
	cens = 0.2	.880	.885	.340	.127
	cens = 0.5	.886	.886	.658	.087
	cens = 0.8	.737	.732	.659	.041

Table 9.8: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Normal, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.049	.049	.088	.091
	cens = 0.2	.050	.050	.073	.092
	cens = 0.5	.048	.047	.055	.079
	cens = 0.8	.065	.058	.051	.055
n = 100	cens = 0	.059	.060	.116	.134
	cens = 0.2	.057	.057	.116	.140
	cens = 0.5	.058	.058	.075	.125
	cens = 0.8	.057	.055	.054	.101
n = 200	cens = 0	.058	.058	.202	.217
	cens = 0.2	.066	.066	.161	.216
	cens = 0.5	.065	.064	.095	.218
	cens = 0.8	.058	.058	.067	.176
15% effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.609	.603	.712	.718
	cens = 0.2	.585	.578	.694	.743
	cens = 0.5	.500	.486	.575	.672
	cens = 0.8	.334	.291	.312	.363
n = 100	cens = 0	.879	.877	.943	.950
	cens = 0.2	.864	.861	.933	.955
	cens = 0.5	.805	.802	.853	.931
	cens = 0.8	.535	.523	.546	.712
n = 200	cens = 0	.996	.995	.999	.999
	cens = 0.2	.993	.993	1.00	1.00
	cens = 0.5	.977	.976	.991	.999
	cens = 0.8	.843	.839	.863	.977

Table 9.9: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Normal, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.071	.071	.144	.160
	cens = 0.2	.077	.075	.124	.124
	cens = 0.5	.063	.061	.075	.073
	cens = 0.8	.059	.054	.053	.036
n = 100	cens = 0	.092	.090	.242	.251
	cens = 0.2	.093	.094	.189	.191
	cens = 0.5	.071	.070	.098	.096
	cens = 0.8	.059	.055	.064	.060
n = 200	cens = 0	.135	.133	.428	.442
	cens = 0.2	.122	.120	.316	.316
	cens = 0.5	.102	.101	.172	.170
	cens = 0.8	.070	.069	.085	.080
No effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.182	.182	.680	.752
	cens = 0.2	.182	.181	.514	.523
	cens = 0.5	.117	.114	.232	.226
	cens = 0.8	.083	.076	.093	.074
n = 100	cens = 0	.334	.330	.942	.963
	cens = 0.2	.277	.275	.809	.811
	cens = 0.5	.182	.177	.399	.399
	cens = 0.8	.101	.097	.131	.123
n = 200	cens = 0	.563	.560	.998	.999
	cens = 0.2	.505	.498	.972	.971
	cens = 0.5	.337	.333	.688	.687
	cens = 0.8	.168	.164	.219	.216

Table 9.10: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Normal, corr = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
15% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.826	.821	.855	.863
	cens = 0.2	.763	.761	.807	.807
	cens = 0.5	.660	.655	.690	.664
	cens = 0.8	.424	.368	.392	.306
n = 100	cens = 0	.980	.978	.986	.990
	cens = 0.2	.971	.971	.984	.984
	cens = 0.5	.935	.934	.946	.944
	cens = 0.8	.684	.665	.684	.642
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.998	.998	.998	.998
	cens = 0.8	.926	.925	.930	.927
15% effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.665	.666	.578	.567
	cens = 0.2	.634	.631	.569	.521
	cens = 0.5	.558	.553	.528	.434
	cens = 0.8	.355	.320	.315	.170
n = 100	cens = 0	.934	.934	.879	.872
	cens = 0.2	.905	.904	.882	.841
	cens = 0.5	.836	.834	.823	.702
	cens = 0.8	.581	.566	.566	.402
n = 200	cens = 0	.999	.999	.992	.993
	cens = 0.2	.997	.997	.991	.983
	cens = 0.5	.990	.990	.987	.960
	cens = 0.8	.874	.872	.868	.705
15% effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.334	.347	.071	.061
	cens = 0.2	.336	.342	.120	.061
	cens = 0.5	.316	.318	.223	.069
	cens = 0.8	.247	.233	.213	.035
n = 100	cens = 0	.595	.602	.093	.082
	cens = 0.2	.621	.628	.223	.075
	cens = 0.5	.527	.531	.357	.063
	cens = 0.8	.392	.385	.340	.043
n = 200	cens = 0	.874	.875	.121	.108
	cens = 0.2	.876	.878	.394	.102
	cens = 0.5	.850	.850	.649	.069
	cens = 0.8	.682	.682	.622	.049

Table 9.11: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Normal, corr = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.080	.080	.146	.156
	cens = 0.2	.072	.074	.110	.138
	cens = 0.5	.060	.057	.072	.112
	cens = 0.8	.075	.063	.067	.078
n = 100	cens = 0	.098	.097	.234	.253
	cens = 0.2	.099	.099	.184	.264
	cens = 0.5	.069	.069	.111	.199
	cens = 0.8	.050	.046	.049	.105
n = 200	cens = 0	.153	.153	.462	.483
	cens = 0.2	.137	.135	.316	.435
	cens = 0.5	.113	.111	.185	.395
	cens = 0.8	.071	.069	.078	.208
15% effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.883	.879	.968	.970
	cens = 0.2	.875	.867	.941	.958
	cens = 0.5	.755	.734	.810	.869
	cens = 0.8	*	*	*	*
n = 100	cens = 0	.995	.994	.999	.999
	cens = 0.2	.995	.994	1.00	1.00
	cens = 0.5	.963	.959	.982	.993
	cens = 0.8	.712	.683	.712	.836
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	.963	.962	.968	.992

* Programs did not converge due to sparse data.

Table 9.12: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Normal, corr = 0.8).

CHAPTER 10

BIVARIATE LOG-NORMAL DISTRIBUTION

10.1 DATA GENERATION

In the univariate setting, we say that X follows a log-normal distribution with parameters μ and σ^2 if $Y = \log(X) \sim N(\mu, \sigma^2)$. This is denoted by $X \sim LN(\mu, \sigma^2)$. Analogously, we say that (X_1, X_2) follow a bivariate log-normal distribution with parameters μ_1, μ_2, σ_1^2, σ_2^2, and ρ if $(Y_1, Y_2)' \overset{d}{=} (\log X_1, \log X_2)' \sim BVN(\mu, \Sigma)$ where $\mu = (\mu_1, \mu_2)'$ and $\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{bmatrix}$ with $\sigma_{12} = Cov(Y_1, Y_2) = \rho\sqrt{\sigma_1^2 + \sigma_2^2}$ and $\overset{d}{=}$ denotes 'equal in distribution'. We will write $(X_1, X_2)' \sim BVLN(\mu, \Sigma)$. The moments of the $X's$ are derived from the moment generating function of the $Y's$. Therefore the first and second order moments are given by

$$E\left[\prod_{k=1}^{2} X_k^{r_k}\right] = E\left[\exp(r_1 X_1 + r_2 X_2)\right]$$

$$= \exp\left(r_1\mu_1 + r_2\mu_2 + \frac{1}{2}\left[r_1\left(r_1\sigma_1^2 + r_2\sigma_{12}\right) + r_2\left(r_2\sigma_2^2 + r_1\sigma_{12}\right)\right]\right).$$

Thus

$$E(X_k) = \exp\left(\mu_k + \frac{1}{2}\sigma_k^2\right)$$

and

$$E\left(X_k^2\right)=\exp\left(2\mu_k+\frac{1}{2}\left(4\sigma_k^2\right)\right) \text{ for } k=1,2$$

while

$$E\left(X_1X_2\right)=\exp\left(\left(\mu_1+\mu_2\right)+\frac{1}{2}\left(\sigma_1^2+2\sigma_{12}+\sigma_2^2\right)\right).$$

From these moments we can obtain

$$Cov\left(X_1,X_2\right)=E\left(X_1X_2\right)-E\left(X_1\right)E\left(X_2\right)$$

and

$$Corr\left(X_1,X_2\right)=\frac{Cov\left(X_1,X_2\right)}{\sqrt{Var\left(X_1\right)Var\left(X_2\right)}}$$

where

$$Var\left(X_k\right)=E\left(X_k^2\right)-\left[E\left(X_k\right)\right]^2 \text{ for } k=1,2\,.$$

We would again like to incorporate additional parameters β_1 and β_2 to allow for different treatment effects on the mean survival time within each cause of failure, and also allowing for different variances within the treatments groups for each cause of failure by incorporating parameters λ_1 and λ_2 into the variance terms. Therefore, in order to generate $\left(X_1,X_2\mid Z\right)'\sim BVLN\left(\mu^*,\Sigma^*\right)$ where

$$\begin{aligned}\mu^* &=\left(\mu_1^*,\mu_2^*\right)'\\ &=\left(\mu_1\exp\left(\beta_1Z\right),\mu_2\exp\left(\beta_2Z\right)\right)'\end{aligned}$$

and

$$\Sigma^*=\begin{bmatrix}\sigma_1^2\exp\left(\lambda_1Z\right) & \rho\sqrt{\sigma_1^2\exp\left(\lambda_1Z\right)\sigma_2^2\exp\left(\lambda_2Z\right)}\\ \rho\sqrt{\sigma_1^2\exp\left(\lambda_1Z\right)\sigma_2^2\exp\left(\lambda_2Z\right)} & \sigma_2^2\exp\left(\lambda_2Z\right)\end{bmatrix}$$

we will first generate $(Y_1, Y_2 \mid Z)' \sim BVN(\mu^*, \Sigma^*)$ using the conditional method described in the previous section. We will then transform the $Y's$ by letting

$$X_k = \exp(Y_k) \text{ for } k = 1,2 .$$

We will then have $(X_1, X_2 \mid Z)' \sim BVLN(\mu^*, \Sigma^*)$ as desired. Therefore

$$E(X_k \mid Z) = \exp\left(\mu_k \exp(\beta_k Z) + \frac{1}{2}\sigma_k^2 \exp(\lambda_k Z) \right),$$

$$E(X_k^2 \mid Z) = \exp\left(2\mu_k \exp(\beta_k Z) + 2\sigma_k^2 \exp(\lambda_k Z)\right)$$

and

$$Var(X_k \mid Z) = E(X_k^2 \mid Z) - [E(X_k \mid Z)]^2 \text{ for } k = 1,2 .$$

Again within each treatment group, we will generate fixed censoring times (d_{1Z}, d_{2Z}) where d_{kZ} is the treatment specific censoring time associated with the generated Y_k variables for $k = 1,2$. These censoring times will be chosen in such a way that they result in censoring probabilities

$$p_1 = P(d_{1Z} < Y_1 \mid Z)$$

and

$$p_2 = P(d_{2Z} < Y_2 \mid Z).$$

Therefore

$$d_{1Z} = \mu_1^* - \Phi^{-1}(p_1)\sqrt{\sigma_1^2 \exp(\lambda_1 Z)}$$

and

$$d_{2Z} = \mu_2^* - \Phi^{-1}(p_2)\sqrt{\sigma_2^2 \exp(\lambda_2 Z)}$$

where $\Phi^{-1}(x)$ denotes the inverse of the standard normal cumulative distribution function evaluated at x. Then to obtain the fixed censoring times (c_{1Z}, c_{2Z}) associated with the $X's$, we will again apply the same transformation used for the $Y's$ by letting $c_{kZ} = \exp(d_{kZ})$ for $k = 1,2$. Due to the order preserving nature of the transformation used, we have that

$$p_1 = P(d_{1Z} < Y_1 \mid Z) = P(c_{1Z} < X_1 \mid Z)$$

and

$$p_2 = P(d_{2Z} < Y_2 \mid Z) = P(c_{2Z} < X_2 \mid Z).$$

10.2 DETERMINING SIMULATION PARAMETERS

Finally, we will describe the method used for determining the parameters required in order to simulate data from a desired bivariate log-normal distribution. Recall that the first and second moments, conditional on Z, are given by

$$E(X_k \mid Z) = \exp\left(\mu_k \exp(\beta_k Z) + \frac{1}{2}\sigma_k^2 \exp(\lambda_k Z)\right)$$

and

$$E(X_k^2 \mid Z) = \exp(2\mu_k \exp(\beta_k Z) + 2\sigma_k^2 \exp(\lambda_k Z)) \text{ for } k = 1,2$$

respectively.

We will again assume that $\lambda_1 = \lambda_2 = 0$ and suppose we wish to generate data having control means and variances given by

$$E[X_k \mid Z = 0] = \mu_{0_k}$$

and

$$Var(X_k \mid Z=0) = E(X_k^2 \mid Z=0) - [E(X_k \mid Z=0)]^2 = \sigma_{0_k}^2 \text{ for } k=1,2$$

respectively. Therefore we have that

$$E[X_k \mid Z=0] = \mu_{0_k} = \exp\left(\mu_k + \frac{1}{2}\sigma_k^2\right)$$

and

$$Var(X_k \mid Z=0) = \sigma_{0_k}^2 = \exp(2\mu_k + 2\sigma_k^2) - (\mu_{0_k})^2 \text{ for } k=1,2.$$

We now have four equations which can be solved for the four unknown parameters (μ_1, μ_2, σ_1^2, and σ_2^2). Thus we are able to generate data with the desired control means and variances by setting

$$\mu_k = 2\log(\mu_{0_k}) - \frac{1}{2}\log(\sigma_{0_k}^2 + \mu_{0_k}^2)$$

and

$$\sigma_k^2 = \log(\sigma_{0_k}^2 + \mu_{0_k}^2) - 2\log(\mu_{0_k}) \text{ for } k=1,2.$$

Again, suppose we wish to generate data for the treatment groups such that they result in treatment means having a specified percentage increase (or decrease) over the control means. We will denote the treatment means by

$$E[X_k \mid Z=1] = \mu_{1_k} \text{ for } k=1,2.$$

The desired percentage increase over the control means will be denoted by r_k for $k=1,2$. Therefore we have that

$$E[X_k \mid Z=1] = \mu_{1_k} = \mu_{0_k}(1+r_k)$$

and

$$E\left[X_k \mid Z=1\right]=\mu_{1_k}=\exp\left(\mu_k \exp\left(\beta_k\right)+\frac{1}{2}\sigma_k^2\right) \text{ for } k=1,2.$$

Thus we are able to generate data with the desired means by setting

$$\beta_k=\log\left\{\frac{1}{\mu_k}\left[\log\left\{\mu_{0_k}\left(1+r_k\right)\right\}-\frac{1}{2}\sigma_k^2\right]\right\} \text{ for } k=1,2.$$

10.3 SIMULATION SUMMARY

In this situation, the proportional hazards assumption is again not met and the event times due to the two competing causes of failure are independent only when the correlation is set to zero.

In Table 10.1, it appears that T^{CPH} and T^{LR} are consistently better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring, regardless of whether the beneficial treatment effect on type 2 failure is small or large. T^{G} and T^{FG} are better at preserving the nominal alpha level when the sample size is small, when the amount of censoring is large, and when the beneficial treatment effect on type 2 failure is small. When we allowed the competing risks to have a small positive correlation in Table 10.4, we again saw similar results. However as the correlation became larger (Tables 10.7 and 10.10), we saw that all tests were better at preserving the nominal alpha level when the sample size was small, when the amount of censoring was large, and when the beneficial treatment effect on type 2 failure was small. Furthermore, each test's ability to preserve the

nominal alpha level appeared to worsen as the positive correlation between the two competing risks increased.

In Table 10.2, it appears that T^{CPH} and T^{LR} are consistently more powerful than T^{G} and T^{FG}. All tests appear to gain power for larger sample sizes. When the effect on type 2 failure is beneficial (panels 2 and 3 of Table 10.2), we see that the power results for T^{CPH} and T^{LR} remain similar to those found when there was no treatment effect on type 2 failure, however T^{G} and T^{FG} lose considerable power as the treatment's beneficial effect on type 2 failure gets larger. When we allowed for the competing risks to be positively correlated (Tables 10.5, 10.8, and 10.11), we again saw that all tests appeared to gain power for larger sample sizes and lose power as the beneficial treatment effect on type 2 failure increased. Although T^{CPH} and T^{LR} are typically the more powerful tests, we see that T^{G} and T^{FG} are more powerful for highly correlated competing risks when there is no treatment effect on type 2 failure. When there is little or no beneficial treatment effect on type 2 failure, all tests appear to gain power as the correlation between the competing risks increased. However when the beneficial treatment effect on type 2 failure is large, all tests appear to lose power as the correlation between the competing risks increased.

In panel 1 of Table 10.3, it appears that T^{CPH} and T^{LR} are typically better at preserving the nominal alpha level when compared to T^{G} and T^{FG}. T^{CPH} and T^{LR} appear to preserve the nominal alpha level for all sample sizes and all amounts of censoring. T^{G} is better at preserving the nominal alpha level when the sample size is small and performs quite well when the amount of censoring is large. T^{FG} is better at preserving the nominal alpha level for smaller sample sizes and the results are

somewhat similar regardless of the amount of censoring. When we allowed for the competing risks to be positively correlated (panel 1 of Tables 10.6, 10.9, and 10.12), we saw that all tests were better at preserving the nominal alpha level for small sample sizes and when the amount of censoring was large. Furthermore, each test's ability to preserve the nominal alpha level appeared to worsen as the positive correlation between the two competing risks increased.

When we consider an adverse treatment effect on type 2 failure, as shown in panel 2 of Table 10.3, we see that T^{G} and T^{FG} are now the more powerful tests when compared to T^{CPH} and T^{LR}. Again, all tests appear to gain power for larger sample sizes. When we allowed for the competing risks to be positively correlated (panel 2 of Tables 10.6, 10.9, and 10.12), we again saw that all tests appeared to gain power for larger sample sizes. Also, the power for each test rose as the correlation between the competing risks increased.

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.056	.056	.088	.107
	cens = 0.2	.048	.051	.078	.080
	cens = 0.5	.051	.052	.066	.066
	cens = 0.8	.058	.055	.055	.046
n = 100	cens = 0	.059	.058	.150	.168
	cens = 0.2	.054	.056	.131	.134
	cens = 0.5	.050	.050	.083	.082
	cens = 0.8	.058	.058	.061	.056
n = 200	cens = 0	.055	.056	.243	.254
	cens = 0.2	.044	.043	.228	.232
	cens = 0.5	.040	.041	.111	.111
	cens = 0.8	.050	.049	.058	.054
No effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.050	.051	.327	.437
	cens = 0.2	.051	.052	.298	.315
	cens = 0.5	.043	.041	.115	.111
	cens = 0.8	.044	.042	.040	.035
n = 100	cens = 0	.052	.052	.663	.735
	cens = 0.2	.048	.049	.555	.559
	cens = 0.5	.046	.046	.201	.201
	cens = 0.8	.052	.050	.059	.055
n = 200	cens = 0	.056	.057	.925	.955
	cens = 0.2	.038	.037	.866	.866
	cens = 0.5	.053	.053	.327	.327
	cens = 0.8	.041	.041	.062	.061

Table 10.1: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Log-Normal, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
15% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.840	.842	.706	.676
	cens = 0.2	.827	.830	.729	.740
	cens = 0.5	.790	.788	.763	.758
	cens = 0.8	.627	.603	.613	.513
n = 100	cens = 0	.974	.974	.939	.929
	cens = 0.2	.987	.987	.968	.969
	cens = 0.5	.980	.980	.968	.969
	cens = 0.8	.904	.895	.890	.873
n = 200	cens = 0	1.00	1.00	.999	.998
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	.999	.999
	cens = 0.8	.988	.988	.987	.987
15% effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.842	.845	.498	.421
	cens = 0.2	.855	.860	.560	.488
	cens = 0.5	.851	.854	.730	.518
	cens = 0.8	.624	.608	.571	.292
n = 100	cens = 0	.991	.991	.746	.723
	cens = 0.2	.994	.994	.859	.789
	cens = 0.5	.982	.982	.939	.792
	cens = 0.8	.888	.879	.858	.626
n = 200	cens = 0	1.00	1.00	.952	.941
	cens = 0.2	1.00	1.00	.987	.962
	cens = 0.5	1.00	1.00	.999	.977
	cens = 0.8	.992	.992	.990	.890
15% effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.864	.875	.180	.096
	cens = 0.2	.859	.866	.254	.098
	cens = 0.5	.822	.830	.598	.086
	cens = 0.8	.659	.651	.601	.057
n = 100	cens = 0	.993	.995	.209	.127
	cens = 0.2	.992	.993	.460	.134
	cens = 0.5	.988	.988	.846	.106
	cens = 0.8	.904	.901	.858	.051
n = 200	cens = 0	1.00	1.00	.320	.257
	cens = 0.2	1.00	1.00	.761	.229
	cens = 0.5	1.00	1.00	.993	.165
	cens = 0.8	.995	.995	.988	.045

Table 10.2: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Log-Normal, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.051	.053	.094	.105
	cens = 0.2	.047	.048	.103	.126
	cens = 0.5	.051	.051	.062	.127
	cens = 0.8	.059	.054	.057	.117
n = 100	cens = 0	.050	.050	.170	.191
	cens = 0.2	.044	.045	.128	.199
	cens = 0.5	.060	.059	.076	.244
	cens = 0.8	.055	.052	.053	.189
n = 200	cens = 0	.057	.060	.285	.298
	cens = 0.2	.052	.054	.225	.334
	cens = 0.5	.051	.051	.084	.423
	cens = 0.8	.051	.049	.046	.396
15% effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.819	.813	.892	.896
	cens = 0.2	.820	.816	.881	.918
	cens = 0.5	.750	.738	.807	.917
	cens = 0.8	.497	.429	.463	.692
n = 100	cens = 0	.984	.984	.992	.994
	cens = 0.2	.988	.987	.995	.998
	cens = 0.5	.963	.962	.981	.999
	cens = 0.8	.782	.769	.800	.971
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	.977	.974	.982	1.00

Table 10.3: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Log-Normal, corr = 0).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.047	.046	.098	.121
	cens = 0.2	.051	.053	.090	.097
	cens = 0.5	.059	.059	.068	.067
	cens = 0.8	.053	.047	.042	.036
n = 100	cens = 0	.058	.056	.175	.195
	cens = 0.2	.058	.058	.171	.173
	cens = 0.5	.057	.057	.081	.080
	cens = 0.8	.052	.052	.060	.058
n = 200	cens = 0	.054	.053	.324	.348
	cens = 0.2	.051	.052	.260	.263
	cens = 0.5	.059	.060	.116	.115
	cens = 0.8	.049	.048	.056	.053
No effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.056	.058	.417	.514
	cens = 0.2	.058	.058	.359	.369
	cens = 0.5	.056	.057	.154	.149
	cens = 0.8	.047	.045	.052	.039
n = 100	cens = 0	.051	.050	.745	.813
	cens = 0.2	.057	.057	.647	.655
	cens = 0.5	.043	.040	.224	.220
	cens = 0.8	.045	.045	.054	.051
n = 200	cens = 0	.086	.086	.985	.992
	cens = 0.2	.074	.072	.900	.901
	cens = 0.5	.067	.066	.428	.426
	cens = 0.8	.052	.050	.092	.087

Table 10.4: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Log-Normal, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
15% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.856	.853	.767	.740
	cens = 0.2	.834	.831	.788	.796
	cens = 0.5	.824	.822	.798	.792
	cens = 0.8	.635	.616	.623	.522
n = 100	cens = 0	.991	.992	.968	.968
	cens = 0.2	.992	.991	.984	.984
	cens = 0.5	.980	.980	.974	.973
	cens = 0.8	.895	.891	.887	.874
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	.996	.996	.997	.996
15% effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.845	.847	.556	.485
	cens = 0.2	.851	.855	.589	.520
	cens = 0.5	.817	.818	.728	.548
	cens = 0.8	.592	.572	.549	.272
n = 100	cens = 0	.989	.989	.802	.792
	cens = 0.2	.991	.991	.889	.831
	cens = 0.5	.982	.982	.953	.810
	cens = 0.8	.885	.881	.853	.606
n = 200	cens = 0	1.00	1.00	.979	.976
	cens = 0.2	1.00	1.00	.994	.986
	cens = 0.5	1.00	1.00	1.00	.971
	cens = 0.8	.993	.993	.992	.909
15% effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.833	.841	.162	.096
	cens = 0.2	.821	.836	.280	.086
	cens = 0.5	.777	.782	.558	.073
	cens = 0.8	.602	.589	.525	.048
n = 100	cens = 0	.987	.988	.199	.122
	cens = 0.2	.982	.982	.488	.136
	cens = 0.5	.974	.976	.842	.102
	cens = 0.8	.881	.876	.817	.037
n = 200	cens = 0	1.00	1.00	.264	.203
	cens = 0.2	1.00	1.00	.789	.226
	cens = 0.5	.999	.999	.990	.137
	cens = 0.8	.993	.993	.980	.070

Table 10.5: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Log-Normal, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.058	.058	.121	.137
	cens = 0.2	.043	.045	.097	.143
	cens = 0.5	.049	.050	.062	.142
	cens = 0.8	.060	.049	.047	.109
n = 100	cens = 0	.046	.046	.185	.210
	cens = 0.2	.055	.056	.148	.215
	cens = 0.5	.057	.055	.081	.271
	cens = 0.8	.067	.066	.067	.209
n = 200	cens = 0	.057	.057	.347	.376
	cens = 0.2	.051	.052	.276	.434
	cens = 0.5	.059	.057	.114	.451
	cens = 0.8	.054	.053	.059	.423
15% effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.836	.834	.921	.924
	cens = 0.2	.849	.841	.910	.932
	cens = 0.5	.791	.777	.826	.939
	cens = 0.8	.562	.478	.521	.709
n = 100	cens = 0	.993	.993	.998	.999
	cens = 0.2	.998	.998	1.00	1.00
	cens = 0.5	.969	.968	.982	.998
	cens = 0.8	.778	.760	.780	.974
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	.978	.978	.980	1.00

Table 10.6: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Log-Normal, corr = 0.2).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.074	.074	.155	.182
	cens = 0.2	.051	.052	.122	.127
	cens = 0.5	.052	.050	.069	.067
	cens = 0.8	.068	.064	.069	.058
n = 100	cens = 0	.072	.073	.274	.298
	cens = 0.2	.052	.051	.205	.208
	cens = 0.5	.053	.051	.103	.102
	cens = 0.8	.050	.045	.058	.053
n = 200	cens = 0	.089	.088	.464	.482
	cens = 0.2	.096	.095	.374	.375
	cens = 0.5	.056	.056	.156	.156
	cens = 0.8	.069	.068	.078	.077
No effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.115	.113	.634	.721
	cens = 0.2	.087	.087	.544	.548
	cens = 0.5	.087	.086	.206	.201
	cens = 0.8	.056	.050	.061	.051
n = 100	cens = 0	.172	.170	.926	.959
	cens = 0.2	.149	.143	.810	.808
	cens = 0.5	.111	.108	.370	.365
	cens = 0.8	.065	.064	.084	.082
n = 200	cens = 0	.272	.265	.999	.999
	cens = 0.2	.259	.253	.977	.977
	cens = 0.5	.166	.161	.648	.647
	cens = 0.8	.085	.084	.156	.151

Table 10.7: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Log-Normal, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
15% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.919	.919	.906	.904
	cens = 0.2	.913	.911	.900	.906
	cens = 0.5	.843	.842	.843	.836
	cens = 0.8	.653	.622	.643	.525
n = 100	cens = 0	.996	.996	.996	.996
	cens = 0.2	.997	.997	.995	.995
	cens = 0.5	.992	.992	.991	.990
	cens = 0.8	.901	.888	.902	.873
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	.996	.996	.996	.996
15% effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.881	.885	.636	.609
	cens = 0.2	.876	.878	.725	.636
	cens = 0.5	.788	.783	.733	.578
	cens = 0.8	.601	.572	.546	.301
n = 100	cens = 0	.994	.994	.912	.899
	cens = 0.2	.994	.994	.960	.925
	cens = 0.5	.980	.980	.962	.852
	cens = 0.8	.885	.878	.864	.623
n = 200	cens = 0	1.00	1.00	.999	.997
	cens = 0.2	1.00	1.00	.999	.997
	cens = 0.5	1.00	1.00	.998	.994
	cens = 0.8	.997	.997	.996	.926
15% effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.792	.805	.164	.096
	cens = 0.2	.757	.772	.261	.085
	cens = 0.5	.732	.737	.518	.062
	cens = 0.8	.514	.499	.464	.045
n = 100	cens = 0	.971	.974	.173	.131
	cens = 0.2	.972	.972	.537	.130
	cens = 0.5	.954	.955	.842	.095
	cens = 0.8	.834	.830	.776	.059
n = 200	cens = 0	1.00	1.00	.242	.190
	cens = 0.2	1.00	1.00	.824	.200
	cens = 0.5	.999	.999	.979	.124
	cens = 0.8	.984	.983	.971	.062

Table 10.8: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Log-Normal, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.062	.064	.161	.188
	cens = 0.2	.058	.060	.136	.199
	cens = 0.5	.050	.049	.070	.162
	cens = 0.8	.048	.042	.047	.087
n = 100	cens = 0	.078	.078	.320	.352
	cens = 0.2	.071	.070	.222	.342
	cens = 0.5	.062	.060	.112	.331
	cens = 0.8	.062	.060	.062	.233
n = 200	cens = 0	.120	.119	.542	.577
	cens = 0.2	.104	.103	.409	.604
	cens = 0.5	.078	.078	.180	.609
	cens = 0.8	.065	.063	.080	.451
15% effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.939	.933	.981	.988
	cens = 0.2	.945	.937	.980	.987
	cens = 0.5	.843	.825	.901	.965
	cens = 0.8	.579	.456	.515	.728
n = 100	cens = 0	1.00	1.00	.999	.999
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.989	.986	.995	1.00
	cens = 0.8	.825	.804	.821	.972
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	.976	.974	.980	1.00

Table 10.9: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Log-Normal, corr = 0.5).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.096	.097	.310	.337
	cens = 0.2	.100	.097	.225	.227
	cens = 0.5	.073	.073	.120	.118
	cens = 0.8	.064	.056	.069	.050
n = 100	cens = 0	.177	.176	.580	.604
	cens = 0.2	.150	.149	.423	.425
	cens = 0.5	.121	.121	.208	.206
	cens = 0.8	.076	.074	.088	.081
n = 200	cens = 0	.260	.257	.870	.881
	cens = 0.2	.276	.273	.739	.739
	cens = 0.5	.183	.181	.377	.372
	cens = 0.8	.077	.076	.104	.100
No effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.302	.294	.935	.977
	cens = 0.2	.255	.246	.823	.827
	cens = 0.5	.193	.190	.393	.387
	cens = 0.8	.110	.107	.123	.105
n = 100	cens = 0	.535	.526	1.00	1.00
	cens = 0.2	.506	.499	.984	.985
	cens = 0.5	.303	.300	.670	.666
	cens = 0.8	.133	.128	.186	.175
n = 200	cens = 0	.849	.847	1.00	1.00
	cens = 0.2	.771	.769	1.00	1.00
	cens = 0.5	.577	.569	.923	.921
	cens = 0.8	.231	.226	.334	.329

Table 10.10: Empirical rejection probabilities at 0.05 level, no effect on type 1 failure (Log-Normal, corr = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
15% effect on type 1 failure, No effect on type 2 failure					
n = 50	cens = 0	.993	.992	.997	.996
	cens = 0.2	.995	.994	.996	.995
	cens = 0.5	.954	.949	.956	.946
	cens = 0.8	.765	.691	.712	.675
n = 100	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.999	.999	1.00	1.00
	cens = 0.8	.961	.949	.958	.939
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	1.00	1.00	1.00	1.00
15% effect on type 1 failure, 5% effect on type 2 failure					
n = 50	cens = 0	.969	.969	.922	.913
	cens = 0.2	.956	.956	.921	.890
	cens = 0.5	.883	.877	.849	.741
	cens = 0.8	.669	.628	.607	.395
n = 100	cens = 0	.999	.999	.999	.998
	cens = 0.2	1.00	1.00	.998	.992
	cens = 0.5	.995	.995	.991	.974
	cens = 0.8	.906	.896	.890	.731
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	.992	.991	.991	.964
15% effect on type 1 failure, 15% effect on type 2 failure					
n = 50	cens = 0	.743	.749	.121	.067
	cens = 0.2	.690	.702	.318	.091
	cens = 0.5	.677	.681	.502	.070
	cens = 0.8	.487	.473	.437	.036
n = 100	cens = 0	.968	.970	.154	.111
	cens = 0.2	.943	.947	.584	.098
	cens = 0.5	.926	.928	.808	.082
	cens = 0.8	.766	.755	.703	.042
n = 200	cens = 0	.999	.999	.232	.190
	cens = 0.2	1.00	1.00	.862	.174
	cens = 0.5	.997	.997	.980	.114
	cens = 0.8	.967	.967	.947	.059

Table 10.11: Empirical rejection probabilities at 0.05 level, positive effect on type 1 failure (Log-Normal, corr = 0.8).

Setting		T^{CPH}	T^{LR}	T^{G}	T^{FG}
No effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	.128	.122	.359	.409
	cens = 0.2	.119	.109	.255	.376
	cens = 0.5	.079	.073	.118	.261
	cens = 0.8	.092	.066	.074	.136
n = 100	cens = 0	.242	.239	.649	.687
	cens = 0.2	.202	.199	.487	.655
	cens = 0.5	.131	.127	.225	.508
	cens = 0.8	.084	.073	.093	.321
n = 200	cens = 0	.416	.411	.931	.936
	cens = 0.2	.358	.355	.783	.927
	cens = 0.5	.242	.240	.452	.849
	cens = 0.8	.120	.113	.133	.593
15% effect on type 1 failure, -5% effect on type 2 failure					
n = 50	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	.996	.995	.998	1.00
	cens = 0.5	.975	.965	.983	.996
	cens = 0.8	.712	.530	.605	.889
n = 100	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	.999	.999	1.00	1.00
	cens = 0.8	.931	.906	.917	.982
n = 200	cens = 0	1.00	1.00	1.00	1.00
	cens = 0.2	1.00	1.00	1.00	1.00
	cens = 0.5	1.00	1.00	1.00	1.00
	cens = 0.8	.997	.996	.998	1.00

Table 10.12: Empirical rejection probabilities at 0.05 level, adverse effect on type 2 failure (Log-Normal, corr = 0.8).

CHAPTER 11

CONCLUSION

Friedlin and Korn (2005) provide results from a special case of general copula models for generating bivariate survival data, using an exponential marginal distribution, a sample size of 200, and no censoring. The corresponding results from our simulations for the Sarmanov bivariate exponential distribution and the Cai-Prentice bivariate exponential distribution are in close agreement with those given by Friedlin and Korn (2005). Additionally, we investigated a variety of bivariate distributions, sample sizes, and censoring probabilities.

We found that the test based on Cox's proportional hazards model and the log-rank test were typically preferred in situations where preserving the nominal level alpha was of interest. In terms of power, we also found these tests to be generally preferred when the assumptions of proportional hazards and independence were met, in situations involving little censoring, and when there was a beneficial treatment effect on the competing risk.

Gray's test and the test due to Fine and Gray were typically preferred in situations where there was an adverse treatment effect on the competing risk. The performance of Gray's test and the test due to Fine and Gray also improved, relative

to the test based on Cox's proportional hazards model and the log-rank test, when the proportional hazards assumption was violated and when the amount of censoring was relatively high. Specifically, we saw that in non-proportional hazards settings, Gray's test and the test due to Fine and Gray were generally preferred when there was no treatment effect, or an adverse treatment effect on the competing risk.

Overall, in terms of power, it appears that the effect of the treatment on the competing risk and the amount of censoring with respect to the cause of interest are strong factors to consider when determining which test to utilize. A positive treatment effect on the competing risk results in a high incidence for the cause of interest in the treatment group. In these situations, the test based on Cox's proportional hazards model and the log-rank test were preferred. However when there is a negative treatment effect on the competing risk, which leads to more censored observations and a low incidence for the cause of interest in the treatment group, Gray's test and the test due to Fine and Gray were preferred. This suggests that the difference in the performance of the tests lies in how each one handles censoring with respect to the cause of interest.

The test based on Cox's proportional hazards model and the log-rank test assume independent censoring, whereas Gray's test and the test due to Fine and Gray do not. Thus when there is little censoring with respect to the cause of interest in the treatment group (e.g. a positive treatment effect for the competing risk), the assumptions for the test based on Cox's proportional hazards model and the log-rank test are only slightly violated and these tests perform quite well. However, in situations were the cause of interest is rarely observed in the treatment group (e.g. a

negative treatment effect for the competing risk), the assumptions for the test based on Cox's proportional hazards model and the log-rank test are greatly violated. The result is a loss of power and we see that Gray's test and the test due to Fine and Gray are actually preferred in such situations.

Future research would include, but not be limited to, further investigation into the performance of Gray's test under the null hypothesis, and the performance of all tests when the proportional hazards assumption is not satisfied. It would also be of interest to quantify the bias associated with the treatment coefficients proposed in our competing risks models. Additionally, one may consider applications of this research to other areas such as "cure-rate" models.

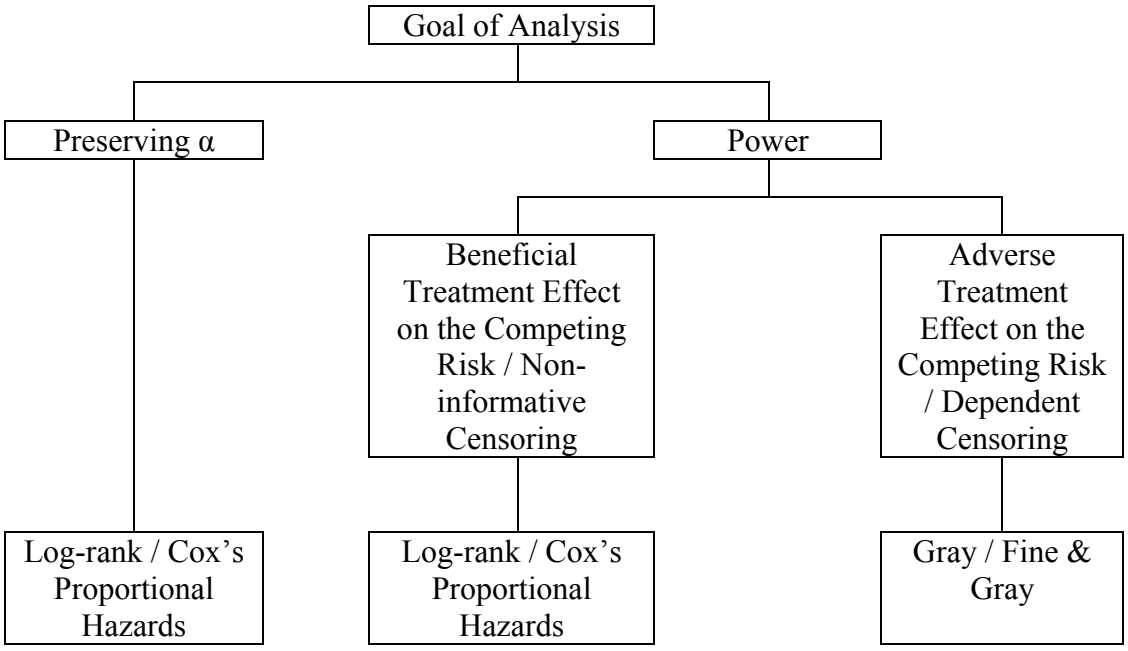

Figure 11.1: Guideline to the Analysis of Competing Risks data.

APPENDIX A

COMPUTER CODE FOR HOEL EXAMPLE

A.1: R 2.0.1 CODE FOR TABLES 4.1 - 4.3 (GRAY)

```
hoel <- read.table ("F:\\Competing Risks\\Paper\\Hoel Data\\hoel.txt",
row.names=NULL, header=T)

ci<-cuminc(hoel$ftime, hoel$fstatus, hoel$group)

ci
```

A.2: R 2.0.1 CODE FOR TABLE 4.4 (FINE & GRAY)

```
crr1<-crr(hoel$ftime, hoel$fstatus, hoel$group, failcode=1)
crr2<-crr(hoel$ftime, hoel$fstatus, hoel$group, failcode=2)
crr3<-crr(hoel$ftime, hoel$fstatus, hoel$group, failcode=3)

crr1
crr2
crr3
```

A.3: R 2.0.1 CODE FOR TABLE 4.5 (COX)

```
n<-nrow(hoel)
cause1<-cause2<-censored<-rep(0,n)

for (i in 1:n) {       cause1[i]<-ifelse(hoel$fstatus[i]==1 , 1, 0)
            cause2[i]<-ifelse(hoel$fstatus[i]==2, 1, 0)
            censored[i]<-ifelse(hoel$fstatus[i]==3, 1, 0)}

cph1<-summary(coxph(Surv(hoel$ftime,cause1)~hoel$group))
cph2<-summary(coxph(Surv(hoel$ftime,cause2)~hoel$group))
cph3<-summary(coxph(Surv(hoel$ftime,censored)~hoel$group))

cph1
cph2
cph3
```

A.4: R 2.0.1 CODE FOR TABLE 4.6 (LOG-RANK)

```
LR1<-survdiff(Surv(hoel$ftime,cause1)~hoel$group, rho=0)
LR2<-survdiff(Surv(hoel$ftime,cause2)~hoel$group, rho=0)
LR3<-survdiff(Surv(hoel$ftime,censored)~hoel$group, rho=0)

LR1
LR2
LR3
```

A.5: R 2.0.1 CODE FOR FIGURES 4.1 - 4.3 (GRAY)

```
max(hoel$ftime)

plot.cuminc(ci[1:2], main="Thymic Lymphoma", curvlab=c("Conventional", "Germ-
free"), xlim=c(0,1020),ylim=c(0,.5),xlab="Days", ylab="Cumulative Incidence",
lwd=1)

plot.cuminc(ci[3:4], main="Reticulum Cell Sarcoma", curvlab=c("Conventional",
"Germ-free"), xlim=c(0,1020), ylim=c(0,.5),xlab="Days", ylab="Cumulative
Incidence", lwd=1)

plot.cuminc(ci[5:6], main="Other Causes", curvlab=c("Conventional", "Germ-free"),
xlim=c(0,1020), ylim=c(0,.5),xlab="Days", ylab="Cumulative Incidence", lwd=1)
```

A.6: STATA CODE FOR FIGURES 4.4 - 4.6 (STATA/SE 9.2)

```
stset ftime, failure(fstatus==1)

stcompet ci_conv=ci hilim_conv=hi lowlim_conv=lo if(group==1), compet1(2)
compet2(3)

stcompet ci_gf=ci hilim_gf=hi lowlim_gf=lo if(group==2), compet1(2) compet2(3)

gen ci_tl_conv=ci_conv if fstatus==1
gen ci_rcs_conv=ci_conv if fstatus==2
gen ci_oth_conv=ci_conv if fstatus==3
gen ci_tl_gf=ci_gf if fstatus==1
gen ci_rcs_gf=ci_gf if fstatus==2
gen ci_oth_gf=ci_gf if fstatus==3

label var ci_tl_conv "Conventional"
label var ci_rcs_conv "Conventional"
```

```
label var ci_oth_conv "Conventional"
label var ci_tl_gf "Germ-free"
label var ci_rcs_gf "Germ-free"
label var ci_oth_gf "Germ-free"

scatter ci_tl_conv ci_tl_gf ftime, m(i i) c(J J) clpattern(solid dash) ylabel(0(.1).5)
title(Thymic Lymphoma) ytitle(Cumulative Incidence) xtitle(Days) sort

scatter ci_rcs_conv ci_rcs_gf ftime, m(i i) c(J J) clpattern(solid dash) ylabel(0(.1).5)
title(Reticulum Cell Sarcoma) ytitle(Cumulative Incidence) xtitle(Days) sort

scatter ci_oth_conv ci_oth_gf ftime, m(i i) c(J J) clpattern(solid dash) ylabel(0(.1).5)
title(Other Causes) ytitle(Cumulative Incidence) xtitle(Days) sort
```

A.7: STATA CODE FOR FIGURES 4.7 - 4.9 (KLEIN & MOESCHBERGER)

Note: The cumulative incidence estimates were first obtained in Excel

```
label var ci_tl_gf "Germ-free"
label var ci_tl_conv "Conventional"
label var ci_rcs_gf "Germ-free"
label var ci_rcs_conv "Conventional"
label var ci_oth_gf "Germ-free"
label var ci_oth_conv "Conventional"


scatter ci_tl_conv ftime_conv, m(i) c(J) clpattern(solid) || scatter ci_tl_gf ftime_gf,
m(i) c(J) clpattern(dash) ylabel(0(.1).5) title(Thymic Lymphoma) ytitle(Cumulative
Incidence) xtitle(Days) sort

scatter ci_rcs_conv ftime_conv, m(i) c(J) clpattern(solid) || scatter ci_rcs_gf ftime_gf,
m(i) c(J) clpattern(dash) ylabel(0(.1).5) title(Reticulum Cell Sarcoma)
ytitle(Cumulative Incidence) xtitle(Days) sort

scatter ci_oth_conv ftime_conv, m(i) c(J) clpattern(solid) || scatter ci_oth_gf ftime_gf,
m(i) c(J) clpattern(dash) ylabel(0(.1).5) title(Other Causes) ytitle(Cumulative
Incidence) xtitle(Days) sort
```

A.8: R 2.0.1 CODE FOR FIGURES 4.10 – 4.11 (GRAY)

```
klein <- read.table ("E:\\Competing Risks\\Paper\\klein_gray_ex.txt",
row.names=NULL, header=T)
```

```
kci<-cuminc(klein$ftime, klein$fstatus, klein$group)

max(klein$ftime)

plot.cuminc(kci[1:2], main="Thymic Lymphoma", curvlab=c("Conventional",
"Germ-free"), xlim=c(0,1000),ylim=c(0,1),xlab="Days", ylab="Cumulative
Incidence", lwd=1)

plot.cuminc(kci[3:4], main="Reticulum Cell Sarcoma", curvlab=c("Conventional",
"Germ-free"), xlim=c(0,1000), ylim=c(0,1),xlab="Days", ylab="Cumulative
Incidence", lwd=1)
```

A.9: SAS CODE FOR FIGURES 4.12 – 4.13 (KLEIN & MOESCHBERGER)

Note: The cumulative incidence estimates were first obtained in SAS

```
PROC IMPORT OUT= WORK.hoel
    DATAFILE= "E:\Competing Risks\Paper\klein_gray_ex.xls"
    DBMS=EXCEL REPLACE;
    SHEET="Sheet1$";
    GETNAMES=YES;
    MIXED=NO;
    SCANTEXT=YES;
    USEDATE=YES;
    SCANTIME=YES;
RUN;

data hoel;
        set hoel;
        cause_tl=(fstatus=1);
        cause_rcs=(fstatus=2);
run;

%macro incid(data,group,cause1,cause2,time,out=);

data lc_one; set &data;
a=&time;
b=&cause1;
c=&cause2;
d=&group;
keep a b c d;

proc sort data=lc_one; by descending a;
```

```
proc iml;
use lc_one; read all into x;
n=nrow(x);
ngrp=max(x[ ,4]);
gnum=J(1,ngrp,0);
do k=1 to n;
gnum[1,x[k,4]] =  gnum[1,x[k,4]]+1;
end;
t=J(1,n,0);
t[1,1]=x[1,1];
ntime=1;
tnow=x[1,1];
do j=2 to n;
if x[j,1] <tnow then do;  ntime=ntime+1; t[1,ntime]=x[j,1]; tnow=x[j,1]; end;
end;
relap=J(ntime,ngrp,0);
trm=J(ntime,ngrp,0);
atrisk=J(ntime,ngrp,0);
do k=1 to ntime;
do j=1 to n;

ax=x[j,1]; at=t[1,k]; ag=x[j,4]; ad=x[j,3]; ar=x[j,2];

if x[j,1] =t[1,k] then do;   relap[k,ag]=relap[k,ag]+ar;
trm[k,ag]=trm[k,ag]+ad; end;
if x[j,1] >= t[1,k] then atrisk[k,ag]=atrisk[k,ag]+1;
end;
end;
lfs=J(ntime,ngrp,-1);
ci_rel=J(ntime,ngrp,-1);
ci_trm=J(ntime,ngrp,-1);
vci_rel=J(ntime,ngrp,-1);
vci_trm=J(ntime,ngrp,-1);
index=J(ntime,1,0);
tt=t(t[1,1:ntime]);

do ig=1 to ngrp;
p=1;
cr=0;
cd=0;
do j=1 to ntime ;
index[j,1]=j;
k=ntime-j+1;

if atrisk[k,ig] >0 then do;
```

```
cr=cr+relap[k,ig]*p/atrisk[k,ig];
cd=cd+trm[k,ig]*p/atrisk[k,ig];
p=p*(1-(trm[k,ig]+relap[k,ig])/atrisk[k,ig]);
lfs[k,ig]=p;
ci_rel[k,ig]=cr;
ci_trm[k,ig]=cd;

end;

else do; lfs[k,ig]=.; ci_rel[k,ig]=.; ci_trm[k,ig]=.; end;
end;
do j=1 to ntime;
know=ntime-j+1;
vr=0;  vd=0;
if ci_rel[know,ig] = . then do;   vci_rel[know,ig]=.; vci_trm[know,ig]=.;end;
else do;
do k=1 to ntime;
jnow=ntime-k+1;
if tt[jnow,1] <= tt[know,1] then do;
wr=(trm[jnow,ig]+relap[jnow,ig])/atrisk[jnow,ig]**2;
wr=wr*lfs[jnow,ig]*(ci_rel[know,ig]-ci_rel[jnow,ig])**2;
q=(relap[jnow,ig]/atrisk[jnow,ig]**2)*lfs[jnow,ig]**2;
q=q*(1-2*(ci_rel[know,ig]-ci_rel[jnow,ig]));
vr=vr+wr+q;
wd=(trm[jnow,ig]+relap[jnow,ig])/atrisk[jnow,ig]**2;
wd=wd*lfs[jnow,ig]*(ci_trm[know,ig]-ci_trm[jnow,ig])**2;
q=trm[jnow,ig]/atrisk[jnow,ig]**2*lfs[jnow,ig]**2;
q=q*(1-2*(ci_trm[know,ig]-ci_trm[jnow,ig]));
vd=vd+wd+q;
end;
end;
end;
vci_rel[know,ig]=sqrt(vr);
vci_trm[know,ig]=sqrt(vd);
end;


end;
nn=ngrp*ntime;
yout=j(nn,6,0);
k=0;
do is=1 to ngrp;
do it=1 to ntime;
k=k+1;
yout[k,1]=tt[it,1];
```

```
yout[k,2]=is;
yout[k,3]=ci_rel[it,is];
yout[k,4]=vci_rel[it,is];
yout[k,5]=ci_trm[it,is];
yout[k,6]=vci_trm[it,is];
end;  end;

create dout from yout;
append from yout;
close dout;
quit;
data io;  set dout;
time=col1;
group=col2;
CI1=col3;
SE_CI1=col4;
CI2=col5;
SE_CI2=col6;
if CI1 =. then se_ci1=.;
if CI2=. then se_ci2=.;
drop col1-col6;
proc sort data=io;  by time;
proc sort data=io;  by group;
proc print data=io;
data &out;  set io;
%mend;

%incid(hoel,group,cause_tl,cause_rcs, ftime,out=ci);
proc print data=ci;
run;
```

Note: The cumulative incidence estimates were plotted in Stata/SE 9.2

```
label var ci_tl_gf "Germ-free"
label var ci_tl_conv "Conventional"
label var ci_rcs_gf "Germ-free"
label var ci_rcs_conv "Conventional"

scatter ci_tl_conv ci_tl_gf time, m(i i) c(J J) clpattern(solid dash) ylabel(0(.1)1)
title(Thymic Lymphoma) ytitle(Cumulative Incidence) xtitle(Days) sort
scatter ci_rcs_conv ci_rcs_gf time, m(i i) c(J J) clpattern(solid dash) ylabel(0(.1)1)
title(Reticulum Cell Sarcoma) ytitle(Cumulative Incidence) xtitle(Days) sort
```

A.10: R 2.0.1 CODE FOR FIGURES 4.14 – 4.16 (FINE & GRAY)

```
hoel <- read.table ("E:\\Competing Risks\\Paper\\hoel.txt", row.names=NULL,
header=T)

crr1.p<-predict(crr1,rbind(1,2))
crr2.p<-predict(crr2,rbind(1,2))
crr3.p<-predict(crr3,rbind(1,2))

plot(crr1.p, lty=1:2, color=1, main="Thymic Lymphoma", xlim=c(0,1020),
ylim=c(0,.5), xlab="Days", ylab="Cumulative Incidence", lwd=1)
legend(0,.5,c("Conventional","Germ-free"),lty=c(1:2))

plot(crr2.p, lty=1:2, color=1, main="Reticulum Cell Sarcoma", xlim=c(0,1020),
ylim=c(0,.5), xlab="Days", ylab="Cumulative Incidence", lwd=1)
legend(0,.5,c("Conventional","Germ-free"),lty=c(1:2))

plot(crr3.p, lty=1:2, color=1, main="Other Causes", xlim=c(0,1020), ylim=c(0,.5),
xlab="Days", ylab="Cumulative Incidence", lwd=1)
legend(0,.5,c("Conventional","Germ-free"),lty=c(1:2))
```

A.11: SAS AND STATA CODE FOR FIGURES 4.17 – 4.19 (ROSTHOJ)

Note: The cumulative incidence estimates were first obtained in SAS

```
%macro CumInc(Data,Strata,Time,Surv);

proc sort data=&Data;
by &Strata;

data nstrat;
set &Data end=last;
by &Strata;
firstS=first.&Strata;
retain nStrata 0;
nStrata+firstS;
if last then call symput('nStrata',nStrata);
drop firstS;
run;

data newData;
set &Data;
start=(&Time eq 0);
retain komb 0;
```

```
komb+start;
med=0;
%do k=0 %to (&nStrata-1) %by 1;
med=med+(komb eq (1+&k*(&nStrata+1)));
%end;
if (med eq 1);
drop start med komb;
run;

proc sort data=newData;
by &Time;

data Time (keep=&Time);
set newData;
%do i=1 %to &nStrata %by 1;
if &Time=lag(&Time) then delete;
%end;

proc sort data=newData;
by &Strata &Time;

data temp1;
set newData; by &Strata;
retain stratum 0;
stratum+first.&Strata;

%do i=1 %to &nStrata %by 1;
data data&i;
set temp1;
if stratum=&i;

data data&i (keep = &Time A&i);
merge data&i Time; by &Time;
retain Surv&i;
if not (&Surv=.) then Surv&i=&Surv;
A&i=-log(Surv&i);

%end;

data A_and_S;
A=0;
%do i=1 %to &nStrata %by 1;
merge data&i; by &Time;
A+A&i;
%end;
```

```
dA=A-lag(A);
if dA eq . then dA=0;

retain S 1;
S+S*(1-dA)-S;
lagS=lag(S);
run;

data temp2 (keep = &Strata &Time);
set temp1;
proc sort data=temp2; by &Time;

data temp3;
merge temp2 A_and_S; by &Time;
proc sort data=temp3; by &Strata &Time;

data data0 (keep = &Time p stratum);
set temp3; by &Strata;
*   lagS=lag(S);

if first.&Strata then do;
lagS=1; end;

retain stratum 0;
stratum+first.&Strata;

%do i=1 %to &nStrata %by 1;
lagA&i=lag(A&i);
if ((stratum eq &i) and (first.&Strata)) then lagA&i=0;
if (stratum ne &i) then lagA&i=0;

l&i=(stratum=&i)*(lagS*(A&i-lagA&i));
retain p&i 0;
p&i+l&i;
if (stratum ne &i) then p&i=0;
%end;

p=0;
%do i=1 %to &nStrata %by 1;
p=p+p&i; %end;
run;

%do i=1 %to &nStrata;
data data&i;
```

```
set data0;
if (stratum eq &i);
p0&i=p; drop p stratum;

proc sort data=data&i;
by &Time;
%end;

data data;
set data1;
%do i=1 %to &nStrata;
data data;
merge data data&i;
by &Time;
%end;

proc sort data=data;
by &Time;

data data;
set data end=last;
by &Time;
n=_N_;
if last then call symput('nobs',n);
drop n;

data data;
set data;
%do i=1 %to &nStrata;
%do j=1 %to &nobs;
dummy=lag(p0&i);
if (p0&i eq .) then p0&i=dummy;
%end;
%end;
drop dummy;
p00=1;
%do i=1 %to &nStrata;
p00=p00-p0&i;
%end;

%mend CumInc;

PROC IMPORT OUT= WORK.hoel
    DATAFILE= "E:\Competing Risks\Paper\Hoel Data\hoel.xls"
    DBMS=EXCEL REPLACE;
```

```
    SHEET="Sheet1$";
    GETNAMES=YES;
    MIXED=NO;
    SCANTEXT=YES;
    USEDATE=YES;
    SCANTIME=YES;
RUN;

data hoel_2;
        set hoel hoel hoel;
        stratum=1 + (_N_ GT 181) + (_N_ GT 362);

        d=(fstatus=1)*(stratum=1)+(fstatus=2)*(stratum=2)+(fstatus=3)*(stratum=3);
            group_tl=group*(stratum=1);
            group_rcs=group*(stratum=2);
            group_oth=group*(stratum=3);
run;

data cov;
        input group_tl group_rcs group_oth;
        cards;
        1 0 0
        0 1 0
        0 0 1
        ;
run;

proc phreg data=hoel_2;
        model ftime*d(0)=group_tl group_rcs group_oth;
        strata stratum;
        baseline out=ciData covariates=cov survival=surv / nomean method=ch;
run;

%CumInc(ciData, stratum, ftime,surv);
proc print data=data;
run;

PROC EXPORT DATA= WORK.DATA
        OUTFILE= "E:\Competing Risks\Paper\Rosthoj\Rosthoj_conv.xls"
        DBMS=EXCEL REPLACE;
    SHEET="Rosthoj_conv";
RUN;

PROC IMPORT OUT= WORK.hoel
    DATAFILE= "E:\Competing Risks\Paper\Hoel Data\hoel.xls"
```

```
    DBMS=EXCEL REPLACE;
    SHEET="Sheet1$";
    GETNAMES=YES;
    MIXED=NO;
    SCANTEXT=YES;
    USEDATE=YES;
    SCANTIME=YES;
RUN;

data hoel_2;
        set hoel hoel hoel;
            stratum=1 + (_N_ GT 181) + (_N_ GT 362);

        d=(fstatus=1)*(stratum=1)+(fstatus=2)*(stratum=2)+(fstatus=3)*(stratum=3);
            group_tl=group*(stratum=1);
            group_rcs=group*(stratum=2);
            group_oth=group*(stratum=3);
run;

data cov;
        input group_tl group_rcs group_oth;
        cards;
        2 0 0
        0 2 0
        0 0 2
        ;
run;

proc phreg data=hoel_2;
        model ftime*d(0)=group_tl group_rcs group_oth;
        strata stratum;
        baseline out=ciData covariates=cov survival=surv / nomean method=ch;
run;

%CumInc(ciData, stratum, ftime,surv);
proc print data=data;
run;

PROC EXPORT DATA= WORK.DATA
        OUTFILE= "E:\Competing Risks\Paper\Rosthoj\Rosthoj_gf.xls"
        DBMS=EXCEL REPLACE;
        SHEET="Rosthoj_gf";
RUN;
```

Note: The cumulative incidence estimates were plotted in Stata/SE 9.2

```
label var tl_gf "Germ-free"
label var tl_conv "Conventional"
label var rcs_gf "Germ-free"
label var rcs_conv "Conventional"
label var oth_gf "Germ-free"
label var oth_conv "Conventional"

scatter tl_conv tl_gf ftime, m(i i) c(J J) clpattern(solid dash) ylabel(0(.1).5)
title(Thymic Lymphoma) ytitle(Cumulative Incidence) xtitle(Days) sort

scatter rcs_conv rcs_gf ftime, m(i i) c(J J) clpattern(solid dash) ylabel(0(.1).5)
title(Reticulum Cell Sarcoma) ytitle(Cumulative Incidence) xtitle(Days) sort

scatter oth_conv oth_gf ftime, m(i i) c(J J) clpattern(solid dash) ylabel(0(.1).5)
title(Other Causes) ytitle(Cumulative Incidence) xtitle(Days) sort
```

APPENDIX B

COMPUTER CODE FOR AML EXAMPLE

B.1: R 2.0.1 CODE FOR TABLE 5.1 (GRAY)

```
aml <- read.table ("E:\\Competing Risks\\AML and CML\\AML edit 07_30_07
v2.txt", row.names=NULL, header=T)

ci<-cuminc(aml$time, aml$event, aml$group1)

ci
```

B.2: R 2.0.1 CODE FOR TABLE 5.2 (FINE & GRAY)

```
crr1<-crr(aml$time, aml$event, aml$group1, failcode=1)
crr2<-crr(aml$time, aml$event, aml$group1, failcode=2)
crr3<-crr(aml$time, aml$event, aml$group1, failcode=3)

crr1
crr2
crr3
```

B.3: R 2.0.1 CODE FOR TABLE 5.3 (COX)

```
n<-nrow(aml)
alive<-relapse<-died<-rep(0,n)

for (i in 1:n) {       alive[i]<-ifelse(aml$event[i]==1 , 1, 0)
            relapse[i]<-ifelse(aml$event[i]==2, 1, 0)
            died[i]<-ifelse(aml$event[i]==3, 1, 0)}

cph1<-summary(coxph(Surv(aml$time,alive)~aml$group1))
cph2<-summary(coxph(Surv(aml$time,relapse)~aml$group1))
cph3<-summary(coxph(Surv(aml$time,died)~aml$group1))

cph1
cph2
cph3
```

B.4: R 2.0.1 CODE FOR TABLE 5.4 (LOG-RANK)

```
LR1<-survdiff(Surv(aml$time,alive)~aml$group1, rho=0)
LR2<-survdiff(Surv(aml$time,relapse)~aml$group1, rho=0)
LR3<-survdiff(Surv(aml$time,died)~aml$group1, rho=0)
```

LR1
LR2
LR3

B.5: R 2.0.1 CODE FOR FIGURES 5.1 – 5.3 (GRAY)

```
max(aml$time)

plot.cuminc(ci[1:2], main="Alive/No Relapse", curvlab=c("1st Remission", "2+ Remissions"), xlim=c(0,22),ylim=c(0,1),xlab="Years", ylab="Cumulative Incidence", lwd=1)

plot.cuminc(ci[3:4], main="Relapsed", curvlab=c("1st Remission", "2+ Remissions"), xlim=c(0,22),ylim=c(0,1),xlab="Years", ylab="Cumulative Incidence", lwd=1)

plot.cuminc(ci[5:6], main="Died", curvlab=c("1st Remission", "2+ Remissions"), xlim=c(0,22),ylim=c(0,1),xlab="Years", ylab="Cumulative Incidence", lwd=1)
```

B.6: R 2.0.1 CODE FOR FIGURES 5.4 – 5.6 (FINE & GRAY)

```
crr1.p<-predict(crr1,rbind(1,2))
crr2.p<-predict(crr2,rbind(1,2))
crr3.p<-predict(crr3,rbind(1,2))

plot(crr1.p, lty=1:2, color=1, main="Alive/No Relapse", xlim=c(0,22), ylim=c(0,1), xlab="Years", ylab="Cumulative Incidence", lwd=1)
legend(0,1,c("1st Remission", "2+ Remissions"),lty=c(1:2))

plot(crr2.p, lty=1:2, color=1, main="Relapsed", xlim=c(0,22), ylim=c(0,1), xlab="Years", ylab="Cumulative Incidence", lwd=1)
legend(0,1,c("1st Remission", "2+ Remissions"),lty=c(1:2))

plot(crr3.p, lty=1:2, color=1, main="Died", xlim=c(0,22), ylim=c(0,1), xlab="Years", ylab="Cumulative Incidence", lwd=1)
legend(0,1,c("1st Remission", "2+ Remissions"),lty=c(1:2))
```

APPENDIX C

SIMULATION PROGRAMMING

C.1: INDEPENDENT BIVARIATE EXPONENTIAL

Note: The following generates data from a specified bivariate distribution.

```
biv.exp.ind<-function(n.trt, n.cont, beta1, beta2, lambda1, lambda2, pcens)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1,n.trt), rep(0,n.cont))

u1<-runif(n, 0, 1)
u2<-runif(n, 0, 1)
uC<-runif(n, 0, 1)

x1<-(-(1/lambda1)* exp(-beta1*z)*log(1-u1))
x2<-(-(1/lambda2)* exp(-beta2*z)*log(1-u2))

lambda.min.trt<-(lambda1*exp(beta1)+lambda2*exp(beta2))
lambda.min.cont<-(lambda1+lambda2)

lambdaC.trt<-(lambda.min.trt*pcens)/(1-pcens)
lambdaC.cont<-(lambda.min.cont*pcens)/(1-pcens)

C<-rep(0,n)

for (i in 1:n.trt)
{C[i]<-(-(1/lambdaC.trt)*log(1-uC[i]))}

for (i in (n.trt+1):n)
        {C[i]<-(-(1/lambdaC.cont)*log(1-uC[i]))}

t<-pmin(x1, x2, C)

delta1<-delta2<-deltaC<-rep(0,n)

for (i in 1:n) {delta1[i]<-ifelse(t[i]==x1[i] , 1, 0)
        delta2[i]<-ifelse(t[i]==x2[i], 2, 0)
        deltaC[i]<-ifelse(t[i]==C[i], 3, 0)}

cause<-delta1+delta2+deltaC

cause1<-cause2<-censored<-rep(0,n)
```

```
for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        censored[i]<-ifelse(cause[i]==3, 1, 0)}

data<-matrix(c(id, t, cause, z, cause1, cause2, censored, x1, x2, C), n, 10)

return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests when censoring is present.

```
power.tests.exp.ind<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1, lambda2,
pcens)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)

{ temp<-biv.exp.ind(n.trt, n.cont, beta1, beta2, lambda1, lambda2, pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
censored<-temp[, 7]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.cens[i]<-summary(coxph(Surv(time,censored)~z))$logtest[3]
signif.cox.cens[i]<-ifelse(pval.cox.cens[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.lrank.cens[i]<-(1-pchisq(survdiff(Surv(time,censored)~z, rho=0)$chisq, 1))
signif.lrank.cens[i]<-ifelse(pval.lrank.cens[i]<alpha, 1, 0)
```

```
pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.cens[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.cens[i]<-ifelse(pval.gray.cens[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.cens[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.cens[i]<-ifelse(pval.fg.cens[i]<alpha, 1, 0)

}
        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.cens<-mean(signif.cox.cens)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.cens<-mean(signif.lrank.cens)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.cens<-mean(signif.gray.cens)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.cens<-mean(signif.fg.cens)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.cens, power.lrank.c1,
power.lrank.c2, power.lrank.cens, power.gray.c1, power.gray.c2, power.gray.cens,
power.fg.c1, power.fg.c2, power.fg.cens), 3, 4)

return(power)
}
```

Note: The following calculates the power associated with four hypothesis tests when no censoring is present.

```
power.tests.exp.ind.nocens<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1, lambda2, pcens)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-rep(0,nsim)

for (i in 1:nsim)

{ temp<-biv.exp.ind(n.trt, n.cont, beta1, beta2, lambda1, lambda2, pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
censored<-temp[, 7]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)
```

```
pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)

power<-matrix(c(power.cox.c1, power.cox.c2, power.lrank.c1, power.lrank.c2,
power.gray.c1, power.gray.c2, power.fg.c1, power.fg.c2), 2, 4)

return(power)}
```

C.2: SARMANOV BIVARIATE EXPONENTIAL

Note: The following generates data from a specified bivariate distribution.

```
biv.exp.sarm<-function(n.trt, n.cont, rho, beta1, beta2, lambda1, lambda2, pcens1,
pcens2)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1, n.trt), rep(0, n.cont))

u1<-runif(n, 0, 1)
u2<-runif(n, 0, 1)
uC1<-runif(n, 0, 1)
uC2<-runif(n, 0, 1)

lambda1.star<-lambda1*exp(beta1*z)
lambda2.star<-lambda2*exp(beta2*z)

w<-(rho*(1+lambda1.star)^2*(1+lambda2.star)^2)/(lambda1.star*lambda2.star)

x1<-(-(1/lambda1.star)*log(1-u1))

x2<-rep(0,n)

k<-(exp(-x1)-(lambda1.star/(1+lambda1.star)))*(lambda2.star*w)/(1+lambda2.star)

for (i in 1:n)
{
f<-function (x2) 1-u2[i]-exp(-lambda2.star[i]*x2)*(1+k[i]*(exp(-x2)-1))
x2[i]<- uniroot(f, low=0, up=10000, tol=1e-5)$root}

lambda1.trt<-(lambda1*exp(beta1))
lambda1.cont<-lambda1

lambda2.trt<-(lambda2*exp(beta2))
lambda2.cont<-lambda2

lambdaC1.trt<-(lambda1.trt*pcens1)/(1-pcens1)
lambdaC1.cont<-(lambda1.cont*pcens1)/(1-pcens1)

lambdaC2.trt<-(lambda2.trt*pcens2)/(1-pcens2)
lambdaC2.cont<-(lambda2.cont*pcens2)/(1-pcens2)

C1<-C2<-rep(0,n)
```

```
for (i in 1:n.trt)
{C1[i]<-(-(1/lambdaC1.trt)*log(1-uC1[i]))}

for (i in (n.trt+1):n)
{C1[i]<-(-(1/lambdaC1.cont)*log(1-uC1[i]))}

for (i in 1:n.trt)
{C2[i]<-(-(1/lambdaC2.trt)*log(1-uC2[i]))}

for (i in (n.trt+1):n)
{C2[i]<-(-(1/lambdaC2.cont)*log(1-uC2[i]))}

t<-pmin(x1, x2, C1, C2)

delta1<-delta2<-deltaC1<-deltaC2<-deltaC<-rep(0,n)

for (i in 1:n) {
        delta1[i]<-ifelse(t[i]==x1[i] , 1, 0)
        delta2[i]<-ifelse(t[i]==x2[i], 2, 0)
        deltaC1[i]<-ifelse(t[i]==C1[i], 3, 0)
        deltaC2[i]<-ifelse(t[i]==C2[i], 4, 0)
        deltaC[i]<-ifelse(t[i]==C1[i] | t[i]==C2[i], 3, 0)}

        cause.1<-delta1+delta2+deltaC1+deltaC2
        cause<-delta1+delta2+deltaC

cause1<-cause2<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        censored[i]<-ifelse(cause[i]==3, 1, 0)}

data<-matrix(c(id, t, cause, cause.1, z, cause1, cause2, censored, x1, x2, C1, C2), n, 12)
return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests when censoring is present.

```
power.tests.exp.sarm<-function(nsim, alpha, n.trt, n.cont, rho, beta1, beta2, lambda1,
lambda2, pcens1, pcens2)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{ temp<-biv.exp.sarm(n.trt, n.cont, rho, beta1, beta2, lambda1, lambda2, pcens1, pcens2)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 5]
cause1<-temp[, 6]
cause2<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.cens[i]<-summary(coxph(Surv(time,censored)~z))$logtest[3]
signif.cox.cens[i]<-ifelse(pval.cox.cens[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.lrank.cens[i]<-(1-pchisq(survdiff(Surv(time,censored)~z, rho=0)$chisq, 1))
signif.lrank.cens[i]<-ifelse(pval.lrank.cens[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)
```

```
pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.cens[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.cens[i]<-ifelse(pval.gray.cens[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.cens[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.cens[i]<-ifelse(pval.fg.cens[i]<alpha, 1, 0)

}
        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.cens<-mean(signif.cox.cens)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.cens<-mean(signif.lrank.cens)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.cens<-mean(signif.gray.cens)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.cens<-mean(signif.fg.cens)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.cens, power.lrank.c1,
power.lrank.c2, power.lrank.cens, power.gray.c1, power.gray.c2, power.gray.cens,
power.fg.c1, power.fg.c2, power.fg.cens), 3, 4)

return(power)

}
```

Note: The following calculates the power associated with four hypothesis tests when no censoring is present.

```
power.tests.exp.sarm.nocens<-function(nsim, alpha, n.trt, n.cont, rho, beta1, beta2,
lambda1, lambda2, pcens1, pcens2)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{ temp<-biv.exp.sarm(n.trt, n.cont, rho, beta1, beta2, lambda1, lambda2, pcens1, pcens2)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 5]
cause1<-temp[, 6]
cause2<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)
```

```
pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)

power<-matrix(c(power.cox.c1, power.cox.c2, power.lrank.c1, power.lrank.c2,
power.gray.c1, power.gray.c2, power.fg.c1, power.fg.c2), 2, 4)

return(power)}
```

C.3: MARSHALL-OLKIN BIVARIATE EXPONENTIAL

Note: The following generates data from a specified bivariate distribution.

```
biv.exp.mo<-function(n.trt, n.cont, beta1, beta2, beta12, lambda1, lambda2, lambda12,
pcens)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1,n.trt), rep(0,n.cont))

u1<-runif(n, 0, 1)
u2<-runif(n, 0, 1)
u12<-runif(n, 0, 1)

y1<-(-(1/lambda1)* exp(-beta1*z)*log(1-u1))
y2<-(-(1/lambda2)* exp(-beta2*z)*log(1-u2))
y12<-(-(1/lambda12)* exp(-beta12*z)*log(1-u12))

x1<-pmin(y1, y12)
x2<-pmin(y2, y12)

lambda.min.trt<-(lambda1*exp(beta1)+lambda2*exp(beta2)+lambda12*exp(beta12))
lambda.min.cont<-(lambda1+lambda2+lambda12)

lambdaC.trt<-(lambda.min.trt*pcens)/(1-pcens)
lambdaC.cont<-(lambda.min.cont*pcens)/(1-pcens)

uC<-runif(n, 0, 1)
C<-rep(0,n)

for (i in 1:n.trt)
{C[i]<-(-(1/lambdaC.trt)*log(1-uC[i]))}

for (i in (n.trt+1):n)
{C[i]<-(-(1/lambdaC.cont)*log(1-uC[i]))}

t<-pmin(x1, x2, C)

delta1<-delta2<-delta3<-deltaC<-rep(0,n)

for (i in 1:n) {delta1[i]<-ifelse((t[i]==x1[i] & x1[i]!=x2[i]) , 1, 0)
        delta2[i]<-ifelse((t[i]==x2[i] & x1[i]!=x2[i]), 2, 0)
        delta3[i]<-ifelse((t[i]==x1[i] & x1[i]==x2[i]), 3, 0)
        deltaC[i]<-ifelse(t[i]==C[i], 4, 0)}
```

```
cause<-delta1+delta2+delta3+deltaC

cause1<-cause2<-cause3<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        cause3[i]<-ifelse(cause[i]==3, 1, 0)
        censored[i]<-ifelse(cause[i]==4, 1, 0)}

data<-matrix(c(id, t, cause, z, cause1, cause2, cause3, censored, x1, x2, C, y1, y2, y12), n,
14)

return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests when censoring is present.

```
power.tests.exp.mo<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, beta12, lambda1,
lambda2, lambda12, pcens)

{
pval.cox.c1<-pval.cox.c2<-pval.cox.c3<-pval.cox.cens<-rep(0,nsim)
signif.cox.c1<-signif.cox.c2<-signif.cox.c3<-signif.cox.cens<-rep(0,nsim)
pval.lrank.c1<-pval.lrank.c2<-pval.lrank.c3<-pval.lrank.cens<-rep(0,nsim)
signif.lrank.c1<-signif.lrank.c2<-signif.lrank.c3<-signif.lrank.cens<-rep(0,nsim)
pval.gray.c1<-pval.gray.c2<-pval.gray.c3<-pval.gray.cens<-rep(0,nsim)
signif.gray.c1<-signif.gray.c2<-signif.gray.c3<-signif.gray.cens<-rep(0,nsim)
pval.fg.c1<-pval.fg.c2<-pval.fg.c3<-pval.fg.cens<-rep(0,nsim)
signif.fg.c1<-signif.fg.c2<-signif.fg.c3<-signif.fg.cens<-rep(0,nsim)

for (i in 1:nsim)
{ temp<-biv.exp.mo(n.trt, n.cont, beta1, beta2, beta12, lambda1, lambda2, lambda12,
pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
cause3<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.c3[i]<-summary(coxph(Surv(time,cause3)~z))$logtest[3]
signif.cox.c3[i]<-ifelse(pval.cox.c3[i]<alpha, 1, 0)

pval.cox.cens[i]<-summary(coxph(Surv(time,censored)~z))$logtest[3]
signif.cox.cens[i]<-ifelse(pval.cox.cens[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)
```

```
pval.lrank.c3[i]<-(1-pchisq(survdiff(Surv(time,cause3)~z, rho=0)$chisq, 1))
signif.lrank.c3[i]<-ifelse(pval.lrank.c3[i]<alpha, 1, 0)

pval.lrank.cens[i]<-(1-pchisq(survdiff(Surv(time,censored)~z, rho=0)$chisq, 1))
signif.lrank.cens[i]<-ifelse(pval.lrank.cens[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.c3[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.c3[i]<-ifelse(pval.gray.c3[i]<alpha, 1, 0)

pval.gray.cens[i]<-cuminc(time, cause, z)$Tests[4,2]
signif.gray.cens[i]<-ifelse(pval.gray.cens[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.c3[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.c3[i]<-ifelse(pval.fg.c3[i]<alpha, 1, 0)

pval.fg.cens[i]<-(1-pchisq((crr(time,cause,z, failcode=4)$coef/sqrt(crr(time,cause,z,
failcode=4)$var))^2,1))
signif.fg.cens[i]<-ifelse(pval.fg.cens[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.c3<-mean(signif.cox.c3)
        power.cox.cens<-mean(signif.cox.cens)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.c3<-mean(signif.lrank.c3)
```

```
        power.lrank.cens<-mean(signif.lrank.cens)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.c3<-mean(signif.gray.c3)
        power.gray.cens<-mean(signif.gray.cens)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.c3<-mean(signif.fg.c3)
        power.fg.cens<-mean(signif.fg.cens)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.c3, power.cox.cens,
power.lrank.c1, power.lrank.c2, power.lrank.c3, power.lrank.cens, power.gray.c1,
power.gray.c2, power.gray.c3, power.gray.cens, power.fg.c1, power.fg.c2, power.fg.c3,
power.fg.cens), 4, 4)

return(power)

}
```

Note: The following calculates the power associated with four hypothesis tests when no censoring is present.

```
power.tests.exp.mo.nocens<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, beta12,
lambda1, lambda2, lambda12, pcens)

{       pval.cox.c1<-pval.cox.c2<-pval.cox.c3<-pval.cox.cens<-rep(0,nsim)
        signif.cox.c1<-signif.cox.c2<-signif.cox.c3<-signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.c3<-pval.lrank.cens<-rep(0,nsim)
        signif.lrank.c1<-signif.lrank.c2<-signif.lrank.c3<-signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.c3<-pval.gray.cens<-rep(0,nsim)
        signif.gray.c1<-signif.gray.c2<-signif.gray.c3<-signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.c3<-pval.fg.cens<-rep(0,nsim)
        signif.fg.c1<-signif.fg.c2<-signif.fg.c3<-signif.fg.cens<-rep(0,nsim)

for (i in 1:nsim)
{ temp<-biv.exp.mo(n.trt, n.cont, beta1, beta2, beta12, lambda1, lambda2, lambda12,
pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
cause3<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.c3[i]<-summary(coxph(Surv(time,cause3)~z))$logtest[3]
signif.cox.c3[i]<-ifelse(pval.cox.c3[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.lrank.c3[i]<-(1-pchisq(survdiff(Surv(time,cause3)~z, rho=0)$chisq, 1))
signif.lrank.c3[i]<-ifelse(pval.lrank.c3[i]<alpha, 1, 0)
```

```
pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.c3[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.c3[i]<-ifelse(pval.gray.c3[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.c3[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.c3[i]<-ifelse(pval.fg.c3[i]<alpha, 1, 0)

}
        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.c3<-mean(signif.cox.c3)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.c3<-mean(signif.lrank.c3)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.c3<-mean(signif.gray.c3)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.c3<-mean(signif.fg.c3)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.c3, power.lrank.c1,
power.lrank.c2, power.lrank.c3, power.gray.c1, power.gray.c2, power.gray.c3,
power.fg.c1, power.fg.c2, power.fg.c3), 3, 4)

return(power)
}
```

C.4: CAI-PRENTICE BIVARIATE EXPONENTIAL

Note: The following generates data from a specified bivariate distribution.

```
biv.exp.cai<-function(n.trt, n.cont, tau, beta1, beta2, lambda1, lambda2, pcens1, pcens2)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1, n.trt), rep(0, n.cont))

u1<-runif(n, 0, 1)
u2<-runif(n, 0, 1)
uC1<-runif(n, 0, 1)
uC2<-runif(n, 0, 1)

theta<-(1-tau)/(2*tau)
a<-(1-u2)^((-theta)^(-1))

x1<-(1/lambda1)*theta*log((1-a)+a*(1-u1)^(-(1+theta)^(-1)))*exp(-beta1*z)
x2<-(-(1/lambda2)*log(1-u2)*exp(-beta2*z))

lambda1.trt<-(lambda1*exp(beta1))
lambda1.cont<-lambda1

lambda2.trt<-(lambda2*exp(beta2))
lambda2.cont<-lambda2

lambdaC1.trt<-(lambda1.trt*pcens1)/(1-pcens1)
lambdaC1.cont<-(lambda1.cont*pcens1)/(1-pcens1)

lambdaC2.trt<-(lambda2.trt*pcens2)/(1-pcens2)
lambdaC2.cont<-(lambda2.cont*pcens2)/(1-pcens2)

C1<-C2<-rep(0,n)

for (i in 1:n.trt)
{C1[i]<-(-(1/lambdaC1.trt)*log(1-uC1[i]))}

for (i in (n.trt+1):n)
{C1[i]<-(-(1/lambdaC1.cont)*log(1-uC1[i]))}

for (i in 1:n.trt)
{C2[i]<-(-(1/lambdaC2.trt)*log(1-uC2[i]))}
```

```
for (i in (n.trt+1):n)
{C2[i]<-(-(1/lambdaC2.cont)*log(1-uC2[i]))}

t<-pmin(x1, x2, C1, C2)

delta1<-delta2<-deltaC1<-deltaC2<-deltaC<-rep(0,n)

for (i in 1:n) {
        delta1[i]<-ifelse(t[i]==x1[i] , 1, 0)
        delta2[i]<-ifelse(t[i]==x2[i], 2, 0)
        deltaC1[i]<-ifelse(t[i]==C1[i], 3, 0)
        deltaC2[i]<-ifelse(t[i]==C2[i], 4, 0)
        deltaC[i]<-ifelse(t[i]==C1[i] | t[i]==C2[i], 3, 0)}

cause.1<-delta1+delta2+deltaC1+deltaC2
cause<-delta1+delta2+deltaC

cause1<-cause2<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        censored[i]<-ifelse(cause[i]==3, 1, 0)}

data<-matrix(c(id, t, cause, cause.1, z, cause1, cause2, censored, x1, x2, C1, C2), n, 12)
return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests when censoring is present.

```
power.tests.exp.cai<-function(nsim, alpha, n.trt, n.cont, tau, beta1, beta2, lambda1,
lambda2, pcens1, pcens2)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.exp.cai(n.trt, n.cont, tau, beta1, beta2, lambda1, lambda2, pcens1, pcens2)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 5]
cause1<-temp[, 6]
cause2<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.cens[i]<-summary(coxph(Surv(time,censored)~z))$logtest[3]
signif.cox.cens[i]<-ifelse(pval.cox.cens[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.lrank.cens[i]<-(1-pchisq(survdiff(Surv(time,censored)~z, rho=0)$chisq, 1))
signif.lrank.cens[i]<-ifelse(pval.lrank.cens[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)
```

```
pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.cens[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.cens[i]<-ifelse(pval.gray.cens[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.cens[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.cens[i]<-ifelse(pval.fg.cens[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.cens<-mean(signif.cox.cens)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.cens<-mean(signif.lrank.cens)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.cens<-mean(signif.gray.cens)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.cens<-mean(signif.fg.cens)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.cens, power.lrank.c1,
power.lrank.c2, power.lrank.cens, power.gray.c1, power.gray.c2, power.gray.cens,
power.fg.c1, power.fg.c2, power.fg.cens), 3, 4)

return(power)
}
```

Note: The following calculates the power associated with four hypothesis tests when no censoring is present.

```
power.tests.exp.cai.nocens<-function(nsim, alpha, n.trt, n.cont, tau, beta1, beta2,
lambda1, lambda2, pcens1, pcens2)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{ temp<-biv.exp.cai(n.trt, n.cont, tau, beta1, beta2, lambda1, lambda2, pcens1, pcens2)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 5]
cause1<-temp[, 6]
cause2<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)
```

```
pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)

power<-matrix(c(power.cox.c1, power.cox.c2, power.lrank.c1, power.lrank.c2,
power.gray.c1, power.gray.c2, power.fg.c1, power.fg.c2), 2, 4)

return(power)}
```

C.5: BIVARIATE EXPONENTIAL WITH A GAMMA FRAILTY

Note: The following generates data from a specified bivariate distribution.

```
biv.exp.frailty<-function(n.trt, n.cont, beta1, beta2, lambda1, lambda2, theta, pcens)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1,n.trt), rep(0,n.cont))

u1<-runif(n, 0, 1)
u2<-runif(n, 0, 1)

w<-rep(0,n)
for (i in 1:n)
{u<-runif(theta, 0, 1)
  v<-(-(1/theta)*log(1-u))
  w[i]<-sum(v)}

y1<-(-(1/lambda1)* exp(-beta1*z)*log(1-u1))
y2<-(-(1/lambda2)* exp(-beta2*z)*log(1-u2))

x1<-w*y1
x2<-w*y2

C<-rep(0,n)
uC<-runif(n, 0, 1)

lambdaC<-rep(0,n)

lambda.min.trt<- rep(0,n.trt)

for (i in 1:n.trt)
{lambda.min.trt[i]<-(1/w[i])*(lambda1*exp(beta1)+lambda2*exp(beta2))
  lambdaC[i]<-(lambda.min.trt[i]*pcens)/(1-pcens)
  C[i]<-(-(1/lambdaC[i])*log(1-uC[i]))}

lambda.min.cont<- rep(0,n.cont)

for (i in (n.trt+1):n)
{lambda.min.cont[i]<-(1/w[i])*(lambda1+lambda2)
  lambdaC[i]<-(lambda.min.cont[i]*pcens)/(1-pcens)
  C[i]<-(-(1/lambdaC[i])*log(1-uC[i]))}

t<-pmin(x1, x2, C)
```

```
delta1<-delta2<-deltaC<-rep(0,n)

for (i in 1:n) {delta1[i]<-ifelse(t[i]==x1[i] , 1, 0)
        delta2[i]<-ifelse(t[i]==x2[i], 2, 0)
        deltaC[i]<-ifelse(t[i]==C[i], 3, 0)}

        cause<-delta1+delta2+deltaC

cause1<-cause2<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        censored[i]<-ifelse(cause[i]==3, 1, 0)}

data<-matrix(c(id, t, cause, z, cause1, cause2, censored, x1, x2, C), n, 10)

return(data)
}
```

Note: The following calculates the power associated with four hypothesis tests when censoring is present.

```
power.tests.exp.frailty<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1,
lambda2, theta, pcens)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.exp.frailty(n.trt, n.cont, beta1, beta2, lambda1, lambda2, theta, pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
censored<-temp[, 7]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.cens[i]<-summary(coxph(Surv(time,censored)~z))$logtest[3]
signif.cox.cens[i]<-ifelse(pval.cox.cens[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.lrank.cens[i]<-(1-pchisq(survdiff(Surv(time,censored)~z, rho=0)$chisq, 1))
signif.lrank.cens[i]<-ifelse(pval.lrank.cens[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)
```

```
pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.cens[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.cens[i]<-ifelse(pval.gray.cens[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.cens[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.cens[i]<-ifelse(pval.fg.cens[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.cens<-mean(signif.cox.cens)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.cens<-mean(signif.lrank.cens)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.cens<-mean(signif.gray.cens)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.cens<-mean(signif.fg.cens)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.cens, power.lrank.c1,
power.lrank.c2, power.lrank.cens, power.gray.c1, power.gray.c2, power.gray.cens,
power.fg.c1, power.fg.c2, power.fg.cens), 3, 4)

return(power)
}
```

Note: The following calculates the power associated with four hypothesis tests when no censoring is present.

```
power.tests.exp.frailty.nocens<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1,
lambda2, theta, pcens)
{
        pval.cox.c1<-pval.cox.c2<- signif.cox.c1<-signif.cox.c2<- rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<- signif.lrank.c1<-signif.lrank.c2<- rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<- signif.gray.c1<-signif.gray.c2<- rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<- signif.fg.c1<-signif.fg.c2<- rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.exp.frailty(n.trt, n.cont, beta1, beta2, lambda1, lambda2, theta, pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
censored<-temp[, 7]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
```

```
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)

power<-matrix(c(power.cox.c1, power.cox.c2, power.lrank.c1, power.lrank.c2,
power.gray.c1, power.gray.c2, power.fg.c1, power.fg.c2), 2, 4)

return(power)}
```

C.6: INDEPENDENT BIVARIATE WEIBULL

Note: The following generates data from a specified bivariate distribution.

```
biv.weib.ind<-function(n.trt, n.cont, beta1, beta2, lambda1, lambda2, alpha1, alpha2,
alphaC, pcens)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1,n.trt), rep(0,n.cont))

u1<-runif(n, 0, 1)
u2<-runif(n, 0, 1)
uC<-runif(n, 0, 1)

x1<-(-(1/lambda1)* exp(-beta1*z)*log(1-u1))^(1/alpha1)
x2<-(-(1/lambda2)* exp(-beta2*z)*log(1-u2))^(1/alpha2)

lambda.min.trt<-(lambda1*exp(beta1)+lambda2*exp(beta2))
lambda.min.cont<-(lambda1+lambda2)

lambdaC.trt<-(lambda.min.trt*pcens)/(1-pcens)
lambdaC.cont<-(lambda.min.cont*pcens)/(1-pcens)

C<-rep(0,n)

for (i in 1:n.trt)
{C[i]<-(-(1/lambdaC.trt)*log(1-uC[i]))^(1/alphaC)}

for (i in (n.trt+1):n)
{C[i]<-(-(1/lambdaC.cont)*log(1-uC[i]))^(1/alphaC)}

t<-pmin(x1, x2, C)

delta1<-delta2<-deltaC<-rep(0,n)

for (i in 1:n) {delta1[i]<-ifelse(t[i]==x1[i] , 1, 0)
        delta2[i]<-ifelse(t[i]==x2[i], 2, 0)
        deltaC[i]<-ifelse(t[i]==C[i], 3, 0)}

        cause<-delta1+delta2+deltaC

cause1<-cause2<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
```

```
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        censored[i]<-ifelse(cause[i]==3, 1, 0)}

data<-matrix(c(id, t, cause, z, cause1, cause2, censored, x1, x2, C), n, 10)

return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests when censoring is present.

```
power.tests.weib.ind<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1, lambda2,
alpha1, alpha2, alphaC, pcens)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.weib.ind(n.trt, n.cont, beta1, beta2, lambda1, lambda2, alpha1, alpha2,
alphaC, pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
censored<-temp[, 7]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.cens[i]<-summary(coxph(Surv(time,censored)~z))$logtest[3]
signif.cox.cens[i]<-ifelse(pval.cox.cens[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.lrank.cens[i]<-(1-pchisq(survdiff(Surv(time,censored)~z, rho=0)$chisq, 1))
signif.lrank.cens[i]<-ifelse(pval.lrank.cens[i]<alpha, 1, 0)
```

```
pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.cens[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.cens[i]<-ifelse(pval.gray.cens[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.cens[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.cens[i]<-ifelse(pval.fg.cens[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.cens<-mean(signif.cox.cens)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.cens<-mean(signif.lrank.cens)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.cens<-mean(signif.gray.cens)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.cens<-mean(signif.fg.cens)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.cens, power.lrank.c1,
power.lrank.c2, power.lrank.cens, power.gray.c1, power.gray.c2, power.gray.cens,
power.fg.c1, power.fg.c2, power.fg.cens), 3, 4)

return(power)
}
```

Note: The following calculates the power associated with four hypothesis tests when no censoring is present.

```
power.tests.weib.ind.nocens<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1,
lambda2, alpha1, alpha2, alphaC, pcens)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.weib.ind(n.trt, n.cont, beta1, beta2, lambda1, lambda2, alpha1, alpha2,
alphaC, pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
censored<-temp[, 7]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)
```

```
pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)

power<-matrix(c(power.cox.c1, power.cox.c2, power.lrank.c1, power.lrank.c2,
power.gray.c1, power.gray.c2, power.fg.c1, power.fg.c2), 2, 4)

return(power)

}
```

C.7: SARMANOV BIVARIATE WEIBULL

Note: The following generates data from a specified bivariate distribution.

```
biv.weib.sarm<-function(n.trt, n.cont, rho, beta1, beta2, lambda1, lambda2, alpha1,
alpha2, alphaC1, alphaC2, pcens1, pcens2)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1, n.trt), rep(0, n.cont))

u1<-runif(n, 0, 1)
u2<-runif(n, 0, 1)
uC1<-runif(n, 0, 1)
uC2<-runif(n, 0, 1)

lambda1.star<-lambda1*exp(beta1*z)
lambda2.star<-lambda2*exp(beta2*z)

w<-(rho*(1+lambda1.star)^2*(1+lambda2.star)^2)/(lambda1.star*lambda2.star)

y1<-(-(1/lambda1.star)*log(1-u1))

y2<-rep(0,n)

k<-(exp(-y1)-(lambda1.star/(1+lambda1.star)))*(lambda2.star*w)/(1+lambda2.star)

for (i in 1:n)
{ f<-function (y2) 1-u2[i]-exp(-lambda2.star[i]*y2)*(1+k[i]*(exp(-y2)-1))
  y2[i]<- uniroot(f, low=0, up=10000, tol=1e-5)$root}

x1<-y1^(1/alpha1)
x2<-y2^(1/alpha2)

lambda1.trt<-(lambda1*exp(beta1))
lambda1.cont<-lambda1

lambda2.trt<-(lambda2*exp(beta2))
lambda2.cont<-lambda2

lambdaC1.trt<-(lambda1.trt*pcens1)/(1-pcens1)
lambdaC1.cont<-(lambda1.cont*pcens1)/(1-pcens1)

lambdaC2.trt<-(lambda2.trt*pcens2)/(1-pcens2)
lambdaC2.cont<-(lambda2.cont*pcens2)/(1-pcens2)
```

```
C1<-C2<-rep(0,n)

for (i in 1:n.trt)
{C1[i]<-(-(1/lambdaC1.trt)*log(1-uC1[i]))^(1/alphaC1)}

for (i in (n.trt+1):n)
{C1[i]<-(-(1/lambdaC1.cont)*log(1-uC1[i]))^(1/alphaC1)}

for (i in 1:n.trt)
{C2[i]<-(-(1/lambdaC2.trt)*log(1-uC2[i]))^(1/alphaC2)}

for (i in (n.trt+1):n)
{C2[i]<-(-(1/lambdaC2.cont)*log(1-uC2[i]))^(1/alphaC2)}

t<-pmin(x1, x2, C1, C2)

delta1<-delta2<-deltaC1<-deltaC2<-deltaC<-rep(0,n)

for (i in 1:n) {
        delta1[i]<-ifelse(t[i]==x1[i] , 1, 0)
        delta2[i]<-ifelse(t[i]==x2[i], 2, 0)
        deltaC1[i]<-ifelse(t[i]==C1[i], 3, 0)
        deltaC2[i]<-ifelse(t[i]==C2[i], 4, 0)
        deltaC[i]<-ifelse(t[i]==C1[i] | t[i]==C2[i], 3, 0)}

        cause.1<-delta1+delta2+deltaC1+deltaC2
        cause<-delta1+delta2+deltaC

cause1<-cause2<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        censored[i]<-ifelse(cause[i]==3, 1, 0)}

data<-matrix(c(id, t, cause, cause.1, z, cause1, cause2, censored, x1, x2, C1, C2), n, 12)
return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests when censoring is present.

```
power.tests.weib.sarm<-function(nsim, alpha, n.trt, n.cont, rho, beta1, beta2, lambda1,
lambda2, alpha1, alpha2, alphaC1, alphaC2, pcens1, pcens2)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{ temp<-biv.weib.sarm(n.trt, n.cont, rho, beta1, beta2, lambda1, lambda2, alpha1, alpha2,
alphaC1, alphaC2, pcens1, pcens2)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 5]
cause1<-temp[, 6]
cause2<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.cens[i]<-summary(coxph(Surv(time,censored)~z))$logtest[3]
signif.cox.cens[i]<-ifelse(pval.cox.cens[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.lrank.cens[i]<-(1-pchisq(survdiff(Surv(time,censored)~z, rho=0)$chisq, 1))
signif.lrank.cens[i]<-ifelse(pval.lrank.cens[i]<alpha, 1, 0)
```

```
pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.cens[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.cens[i]<-ifelse(pval.gray.cens[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.cens[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.cens[i]<-ifelse(pval.fg.cens[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.cens<-mean(signif.cox.cens)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.cens<-mean(signif.lrank.cens)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.cens<-mean(signif.gray.cens)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.cens<-mean(signif.fg.cens)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.cens, power.lrank.c1,
power.lrank.c2, power.lrank.cens, power.gray.c1, power.gray.c2, power.gray.cens,
power.fg.c1, power.fg.c2, power.fg.cens), 3, 4)

return(power)
}
```

Note: The following calculates the power associated with four hypothesis tests when no censoring is present.

```
power.tests.weib.sarm.nocens<-function(nsim, alpha, n.trt, n.cont, rho, beta1, beta2,
lambda1, lambda2, alpha1, alpha2, alphaC1, alphaC2, pcens1, pcens2)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{ temp<-biv.weib.sarm(n.trt, n.cont, rho, beta1, beta2, lambda1, lambda2, alpha1, alpha2,
alphaC1, alphaC2, pcens1, pcens2)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 5]
cause1<-temp[, 6]
cause2<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)
```

```
pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)
}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)

power<-matrix(c(power.cox.c1, power.cox.c2, power.lrank.c1, power.lrank.c2,
power.gray.c1, power.gray.c2, power.fg.c1, power.fg.c2), 2, 4)

return(power)

}
```

C.8: MARSHALL-OLKIN BIVARIATE WEIBULL

Note: The following generates data from a specified bivariate distribution.

```
biv.weib.mo<-function(n.trt, n.cont, beta1, beta2, beta12, lambda1, lambda2, lambda12,
alpha1, alpha2, alpha12, alphaC, pcens)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1,n.trt), rep(0,n.cont))

u1<-runif(n, 0, 1)
u2<-runif(n, 0, 1)
u12<-runif(n, 0, 1)

y1<-(-(1/lambda1)* exp(-beta1*z)*log(1-u1))^(1/alpha1)
y2<-(-(1/lambda2)* exp(-beta2*z)*log(1-u2))^(1/alpha2)
y12<-(-(1/lambda12)* exp(-beta12*z)*log(1-u12))^(1/alpha12)

x1<-pmin(y1, y12)
x2<-pmin(y2, y12)

lambda.min.trt<-(lambda1*exp(beta1)+lambda2*exp(beta2)+lambda12*exp(beta12))
lambda.min.cont<-(lambda1+lambda2+lambda12)

lambdaC.trt<-(lambda.min.trt*pcens)/(1-pcens)
lambdaC.cont<-(lambda.min.cont*pcens)/(1-pcens)

uC<-runif(n, 0, 1)
C<-rep(0,n)

for (i in 1:n.trt)
{C[i]<-(-(1/lambdaC.trt)*log(1-uC[i]))^(1/alphaC)}

for (i in (n.trt+1):n)
{C[i]<-(-(1/lambdaC.cont)*log(1-uC[i]))^(1/alphaC)}

t<-pmin(x1, x2, C)

delta1<-delta2<-delta3<-deltaC<-rep(0,n)

for (i in 1:n) {delta1[i]<-ifelse((t[i]==x1[i] & x1[i]!=x2[i]) , 1, 0)
        delta2[i]<-ifelse((t[i]==x2[i] & x1[i]!=x2[i]), 2, 0)
        delta3[i]<-ifelse((t[i]==x1[i] & x1[i]==x2[i]), 3, 0)
        deltaC[i]<-ifelse(t[i]==C[i], 4, 0)}
```

```
cause<-delta1+delta2+delta3+deltaC

cause1<-cause2<-cause3<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        cause3[i]<-ifelse(cause[i]==3, 1, 0)
        censored[i]<-ifelse(cause[i]==4, 1, 0)}


data<-matrix(c(id, t, cause, z, cause1, cause2, cause3, censored, x1, x2, C, y1, y2, y12), n,
14)

return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests when censoring is present.

```
power.tests.weib.mo<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, beta12, lambda1,
lambda2, lambda12, alpha1, alpha2, alpha12, alphaC, pcens)

{
        pval.cox.c1<-pval.cox.c2<-pval.cox.c3<-pval.cox.cens<-rep(0,nsim)
signif.cox.c1<-signif.cox.c2<-signif.cox.c3<-signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.c3<-pval.lrank.cens<-rep(0,nsim)
signif.lrank.c1<-signif.lrank.c2<-signif.lrank.c3<-signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.c3<-pval.gray.cens<-rep(0,nsim)
signif.gray.c1<-signif.gray.c2<-signif.gray.c3<-signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.c3<-pval.fg.cens<-rep(0,nsim)
signif.fg.c1<-signif.fg.c2<-signif.fg.c3<-signif.fg.cens<-rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.weib.mo(n.trt,n.cont,beta1,beta2,beta12,lambda1,lambda2,lambda12,alpha1,
alpha2,alpha12,alphaC, pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
cause3<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.c3[i]<-summary(coxph(Surv(time,cause3)~z))$logtest[3]
signif.cox.c3[i]<-ifelse(pval.cox.c3[i]<alpha, 1, 0)

pval.cox.cens[i]<-summary(coxph(Surv(time,censored)~z))$logtest[3]
signif.cox.cens[i]<-ifelse(pval.cox.cens[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)
```

```
pval.lrank.c3[i]<-(1-pchisq(survdiff(Surv(time,cause3)~z, rho=0)$chisq, 1))
signif.lrank.c3[i]<-ifelse(pval.lrank.c3[i]<alpha, 1, 0)

pval.lrank.cens[i]<-(1-pchisq(survdiff(Surv(time,censored)~z, rho=0)$chisq, 1))
signif.lrank.cens[i]<-ifelse(pval.lrank.cens[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.c3[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.c3[i]<-ifelse(pval.gray.c3[i]<alpha, 1, 0)

pval.gray.cens[i]<-cuminc(time, cause, z)$Tests[4,2]
signif.gray.cens[i]<-ifelse(pval.gray.cens[i]<alpha, 1, 0)


pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.c3[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.c3[i]<-ifelse(pval.fg.c3[i]<alpha, 1, 0)

pval.fg.cens[i]<-(1-pchisq((crr(time,cause,z, failcode=4)$coef/sqrt(crr(time,cause,z,
failcode=4)$var))^2,1))
signif.fg.cens[i]<-ifelse(pval.fg.cens[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.c3<-mean(signif.cox.c3)
        power.cox.cens<-mean(signif.cox.cens)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
```

```
        power.lrank.c3<-mean(signif.lrank.c3)
        power.lrank.cens<-mean(signif.lrank.cens)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.c3<-mean(signif.gray.c3)
        power.gray.cens<-mean(signif.gray.cens)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.c3<-mean(signif.fg.c3)
        power.fg.cens<-mean(signif.fg.cens)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.c3, power.cox.cens,
power.lrank.c1, power.lrank.c2, power.lrank.c3, power.lrank.cens, power.gray.c1,
power.gray.c2, power.gray.c3, power.gray.cens, power.fg.c1, power.fg.c2, power.fg.c3,
power.fg.cens), 4, 4)

return(power)

}
```

Note: The following calculates the power associated with four hypothesis tests when no censoring is present.

```
power.tests.weib.mo.nocens<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, beta12,
lambda1, lambda2, lambda12, alpha1, alpha2, alpha12, alphaC, pcens)

{       pval.cox.c1<-pval.cox.c2<-pval.cox.c3<-pval.cox.cens<-rep(0,nsim)
signif.cox.c1<-signif.cox.c2<-signif.cox.c3<-signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.c3<-pval.lrank.cens<-rep(0,nsim)
signif.lrank.c1<-signif.lrank.c2<-signif.lrank.c3<-signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.c3<-pval.gray.cens<-rep(0,nsim)
signif.gray.c1<-signif.gray.c2<-signif.gray.c3<-signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.c3<-pval.fg.cens<-rep(0,nsim)
signif.fg.c1<-signif.fg.c2<-signif.fg.c3<-signif.fg.cens<-rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.weib.mo(n.trt,n.cont,beta1,beta2,beta12,lambda1,lambda2,lambda12,alpha1,
alpha2,alpha12,alphaC, pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
cause3<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.c3[i]<-summary(coxph(Surv(time,cause3)~z))$logtest[3]
signif.cox.c3[i]<-ifelse(pval.cox.c3[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.lrank.c3[i]<-(1-pchisq(survdiff(Surv(time,cause3)~z, rho=0)$chisq, 1))
signif.lrank.c3[i]<-ifelse(pval.lrank.c3[i]<alpha, 1, 0)
```

```
pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.c3[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.c3[i]<-ifelse(pval.gray.c3[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.c3[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.c3[i]<-ifelse(pval.fg.c3[i]<alpha, 1, 0)
}
        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.c3<-mean(signif.cox.c3)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.c3<-mean(signif.lrank.c3)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.c3<-mean(signif.gray.c3)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.c3<-mean(signif.fg.c3)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.c3, power.lrank.c1,
power.lrank.c2, power.lrank.c3, power.gray.c1, power.gray.c2, power.gray.c3,
power.fg.c1, power.fg.c2, power.fg.c3), 3, 4)

return(power)
}
```

C.9: CAI-PRENTICE BIVARIATE WEIBULL

Note: The following generates data from a specified bivariate distribution.

```
biv.weib.cai<-function(n.trt, n.cont, tau, beta1, beta2, lambda1, lambda2, alpha1, alpha2,
alphaC1, alphaC2, pcens1, pcens2)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1, n.trt), rep(0, n.cont))

u1<-runif(n, 0, 1)
u2<-runif(n, 0, 1)
uC1<-runif(n, 0, 1)
uC2<-runif(n, 0, 1)

theta<-(1-tau)/(2*tau)
a<-(1-u2)^((-theta)^(-1))

x1<-((1/lambda1)*theta*log((1-a)+a*(1-u1)^(-(1+theta)^(-1)))*exp(-
beta1*z))^(1/alpha1)
x2<-(-(1/lambda2)*log(1-u2)*exp(-beta2*z))^(1/alpha2)

lambda1.trt<-(lambda1*exp(beta1))
lambda1.cont<-lambda1

lambda2.trt<-(lambda2*exp(beta2))
lambda2.cont<-lambda2

lambdaC1.trt<-(lambda1.trt*pcens1)/(1-pcens1)
lambdaC1.cont<-(lambda1.cont*pcens1)/(1-pcens1)

lambdaC2.trt<-(lambda2.trt*pcens2)/(1-pcens2)
lambdaC2.cont<-(lambda2.cont*pcens2)/(1-pcens2)

C1<-C2<-rep(0,n)

for (i in 1:n.trt)
{C1[i]<-(-(1/lambdaC1.trt)*log(1-uC1[i]))^(1/alphaC1)}

for (i in (n.trt+1):n)
{C1[i]<-(-(1/lambdaC1.cont)*log(1-uC1[i]))^(1/alphaC1)}

for (i in 1:n.trt)
{C2[i]<-(-(1/lambdaC2.trt)*log(1-uC2[i]))^(1/alphaC2)}
```

```
for (i in (n.trt+1):n)
{C2[i]<-(-(1/lambdaC2.cont)*log(1-uC2[i]))^(1/alphaC2)}

t<-pmin(x1, x2, C1, C2)

delta1<-delta2<-deltaC1<-deltaC2<-deltaC<-rep(0,n)

for (i in 1:n) {
        delta1[i]<-ifelse(t[i]==x1[i] , 1, 0)
        delta2[i]<-ifelse(t[i]==x2[i], 2, 0)
        deltaC1[i]<-ifelse(t[i]==C1[i], 3, 0)
        deltaC2[i]<-ifelse(t[i]==C2[i], 4, 0)
        deltaC[i]<-ifelse(t[i]==C1[i] | t[i]==C2[i], 3, 0)}

        cause.1<-delta1+delta2+deltaC1+deltaC2
        cause<-delta1+delta2+deltaC

cause1<-cause2<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        censored[i]<-ifelse(cause[i]==3, 1, 0)}

data<-matrix(c(id, t, cause, cause.1, z, cause1, cause2, censored, x1, x2, C1, C2), n, 12)

return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests when censoring is present.

```
power.tests.weib.cai<-function(nsim, alpha, n.trt, n.cont, tau, beta1, beta2, lambda1,
lambda2, alpha1, alpha2, alphaC1, alphaC2, pcens1, pcens2)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.weib.cai(n.trt,n.cont,tau, beta1, beta2, lambda1, lambda2, alpha1, alpha2,
alphaC1, alphaC2,pcens1, pcens2)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 5]
cause1<-temp[, 6]
cause2<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.cens[i]<-summary(coxph(Surv(time,censored)~z))$logtest[3]
signif.cox.cens[i]<-ifelse(pval.cox.cens[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.lrank.cens[i]<-(1-pchisq(survdiff(Surv(time,censored)~z, rho=0)$chisq, 1))
signif.lrank.cens[i]<-ifelse(pval.lrank.cens[i]<alpha, 1, 0)
```

```
pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.cens[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.cens[i]<-ifelse(pval.gray.cens[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.cens[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.cens[i]<-ifelse(pval.fg.cens[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.cens<-mean(signif.cox.cens)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.cens<-mean(signif.lrank.cens)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.cens<-mean(signif.gray.cens)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.cens<-mean(signif.fg.cens)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.cens, power.lrank.c1,
power.lrank.c2, power.lrank.cens, power.gray.c1, power.gray.c2, power.gray.cens,
power.fg.c1, power.fg.c2, power.fg.cens), 3, 4)

return(power)
}
```

Note: The following calculates the power associated with four hypothesis tests when no censoring is present.

```
power.tests.weib.cai.nocens<-function(nsim, alpha, n.trt, n.cont, tau, beta1, beta2,
lambda1, lambda2, alpha1, alpha2, alphaC1, alphaC2, pcens1, pcens2)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.weib.cai(n.trt,n.cont,tau, beta1, beta2, lambda1, lambda2, alpha1, alpha2,
alphaC1, alphaC2,pcens1, pcens2)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 5]
cause1<-temp[, 6]
cause2<-temp[, 7]
censored<-temp[, 8]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)
```

```
pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)

power<-matrix(c(power.cox.c1, power.cox.c2, power.lrank.c1, power.lrank.c2,
power.gray.c1, power.gray.c2, power.fg.c1, power.fg.c2), 2, 4)

return(power)

}
```

C.10: BIVARIATE WEIBULL WITH A GAMMA FRAILTY

Note: The following generates data from a specified bivariate distribution.

```
biv.weib.frailty<-function(n.trt, n.cont, beta1, beta2, lambda1, lambda2, theta, alpha1,
alpha2, alphaC, pcens)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1,n.trt), rep(0,n.cont))

u1<-runif(n, 0, 1)
u2<-runif(n, 0, 1)

w<-rep(0,n)
for (i in 1:n)
{u<-runif(theta, 0, 1)
  v<-(-(1/theta)*log(1-u))
  w[i]<-sum(v)}

y1<-(-(1/lambda1)* exp(-beta1*z)*log(1-u1))^(1/alpha1)
y2<-(-(1/lambda2)* exp(-beta2*z)*log(1-u2))^(1/alpha2)

x1<-w*y1
x2<-w*y2

C<-rep(0,n)
uC<-runif(n, 0, 1)

lambdaC<-rep(0,n)

lambda.min.trt<- rep(0,n.trt)
for (i in 1:n.trt)
{lambda.min.trt[i]<-(1/w[i])*(lambda1*exp(beta1)+lambda2*exp(beta2))
  lambdaC[i]<-(lambda.min.trt[i]*pcens)/(1-pcens)
  C[i]<-(-(1/lambdaC[i])*log(1-uC[i]))^(1/alphaC)}

lambda.min.cont<- rep(0,n.cont)
for (i in (n.trt+1):n)
{lambda.min.cont[i]<-(1/w[i])*(lambda1+lambda2)
  lambdaC[i]<-(lambda.min.cont[i]*pcens)/(1-pcens)
  C[i]<-(-(1/lambdaC[i])*log(1-uC[i]))^(1/alphaC)}

t<-pmin(x1, x2, C)
```

```
delta1<-delta2<-deltaC<-rep(0,n)

for (i in 1:n) {delta1[i]<-ifelse(t[i]==x1[i] , 1, 0)
        delta2[i]<-ifelse(t[i]==x2[i], 2, 0)
        deltaC[i]<-ifelse(t[i]==C[i], 3, 0)}

cause<-delta1+delta2+deltaC

cause1<-cause2<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        censored[i]<-ifelse(cause[i]==3, 1, 0)}

data<-matrix(c(id, t, cause, z, cause1, cause2, censored, x1, x2, C), n, 10)

return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests when censoring is present.

```
power.tests.weib.frailty<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1,
lambda2, theta, alpha1, alpha2, alphaC, pcens)
{
        pval.cox.c1<-pval.cox.c2<-pval.cox.cens<-signif.cox.c1<-signif.cox.c2<-
signif.cox.cens<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-pval.lrank.cens<-signif.lrank.c1<-signif.lrank.c2<-
signif.lrank.cens<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-pval.gray.cens<-signif.gray.c1<-signif.gray.c2<-
signif.gray.cens<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-pval.fg.cens<-signif.fg.c1<-signif.fg.c2<-signif.fg.cens<-
rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.weib.frailty(n.trt, n.cont, beta1, beta2, lambda1, lambda2, theta, alpha1,
alpha2, alphaC, pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
censored<-temp[, 7]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.cox.cens[i]<-summary(coxph(Surv(time,censored)~z))$logtest[3]
signif.cox.cens[i]<-ifelse(pval.cox.cens[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.lrank.cens[i]<-(1-pchisq(survdiff(Surv(time,censored)~z, rho=0)$chisq, 1))
signif.lrank.cens[i]<-ifelse(pval.lrank.cens[i]<alpha, 1, 0)
```

```
pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.gray.cens[i]<-cuminc(time, cause, z)$Tests[3,2]
signif.gray.cens[i]<-ifelse(pval.gray.cens[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)

pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)

pval.fg.cens[i]<-(1-pchisq((crr(time,cause,z, failcode=3)$coef/sqrt(crr(time,cause,z,
failcode=3)$var))^2,1))
signif.fg.cens[i]<-ifelse(pval.fg.cens[i]<alpha, 1, 0)

}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)
        power.cox.cens<-mean(signif.cox.cens)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)
        power.lrank.cens<-mean(signif.lrank.cens)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)
        power.gray.cens<-mean(signif.gray.cens)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)
        power.fg.cens<-mean(signif.fg.cens)

power<-matrix(c(power.cox.c1, power.cox.c2, power.cox.cens, power.lrank.c1,
power.lrank.c2, power.lrank.cens, power.gray.c1, power.gray.c2, power.gray.cens,
power.fg.c1, power.fg.c2, power.fg.cens), 3, 4)

return(power)
}
```

Note: The following calculates the power associated with four hypothesis tests when no censoring is present.

```
power.tests.weib.frailty.nocens<-function(nsim, alpha, n.trt, n.cont, beta1, beta2,
lambda1, lambda2, theta, alpha1, alpha2, alphaC, pcens)
{
        pval.cox.c1<-pval.cox.c2<- signif.cox.c1<-signif.cox.c2<- rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<- signif.lrank.c1<-signif.lrank.c2<- rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<- signif.gray.c1<-signif.gray.c2<- rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<- signif.fg.c1<-signif.fg.c2<- rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.weib.frailty(n.trt, n.cont, beta1, beta2, lambda1, lambda2, theta, alpha1,
alpha2, alphaC, pcens)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
censored<-temp[, 7]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)

pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)
```

```
pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)
}

        power.cox.c1<-mean(signif.cox.c1)
        power.cox.c2<-mean(signif.cox.c2)

        power.lrank.c1<-mean(signif.lrank.c1)
        power.lrank.c2<-mean(signif.lrank.c2)

        power.gray.c1<-mean(signif.gray.c1)
        power.gray.c2<-mean(signif.gray.c2)

        power.fg.c1<-mean(signif.fg.c1)
        power.fg.c2<-mean(signif.fg.c2)

power<-matrix(c(power.cox.c1, power.cox.c2, power.lrank.c1, power.lrank.c2,
power.gray.c1, power.gray.c2, power.fg.c1, power.fg.c2), 2, 4)

return(power)

}
```

C.11: BIVARIATE NORMAL

Note: The following generates data from a specified bivariate distribution.

```
biv.norm<-function(n.trt, n.cont, beta1, beta2, mu1, mu2, var1, var2, lambda1, lambda2,
rho, pcens1, pcens2)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1, n.trt), rep(0, n.cont))

mu1.trt<- mu1*exp(beta1)
mu2.trt<- mu2*exp(beta2)
mu.trt<-c(mu1.trt, mu2.trt)
mu1.cont<-mu1
mu2.cont<-mu2
mu.cont<-c(mu1.cont, mu2.cont)
var1.trt<-var1*exp(lambda1)
var1.cont<-var1
var2.trt<-var2*exp(lambda2)
var2.cont<-var2
cov.trt<-rho*sqrt(var1.trt*var2.trt)
cov.cont<-rho*sqrt(var1.cont*var2.cont)
sigma.trt<-matrix(c(var1.trt, cov.trt, cov.trt, var2.trt),2,2)
sigma.cont<-matrix(c(var1.cont, cov.cont, cov.cont, var2.cont),2,2)

x1.trt<-rnorm(n=n.trt, mu1.trt, sd=sqrt(var1.trt))
x2.given.x1.trt<-rnorm(n=n.trt, mu2.trt+rho*sqrt(var2.trt/var1.trt)*(x1.trt-mu1.trt),
sd=sqrt(var2.trt*(1-rho^2)))
x.trt<-matrix(0, n.trt, 2)
x.trt[, 1]<-x1.trt
x.trt[, 2]<-x2.given.x1.trt

x1.cont<-rnorm(n=n.cont, mu1.cont, sd=sqrt(var1.cont))
x2.given.x1.cont<-rnorm(n=n.cont, mu2.cont+rho*sqrt(var2.cont/var1.cont)*(x1.cont-
mu1.cont), sd=sqrt(var2.cont*(1-rho^2)))
x.cont<-matrix(0, n.cont, 2)
x.cont[, 1]<-x1.cont
x.cont[, 2]<-x2.given.x1.cont

x<-rbind(x.trt, x.cont)

x1<-x[, 1]
x2<-x[, 2]
```

```
c1<-c2<-rep(0,n)

for (i in 1:n.trt)
{c1[i]<-2*mu1.trt-qnorm(pcens1, mu1.trt, sqrt(var1.trt))}

for (i in (n.trt+1):n)
{c1[i]<- 2*mu1.cont-qnorm(pcens1, mu1.cont, sqrt(var1.cont))}

for (i in 1:n.trt)
{c2[i]<-2*mu2.trt-qnorm(pcens2, mu2.trt, sqrt(var2.trt))}

for (i in (n.trt+1):n)
{c2[i]<- 2*mu2.cont-qnorm(pcens2, mu2.cont, sqrt(var2.cont))}

t<-pmax(0,pmin(x1, x2, c1, c2))

delta1<-delta2<-deltac1<-deltac2<-deltac<-rep(0,n)

for (i in 1:n) {
        delta1[i]<-ifelse(t[i]==x1[i] , 1, 0)
        delta2[i]<-ifelse(t[i]==x2[i], 2, 0)
        deltac1[i]<-ifelse(t[i]==c1[i], 3, 0)
        deltac2[i]<-ifelse(t[i]==c2[i], 4, 0)
        deltac[i]<-ifelse(t[i]==c1[i] | t[i]==c2[i], 3, 0)}

        cause.1<-delta1+delta2+deltac1+deltac2
        cause<-delta1+delta2+deltac

cause1<-cause2<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        censored[i]<-ifelse(cause[i]==3, 1, 0)}

data<-matrix(c(id, t, cause, z, cause1, cause2, censored, x1, x2, c1, c2), n, 11)
return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests.

```
power.tests.norm<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, mu1, mu2, var1, var2,
lambda1, lambda2, rho, pcens1, pcens2)
{
        pval.cox.c1<-pval.cox.c2<-signif.cox.c1<-signif.cox.c2<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-signif.lrank.c1<-signif.lrank.c2<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-signif.gray.c1<-signif.gray.c2<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-signif.fg.c1<-signif.fg.c2<-rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.norm(n.trt, n.cont, beta1, beta2, mu1, mu2, var1, var2, lambda1, lambda2,
rho, pcens1, pcens2)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
censored<-temp[, 7]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)


pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)
```

```
pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)
}

            power.cox.c1<-mean(signif.cox.c1)
            power.cox.c2<-mean(signif.cox.c2)

            power.lrank.c1<-mean(signif.lrank.c1)
            power.lrank.c2<-mean(signif.lrank.c2)

            power.gray.c1<-mean(signif.gray.c1)
            power.gray.c2<-mean(signif.gray.c2)

            power.fg.c1<-mean(signif.fg.c1)
            power.fg.c2<-mean(signif.fg.c2)

power<-matrix(c(power.cox.c1, power.cox.c2, power.lrank.c1, power.lrank.c2,
power.gray.c1, power.gray.c2, power.fg.c1, power.fg.c2), 2, 4)

return(power)

}
```

C.12: BIVARIATE LOG-NORMAL

Note: The following generates data from a specified bivariate distribution.

```
biv.lnorm<-function(n.trt, n.cont, beta1, beta2, mu1, mu2, var1, var2, lambda1, lambda2,
rho, pcens1, pcens2)
{
n<-n.trt+n.cont
id<-c(1:n)
z<-c(rep(1, n.trt), rep(0, n.cont))

mu1.trt<- mu1*exp(beta1)
mu2.trt<- mu2*exp(beta2)
mu.trt<-c(mu1.trt, mu2.trt)
mu1.cont<-mu1
mu2.cont<-mu2
mu.cont<-c(mu1.cont, mu2.cont)
var1.trt<-var1*exp(lambda1)
var1.cont<-var1
var2.trt<-var2*exp(lambda2)
var2.cont<-var2
cov.trt<-rho*sqrt(var1.trt*var2.trt)
cov.cont<-rho*sqrt(var1.cont*var2.cont)
sigma.trt<-matrix(c(var1.trt, cov.trt, cov.trt, var2.trt),2,2)
sigma.cont<-matrix(c(var1.cont, cov.cont, cov.cont, var2.cont),2,2)

y1.trt<-rnorm(n=n.trt, mu1.trt, sd=sqrt(var1.trt))
y2.given.y1.trt<-rnorm(n=n.trt, mu2.trt+rho*sqrt(var2.trt/var1.trt)*(y1.trt-mu1.trt),
sd=sqrt(var2.trt*(1-rho^2)))
y.trt<-matrix(0, n.trt, 2)
y.trt[, 1]<-y1.trt
y.trt[, 2]<-y2.given.y1.trt

y1.cont<-rnorm(n=n.cont, mu1.cont, sd=sqrt(var1.cont))
y2.given.y1.cont<-rnorm(n=n.cont, mu2.cont+rho*sqrt(var2.cont/var1.cont)*(y1.cont-
mu1.cont), sd=sqrt(var2.cont*(1-rho^2)))
y.cont<-matrix(0, n.cont, 2)
y.cont[, 1]<-y1.cont
y.cont[, 2]<-y2.given.y1.cont

x.trt<-exp(y.trt)
x.cont<-exp(y.cont)

x<-rbind(x.trt, x.cont)
```

```
x1<-x[, 1]
x2<-x[, 2]

c1<-c2<-rep(0,n)

for (i in 1:n.trt)
{c1[i]<-exp(2*mu1.trt-qnorm(pcens1, mu1.trt, sqrt(var1.trt)))}

for (i in (n.trt+1):n)
{c1[i]<- exp(2*mu1.cont-qnorm(pcens1, mu1.cont, sqrt(var1.cont)))}

for (i in 1:n.trt)
{c2[i]<-exp(2*mu2.trt-qnorm(pcens2, mu2.trt, sqrt(var2.trt)))}

for (i in (n.trt+1):n)
{c2[i]<- exp(2*mu2.cont-qnorm(pcens2, mu2.cont, sqrt(var2.cont)))}

t<-pmax(0,pmin(x1, x2, c1, c2))

delta1<-delta2<-deltac1<-deltac2<-deltac<-rep(0,n)

for (i in 1:n) {
        delta1[i]<-ifelse(t[i]==x1[i] , 1, 0)
        delta2[i]<-ifelse(t[i]==x2[i], 2, 0)
        deltac1[i]<-ifelse(t[i]==c1[i], 3, 0)
        deltac2[i]<-ifelse(t[i]==c2[i], 4, 0)
        deltac[i]<-ifelse(t[i]==c1[i] | t[i]==c2[i], 3, 0)}

        cause.1<-delta1+delta2+deltac1+deltac2
        cause<-delta1+delta2+deltac

cause1<-cause2<-censored<-rep(0,n)

for (i in 1:n) {cause1[i]<-ifelse(cause[i]==1 , 1, 0)
        cause2[i]<-ifelse(cause[i]==2, 1, 0)
        censored[i]<-ifelse(cause[i]==3, 1, 0)}

data<-matrix(c(id, t, cause, z, cause1, cause2, censored, x1, x2, c1, c2), n, 11)

return(data)

}
```

Note: The following calculates the power associated with four hypothesis tests.

```
power.tests.lnorm<-function(nsim, alpha, n.trt, n.cont, beta1, beta2, mu1, mu2, var1,
var2, lambda1, lambda2, rho, pcens1, pcens2)
{
        pval.cox.c1<-pval.cox.c2<-signif.cox.c1<-signif.cox.c2<-rep(0,nsim)
        pval.lrank.c1<-pval.lrank.c2<-signif.lrank.c1<-signif.lrank.c2<-rep(0,nsim)
        pval.gray.c1<-pval.gray.c2<-signif.gray.c1<-signif.gray.c2<-rep(0,nsim)
        pval.fg.c1<-pval.fg.c2<-signif.fg.c1<-signif.fg.c2<-rep(0,nsim)

for (i in 1:nsim)
{temp<-biv.lnorm(n.trt, n.cont, beta1, beta2, mu1, mu2, var1, var2, lambda1, lambda2,
rho, pcens1, pcens2)

time<-temp[, 2]
cause<-temp[, 3]
z<-temp[, 4]
cause1<-temp[, 5]
cause2<-temp[, 6]
censored<-temp[, 7]

pval.cox.c1[i]<-summary(coxph(Surv(time,cause1)~z))$logtest[3]
signif.cox.c1[i]<-ifelse(pval.cox.c1[i]<alpha, 1, 0)

pval.cox.c2[i]<-summary(coxph(Surv(time,cause2)~z))$logtest[3]
signif.cox.c2[i]<-ifelse(pval.cox.c2[i]<alpha, 1, 0)

pval.lrank.c1[i]<-(1-pchisq(survdiff(Surv(time,cause1)~z, rho=0)$chisq, 1))
signif.lrank.c1[i]<-ifelse(pval.lrank.c1[i]<alpha, 1, 0)

pval.lrank.c2[i]<-(1-pchisq(survdiff(Surv(time,cause2)~z, rho=0)$chisq, 1))
signif.lrank.c2[i]<-ifelse(pval.lrank.c2[i]<alpha, 1, 0)


pval.gray.c1[i]<-cuminc(time, cause, z)$Tests[1,2]
signif.gray.c1[i]<-ifelse(pval.gray.c1[i]<alpha, 1, 0)

pval.gray.c2[i]<-cuminc(time, cause, z)$Tests[2,2]
signif.gray.c2[i]<-ifelse(pval.gray.c2[i]<alpha, 1, 0)

pval.fg.c1[i]<-(1-pchisq((crr(time,cause,z, failcode=1)$coef/sqrt(crr(time,cause,z,
failcode=1)$var))^2,1))
signif.fg.c1[i]<-ifelse(pval.fg.c1[i]<alpha, 1, 0)
```

```
pval.fg.c2[i]<-(1-pchisq((crr(time,cause,z, failcode=2)$coef/sqrt(crr(time,cause,z,
failcode=2)$var))^2,1))
signif.fg.c2[i]<-ifelse(pval.fg.c2[i]<alpha, 1, 0)
}

            power.cox.c1<-mean(signif.cox.c1)
            power.cox.c2<-mean(signif.cox.c2)

            power.lrank.c1<-mean(signif.lrank.c1)
            power.lrank.c2<-mean(signif.lrank.c2)

            power.gray.c1<-mean(signif.gray.c1)
            power.gray.c2<-mean(signif.gray.c2)

            power.fg.c1<-mean(signif.fg.c1)
            power.fg.c2<-mean(signif.fg.c2)

power<-matrix(c(power.cox.c1, power.cox.c2, power.lrank.c1, power.lrank.c2,
power.gray.c1, power.gray.c2, power.fg.c1, power.fg.c2), 2, 4)

return(power)

}
```

APPENDIX D

COMPUTER CODE FOR DETERMINING SIMULATION PARAMETERS

D.1: INDEPENDENT BIVARIATE EXPONENTIAL

Note: This also works for the Sarmanov Bivariate Exponential, the Cai-Prentice Bivariate Exponential, and the Bivariate Exponential with a Gamma Frailty.

```
cmean1<-10
p1<-c(0, 0.20, 0.60, -.20)
tmean1<-cmean1*(1+p1)

cmean2<-10
p2<-c(0, 0.20, 0.60, -.20)
tmean2<-cmean2*(1+p2)

lambda1<-1/cmean1
lambda2<-1/cmean2

beta1<-log(1/(cmean1*(1+p1)*lambda1))
beta2<-log(1/(cmean2*(1+p2)*lambda2))

lambda1
lambda2
beta1
beta2
```

D.2: MARSHALL-OLKIN BIVARIATE EXPONENTIAL

```
px1x2<-0.20

cmean1<-10
p1<-c(0,0,.6,.6,.6,0,.6)
tmean1<-cmean1*(1+p1)

cmean2<-10
p2<-c(.2,.6,0,.2,.6,-.2,-.2)
tmean2<-cmean2*(1+p2)

lambda12<-((1/cmean1)+(1/cmean2))*(px1x2/(1+px1x2))
lambda1<-(1/cmean1)-lambda12
lambda2<-(1/cmean2)-lambda12

beta12<-log((1/(lambda12*(1+1/px1x2)))*(1/(cmean1*(1+p1))+1/(cmean2*(1+p2))))
beta1<-log(1/(cmean1*(1+p1)*lambda1)- (lambda12/lambda1)*exp(beta12))
beta2<-log(1/(cmean2*(1+p2)*lambda2)- (lambda12/lambda2)*exp(beta12))
```

```
lambda12
lambda1
lambda2
beta12
beta1
beta2
```

D.3: INDEPENDENT BIVARIATE WEIBULL

Note: This also works for the Sarmanov Bivariate Weibull, the Cai-Prentice Bivariate Weibull, and the Bivariate Weibull with a Gamma Frailty.

```
alpha<-2

cmean1<-10
p1<-c(0, 0.20, 0.60, -.20)
tmean1<-cmean1*(1+p1)

cmean2<-10
p2<-c(0, 0.20, 0.60, -.20)
tmean2<-cmean2*(1+p2)

lambda1<-(cmean1/gamma(1+1/alpha))^(-alpha)
lambda2<-(cmean2/gamma(1+1/alpha))^(-alpha)

beta1<-log((1/lambda1)*(cmean1*(1+p1)/gamma(1+1/alpha))^(-alpha))
beta2<-log((1/lambda2)*(cmean2*(1+p2)/gamma(1+1/alpha))^(-alpha))

lambda1
lambda2
beta1
beta2
```

D.4: MARSHALL-OLKIN BIVARIATE WEIBULL

```
alpha<-2

px1x2<-0.20

cmean1<-10
p1<-c(0,0,.6,.6,.6,0,.6)
tmean1<-cmean1*(1+p1)
```

```
cmean2<-10
p2<-c(.2,.6,0,.2,.6,-.2,-.2)
tmean2<-cmean2*(1+p2)

lambda12<-((cmean1/gamma(1+1/alpha))^(-alpha)+(cmean2/gamma(1+1/alpha))^(-
alpha))*(px1x2/(1+px1x2))
lambda1<-(cmean1/gamma(1+1/alpha))^(-alpha)-lambda12
lambda2<-(cmean2/gamma(1+1/alpha))^(-alpha)-lambda12

beta12<-log((1/(lambda12*(1+1/px1x2)))*((cmean1*(1+p1)/gamma(1+1/alpha))^(-
alpha)+(cmean2*(1+p2)/gamma(1+1/alpha))^(-alpha)))
beta1<-log((1/lambda1)*(cmean1*(1+p1)/gamma(1+1/alpha))^(-alpha) -
(lambda12/lambda1)*exp(beta12))
beta2<-log((1/lambda2)*(cmean2*(1+p2)/gamma(1+1/alpha))^(-alpha) -
(lambda12/lambda2)*exp(beta12))

lambda12
lambda1
lambda2
beta12
beta1
beta2
```

D.5: BIVARIATE NORMAL

```
cmean1<-10
p1<-c(0, 0.05, 0.15, -0.05)
tmean1<-cmean1*(1+p1)

cmean2<-10
p2<-c(0, 0.05, 0.15, -0.05)
tmean2<-cmean2*(1+p2)

beta1<-log(1+p1)
beta2<-log(1+p2)

beta1
beta2
```

D.6: BIVARIATE LOG-NORMAL

```
cmean1<-10
cvar1<-sqrt(10)
p1<-c(0, 0.05, 0.15, -.05)
tmean1<-cmean1*(1+p1)

cmean2<-10
cvar2<-sqrt(10)
p2<-c(0, 0.05, 0.15, -.05)
tmean2<-cmean2*(1+p2)

mu1<-2*log(cmean1)-(1/2)*log(cvar1+cmean1^2)
mu2<-2*log(cmean2)-(1/2)*log(cvar2+cmean2^2)

var1<-log(cvar1+cmean1^2)-2*log(cmean1)
var2<-log(cvar2+cmean2^2)-2*log(cmean2)

beta1<-log((1/mu1)*(log(cmean1*(1+p1))-(1/2)*var1))
beta2<-log((1/mu2)*(log(cmean2*(1+p2))-(1/2)*var2))

mu1
mu2
var1
var2
beta1
beta2
```

APPENDIX E

COMPUTER CODE FOR PRODUCING THE POWER TABLES

E.1: INDEPENDENT BIVARIATE EXPONENTIAL

power.tests.exp.ind(nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1, lambda2, pcens)

No treatment effect for X_1, 20% treatment effect for X_2.

power.tests.exp.ind.nocens(1000, .05, 50, 50, 0,-0.1823216, .1, .1, 0)
power.tests.exp.ind(1000, .05, 50, 50, 0, -0.1823216, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 50, 50, 0, -0.1823216, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 50, 50, 0, -0.1823216, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 100, 100, 0,-0.1823216, .1, .1, 0)
power.tests.exp.ind(1000, .05, 100, 100, 0, -0.1823216, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 100, 100, 0, -0.1823216, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 100, 100, 0, -0.1823216, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 200, 200, 0, -0.1823216, .1, .1, 0)
power.tests.exp.ind(1000, .05, 200, 200, 0, -0.1823216, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 200, 200, 0, -0.1823216, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 200, 200, 0, -0.1823216, .1, .1, 0.80)

No treatment effect for X_1, 60% treatment effect for X_2.

power.tests.exp.ind.nocens(1000, .05, 50, 50, 0, -0.4700036, .1, .1, 0)
power.tests.exp.ind(1000, .05, 50, 50, 0, -0.4700036, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 50, 50, 0, -0.4700036, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 50, 50, 0, -0.4700036, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 100, 100, 0, -0.4700036, .1, .1, 0)
power.tests.exp.ind(1000, .05, 100, 100, 0, -0.4700036, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 100, 100, 0, -0.4700036, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 100, 100, 0, -0.4700036, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 200, 200, 0, -0.4700036, .1, .1, 0)
power.tests.exp.ind(1000, .05, 200, 200, 0, -0.4700036, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 200, 200, 0, -0.4700036, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 200, 200, 0, -0.4700036, .1, .1, 0.80)

60% treatment effect for X_1, no treatment effect for X_2.

(Use the results for X_2 from the above programs)

60% treatment effect for X_1, 20% treatment effect for X_2.

power.tests.exp.ind.nocens(1000, .05, 50, 50, -0.4700036, -0.1823216, .1, .1, 0)
power.tests.exp.ind(1000, .05, 50, 50, -0.4700036, -0.1823216, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 50, 50, -0.4700036, -0.1823216, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 50, 50, -0.4700036, -0.1823216, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 100, 100, -0.4700036, -0.1823216, .1, .1, 0)
power.tests.exp.ind(1000, .05, 100, 100, -0.4700036, -0.1823216, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 100, 100, -0.4700036, -0.1823216, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 100, 100, -0.4700036, -0.1823216, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 200, 200, -0.4700036, -0.1823216, .1, .1, 0)
power.tests.exp.ind(1000, .05, 200, 200, -0.4700036, -0.1823216, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 200, 200, -0.4700036, -0.1823216, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 200, 200, -0.4700036, -0.1823216, .1, .1, 0.80)

60% treatment effect for X_1, 60% treatment effect for X_2.

power.tests.exp.ind.nocens(1000, .05, 50, 50, -0.4700036, -0.4700036, .1, .1, 0)
power.tests.exp.ind(1000, .05, 50, 50, -0.4700036, -0.4700036, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 50, 50, -0.4700036, -0.4700036, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 50, 50, -0.4700036, -0.4700036, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 100, 100, -0.4700036, -0.4700036, .1, .1, 0)
power.tests.exp.ind(1000, .05, 100, 100, -0.4700036, -0.4700036, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 100, 100, -0.4700036, -0.4700036, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 100, 100, -0.4700036, -0.4700036, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 200, 200, -0.4700036, -0.4700036, .1, .1, 0)
power.tests.exp.ind(1000, .05, 200, 200, -0.4700036, -0.4700036, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 200, 200, -0.4700036, -0.4700036, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 200, 200, -0.4700036, -0.4700036, .1, .1, 0.80)

No treatment effect for X_1, -20% treatment effect for X_2.

power.tests.exp.ind.nocens(1000, .05, 50, 50, 0, 0.2231436, .1, .1, 0)
power.tests.exp.ind(1000, .05, 50, 50, 0, 0.2231436, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 50, 50, 0, 0.2231436, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 50, 50, 0, 0.2231436, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 100, 100, 0, 0.2231436, .1, .1, 0)
power.tests.exp.ind(1000, .05, 100, 100, 0, 0.2231436, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 100, 100, 0, 0.2231436, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 100, 100, 0, 0.2231436, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 200, 200, 0, 0.2231436, .1, .1, 0)
power.tests.exp.ind(1000, .05, 200, 200, 0, 0.2231436, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 200, 200, 0, 0.2231436, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 200, 200, 0, 0.2231436, .1, .1, 0.80)

60% treatment effect for X_1, -20% treatment effect for X_2.

power.tests.exp.ind.nocens(1000, .05, 50, 50, -0.4700036, 0.2231436, .1, .1, 0)
power.tests.exp.ind(1000, .05, 50, 50, -0.4700036, 0.2231436, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 50, 50, -0.4700036, 0.2231436, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 50, 50, -0.4700036, 0.2231436, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 100, 100, -0.4700036, 0.2231436, .1, .1, 0)
power.tests.exp.ind(1000, .05, 100, 100, -0.4700036, 0.2231436, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 100, 100, -0.4700036, 0.2231436, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 100, 100, -0.4700036, 0.2231436, .1, .1, 0.80)

power.tests.exp.ind.nocens(1000, .05, 200, 200, -0.4700036, 0.2231436, .1, .1, 0)
power.tests.exp.ind(1000, .05, 200, 200, -0.4700036, 0.2231436, .1, .1, 0.20)
power.tests.exp.ind(1000, .05, 200, 200, -0.4700036, 0.2231436, .1, .1, 0.50)
power.tests.exp.ind(1000, .05, 200, 200, -0.4700036, 0.2231436, .1, .1, 0.80)

E.2: SARMANOV BIVARIATE EXPONENTIAL

power.tests.exp.sarm(nsim, alpha, n.trt, n.cont, rho, beta1, beta2, lambda1, lambda2, pcens1, pcens2)

Sarmanov Bivariate Exponential (rho = 0)

No treatment effect for X_1, 20% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0, 0, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0, 0, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0, 0, -0.1823216, .1, .1, 0.2, 0.2)

power.tests.exp.sarm(1000, .05, 200, 200, 0, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0, 0, -0.1823216, .1, .1, 0.8, 0.8)

No treatment effect for X_1, 60% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0, 0, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0, 0, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0, 0, -0.4700036, .1, .1, 0.8, 0.8)

60% treatment effect for X_1, no treatment effect for X_2.

(Use the results for X_2 from the above programs)

60% treatment effect for X_1, 20% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

60% treatment effect for X_1, 60% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0, -0.4700036, -0.4700036, .1, .1, 0, 0)

power.tests.exp.sarm(1000, .05, 50, 50, 0, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

No treatment effect for X_1, -20% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0, 0, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0, 0, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0, 0, 0.2231436, .1, .1, 0.8, 0.8)

60% treatment effect for X_1, -20% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)

power.tests.exp.sarm(1000, .05, 200, 200, 0, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

Sarmanov Bivariate Exponential (rho = 0.2)

No treatment effect for X_1, 20% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.2, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, 0, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.2, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, 0, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.2, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, 0, -0.1823216, .1, .1, 0.8, 0.8)

No treatment effect for X_1, 60% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.2, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, 0, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.2, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, 0, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.2, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, 0, -0.4700036, .1, .1, 0.8, 0.8)

60% treatment effect for X_1, no treatment effect for X_2.

(Use the results for X_2 from the above programs)

60% treatment effect for X_1, 20% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.2, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.2, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.2, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

60% treatment effect for X_1, 60% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.2, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.2, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.2, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

No treatment effect for X_1, -20% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.2, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, 0, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.2, 0, 0.2231436, .1, .1, 0, 0)

power.tests.exp.sarm(1000, .05, 100, 100, 0.2, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, 0, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.2, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, 0, 0.2231436, .1, .1, 0.8, 0.8)

60% treatment effect for X_1, -20% treatment effect for X_2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.2, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.2, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.2, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.2, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.2, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.2, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

Sarmanov Bivariate Exponential (rho = 0.5)

No treatment effect for x1, 20% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.5, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, 0, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.5, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, 0, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.5, 0, -0.1823216, .1, .1, 0, 0)

power.tests.exp.sarm(1000, .05, 200, 200, 0.5, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, 0, -0.1823216, .1, .1, 0.8, 0.8)

No treatment effect for x1, 60% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.5, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, 0, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.5, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, 0, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.5, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, 0, -0.4700036, .1, .1, 0.8, 0.8)

60% treatment effect for x1, no treatment effect for x2.

(Use the results for x2 from the above programs)

60% treatment effect for x1, 20% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.5, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.5, -0.4700036, -0.1823216, .1, .1, 0,
0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.5, -0.4700036, -0.1823216, .1, .1, 0,
0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

60% treatment effect for x1, 60% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.5, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.5, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.5, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

No treatment effect for x1, -20% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.5, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, 0, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.5, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, 0, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.5, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, 0, 0.2231436, .1, .1, 0.8, 0.8)

60% treatment effect for x1, -20% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.5, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.5, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.5, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.5, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)

power.tests.exp.sarm(1000, .05, 100, 100, 0.5, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.5, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.5, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

Sarmanov Bivariate Exponential (rho = 0.8)

No treatment effect for x1, 20% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.8, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, 0, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.8, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, 0, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.8, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, 0, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, 0, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, 0, -0.1823216, .1, .1, 0.8, 0.8)

No treatment effect for x1, 60% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.8, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, 0, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.8, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, 0, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.8, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, 0, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, 0, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, 0, -0.4700036, .1, .1, 0.8, 0.8)

60% treatment effect for x1, no treatment effect for x2.

(Use the results for x2 from the above programs)

60% treatment effect for x1, 20% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.8, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.8, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.8, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, -0.4700036, -0.1823216, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, -0.4700036, -0.1823216, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, -0.4700036, -0.1823216, .1, .1, 0.8, 0.8)

60% treatment effect for x1, 60% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.8, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.8, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.8, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, -0.4700036, -0.4700036, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, -0.4700036, -0.4700036, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, -0.4700036, -0.4700036, .1, .1, 0.8, 0.8)

No treatment effect for x1, -20% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.8, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, 0, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.8, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, 0, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.8, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, 0, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, 0, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, 0, 0.2231436, .1, .1, 0.8, 0.8)

60% treatment effect for x1, -20% treatment effect for x2.

power.tests.exp.sarm.nocens(1000, .05, 50, 50, 0.8, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 50, 50, 0.8, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 100, 100, 0.8, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 100, 100, 0.8, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

power.tests.exp.sarm.nocens(1000, .05, 200, 200, 0.8, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, -0.4700036, 0.2231436, .1, .1, 0.2, 0.2)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, -0.4700036, 0.2231436, .1, .1, 0.5, 0.5)
power.tests.exp.sarm(1000, .05, 200, 200, 0.8, -0.4700036, 0.2231436, .1, .1, 0.8, 0.8)

E.3: MARSHALL-OLKIN BIVARIATE EXPONENTIAL

power.tests.exp.mo(nsim, alpha, n.trt, n.cont, beta1, beta2, beta12, lambda1, lambda2, lambda12, pcens)

The following yield P(X1=X2) = .20, no treatment effect for x1, and a 20% treatment effect for x2.

power.tests.exp.mo.nocens(1000, .05, 50, 50, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0)

power.tests.exp.mo(1000, .05, 50, 50, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 50, 50, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 50, 50, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 100, 100, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 100, 100, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 100, 100, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 100, 100, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 200, 200, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 200, 200, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 200, 200, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 200, 200, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0.8)

The following yield P(X1=X2) = .20, no treatment effect for x1, and a 60% treatment effect for x2.

power.tests.exp.mo.nocens(1000, .05, 50, 50, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 50, 50, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 50, 50, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 50, 50, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 100, 100, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 100, 100, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 100, 100, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 100, 100, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 200, 200, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 200, 200, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 200, 200, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 200, 200, 0.08961216, -0.63252256, -0.20763936, 0.06666667, 0.06666667, 0.03333333, 0.8)

The following yield P(X1=X2) = .20, a 60% treatment effect for x1, and no treatment effect for x2.

(Use the results for x2 from the above programs)

The following yield P(X1=X2) = .20, a 60% treatment effect for x1, and a 20% treatment effect for x2.

power.tests.exp.mo.nocens(1000, .05, 50, 50, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 50, 50, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 50, 50, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 50, 50, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 100, 100, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 100, 100, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 100, 100, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 100, 100, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 200, 200, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 200, 200, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 200, 200, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 200, 200, -0.55701501, -0.12169693, -0.31585295, 0.06666667, 0.06666667, 0.03333333, 0.8)

The following yield P(X1=X2) = .20, a 60% treatment effect for x1, and a 60% treatment effect for x2.

power.tests.exp.mo.nocens(1000, .05, 50, 50, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 50, 50, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 50, 50, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 50, 50, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 100, 100, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 100, 100, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 100, 100, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 100, 100, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 200, 200, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 200, 200, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 200, 200, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 200, 200, -0.47000363, -0.47000363, -0.47000363, 0.06666667, 0.06666667, 0.03333333, 0.8)

The following yield P(X1=X2) = .20, no effect for x1, and a -20% treatment effect for x2.

power.tests.exp.mo.nocens(1000, .05, 50, 50, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 50, 50, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 50, 50, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 50, 50, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 100, 100, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0)

power.tests.exp.mo(1000, .05, 100, 100, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 100, 100, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 100, 100, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 200, 200, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 200, 200, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 200, 200, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 200, 200, -0.06453852, 0.27193372, 0.11778304, 0.06666667, 0.06666667, 0.03333333, 0.8)

The following yield P(X1=X2) = .20, a 60% treatment effect for x1, and a -20% treatment effect for x2.

power.tests.exp.mo.nocens(1000, .05, 50, 50, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 50, 50, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 50, 50, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 50, 50, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 100, 100, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 100, 100, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 100, 100, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 100, 100, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0.8)

power.tests.exp.mo.nocens(1000, .05, 200, 200, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0)
power.tests.exp.mo(1000, .05, 200, 200, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0.2)
power.tests.exp.mo(1000, .05, 200, 200, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0.5)
power.tests.exp.mo(1000, .05, 200, 200, -0.75768570, 0.34092659, -0.06453852, 0.06666667, 0.06666667, 0.03333333, 0.8)

E.4: CAI-PRENTICE BIVARIATE EXPONENTIAL

power.tests.exp.cai(nsim, alpha, n.trt, n.cont, tau, beta1, beta2, lambda1, lambda2, pcens1, pcens2)

No treatment effect for x1, 20% treatment effect for x2, tau=0.01.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.01, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, 0, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.01, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, 0, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.01, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, 0, -0.1823216, .1, .1, 0.80, 0.80)

No treatment effect for x1, 60% treatment effect for x2, tau=0.01.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.01, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, 0, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.01, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, 0, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.01, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, 0, -0.4700036, .1, .1, 0.80, 0.80)

60% treatment effect for x1, no treatment effect for x2, tau=0.01.

(Use the results for x2 from the above programs)

60% treatment effect for x1, 20% treatment effect for x2, tau=0.01.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.01, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.01, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.01, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

60% treatment effect for x1, 60% treatment effect for x2, tau=0.01.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.01, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.01, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.01, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

No treatment effect for x1, -20% treatment effect for x2, tau=0.01.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.01, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, 0, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.01, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, 0, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.01, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, 0, 0.2231436, .1, .1, 0.80, 0.80)

60% treatment effect for x1, -20% treatment effect for x2, tau=0.01.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.01, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.01, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.01, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.01, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.01, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.01, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

No treatment effect for x1, 20% treatment effect for x2, tau=0.2.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.2, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, 0, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.2, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, 0, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.2, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, 0, -0.1823216, .1, .1, 0.80, 0.80)

No treatment effect for x1, 60% treatment effect for x2, tau=0.2.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.2, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, 0, -0.4700036, .1, .1, 0.20, 0.20)

power.tests.exp.cai(1000, .05, 50, 50, 0.2, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, 0, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.2, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, 0, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.2, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, 0, -0.4700036, .1, .1, 0.80, 0.80)

60% treatment effect for x1, no treatment effect for x2, tao=0.2.

(Use the results for x2 from the above programs)

60% treatment effect for x1, 20% treatment effect for x2, tau=0.2

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.2, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.2, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.2, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

60% treatment effect for x1, 60% treatment effect for x2, tau=0.2.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.2, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.2, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.2, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

No treatment effect for x1, -20% treatment effect for x2, tau=0.2.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.2, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, 0, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.2, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, 0, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.2, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, 0, 0.2231436, .1, .1, 0.80, 0.80)

60% treatment effect for x1, -20% treatment effect for x2, tau=0.2.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.2, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.2, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.2, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.2, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.2, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.2, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

No treatment effect for x1, 20% treatment effect for x2, tau=0.5.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.5, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, 0, -0.1823216, .1, .1, 0.50, 0.50)

power.tests.exp.cai(1000, .05, 50, 50, 0.5, 0, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.5, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, 0, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.5, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, 0, -0.1823216, .1, .1, 0.80, 0.80)

No treatment effect for x1, 60% treatment effect for x2, tau=0.5.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.5, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, 0, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.5, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, 0, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.5, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, 0, -0.4700036, .1, .1, 0.80, 0.80)

60% treatment effect for x1, no treatment effect for x2, tao=0.5.

(Use the results for x2 from the above programs)

60% treatment effect for x1, 20% treatment effect for x2, tau=0.5.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.5, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.5, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.5, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

60% treatment effect for x1, 60% treatment effect for x2, tau=0.5.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.5, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.5, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.5, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

No treatment effect for x1, -20% treatment effect for x2, tau=0.5.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.5, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, 0, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.5, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, 0, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.5, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, 0, 0.2231436, .1, .1, 0.80, 0.80)

60% treatment effect for x1, -20% treatment effect for x2, tau=0.5.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.5, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.5, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.5, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.5, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.5, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.5, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

No treatment effect for x1, 20% treatment effect for x2, tau=0.8.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.8, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, 0, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.8, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, 0, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.8, 0, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, 0, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, 0, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, 0, -0.1823216, .1, .1, 0.80, 0.80)

No treatment effect for x1, 60% treatment effect for x2, tau=0.8.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.8, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, 0, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.8, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, 0, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.8, 0, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, 0, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, 0, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, 0, -0.4700036, .1, .1, 0.80, 0.80)

60% treatment effect for x1, no treatment effect for x2, tau=0.8.

(Use the results for x2 from the above programs)

60% treatment effect for x1, 20% treatment effect for x2, tau=0.8.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.8, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.8, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.8, -0.4700036, -0.1823216, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, -0.4700036, -0.1823216, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, -0.4700036, -0.1823216, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, -0.4700036, -0.1823216, .1, .1, 0.80, 0.80)

60% treatment effect for x1, 60% treatment effect for x2, tau=0.8.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.8, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.8, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.8, -0.4700036, -0.4700036, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, -0.4700036, -0.4700036, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, -0.4700036, -0.4700036, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, -0.4700036, -0.4700036, .1, .1, 0.80, 0.80)

No treatment effect for x1, -20% treatment effect for x2, tau=0.8.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.8, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, 0, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.8, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, 0, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.8, 0, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, 0, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, 0, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, 0, 0.2231436, .1, .1, 0.80, 0.80)

60% treatment effect for x1, -20% treatment effect for x2, tau=0.8.

power.tests.exp.cai.nocens(1000, .05, 50, 50, 0.8, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 50, 50, 0.8, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 100, 100, 0.8, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 100, 100, 0.8, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

power.tests.exp.cai.nocens(1000, .05, 200, 200, 0.8, -0.4700036, 0.2231436, .1, .1, 0, 0)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, -0.4700036, 0.2231436, .1, .1, 0.20, 0.20)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, -0.4700036, 0.2231436, .1, .1, 0.50, 0.50)
power.tests.exp.cai(1000, .05, 200, 200, 0.8, -0.4700036, 0.2231436, .1, .1, 0.80, 0.80)

E.5: BIVARIATE EXPONENTIAL WITH A GAMMA FRAILTY

power.tests.exp.frailty(nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1, lambda2, theta, pcens)

No trt effect for x1, 20% trt effect for x2, theta=2

power.tests.exp.frailty.nocens(1000, .05, 50, 50, 0, -0.1823216, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 50, 50, 0, -0.1823216, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 50, 50, 0, -0.1823216, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 50, 50, 0, -0.1823216, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 100, 100, 0, -0.1823216, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 100, 100, 0, -0.1823216, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 100, 100, 0, -0.1823216, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 100, 100, 0, -0.1823216, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 200, 200, 0, -0.1823216, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 200, 200, 0, -0.1823216, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 200, 200, 0, -0.1823216, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 200, 200, 0, -0.1823216, .1, .1, 2, 0.80)

No trt effect for x1, 60% trt effect for x2, theta=2

power.tests.exp.frailty.nocens(1000, .05, 50, 50, 0, -0.4700036, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 50, 50, 0, -0.4700036, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 50, 50, 0, -0.4700036, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 50, 50, 0, -0.4700036, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 100, 100, 0, -0.4700036, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 100, 100, 0, -0.4700036, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 100, 100, 0, -0.4700036, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 100, 100, 0, -0.4700036, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 200, 200, 0, -0.4700036, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 200, 200, 0, -0.4700036, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 200, 200, 0, -0.4700036, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 200, 200, 0, -0.4700036, .1, .1, 2, 0.80)

60% trt effect for x1, No trt effect for x2, theta=2

(Use the results for x2 from the above program for these)

60% trt effect for x1, 20% trt effect for x2, theta=2

power.tests.exp.frailty.nocens(1000, .05, 50, 50, -0.4700036, -0.1823216, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 50, 50, -0.4700036, -0.1823216, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 50, 50, -0.4700036, -0.1823216, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 50, 50, -0.4700036, -0.1823216, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 100, 100, -0.4700036, -0.1823216, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 100, 100, -0.4700036, -0.1823216, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 100, 100, -0.4700036, -0.1823216, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 100, 100, -0.4700036, -0.1823216, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 200, 200, -0.4700036, -0.1823216, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 200, 200, -0.4700036, -0.1823216, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 200, 200, -0.4700036, -0.1823216, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 200, 200, -0.4700036, -0.1823216, .1, .1, 2, 0.80)

60% trt effect for x1, 60% trt effect for x2, theta=2

power.tests.exp.frailty.nocens(1000, .05, 50, 50, -0.4700036, -0.4700036, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 50, 50, -0.4700036, -0.4700036, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 50, 50, -0.4700036, -0.4700036, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 50, 50, -0.4700036, -0.4700036, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 100, 100, -0.4700036, -0.4700036, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 100, 100, -0.4700036, -0.4700036, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 100, 100, -0.4700036, -0.4700036, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 100, 100, -0.4700036, -0.4700036, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 200, 200, -0.4700036, -0.4700036, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 200, 200, -0.4700036, -0.4700036, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 200, 200, -0.4700036, -0.4700036, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 200, 200, -0.4700036, -0.4700036, .1, .1, 2, 0.80)

No trt effect for x1, -20% trt effect for x2, theta=2

power.tests.exp.frailty.nocens(1000, .05, 50, 50, 0, 0.2231436, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 50, 50, 0, 0.2231436, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 50, 50, 0, 0.2231436, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 50, 50, 0, 0.2231436, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 100, 100, 0, 0.2231436, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 100, 100, 0, 0.2231436, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 100, 100, 0, 0.2231436, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 100, 100, 0, 0.2231436, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 200, 200, 0, 0.2231436, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 200, 200, 0, 0.2231436, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 200, 200, 0, 0.2231436, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 200, 200, 0, 0.2231436, .1, .1, 2, 0.80)

60% trt effect for x1, -20% trt effect for x2, theta=2

power.tests.exp.frailty.nocens(1000, .05, 50, 50, -0.4700036, 0.2231436, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 50, 50, -0.4700036, 0.2231436, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 50, 50, -0.4700036, 0.2231436, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 50, 50, -0.4700036, 0.2231436, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 100, 100, -0.4700036, 0.2231436, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 100, 100, -0.4700036, 0.2231436, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 100, 100, -0.4700036, 0.2231436, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 100, 100, -0.4700036, 0.2231436, .1, .1, 2, 0.80)

power.tests.exp.frailty.nocens(1000, .05, 200, 200, -0.4700036, 0.2231436, .1, .1, 2, 0)
power.tests.exp.frailty(1000, .05, 200, 200, -0.4700036, 0.2231436, .1, .1, 2, 0.20)
power.tests.exp.frailty(1000, .05, 200, 200, -0.4700036, 0.2231436, .1, .1, 2, 0.50)
power.tests.exp.frailty(1000, .05, 200, 200, -0.4700036, 0.2231436, .1, .1, 2, 0.80)

E.6: INDEPENDENT BIVARIATE WEIBULL

power.tests.weib.ind (nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1, lambda2, alpha1, alpha2, alphaC, pcens)

No trt effect for x1, 20% trt effect for x2

power.tests.weib.ind.nocens(1000, .05, 50, 50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 50, 50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 50, 50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 50, 50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 100, 100, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 100, 100, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 100, 100, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 100, 100, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 200, 200, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 200, 200, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 200, 200, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 200, 200, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

No trt effect for x1, 60% trt effect for x2

power.tests.weib.ind.nocens(1000, .05, 50, 50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0)

power.tests.weib.ind(1000, .05, 50, 50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 50, 50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 50, 50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 100, 100, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 100, 100, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 100, 100, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 100, 100, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 200, 200, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 200, 200, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 200, 200, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 200, 200, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

60% trt effect for x1, No trt effect for x2, theta=2

(Use the results for x2 from the above program for these)

60% trt effect for x1, 20% trt effect for x2

power.tests.weib.ind.nocens(1000, .05, 50, 50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 50, 50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 50, 50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 50, 50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 100, 100, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 100, 100, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.20)

power.tests.weib.ind(1000, .05, 100, 100, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 100, 100, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 200, 200, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 200, 200, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 200, 200, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 200, 200, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

60% trt effect for x1, 60% trt effect for x2

power.tests.weib.ind.nocens(1000, .05, 50, 50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 50, 50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 50, 50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 50, 50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 100, 100, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 100, 100, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 100, 100, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 100, 100, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 200, 200, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 200, 200, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 200, 200, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 200, 200, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

No trt effect for x1, -20% trt effect for x2

power.tests.weib.ind.nocens(1000, .05, 50, 50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 50, 50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 50, 50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 50, 50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 100, 100, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 100, 100, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 100, 100, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 100, 100, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 200, 200, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 200, 200, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 200, 200, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 200, 200, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

60% trt effect for x1, -20% trt effect for x2

power.tests.weib.ind.nocens(1000, .05, 50, 50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 50, 50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 50, 50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 50, 50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 100, 100, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 100, 100, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 100, 100, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.50)

power.tests.weib.ind(1000, .05, 100, 100, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

power.tests.weib.ind.nocens(1000, .05, 200, 200, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0)
power.tests.weib.ind(1000, .05, 200, 200, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.20)
power.tests.weib.ind(1000, .05, 200, 200, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.50)
power.tests.weib.ind(1000, .05, 200, 200, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 0.80)

E.7: SARMANOV BIVARIATE WEIBULL

power.tests.weib.sarm(nsim, alpha, n.trt, n.cont, rho, beta1, beta2, lambda1, lambda2, alpha1, alpha2, alphaC1, alphaC2, pcens1, pcens2)

Sarmanov Weibull (rho = 0)

No trt effect for x1, 20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)

power.tests.weib.sarm(1000, .05, 200, 200, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

No trt effect for x1, 60% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

60% trt effect for x1, 20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)

power.tests.weib.sarm(1000, .05, 50, 50, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, 60% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)

power.tests.weib.sarm(1000, .05, 200, 200, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

No trt effect for x1, -20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, -20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)

power.tests.weib.sarm(1000, .05, 50, 50, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

Sarmanov Weibull (rho = 0.20)

No trt effect for x1, 20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

No trt effect for x1, 60% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

60% trt effect for x1, 20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, 60% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)

power.tests.weib.sarm(1000, .05, 100, 100, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

No trt effect for x1, -20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, -20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)

power.tests.weib.sarm(1000, .05, 50, 50, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.2, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

Sarmanov Weibull (rho = 0.50)

No trt effect for x1, 20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)

power.tests.weib.sarm(1000, .05, 100, 100, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

No trt effect for x1, 60% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

60% trt effect for x1, 20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, 60% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)

power.tests.weib.sarm(1000, .05, 100, 100, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

No trt effect for x1, -20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, -20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.5, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

Sarmanov Weibull (rho = 0.80)

No trt effect for x1, 20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)

power.tests.weib.sarm(1000, .05, 100, 100, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

No trt effect for x1, 60% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above programs)

60% trt effect for x1, 20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, 60% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

No trt effect for x1, -20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)

power.tests.weib.sarm(1000, .05, 200, 200, 0.8, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

60% trt effect for x1, -20% trt effect for x2

power.tests.weib.sarm.nocens(1000, .05, 50, 50, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 50, 50, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 100, 100, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 100, 100, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

power.tests.weib.sarm.nocens(1000, .05, 200, 200, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5, 0.5)
power.tests.weib.sarm(1000, .05, 200, 200, 0.8, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8, 0.8)

E.8: MARSHALL-OLKIN BIVARIATE WEIBULL

power.tests.weib.mo(nsim, alpha, n.trt, n.cont, beta1, beta2, beta12, lambda1, lambda2, lambda12, alpha1, alpha2, alpha12, alphaC, pcens)

The following yield P(X1=X2) = .20, No trt effect for x1, and a 20% treatment effect for x2:

power.tests.weib.mo.nocens(1000, .05, 50, 50, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 50, 50, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)

power.tests.weib.mo(1000, .05, 50, 50, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 50, 50, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 100, 100, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 100, 100, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 100, 100, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 100, 100, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 200, 200, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 200, 200, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 200, 200, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 200, 200, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

The following yield P(X1=X2) = .20, No trt effect for x1, and a 60% treatment effect for x2:

power.tests.weib.mo.nocens(1000, .05, 50, 50, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 50, 50, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 50, 50, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 50, 50, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 100, 100, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 100, 100, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 100, 100, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 100, 100, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 200, 200, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 200, 200, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 200, 200, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 200, 200, 0.1417979, -1.4343036, -0.36339389, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

The following yield P(X1=X2) = .20, 60% trt effect for x1, and No treatment effect for x2:

(Use the results for x2 from the above programs)

The following yield P(X1=X2) = .20, 60% trt effect for x1, and a 20% treatment effect for x2:

power.tests.weib.mo.nocens(1000, .05, 50, 50, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 50, 50, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 50, 50, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 50, 50, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 100, 100, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 100, 100, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 100, 100, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 100, 100, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 200, 200, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 200, 200, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 200, 200, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 200, 200, -1.15623037, -0.2608463, -0.61150319, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

The following yield P(X1=X2) = .20, 60% trt effect for x1, and a 60% treatment effect for x2:

power.tests.weib.mo.nocens(1000, .05, 50, 50, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 50, 50, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 50, 50, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 50, 50, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 100, 100, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 100, 100, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 100, 100, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 100, 100, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 200, 200, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 200, 200, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 200, 200, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 200, 200, -0.94000726, -0.94000726, -0.94000726, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

The following yield P(X1=X2) = .20, No trt effect for x1, and a -20% treatment effect for x2:

power.tests.weib.mo.nocens(1000, .05, 50, 50, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 50, 50, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 50, 50, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 50, 50, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 100, 100, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)

power.tests.weib.mo(1000, .05, 100, 100, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 100, 100, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 100, 100, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 200, 200, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 200, 200, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 200, 200, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 200, 200, -0.15154990, 0.5324648, 0.24783616, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

The following yield P(X1=X2) = .20, 60% trt effect for x1, and a -20% treatment effect for x2:

power.tests.weib.mo.nocens(1000, .05, 50, 50, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 50, 50, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 50, 50, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 50, 50, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 100, 100, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 100, 100, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 100, 100, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 100, 100, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

power.tests.weib.mo.nocens(1000, .05, 200, 200, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0)
power.tests.weib.mo(1000, .05, 200, 200, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)
power.tests.weib.mo(1000, .05, 200, 200, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.50)
power.tests.weib.mo(1000, .05, 200, 200, -2.32630162, 0.6181374, -0.02371653, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.80)

E.9: CAI-PRENTICE BIVARIATE WEIBULL

power.tests.weib.cai(nsim, alpha, n.trt, n.cont, tau, beta1, beta2, lambda1, lambda2, alpha1, alpha2, alphaC1, alphaC2, pcens1, pcens2)

No trt effect for x1, 20% trt effect for x2, tau=0.01

power.tests.weib.cai.nocens(1000, .05, 50, 50, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, 60% trt effect for x2, tau=0.01

power.tests.weib.cai.nocens(1000, .05, 50, 50, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)

power.tests.weib.cai(1000, .05, 50, 50, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, No trt effect for x2, tau=0.01

(Use the results for x2 from the above programs)

60% trt effect for x1, 20% trt effect for x2, tau=0.01

power.tests.weib.cai.nocens(1000, .05, 50, 50, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, 60% trt effect for x2, tau=0.01

power.tests.weib.cai.nocens(1000, .05, 50, 50, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, -20% trt effect for x2, tau=0.01

power.tests.weib.cai.nocens(1000, .05, 50, 50, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)

power.tests.weib.cai(1000, .05, 50, 50, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, -20% trt effect for x2, tau=0.01

power.tests.weib.cai.nocens(1000, .05, 50, 50, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, 0.01, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, 20% trt effect for x2, tau=.2

power.tests.weib.cai.nocens(1000, .05, 50, 50, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .20, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, 60% trt effect for x2, tau=.2

power.tests.weib.cai.nocens(1000, .05, 50, 50, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)

power.tests.weib.cai(1000, .05, 50, 50, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .20, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, No trt effect for x2, tau=.2

(Use the results for x2 from the above programs)

60% trt effect for x1, 20% trt effect for x2, tau=.2

power.tests.weib.cai.nocens(1000, .05, 50, 50, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)

power.tests.weib.cai(1000, .05, 100, 100, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .20, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, 60% trt effect for x2, tau=.2

power.tests.weib.cai.nocens(1000, .05, 50, 50, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .20, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, -20% trt effect for x2, tau=.2

power.tests.weib.cai.nocens(1000, .05, 50, 50, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)

power.tests.weib.cai(1000, .05, 50, 50, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .20, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, -20% trt effect for x2, tau=.2

power.tests.weib.cai.nocens(1000, .05, 50, 50, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .20, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, 20% trt effect for x2, tau=.5

power.tests.weib.cai.nocens(1000, .05, 50, 50, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, 60% trt effect for x2, tau=.5

power.tests.weib.cai.nocens(1000, .05, 50, 50, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)

power.tests.weib.cai(1000, .05, 50, 50, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, No trt effect for x2, tau=.5

(Use the results for x2 from the above programs)

60% trt effect for x1, 20% trt effect for x2, tau=.5

power.tests.weib.cai.nocens(1000, .05, 50, 50, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)

power.tests.weib.cai(1000, .05, 100, 100, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, 60% trt effect for x2, tau=.5

power.tests.weib.cai.nocens(1000, .05, 50, 50, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, -20% trt effect for x2, tau=.5

power.tests.weib.cai.nocens(1000, .05, 50, 50, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)

power.tests.weib.cai(1000, .05, 50, 50, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, -20% trt effect for x2, tau=.5

power.tests.weib.cai.nocens(1000, .05, 50, 50, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, 20% trt effect for x2, tau=.8

power.tests.weib.cai.nocens(1000, .05, 50, 50, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .80, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, 60% trt effect for x2, tau=.8

power.tests.weib.cai.nocens(1000, .05, 50, 50, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)

power.tests.weib.cai(1000, .05, 50, 50, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .80, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, No trt effect for x2, tau=.8

(Use the results for x2 from the above program for these)

60% trt effect for x1, 20% trt effect for x2, tau=.8

power.tests.weib.cai.nocens(1000, .05, 50, 50, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)

power.tests.weib.cai(1000, .05, 100, 100, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .80, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, 60% trt effect for x2, tau=.8

power.tests.weib.cai.nocens(1000, .05, 50, 50, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .80, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

No trt effect for x1, -20% trt effect for x2, tau=.8

power.tests.weib.cai.nocens(1000, .05, 50, 50, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)

power.tests.weib.cai(1000, .05, 50, 50, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .80, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

60% trt effect for x1, -20% trt effect for x2, tau=.8

power.tests.weib.cai.nocens(1000, .05, 50, 50, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 50, 50, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 50, 50, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 50, 50, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 100, 100, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 100, 100, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 100, 100, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 100, 100, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

power.tests.weib.cai.nocens(1000, .05, 200, 200, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0, 0)
power.tests.weib.cai(1000, .05, 200, 200, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)
power.tests.weib.cai(1000, .05, 200, 200, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.50, 0.50)
power.tests.weib.cai(1000, .05, 200, 200, .80, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.80, 0.80)

E.10: BIVARIATE WEIBULL WITH A GAMMA FRAILTY

power.tests.weib.frailty(nsim, alpha, n.trt, n.cont, beta1, beta2, lambda1, lambda2, theta, alpha1, alpha2, alphaC, pcens)

No trt effect for x1, 20% trt effect for x2, theta=2

power.tests.weib.frailty.nocens(1000, .05, 50, 50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 50, 50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 50, 50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 50, 50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 100, 100, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 100, 100, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 100, 100, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 100, 100, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 200, 200, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 200, 200, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 200, 200, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 200, 200, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

No trt effect for x1, 60% trt effect for x2, theta=2

power.tests.weib.frailty.nocens(1000, .05, 50, 50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 50, 50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 50, 50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 50, 50, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 100, 100, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 100, 100, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 100, 100, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 100, 100, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 200, 200, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 200, 200, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 200, 200, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 200, 200, 0, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

60% trt effect for x1, No trt effect for x2, theta=2

(Use the results for x2 from the above program for these)

60% trt effect for x1, 20% trt effect for x2, theta=2

power.tests.weib.frailty.nocens(1000, .05, 50, 50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 50, 50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 50, 50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 50, 50, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 100, 100, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)

power.tests.weib.frailty(1000, .05, 100, 100, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 100, 100, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 100, 100, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 200, 200, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 200, 200, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 200, 200, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 200, 200, -0.9400073, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

60% trt effect for x1, 60% trt effect for x2, theta=2

power.tests.weib.frailty.nocens(1000, .05, 50, 50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 50, 50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 50, 50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 50, 50, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 100, 100, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 100, 100, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 100, 100, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 100, 100, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 200, 200, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 200, 200, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 200, 200, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 200, 200, -0.9400073, -0.9400073, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

No trt effect for x1, -20% trt effect for x2, theta=2

power.tests.weib.frailty.nocens(1000, .05, 50, 50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 50, 50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 50, 50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 50, 50, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 100, 100, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 100, 100, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 100, 100, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 100, 100, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 200, 200, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 200, 200, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 200, 200, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 200, 200, 0, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

60% trt effect for x1, -20% trt effect for x2, theta=2

power.tests.weib.frailty.nocens(1000, .05, 50, 50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 50, 50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 50, 50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 50, 50, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 100, 100, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 100, 100, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)

power.tests.weib.frailty(1000, .05, 100, 100, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 100, 100, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

power.tests.weib.frailty.nocens(1000, .05, 200, 200, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0)
power.tests.weib.frailty(1000, .05, 200, 200, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)
power.tests.weib.frailty(1000, .05, 200, 200, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.5)
power.tests.weib.frailty(1000, .05, 200, 200, -0.9400073, 0.4462871, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.8)

E.11: BIVARIATE NORMAL

power.tests.norm(nsim, alpha, n.trt, n.cont, beta1, beta2, mu1, mu2, var1, var2, lambda1, lambda2, rho, pcens1, pcens2)

Bivariate Normal (rho = 0)

No trt effect for x1, 5% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

No trt effect for x1, 15% effect for x2

power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

```
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)
```

15% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

15% trt effect for x1, 5% trt effect for x2

```
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.20,
0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.50,
0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0.80,
0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0,
0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0,
0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0,
0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0,
0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0, 0,
0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0,
0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0,
0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, 0,
0.80, 0.80)
```

15% trt effect for x1, 15% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

No trt effect for x1, -5% effect for x2

power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

15% trt effect for x1, -5% effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, 0, 0.80, 0.80)

Bivariate Normal (rho = 0.20)

power.tests.norm(nsim, alpha, n.trt, n.cont, beta1, beta2, mu1, mu2, var1, var2, lambda1, lambda2, rho, pcens1, pcens2)

No trt effect for x1, 5% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

No trt effect for x1, 15% effect for x2

power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

15% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

15% trt effect for x1, 5% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0,
0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2,
0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2,
0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2,
0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0,
0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2,
0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2,
0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2,
0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

15% trt effect for x1, 15% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

No trt effect for x1, -5% effect for x2

power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

15% trt effect for x1, -5% effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .2, 0.80, 0.80)

Bivariate Normal (rho = 0.50)

No trt effect for x1, 5% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

No trt effect for x1, 15% effect for x2

power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

15% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

15% trt effect for x1, 5% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)

power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

15% trt effect for x1, 15% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)

power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

No trt effect for x1, -5% effect for x2

power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

15% trt effect for x1, -5% effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0, 0)

power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .5, 0.80, 0.80)

Bivariate Normal (rho = 0.80)

No trt effect for x1, 5% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

No trt effect for x1, 15% effect for x2

power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

15% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

15% trt effect for x1, 5% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.04879016, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

15% trt effect for x1, 15% trt effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0, 0)

power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, 0.13976194, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

No trt effect for x1, -5% effect for x2

power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

15% trt effect for x1, -5% effect for x2

power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 50, 50, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 100, 100, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0, 0)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.20, 0.20)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.50, 0.50)
power.tests.norm(1000, .05, 200, 200, 0.13976194, -0.05129329, 10, 10, 10, 10, 0, 0, .8, 0.80, 0.80)

E.12: BIVARIATE LOG-NORMAL

power.tests.lnorm(nsim, alpha, n.trt, n.cont, beta1, beta2, mu1, mu2, var1, var2, lambda1, lambda2, rho, pcens1, pcens2)

Bivariate Log-Normal (rho = 0)

No trt effect for x1, 5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

No trt effect for x1, 15% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

15% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

15% trt effect for x1, 5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

15% trt effect for x1, 15% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

No trt effect for x1, -5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

15% trt effect for x1, -5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0, 0.80, 0.80)

Bivariate Log-Normal (rho = 0.20)

No trt effect for x1, 5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

No trt effect for x1, 15% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

15% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

15% trt effect for x1, 5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.2, 0.80, 0.80)

15% trt effect for x1, 15% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

No trt effect for x1, -5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

15% trt effect for x1, -5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .2, 0.80, 0.80)

Bivariate Log-Normal (rho = 0.50)

No trt effect for x1, 5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

No trt effect for x1, 15% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

15% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

15% trt effect for x1, 5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.5, 0.80, 0.80)

15% trt effect for x1, 15% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

No trt effect for x1, -5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)

power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

15% trt effect for x1, -5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.80, 0.80)

Bivariate Log-Normal (rho = 0.80)

No trt effect for x1, 5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)

power.tests.lnorm(1000, .05, 50, 50, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

No trt effect for x1, 15% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)

power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

15% trt effect for x1, No trt effect for x2

(Use the results for x2 from the above program for these)

15% trt effect for x1, 5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, 0.8, 0.80, 0.80)

15% trt effect for x1, 15% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, 0.05931645, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

No trt effect for x1, -5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)

power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

15% trt effect for x1, -5% trt effect for x2

power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 50, 50, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 100, 100, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0, 0)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.20, 0.20)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.50, 0.50)
power.tests.lnorm(1000, .05, 200, 200, 0.05931645, -0.02268335, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .8, 0.80, 0.80)

APPENDIX F

TEST RESULTS FOR DATA GENERATION PROGRAMS

F.1: INDEPENDENT BIVARIATE EXPONENTIAL

biv.exp.ind<-function(n.trt, n.cont, beta1, beta2, lambda1, lambda2, pcens)

temp<-biv.exp.ind(200, 200, 0, -0.1823216, .1, .1, 0.20)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 12$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 10^2$
$Var(X_1 \mid Z = 0) = 10^2$
$Var(X_2 \mid Z = 1) = 12^2$
$Var(X_2 \mid Z = 0) = 10^2$
$Corr(X_1, X_2 \mid Z = 1) = 0$
$Corr(X_1, X_2 \mid Z = 0) = 0$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 8.619$
$\hat{E}(X_1 \mid Z = 0) = 9.779$
$\hat{E}(X_2 \mid Z = 1) = 12.546$
$\hat{E}(X_2 \mid Z = 0) = 11.052$
$\hat{Var}(X_1 \mid Z = 1) = 8.260^2$
$\hat{Var}(X_1 \mid Z = 0) = 9.137^2$
$\hat{Var}(X_2 \mid Z = 1) = 12.304^2$
$\hat{Var}(X_2 \mid Z = 0) = 10.582^2$
$\hat{Corr}(X_1, X_2 \mid Z = 1) = 0.057$
$\hat{Corr}(X_1, X_2 \mid Z = 0) = -0.009$
$\hat{P}(censoring) = 0.183$

F.2: SARMANOV BIVARIATE EXPONENTIAL

biv.exp.sarm<-function(n.trt, n.cont, rho, beta1, beta2, lambda1, lambda2, pcens1, pcens2)

temp<-biv.exp.sarm(200, 200, 0.5, 0, -0.1823216, .1, .1, 0.2, 0.2)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 12$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 10^2$
$Var(X_1 \mid Z = 0) = 10^2$
$Var(X_2 \mid Z = 1) = 12^2$
$Var(X_2 \mid Z = 0) = 10^2$
$Corr(X_1, X_2 \mid Z = 1) = 0.50$
$Corr(X_1, X_2 \mid Z = 0) = 0.50$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 8.886$
$\hat{E}(X_1 \mid Z = 0) = 10.618$
$\hat{E}(X_2 \mid Z = 1) = 13.035$
$\hat{E}(X_2 \mid Z = 0) = 12.592$
$\hat{Var}(X_1 \mid Z = 1) = 9.166^2$
$\hat{Var}(X_1 \mid Z = 0) = 10.511^2$
$\hat{Var}(X_2 \mid Z = 1) = 11.847^2$
$\hat{Var}(X_2 \mid Z = 0) = 10.561^2$
$\hat{Corr}(X_1, X_2 \mid Z = 1) = 0.271$
$\hat{Corr}(X_1, X_2 \mid Z = 0) = 0.317$
$\hat{P}(censoring) = 0.273$

F.3: MARSHALL-OLKIN BIVARIATE EXPONENTIAL

biv.exp.mo<-function(n.trt, n.cont, beta1, beta2, beta12, lambda1, lambda2, lambda12, pcens)

temp<-biv.exp.mo(200, 200, 0.04082199, -0.23361485, -0.08701138, 0.06666667, 0.06666667, 0.03333333, 0.2)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 12$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 10^2$
$Var(X_1 \mid Z = 0) = 10^2$
$Var(X_2 \mid Z = 1) = 12^2$
$Var(X_2 \mid Z = 0) = 10^2$
$P(X_1 = X_2) = 0.20$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 9.356$
$\hat{E}(X_1 \mid Z = 0) = 9.141$
$\hat{E}(X_2 \mid Z = 1) = 10.489$
$\hat{E}(X_2 \mid Z = 0) = 9.255$
$\hat{Var}(X_1 \mid Z = 1) = 9.285^2$
$\hat{Var}(X_1 \mid Z = 0) = 8.338^2$
$\hat{Var}(X_2 \mid Z = 1) = 10.809^2$
$\hat{Var}(X_2 \mid Z = 0) = 9.579^2$
$\hat{P}(X_1 = X_2) = 0.220$
$\hat{P}(censoring) = 0.158$

F.4: CAI-PRENTICE BIVARIATE EXPONENTIAL

biv.exp.cai<-function(n.trt, n.cont, tau, beta1, beta2, lambda1, lambda2, pcens1, pcens2)

temp<-biv.exp.cai(200,200, 0.5, 0, -0.1823216, .1, .1, 0.20, 0.20)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 12$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 10^2$
$Var(X_1 \mid Z = 0) = 10^2$
$Var(X_2 \mid Z = 1) = 12^2$
$Var(X_2 \mid Z = 0) = 10^2$
$Tau(X_1, X_2 \mid Z = 1) = 0.50$
$Tau(X_1, X_2 \mid Z = 0) = 0.50$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 10.392$
$\hat{E}(X_1 \mid Z = 0) = 10.223$
$\hat{E}(X_2 \mid Z = 1) = 12.615$
$\hat{E}(X_2 \mid Z = 0) = 10.302$
$\hat{Var}(X_1 \mid Z = 1) = 10.506^2$
$\hat{Var}(X_1 \mid Z = 0) = 11.235^2$
$\hat{Var}(X_2 \mid Z = 1) = 12.341^2$
$\hat{Var}(X_2 \mid Z = 0) = 11.094^2$
$\hat{Tau}(X_1, X_2 \mid Z = 1) = 0.560$
$\hat{Tau}(X_1, X_2 \mid Z = 0) = 0.461$
$\hat{P}(censoring) = 0.280$

F.5: BIVARIATE EXPONENTIAL WITH A GAMMA FRAILTY

biv.exp.frailty(n.trt, n.cont, beta1, beta2, lambda1, lambda2, theta, pcens)

temp<-biv.exp.frailty(200, 200, 0, -0.1823216, .1, .1, 2, 0.20)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 12$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 14.142^2$
$Var(X_1 \mid Z = 0) = 14.142^2$
$Var(X_2 \mid Z = 1) = 16.971^2$
$Var(X_2 \mid Z = 0) = 14.142^2$
$Corr(X_1, X_2 \mid Z = 1) > 0$
$Corr(X_1, X_2 \mid Z = 0) > 0$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 10.261$
$\hat{E}(X_1 \mid Z = 0) = 9.528$
$\hat{E}(X_2 \mid Z = 1) = 11.520$
$\hat{E}(X_2 \mid Z = 0) = 9.084$
$\hat{V}ar(X_1 \mid Z = 1) = 11.934^2$
$\hat{V}ar(X_1 \mid Z = 0) = 12.826^2$
$\hat{V}ar(X_2 \mid Z = 1) = 15.224^2$
$\hat{V}ar(X_2 \mid Z = 0) = 10.695^2$
$\hat{C}orr(X_1, X_2 \mid Z = 1) = 0.265$
$\hat{C}orr(X_1, X_2 \mid Z = 0) = 0.279$
$\hat{P}(censoring) = 0.218$

F.6: INDEPENDENT BIVARIATE WEIBULL

biv.weib.ind<-function(n.trt, n.cont, beta1, beta2, lambda1, lambda2, alpha1, alpah2, alphaC, pcens)

temp<-biv.weib.ind(200, 200, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 0.20)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 12$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 5.227^2$
$Var(X_1 \mid Z = 0) = 5.227^2$
$Var(X_2 \mid Z = 1) = 6.273^2$
$Var(X_2 \mid Z = 0) = 5.227^2$
$Corr(X_1, X_2 \mid Z = 1) = 0$
$Corr(X_1, X_2 \mid Z = 0) = 0$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 10.018$
$\hat{E}(X_1 \mid Z = 0) = 10.193$
$\hat{E}(X_2 \mid Z = 1) = 11.962$
$\hat{E}(X_2 \mid Z = 0) = 9.945$
$\hat{Var}(X_1 \mid Z = 1) = 5.137^2$
$\hat{Var}(X_1 \mid Z = 0) = 5.585^2$
$\hat{Var}(X_2 \mid Z = 1) = 6.610^2$
$\hat{Var}(X_2 \mid Z = 0) = 5.040^2$
$\hat{Corr}(X_1, X_2 \mid Z = 1) = 0.049$
$\hat{Corr}(X_1, X_2 \mid Z = 0) = -0.061$
$\hat{P}(censoring) = 0.208$

F.7: SARMANOV BIVARIATE WEIBULL

biv.weib.sarm<-function(n.trt, n.cont, rho, beta1, beta2, lambda1, lambda2, alpha1, alpha2, alphaC1, alphaC2, pcens1, pcens2)

temp<-biv.weib.sarm(200, 200, 0.5, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2, 0.2)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 12$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 5.227^2$
$Var(X_1 \mid Z = 0) = 5.227^2$
$Var(X_2 \mid Z = 1) = 6.273^2$
$Var(X_2 \mid Z = 0) = 5.227^2$
$Corr(X_1, X_2 \mid Z = 1) = 0.50$
$Corr(X_1, X_2 \mid Z = 0) = 0.50$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 10.122$
$\hat{E}(X_1 \mid Z = 0) = 9.807$
$\hat{E}(X_2 \mid Z = 1) = 14.026$
$\hat{E}(X_2 \mid Z = 0) = 12.101$
$\hat{V}ar(X_1 \mid Z = 1) = 5.281^2$
$\hat{V}ar(X_1 \mid Z = 0) = 5.343^2$
$\hat{V}ar(X_2 \mid Z = 1) = 5.369^2$
$\hat{V}ar(X_2 \mid Z = 0) = 4.380^2$
$\hat{C}orr(X_1, X_2 \mid Z = 1) = 0.258$
$\hat{C}orr(X_1, X_2 \mid Z = 0) = 0.188$
$\hat{P}(censoring) = 0.225$

F.8: MARSHALL-OLKIN BIVARIATE WEIBULL

biv.weib.mo<-function(n.trt, n.cont, beta1, beta2, beta12, lambda1, lambda2, lambda12, alpha1, alpha2, alpha12, alphaC, pcens)

temp<-biv.weib.mo(200, 200, 0.07361182, -0.4811769, -0.16579225, 0.005235988, 0.005235988, 0.002617994, 2, 2, 2, 2, 0.20)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 12$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 5.227^2$
$Var(X_1 \mid Z = 0) = 5.227^2$
$Var(X_2 \mid Z = 1) = 6.273^2$
$Var(X_2 \mid Z = 0) = 5.227^2$
$P(X_1 = X_2) = 0.20$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 9.325$
$\hat{E}(X_1 \mid Z = 0) = 10.103$
$\hat{E}(X_2 \mid Z = 1) = 12.090$
$\hat{E}(X_2 \mid Z = 0) = 10.124$
$\hat{Var}(X_1 \mid Z = 1) = 4.712^2$
$\hat{Var}(X_1 \mid Z = 0) = 5.632^2$
$\hat{Var}(X_2 \mid Z = 1) = 6.200^2$
$\hat{Var}(X_2 \mid Z = 0) = 5.251^2$
$\hat{P}(X_1 = X_2) = 0.140$
$\hat{P}(censoring) = 0.213$

F.9: CAI-PRENTICE BIVARIATE WEIBULL

biv.weib.cai<-function(n.trt, n.cont, tau, beta1, beta2, lambda1, lambda2, alpha1, alpha2, alphaC1, alphaC2, pcens1, pcens2)

temp<-biv.weib.cai(200, 200, .50, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.20, 0.20)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 12$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 5.227^2$
$Var(X_1 \mid Z = 0) = 5.227^2$
$Var(X_2 \mid Z = 1) = 6.273^2$
$Var(X_2 \mid Z = 0) = 5.227^2$
$Tau(X_1, X_2 \mid Z = 1) = 0.50$
$Tau(X_1, X_2 \mid Z = 0) = 0.50$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 10.148$
$\hat{E}(X_1 \mid Z = 0) = 10.184$
$\hat{E}(X_2 \mid Z = 1) = 12.402$
$\hat{E}(X_2 \mid Z = 0) = 10.002$
$\hat{Var}(X_1 \mid Z = 1) = 5.019^2$
$\hat{Var}(X_1 \mid Z = 0) = 5.091^2$
$\hat{Var}(X_2 \mid Z = 1) = 6.503^2$
$\hat{Var}(X_2 \mid Z = 0) = 5.497^2$
$\hat{Tau}(X_1, X_2 \mid Z = 1) = 0.496$
$\hat{Tau}(X_1, X_2 \mid Z = 0) = 0.519$
$\hat{P}(censoring) = 0.275$

F.10: BIVARIATE WEIBULL WITH A GAMMA FRAILTY

biv.weib.frailty(n.trt, n.cont, beta1, beta2, lambda1, lambda2, theta, alpha1, alpha2, alphaC, pcens)

temp<-biv.weib.frailty(200, 200, 0, -0.3646431, 0.007853982, 0.007853982, 2, 2, 2, 2, 0.2)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 12$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 9.539^2$
$Var(X_1 \mid Z = 0) = 9.539^2$
$Var(X_2 \mid Z = 1) = 11.446^2$
$Var(X_2 \mid Z = 0) = 9.539^2$
$Corr(X_1, X_2 \mid Z = 1) > 0$
$Corr(X_1, X_2 \mid Z = 0) > 0$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 9.123$
$\hat{E}(X_1 \mid Z = 0) = 10.854$
$\hat{E}(X_2 \mid Z = 1) = 12.083$
$\hat{E}(X_2 \mid Z = 0) = 10.613$
$\hat{V}ar(X_1 \mid Z = 1) = 8.177^2$
$\hat{V}ar(X_1 \mid Z = 0) = 10.611^2$
$\hat{V}ar(X_2 \mid Z = 1) = 10.811^2$
$\hat{V}ar(X_2 \mid Z = 0) = 10.546^2$
$\hat{C}orr(X_1, X_2 \mid Z = 1) = 0.609$
$\hat{C}orr(X_1, X_2 \mid Z = 0) = 0.634$
$\hat{P}(censoring) = 0.190$

F.11: BIVARIATE NORMAL

biv.norm<-function(n.trt, n.cont, beta1, beta2, mu1, mu2, var1, var2, lambda1, lambda2, rho, pcens1, pcens2)

temp<-biv.norm(200, 200, 0, 0.04879016, 10, 10, 10, 10, 0, 0, .5, 0.20, 0.20)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 10.5$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 10$
$Var(X_1 \mid Z = 0) = 10$
$Var(X_2 \mid Z = 1) = 10$
$Var(X_2 \mid Z = 0) = 10$
$Corr(X_1, X_2 \mid Z = 1) = 0.50$
$Corr(X_1, X_2 \mid Z = 0) = 0.50$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 10.054$
$\hat{E}(X_1 \mid Z = 0) = 9.620$
$\hat{E}(X_2 \mid Z = 1) = 10.426$
$\hat{E}(X_2 \mid Z = 0) = 10.000$
$\hat{V}ar(X_1 \mid Z = 1) = 10.659$
$\hat{V}ar(X_1 \mid Z = 0) = 12.764$
$\hat{V}ar(X_2 \mid Z = 1) = 8.971$
$\hat{V}ar(X_2 \mid Z = 0) = 10.581$
$\hat{C}orr(X_1, X_2 \mid Z = 1) = 0.451$
$\hat{C}orr(X_1, X_2 \mid Z = 0) = 0.554$
$\hat{P}(censoring) = 0.085$

F.12: BIVARIATE LOG-NORMAL

biv.lnorm<-function(n.trt, n.cont, beta1, beta2, mu1, mu2, var1, var2, lambda1, lambda2, rho, pcens1, pcens2)

temp<-biv.lnorm(200, 200, 0, 0.02110915, 2.287019, 2.287019, 0.03113307, 0.03113307, 0, 0, .5, 0.20, 0.20)

Generating data from a distribution with the following parameters:

$E(X_1 \mid Z = 1) = 10$
$E(X_1 \mid Z = 0) = 10$
$E(X_2 \mid Z = 1) = 10.5$
$E(X_2 \mid Z = 0) = 10$
$Var(X_1 \mid Z = 1) = 3.162$
$Var(X_1 \mid Z = 0) = 3.162$
$Var(X_2 \mid Z = 1) = 3.486$
$Var(X_2 \mid Z = 0) = 3.162$
$Corr(X_1, X_2 \mid Z = 1) = 0.50$
$Corr(X_1, X_2 \mid Z = 0) = 0.50$
$P(censoring) = 0.20$

Observed sample statistics:

$\hat{E}(X_1 \mid Z = 1) = 10.132$
$\hat{E}(X_1 \mid Z = 0) = 9.781$
$\hat{E}(X_2 \mid Z = 1) = 10.751$
$\hat{E}(X_2 \mid Z = 0) = 9.888$
$\hat{Var}(X_1 \mid Z = 1) = 3.157$
$\hat{Var}(X_1 \mid Z = 0) = 3.279$
$\hat{Var}(X_2 \mid Z = 1) = 3.629$
$\hat{Var}(X_2 \mid Z = 0) = 3.502$
$\hat{Corr}(X_1, X_2 \mid Z = 1) = 0.563$
$\hat{Corr}(X_1, X_2 \mid Z = 0) = 0.527$
$\hat{P}(censoring) = 0.123$

BIBLIOGRAPHY

Aalen, O. O. (1989) A linear regression model for analysis of lifetimes. *Statistics in Medicine* **8:** 907-925.

Abbring, J. H. & van den Berg, G. J. (2003) The identifiability of the mixed proportional hazards competing risks model. *Journal of the Royal Statistical Society, Series B* **65**: 701-710.

Agresti, A. (1990) Categorical Data Analysis. John Wiley and Sons: New York.

Andersen, P. K., Abildstrom, S. Z., & Rosthoj, S. (2002) Competing risks as a multi-state model. *Statistical Methods in Medical Research* **11**: 203-215.

Anderson, P. K., Klein, J. P., & Rosthoj, S. (2003) Generalized linear models for correlated pseudo-observations, with applications to multi-state models. *Biometrics* **90**: 15-27.

Barlow, R.E. & Proschan, F. (1981) Statistical Theory of Reliability and Life Testing. McArdle Press.

Benichou, J. & Gail, M.H. (1990) Estimates of Absolute Cause-Specific Risk in Cohort Studies. *Biometrics* **46**: 813-826.

Bernoulli, D. (1766) Essai d'une nouvelle analyse de la mortalité causée par la petite vérole. *Mém. Math. Phys. Acad. Roy. Sci., Paris*, 1-45. (English translation entitled 'An attempt at a new analysis of the mortality caused by smallpox and of the advantages of inoculation to prevent it' in: Bradley, L. (1971) Smallpox Inoculation: An Eighteenth Century Mathematical Controversy.)

Brent, R. (1973) Algorithms for Minimization without Derivatives. Prentice-Hall: New Jersey.

Cai, J. & Prentice, R.L. (1995) Estimating Equations for Hazard Ratio Parameters Based on Correlated Failure Time Data. *Biometrika* **82**: 151-164.

Casella, G. & Berger, R.L. (2002) Statistical Inference-2nd edition. Duxbury.

Clayton, D.G. (1978) A Model for Association in Bivariate Life Tables and its Application in Epidemiological Studies of Familial Tendency in Chronic Disease Incidence. *Biometrika* **65**: 141-151.

Clayton, D.G., Cuzick, J. (1985) Multivariate Generalizations of the Proportional Hazards Model. *Journal of the Royal Statistical Society, Series A* **148**: 82-117.

Cox, D. R. (1972) Regression models and life tables (with discussion). *Journal of the Royal Statistical Society, Series B* **34**: 187-220.

David, H. A. & Moeschberger, M. L. (1978) The theory of competing risks. Charles Griffin: London.

Devroye, L. (1986) Non-Uniform Random Variate Generation. Springer-Verlag: New York.

Escarela, G. & Carriere, J. F. (2003) Fitting competing risks with an assumed copula. *Statistical Methods in Medical Research* **12**: 333-349.

Fine, J. P. (2001) Regression modeling of competing crude failure probabilities. *Biostatistics* **2**: 85-97.

Fine, J. P. & Gray, R. J. (1999) A proportional hazards model for the sub-distribution of a competing risk. *Journal of the American Statistical Association* **94**: 496-509.

Freidlin, B. & Korn, E. (2005) Testing treatment effects in the presence of competing risks. *Statistics in Medicine* **24:** 1703-1712.

Gail, M. (1975) A review and critique of some models used in competing risk analysis. *Biometrics* **31**: 209-222.

Gichangi, A. & Vach, W. (2005) The analysis of competing risks data: A guided tour. *Statistics in Medicine*, under review.

Gilbert, P. B., Mckeague, I. W., & Sun, Y. (2004) Test for Comparing Mark-Specific Hazards and Cumulative Incidence Functions. *Lifetime Data Analysis* **10**: 5-28.

Gooley, T. A., Leisenring, W., Crowley, J., & Storer, B. E. (1999) Estimation of Failure Probabilities in the Presence of Competing Risks: new Representations of Old Estimators. *Statistics in Medicine* **18**:695-706.

Gray, B. (2004) The cmprsk Package.

Gray, R. J. (1988) A class of k-sample tests for comparing the cumulative incidence of a competing risk. *Annals of Statistics* **16**: 1141-1154.

Heckman, J. J. and Honore, B. E. (1989) The identifiability of the competing risks model, *Biometrika* **76**: 325-330.

Hoel, D.G. (1972) A representation of mortality data by competing risks. *Biometrics* **28**: 475-488.

Houggard, P. (2000) Analysis of Multivariate Survival Data. Springer-Verlag: New York.

Insightful Corporation. 1700 Westlake Ave. N., Suite 500. Seattle, WA 98109. http://www.insightful.com.

Jewell, N. P., van der Laan, M., & Henneman, T. (2003) Nonparametric estimation from current status data with competing risks. *Biometrika* **90**: 183-197.

Kalbfleisch, J. D. & Prentice, R. L. (2002) The Statistical Analysis of Failure Time Data- 2nd edition. John Wiley and Sons: New York.

Kim, H.T. (2007) Cumulative Incidence in Competing Risks Data and Competing Risks Regression Analysis. *Clinical Cancer Research* **13**: 559-565.

Klein, J. P. (2006) Modeling competing risks in cancer studies. *Statistics in Medicine* **25:** 1015-1034. John Wiley and Sons.

Klein, J. & Andersen, P.K. (2005) Regression modeling of competing risks data based on pseudo values of the cumulative incidence function. *Biometrics* **61**: 223-229.

Klein, J.P. & Moeschberger, M.L. (1988) Bounds on Net Survival Probabilities for Dependent Competing Risks. *Biometrics* **44**: 529-538.

Klein, J.P. & Moeschberger, M.L. (2003) Survival Analysis: Statistical Methods for Censored and Truncated Data-2nd edition. Springer: New York.

Kotz, S., Balakrishnan, N., Johnson, N.L. (2000) Continuous Multivariate Distributions-2nd edition. John Wiley and Sons: New York.

Latouche, A., Porcher, R., & Chevret, S. (2004) Sample size formula for proportional hazards modeling of competing risks. *Statistics in Medicine* **23**: 3263-3274.

Lee, M.T. (1996) Properties and Applications of the Sarmanov Family of Bivariate Distributions. *Communications in Statistics-Theory and Methods* **25**: 1207-1222.

Lunn, M. & McNeil D. (1995) Applying Cox Regression to Competing Risks. *Biometrics* **51**: 524-532.

Mantel, N. (1966) Evaluation of Survival Data and Two New Rank Order Statistics Arising in its Consideration. *Cancer Chemotherapy Reports* **50**: 163-170.

Marubini, E. & Valsecchi, M.G. (1995) Analyzing Survival Data from Clinical Trials and Observational Studies. John Wiley and Sons: New York.

McKeague, I.W. (1988) Asymptotic theory for weighted least-squares estimators in Aalen's additive risk model. *Contemporary Mathematics* **80**: 139-152.

Moeschberger, M.L. (1974) Life Tests under Dependent Competing Causes of Failure. *Technometrics* **16**: 39-47.

Pepe, M.S. (1991) Inference for Events with Dependent Risks in Multiple Endpoint Studies. *Journal of the American Statistical Association* **86**: 770-778.

Pepe, M.S. & Mori, M. (1993) Kaplan-Meier, marginal or conditional probability curves in summarizing competing risks failure time data? *Statistics in Medicine* **12**: 737-751.

Pintilie, M. (2002) Dealing with competing risks: testing covariates and calculating sample size. *Statistics in Medicine* **21**: 3317-3324.

Prentice, R.L., Cai, J. (1992) Covariance and Survivor Function Estimation Using Censored Multivariate Failure Time Data. *Biometrika* **79**: 495-512.

Prentice, R.L., Kalbfleisch, J.D., Peterson, A.V., Flournoy, N., Farewell, V.T. & Breslow, N.E. (1978) The analysis of failure times in the presence of competing risks. *Biometrics* **34**: 541-554.

Rosthoj, S., Andersen, P. K., & Abildstrom, S. Z. (2004) SAS macros for estimation of the cumulative incidence functions based on a Cox regression model for competing risks survival data. *Computer Methods and Programs in Biomedicine* **74**: 69-75.

SAS Institute Inc. 100 SAS Campus Drive. Cary, NC 27513. http://www.sas.com.

Satagopan, J.M., Ben-Porat, L., Berwick, M., Robson, M., Kutler D., & Auerbach, A. D. (2004) A note on competing risks in survival data analysis. *British Journal of Cancer* **91**: 1229-1235.

StataCorp LP. 4905 Lakeway Drive. College Station, TX 77845. http://www.stata.com.

Steele, F. Goldstein, H., & Brown, W. (2004) A general multilevel multistate competing risks model for event history data, with an application to a study of contraceptive use dynamics. *Statistical Modeling* **4**: 145-159.

Sun, J., Sun, L. & Flournoy, N. (2004) Additive hazards model for competing risks analysis of the case-cohort design. *Communication in Statistics* **33**: 351-366.

Tai B-C, Machin, D., White, I., & Gebski V. (2001) Competing risks analysis of patients with osteosarcoma: a comparison of four different approaches. *Statistics in Medicine* **20**: 661-684.

The R Foundation for Statistical Computing. Technische Universitat Wien. 1040 Vienna, Austria. http://www.r-project.org/.

Tsiatis, A. A. (1975) Aspect of the problem of competing risks. *Proceedings of the National Academy of Sciences* **72**: 20-22.