Journal of Statistical Software

September 2023, Volume 107, Issue 2. doi: 10.18637/jss.v107.i02

carat: An R Package for Covariate-Adaptive Randomization in Clinical Trials

Wei Ma
Renmin University of China

Xiaoqing Ye
Renmin University of China

Fuyi Tu
Renmin University of China

Feifang Hu
George Washington University

Abstract

Covariate-adaptive randomization is gaining popularity in clinical trials because they enable the generation of balanced allocations with respect to covariates. Over the past decade, substantial progress has been made in both new innovative randomization procedures and the theoretical properties of associated inferences. However, these results are scattered across the literature, and a single tool kit does not exist for use by clinical trial practitioners and researchers to conduct and evaluate these methods. The R package **carat** is proposed to address this need. It facilitates a broad range of covariate-adaptive randomization and testing procedures, such as the most common and classical methods, and also reflects recent developments in the field. The package contains comprehensive evaluation and comparison tools for use in both randomization procedures and tests. This enables power analysis to be conducted to assist the planning of a covariate-adaptive clinical trial. The package also implements a command-line interface to allow for an interactive allocation procedure, which is typically the case in real-world applications. In this paper, the features and functionalities of **carat** are presented.

Keywords: **carat**, clinical trial, covariate-adaptive randomization, hypothesis test, command-line interface, power analysis, R, **Rcpp**, **OpenMP**.

1. Introduction

Covariate-adaptive randomization is a class of randomization procedures in which allocation probabilities are sequentially modified to achieve balanced covariates across different treatment groups. A clinical trial with well-balanced covariates enhances the credibility of analysis and improves statistical efficiency (Kundt 2009). Therefore, covariate-adaptive randomization has become increasingly used in clinical trials. In a survey of 224 randomized clinical trials published in 2014 in leading medical journals, 183 (82%) were found to have used covariate-adaptive randomization procedures (Lin, Zhu, and Su 2015). These procedures were also

used in recent COVID-19 treatment and vaccine trials (Wang *et al.* 2020; Baden *et al.* 2021). Stratified randomization is the most common and straightforward way of obtaining a balanced allocation, in which a restricted randomization procedure is applied within each stratum created by grouping patients' covariate values. Among stratified randomization methods (for example, Shao, Yu, and Zhong 2010; Baldi Antognini and Zagoraiou 2011), stratified permuted block randomization (Zelen 1974) is the most commonly used randomization method in clinical trials. Minimization (Taves 1974; Pocock and Simon 1975) is an alternative method that is used when balance over covariates' margins is desired, and especially when the number of covariates is relatively large compared to the sample size. More recently, Hu and Hu (2012) have proposed a general family of covariate-adaptive designs to control imbalances of different types (within-stratum, within-covariate-margin, and overall). Covariate balance is also approached with the use of optimal design theory (Atkinson 1982). In addition, model-based approaches have been proposed to improve efficiency with the use of optimum design theory (Atkinson 1982; Atkinson 1999; Smith 1984; Begg and Kalish 1984). However, model-based approaches may not necessarily lead to balance in some cases, and they are less intuitive for practical use (Rosenberger and Sverdlov 2008; Hu and Hu 2012). In this paper, of all the model-based approaches, we only consider the Atkinson's D_A-optimal biased coin design (Atkinson 1982), which leads to a balanced allocation. We provide a brief introduction of all the considered methods in Section 2.1, but refer readers to Rosenberger and Lachin (2015) for a more comprehensive review.

Due to the increased popularity of covariate-adaptive randomization in practice and the development of new methodologies in the literature, there is a pressing need to provide clinical trial designers with easy-to-use software that covers a broad range of covariate-adaptive randomization procedures, and offers tools for users to evaluate and compare the performance of different methods, to select the best one according to their needs. However, most currently available randomization tools focus on complete randomization and block randomization, including standalone software such as **Clinstat** (Bland 2004) and **Random allocation software** (Saghaei 2004), web-based allocation systems such as `Randomization.com` (Dallal 2003) and GraphPad **QuickCalcs** (GraphPad Software Inc. 2017), R packages (R Core Team 2023) such as **blockrand** (Greg 2020), **randomizr** (Coppock, Cooper, and Fultz 2023), and **randomizeR** (Schindler, Uschner, Hilgers, and Heussen 2023), and Stata modules such as **ralloc** (Ryan 2018). However, the aforementioned tools do not include covariate-adaptive randomization, which achieves covariate balance and is preferred in practice. Of the tools available for covariate-adaptive randomization, the choices are limited to Pocock and Simon's minimization, including programs such as **Minim** (Evans, Day, and Royston 1995) and **Minimpy** (Saghaei and Saghaei 2011), a web-based system **MagMin** (Cai, Xia, Gao, and Cao 2010), and a **Bioconductor** package (Gentleman *et al.* 2004) **randPack** (Carey and Gentleman 2023). Moreover, no assessment tools are provided, so it is not possible to directly compare different randomization procedures. The package **carat** is designed to meet this need by comprising a broad spectrum of covariate-adaptive randomization procedures, including both classical and newly developed methods. This package is equipped with comprehensive tools for assessment and comparison of various randomization procedures in terms of their balancing performances. A command-line user interface is also implemented to handle the adaptive feature of the randomization procedures, and to facilitate real-world applications. Package **carat** (Tu, Ye, Ma, and Hu 2023) is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=carat`.

In addition to being a comprehensive tool for conducting covariate-adaptive randomization from the design aspects, the package **carat** is also created to enable reliable statistical inference following those procedures, which is often a missing component of the aforementioned packages. There have been concerns about whether standard statistical models can adequately reflect a complicated randomization scheme under covariate-adaptive designs. For example, even the usual two-sample t test is conservative under the covariate-adaptive biased coin design (Shao *et al.* 2010). However, over the past decade, several valid and robust testing methods for detecting treatment effects under covariate-adaptive randomization have been proposed, such as the bootstrap t test (Shao *et al.* 2010; Shao and Yu 2013), the corrected t test (Ma, Hu, and Zhang 2015; Ma, Qin, Li, and Hu 2020), and the randomization test (Rosenberger, Uschner, and Wang 2019), as outlined in Section 2.3. Unfortunately, these methods are scattered across the literature, and there is no single tool kit for both practitioners and researchers to use to perform and evaluate these methods. Some existing methods in R packages for bootstrapping and stratified permutation tests such as **boot** (Canty and Ripley 2022) and **coin** (Hothorn, Hornik, Van de Wiel, and Zeileis 2006) are not suitable for making statistical inferences under covariate-adaptive randomization because they assume that the outcomes are independent. However, the outcomes are correlated under covariate-adaptive randomization and treatment reassignment is required for valid tests. Our package provides functions for the tests associated with covariate-adaptive randomization, while being cognizant of the fact that the field is still expanding. We believe that these tools will promote the acceptance of covariate-adaptive randomization, provide easy access to valid ways of analyzing data from such trials, stimulate related research by providing benchmarks for the comparison of different tests, and enable the development of more innovative statistical methods.

Moreover, building upon the tools for conducting a variety of randomization and testing procedures, the package **carat** is equipped with comprehensive power analysis tools based on the Monte Carlo method. Users can thus easily evaluate the performance of any included test under different data generation mechanisms and randomization methods. This is particularly important in the planning stage of a clinical trial to determine a proper sample size. The package provides complete solutions and is a one-stop source for planning, conducting, and analyzing covariate-adaptive clinical trials. Finally, the computational tasks in the package are not trivial because of iterative updates during the process of adaptive randomization and sampling-based tests. This is most apparent when performing the Monte Carlo simulations. The integration of R with C++ (Stroustrup 2013) via **Rcpp** (Eddelbuettel and François 2011; Eddelbuettel *et al.* 2023) and parallel computing tools **OpenMP** (Dagum and Menon 1998) are extensively used throughout the package to provide computationally efficient functions and hence a better user experience. See Section 5 for further discussion.

The paper is organized as follows. In Section 2, we give an overview of the relevant statistical background for covariate-adaptive randomization procedures and associated hypothesis testing. Section 3 provides the description of our package **carat** and illustrations of some examples. In Section 4, we apply the package **carat** to the cognitive behavioral-analysis system of psychotherapy (CBASP) clinical trial data. Computational issues are discussed in Section 5, before concluding with a summary and discussion of possible extensions in Section 6.

2. Statistical background

The **carat** package constitutes a comprehensive tool kit for designing and analyzing covariate-adaptive randomized clinical trials. In this section, we provide a brief overview of the necessary statistical background material, with reference to the original literature. In Section 2.1, we introduce the covariate-adaptive randomization procedures implemented in the package. We describe the criteria for the evaluation of various randomization procedures in Section 2.2, we detail three valid and robust methods for testing treatment effects under covariate-adaptive randomization in Section 2.3, and we briefly present power calculations in Section 2.4.

2.1. Covariate-adaptive randomization

We consider the case of two-armed clinical trials conducted with covariate-adaptive randomization, and use notations as in Hu and Hu (2012). Suppose there are I covariates and m_i levels for the i-th covariate, and hence in total $m = \prod_{i=1}^{I} m_i$ strata. Let T_j be the assignment of the j-th patient, $j = 1, \ldots, n$, such that $T_j = 1$ for treatment 1 and $T_j = 0$ for treatment 2. Let Z_j indicate the covariate profile of the j-th patient, such that $Z_j = (k_1, \ldots, k_I)^\top$ if the i-th covariate is at level k_i, $1 \leq i \leq I$ and $1 \leq k_i \leq m_i$. For notational simplicity, let $N_n^{(t)}$, $N_n^{(t)}(i; k_i)$ and $N_n^{(t)}(k_1, \ldots, k_I)$, $t = 1, 2$, be the overall number of patients, the number of patients with level k_i of the i-th covariate and the number of patients in the stratum $(k_1, \ldots, k_I)$ assigned to treatment t after n patients. Then, define $D_n = N_n^{(1)} - N_n^{(2)}$ as the imbalance between the two treatments. The within-stratum imbalance $D_n(k_1, \ldots, k_I)$ and the within-covariate-margin imbalance $D_n(i; k_i)$ are defined similarly. Furthermore, if the n-th patient were assigned to treatment 1, then $D_n^{(1)} = D_{n-1} + 1$ would be the "potential" overall difference in the two groups; similarly, $D_n^{(1)}(i; k_i) = D_{n-1}(i; k_i) + 1$ and $D_n^{(1)}(k_1, \ldots, k_I) = D_{n-1}(k_1, \ldots, k_I) + 1$ would be the potential differences on margin $(i; k_i)$ and within stratum $(k_1, \ldots, k_I)$, respectively. Also, if the n-th patient were assigned to treatment 2, the three types of potential differences $D_n^{(2)}$, $D_n^{(2)}(i; k_i)$, and $D_n^{(2)}(k_1, \ldots, k_I)$ are defined analogously with $+1$ replaced by -1.

In the literature, a variety of covariate-adaptive randomization procedures are proposed to reduce different aspects of these imbalances with respect to the covariates. Among all the implemented procedures, Hu and Hu's general covariate-adaptive randomization (Hu and Hu 2012) takes all three aspects of imbalance (within-stratum, within-covariate-margin, and overall) into account, while Pocock and Simon's minimization (Pocock and Simon 1975) and stratified biased coin design (Shao *et al.* 2010) can be considered as two special cases of Hu and Hu's general method and target the within-covariate-margin and within-stratum imbalances, respectively. Stratified randomization methods also include stratified permuted block randomization (Zelen 1974) and covariate-adjusted biased randomization (Baldi Antognini and Zagoraiou 2011). In addition, Atkinson's D_A-optimal biased coin design (Atkinson 1982) exemplifies the class of model-based approaches. This method achieves covariate balance by minimizing the variance of estimated treatment effects, rather than by directly minimizing any specific imbalance measure.

Hu and Hu's general covariate-adaptive randomization

In Hu and Hu (2012), a general family of covariate-adaptive randomization procedures is proposed to control various types of imbalances (within-stratum, within-covariate-margin,

and overall). If the $(n+1)$-th patient were allocated to treatment t, $t = 1, 2$, as indicated by the superscript, the "potential" imbalance measure would be defined as

$$\text{Imb}_{n+1}^{(t)} = \omega_o[D_{n+1}^{(t)}]^2 + \omega_s[D_{n+1}^{(t)}(k_1, \ldots, k_I)]^2 + \sum_{i=1}^{I} \omega_{m,i}[D_{n+1}^{(t)}(i; k_i)]^2,$$

where ω_o, ω_s and $\omega_{m,i}$ are the non-negative weights placed on overall imbalance, within-stratum imbalance, and within-covariate-margin imbalance, respectively. The subscript m attached to a weight $\omega_{m,i}$, $i = 1, \ldots, I$, indicates that the weight is imposed on a margin. For simplicity but without loss of generality, it is assumed that $\omega_o + \omega_s + \sum_{i=1}^{I} \omega_{m,i} = 1$ in Hu and Hu (2012). In our package, the weights are normalized so that their sum equals 1 if the assumption is not satisfied. The general guidance on the choice of weights is that if a covariate is considered important, more weight can be imposed on the within-stratum and within-covariate-margin imbalances of the covariate. Then, conditional on the assignments and the covariate profiles of the first n patients, the procedure assigns the $(n+1)$-th patient to treatment 1 with the probability

$$\mathsf{P}(T_{n+1} = 1 \mid \mathbf{Z}_{n+1}, \boldsymbol{T}_n) = \begin{cases} q, & \text{if } \text{Imb}_{n+1}^{(1)} > \text{Imb}_{n+1}^{(2)}, \\ p, & \text{if } \text{Imb}_{n+1}^{(1)} < \text{Imb}_{n+1}^{(2)}, \\ 0.5, & \text{otherwise}, \end{cases}$$

where $n \geq 1$, $0 < q < p < 1$, $p + q = 1$, $\mathbf{Z}_n = (Z_1, \ldots, Z_n)$ and $\boldsymbol{T}_n = (T_1, \ldots, T_n)^\top$. The literature suggests that larger values of p, such as 0.85, 0.90, and 0.95, should be used for covariate-adaptive randomization. See Hu and Hu (2012) and the references therein for details.

Pocock and Simon's minimization

Pocock and Simon's minimization (Pocock and Simon 1975) is one of the most classical and widely used covariate-adaptive randomization methods. Supposing that n patients have been assigned and the covariate profile of the $(n+1)$-th patient is $Z_{n+1} = (k_1, \ldots, k_I)^\top$, Pocock and Simon (1975) defined a total amount of imbalance, $G^{(t)}$, $t = 1, 2$, that would result if the new patient was assigned to treatment t. This is expressed in the form

$$G^{(t)} = G(D_{n+1}^{(t)}(i; k_i), i = 1, \ldots, I) = \sum_{i=1}^{I} \omega_i d(D_{n+1}^{(t)}(i; k_i)),$$

where $d(\cdot)$ is a function of the potential within-covariate-margin difference $D_{n+1}^{(t)}(i; k_i)$, and ω_i is the non-negative weight placed on the i-th margin, $i = 1, \ldots, I$. The consideration of $\omega_i, i = 1, \ldots, I$, is similar to that of the weights for Hu and Hu's general covariate-adaptive randomization, and more weight can be assigned to the within-covariate-margin imbalances of the important covariates. Then, the new patient is assigned to treatment 1 with the probability

$$\mathsf{P}(T_{n+1} = 1 \mid \mathbf{Z}_{n+1}, \boldsymbol{T}_n) = \begin{cases} q, & \text{if } G^{(1)} > G^{(2)}, \\ p, & \text{if } G^{(1)} < G^{(2)}, \\ 0.5, & \text{otherwise}, \end{cases}$$

where p and q satisfy the same conditions as above. In our package, Pocock and Simon's minimization is specified by choosing $d(\cdot)$ as the square function, and thus reduces to a special case of Hu and Hu's general covariate-adaptive randomization with $\omega_o = \omega_s = 0$.

Stratified biased coin design

Shao *et al.* (2010) apply the biased coin design (Efron 1971) to assign the $(n+1)$-th patient with the covariate profile $Z_{n+1} = (k_1, \ldots, k_I)$ based on the current imbalance level within the same stratum. The probability of assigning the patient to treatment 1 is

$$\mathsf{P}(T_{n+1} = 1 \mid \mathbf{Z}_{n+1}, \boldsymbol{T}_n) = \begin{cases} q, & \text{if } (D^{(1)}_{n+1}(k_1, \ldots, k_I))^2 > (D^{(2)}_{n+1}(k_1, \ldots, k_I))^2, \\ p, & \text{if } (D^{(1)}_{n+1}(k_1, \ldots, k_I))^2 < (D^{(2)}_{n+1}(k_1, \ldots, k_I))^2, \\ 0.5, & \text{otherwise,} \end{cases}$$

where the same conditions are imposed on p and q. It is easily seen that the stratified biased coin design is also a special case of Hu and Hu's general covariate-adaptive randomization with $\omega_o = \omega_{m,i} = 0,\ i = 1, \ldots, I$. In this package, the default value of p is set to be 0.85 for the above three methods.

Stratified permuted block randomization

Stratified permuted block randomization (Zelen 1974) is the most commonly used method in clinical trials to achieve balance with respect to the covariates, for which permuted block randomization is used within each stratum. In this package, we consider permuted block randomization with fixed block sizes; for example, the default block size is set to be 4, which is commonly used in practice, although other values can also be used. The method works most efficiently when the number of strata is small (Kalish and Begg 1985). However, it may cause severe overall imbalance when there are too many strata, in which case the within-covariate-margin approaches may be preferred (Hu and Hu 2012).

Covariate-adjusted biased coin design

Covariate-adjusted biased coin design (Baldi Antognini and Zagoraiou 2011) is a new class of stratified randomization methods. Suppose that n patients have been assigned and the covariate profile of the $(n+1)$-th patient is $Z_{n+1} = (k_1, \ldots, k_I)$. Then, conditional on the assignments and the covariate profiles of the first n patients, the new patient is assigned to treatment 1 with the probability

$$\mathsf{P}(T_{n+1} = 1 \mid \mathbf{Z}_{n+1}, \boldsymbol{T}_n) = F(D_n(k_1, \ldots, k_I)),\ 1 \le k_i \le m_i,\ i = 1, \ldots, I,$$

where the generating function $F(\cdot)$, as a map from integers to $[0, 1]$, is a decreasing and symmetric function with $F(x) = 1 - F(-x)$. In this package, the same generating function is applied across all strata, which is given by

$$F^a(x) = \begin{cases} 1/2, & \text{if } x = 0, \\ (x^a + 1)^{-1}, & \text{if } x \ge 1. \end{cases}$$

The parameter $a > 0$ controls the degree of randomness, and thus the method tends toward complete randomization as $a \to 0$, and the assignment becomes more deterministic as $a \to \infty$. In this package, the default value of a is 3. An alternative choice of a is $t^{-1} - 1$, where t is the reciprocal of the number of strata (Baldi Antognini and Zagoraiou 2011).

Atkinson's D_A-optimal biased coin design

Atkinson's D_A-optimal biased coin design (Atkinson 1982) is a model-based approach to balance allocations across covariates. By assuming a linear model between response and

covariates, the design sequentially assigns patients to minimize the variance of estimated treatment effects. Supposing n patients have already been assigned, the $(n+1)$-th patient is assigned to treatment 1 with the probability (Smith 1984)

$$\frac{\left[1-(1;Z_{n+1})(\mathbf{F}_n^\top\mathbf{F}_n)^{-1}\mathbf{b}_n\right]^2}{\left[1-(1;Z_{n+1})(\mathbf{F}_n^\top\mathbf{F}_n)^{-1}\mathbf{b}_n\right]^2+\left[1+(1;Z_{n+1})(\mathbf{F}_n^\top\mathbf{F}_n)^{-1}\mathbf{b}_n\right]^2},$$

where $\mathbf{F}_n = (\mathbf{1}_n;\ \mathbf{Z}_n^\top)$, and $\mathbf{b}_n^\top = (2\boldsymbol{T}_n - \mathbf{1}_n)^\top\mathbf{F}_n$.

2.2. Evaluation of covariate-adaptive randomization

The primary goal of using covariate-adaptive randomization in practice is to achieve balance both with respect to the key covariates and the overall treatment assignments within a clinical trial. The enhanced similarity of covariates between different treatment groups also gains statistical efficiency or power (Kundt 2009). Therefore, when it comes to evaluating covariate-adaptive randomization methods, the balancing behavior of the method is arguably the most relevant measure. This is reflected in the development of our package. As it is often the case that multiple discrete covariates are used in randomization, different levels of imbalance need to be considered, such as within-stratum, within-covariate-margin, and overall imbalances. These measures are extensively used in the literature to evaluate and compare covariate-adaptive randomization methods (e.g., Hu and Hu 2012). Other criteria that used for evaluation of general randomization methods, such as selection bias, are of less concern when implementing covariate-adaptive randomization, especially in a double-blinded setting. These criteria are therefore not included in the current version of the package.

The Monte Carlo method provides a relatively straightforward and powerful way to quantify the balancing behavior of covariate-adaptive randomization methods. We use this method in our package, as it is preferable to the theoretical description because of the following reasons. First, rather than assessing the imbalance with a single quantity (e.g., variance), a Monte Carlo simulation experiment is used to determine four quantities (i.e., maximal, 95% quantile, median and mean) of the absolute value of the imbalance, which has been acknowledged to be a better solution for the process of estimation with considerable uncertainty (Metropolis and Ulam 1949). The whole distribution of imbalance is easily illustrated with the aid of visualization tools such as box plots, as detailed in Section 3.3. Second, although the theoretical properties are well studied for some randomization methods (e.g., Baldi Antognini and Zagoraiou 2011; Hu and Hu 2012), the balancing properties are less clear for other methods, such as Pocock and Simon's minimization. Even for those well-studied methods, the theoretical properties rely on asymptotic theory and may not be appropriate empirically. These issues can be resolved by using the Monte Carlo method. Moreover, the method can be readily applied for a newly proposed covariate-adaptive randomization method, which allows the package to extend more easily. Lastly, the computationally intensive issues caused by Monte Carlo simulations can be reduced by modern computational tools, such as **Rcpp** and **OpenMP**, as detailed in Section 5.

2.3. Inference under covariate-adaptive randomization

Detecting treatment effect is usually the primary interest in covariate-adaptive clinical trials. However, some researchers have questioned whether some classical statistical methods remain

valid under covariate-adaptive randomization. For example, Shao *et al.* (2010) proved that under the stratified biased coin design, the usual two-sample t test is conservative, in the sense that it is less likely to be rejected under the null hypothesis than the nominal level, and a bootstrap t test was used to correct the Type I error. Over the past decade, significant progress has been made in determining the theoretical basis of inference under covariate-adaptive randomization. In particular, many valid hypothesis testing methods have been proposed for a broad range of covariate-adaptive randomization procedures. However, the results are scattered throughout the literature, and a comprehensive comparison among these methods is lacking. The package **carat** offers three tests to compare inference methods, thereby enabling valid and robust inferences under covariate-adaptive randomization to be drawn from clinical trial data. This is reviewed in this section. Users can apply any other standard methods, such as the t test or ANOVA, to analyze data from a covariate-adaptive randomized clinical trial. However, it is recommended that a comparison between different tests, be conducted using the three methods included in the package, to reveal which one is best in particular settings. For details, see Section 2.4.

Bootstrap t test

The general t test is of the form

$$\frac{\bar{Y}_1 - \bar{Y}_0}{\sqrt{\widehat{\mathrm{Var}}(\bar{Y}_1 - \bar{Y}_0)}},$$

where $\bar{Y}_1$ and $\bar{Y}_0$ are the sample means of two treatment groups. However, it is shown in Shao *et al.* (2010) that the naive variance estimator used in the usual t test overestimates the true variance of $\bar{Y}_1 - \bar{Y}_0$ due to the dependence of the two sample means induced by covariate-adaptive randomization. Therefore, it is critical to have a proper variance estimator to ensure a valid Type I error rate is obtained. For this, Shao *et al.* (2010) proposed a bootstrap method to approximate the variance of $\bar{Y}_1 - \bar{Y}_0$, as described below. Another model-based approach is described in the next section.

1) Generate bootstrap data $(Y_1^*, Z_1^*), \ldots, (Y_n^*, Z_n^*)$ as a simple random sample with replacement from the original data $(Y_1, Z_1), \ldots, (Y_n, Z_n)$, where Y_i denotes the outcome, and Z_i denotes the covariate profile of the i-th patient.

2) Apply the same covariate-adaptive randomization procedure on $Z_1^*, \ldots, Z_n^*$ to obtain the bootstrap analogs of treatment assignments $T_1^*, \ldots, T_n^*$, and define

$$\hat{\theta}^* = \frac{1}{n_1^*} \sum_{i=1}^{n} Y_i^* T_i^* - \frac{1}{n_0^*} \sum_{i=1}^{n} Y_i^* (1 - T_i^*),$$

where n_1^* is the number of patients assigned to treatment 1, and n_0^* is the number of patients assigned to treatment 2 in a bootstrap sample.

3) Repeat step 2 for B times, and generate B independent bootstrap samples to obtain $\hat{\theta}_b^*, b = 1, \ldots, B$. Then, the variance of $\bar{Y}_1 - \bar{Y}_0$ is estimated by the sample variance of $\hat{\theta}_b^*$, and a normal approximation is used to determine the p value.

This method depends on the reassignment of treatments, and in this way it differs from classical bootstrap methods.

Corrected t test

Another approach to modifying the variance estimator used for the t test is to rely on the asymptotic distribution of $\bar{Y}_1 - \bar{Y}_0$ under a covariate-adaptive randomization procedure. In the linear model framework, the asymptotic normality of $\bar{Y}_1 - \bar{Y}_0$ is established by Ma *et al.* (2015), provided that the overall imbalance and within-covariate-margin imbalances for all covariates are bounded in probability. The balancing properties are satisfied by a broad range of covariate-adaptive randomization procedures, including all of the procedures involved in the package except for Atkinson's D_A-optimal biased coin design. The corrected t test has been used in several recent papers (e.g., Zhu and Hu 2019; Yu and Lai 2019; Ma *et al.* 2020) and has been shown to be robust to model misspecification. It is also more powerful than other tests, such as randomization tests.

Randomization test

The randomization test is an alternative method to preserve the Type I error rate under covariate-adaptive randomization. By fixing patients' covariate profiles and outcomes, it ensures that each time the covariate-adaptive randomization is performed there will be different assignments for the patients, and hence different values of a given test statistic. We can generate the null distribution of the test statistic by repetition of the randomization procedure and reject the null hypothesis if the observed test statistic lies in the extreme section of all possible values. However, performing the exact randomization test can be a large computational burden when n is large. Hence, the usage of finite replications to approximate the distribution of test statistics is preferred in practice (Rosenberger and Lachin 2015). Unlike the model-based tests, this test contains no assumption about the underlying model, although it is more computationally intensive and may cause a loss of power.

The randomization test used in **carat** is described as follows:

1) For the observed responses $Y_1, \ldots, Y_n$ and the treatment assignments $T_1, \ldots, T_n$, compute the observed value for the test statistic

$$S_{\text{obs}} = \frac{1}{n_1} \sum_{i=1}^{n} Y_i T_i - \frac{1}{n_0} \sum_{i=1}^{n} Y_i (1 - T_i),$$

where n_1 is the number of patients assigned to treatment 1, and n_0 is the number of patients assigned to treatment 2.

2) Perform the covariate-adaptive randomization procedure on the same patients' covariate profiles to obtain new treatment assignments and calculate the corresponding test statistic values S_l, and repeat L times.

3) Calculate the two-sided p value

$$p = \frac{1}{L} \sum_{l=1}^{L} I(|S_l| \geq |S_{\text{obs}}|),$$

and reject the null hypothesis if p has an extreme value.

2.4. Power calculation

As there are several tests that guarantee control of the Type I error rates, an important way of assessing their relative performance is to calculate and compare their power under different

scenarios. Moreover, such a power calculation is important in planning a clinical trial because it gives a basis for choosing the sample size required to determine a meaningful treatment effect. In the package **carat**, the Type I error, as well as power, is calculated through the Monte Carlo method. The current version supports two types of endpoints (continuous and binary), and the corresponding data-generating models:

- Linear model for continuous endpoints:

$$Y_i = \mu_1 T_i + \mu_2(1 - T_i) + \beta^\top Z_i + \epsilon_i,$$

- Logit model for binary endpoints :

$$\text{logit}\{\mathsf{P}(Y_i = 1 \mid Z_i)\} = \mu_1 T_i + \mu_2(1 - T_i) + \beta^\top Z_i,$$

where μ_1 and μ_2 are the main effects of two treatments, β is the vector of coefficients for the covariates, and ϵ_i is the random error. Additional data types may be added in later versions of the package.

3. The carat package

3.1. Overview

The **carat** package comprises 27 functions. These functions are implemented to generate randomization sequences, evaluate and compare different randomization procedures, and enable reliable statistical inferences in the framework of covariate-adaptive randomization. Package **carat** joins a growing list of tools for realizing randomized clinical trials; however, it offers a comprehensive and single package for covariate-adaptive randomization.

Table 1 illustrates the functions and methods available in the **carat** package, which consists of two closely connected parts: The design and analysis of covariate-adaptive randomized clinical trials. The first part, design of clinical trials, includes functions for generating randomization sequences using one of the covariate-adaptive randomization procedures, with complete covariate data or a covariate data-generating mechanism. Especially, **carat** offers a series of command-line user interface functions for sequentially accrued covariate data. In addition to implementing randomization procedures, the package also provides functions for assessing and comparing of different covariate-adaptive randomization procedures.

In the second part, analysis of clinical trials, the package provides three hypothesis testing methods proposed specifically for covariate-adaptive randomization; these methods facilitate the inference of treatment effects under the included randomization procedures. The package also provides functions for power analysis and visualization to facilitate the evaluation of the performance of these tests under different randomization schemes.

Rather than sealing the parameters in an input-generating mechanism, we directly view the parameters as the input of the functions in **carat**. From Table 2, we observe that the input of each main function comprises two parts: The common and specific parameters. Details of these parameters are listed in the rest of this section. All of the main results in **carat** are implemented using the S3 method. For the functions to generate randomization sequences, the outputs are printed in our own defined class '`carandom`', as Table 2 outlined.

	Usage	Functions
Design	Randomization sequence generation with complete covariate data	`HuHuCAR()`, `PocSimMIN()`, `StrBCD()`, `StrPBR()`, `AdjBCD()`, `DoptBCD()`
	Randomization sequence generation with a covariate data-generating mechanism	`HuHuCAR.sim()`, `PocSimMIN.sim()`, `StrBCD.sim()`, `StrPBR.sim()`, `AdjBCD.sim()`, `DoptBCD.sim()`
	Command-line user interface	`HuHuCAR.ui()`, `PocSimMIN.ui()`, `StrBCD.ui()`, `StrPBR.ui()`, `AdjBCD.ui()`, `DoptBCD.ui()`
	Evaluation of randomization procedures	`evalRand()`, `evalRand.sim()`
	Comparison of randomization procedures	`compRand()`
Analysis	Data generation	`getData()`
	Statistical inference	`boot.test()`, `corr.test()`, `rand.test()`
	Power analysis	`evalPower()`, `compPower()`

Table 1: Functions and S3 methods in the **carat** package.

As for the evaluation functions, the outputs are sealed in our own defined class '`careval`' in **carat**. The objects outputted as '`careval`' class are directly applied to the comparison functions. It is presumably easy to extend our package using these classes. When a new covariate-adaptive randomization procedure is proposed, we can print the randomization sequences and the procedure's evaluation in the classes '`carandom`' and '`careval`', respectively. Naturally, the newly proposed randomization procedure can be compared with the existing randomization schemes by the comparison functions in **carat**. Motivated by the function `t.test()` in the base-R package **stats**, the results of the testing functions are printed as the class '`htest`', wherein the outputs of many other commonly used inference functions such as `wilcox.test()` and `shapiro.test()` are printed. Therefore, not only can our results be easily compared with the results obtained from existing inference functions in R but they also

	Randomization/test	Functions	Specific parameters	Common parameters	Output/S3 method
Design	Hu and Hu's randomization	`HuHuCAR()` `HuHuCAR.sim()`	`omega`, `p`	`data` for `Design`; `n`, `cov_num`, `level_num`, and `pr` for `Design.sim`	'carandom'
	Pocock and Simon's minimization	`PocSimMIN()` `PocSimMIN.sim()`	`weight`, `p`		
	Stratified biased coin design	`StrBCD()` `StrBCD.sim()`	`p`		
	Stratified permuted block randomization	`StrPBR()` `StrPBR.sim()`	`bsize`		
	Covariate-adjusted biased coin design	`AdjBCD()` `AdjBCD.sim()`	`a`		
	Atkinson's D_A-optimal biased coin design	`DoptBCD()` `DoptBCD.sim()`	/		
Test	Bootstrapt t-test	`boot.test()`	`B`	`data`, `method`, to be passed to methods. `conf`, `...`; Note that `...` are arguments	'htest'
	Corrected t-test	`corr.test()`	/		
	Randomization test	`rand.test()`	`Reps`, `plot`, `binwidth`		

Table 2: The inputs and outputs of the main functions in **carat**.

address R's requirements for inference functions. Moreover, we provide a power evaluation function `evalPower()` that builds upon the data generation function and inference functions and returns the simulated power. The powers under different randomization methods and inference functions can be visually compared for intuitive understanding through function `compPower()`, as long as they have the same format as the outputs of `evalPower()`.

We apply **ggplot2** (Wickham 2016) to drawing graphs in our package. The **carat** package is mainly coded in C++ using **Rcpp** (Eddelbuettel *et al.* 2023) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014). Furthermore, multi-core machine users can use an **OpenMP**-supported version of **carat**. For details, see Section 5.

In the rest of this section, we present several examples to illustrate the usage of these functions more detailed. The current version 2.2.1 of **carat** is based on R 4.3.1. It can be loaded in an R session via:

```
R> install.packages("carat")
R> library("carat")
```

3.2. Randomization

The package **carat** offers a variety of ways to conduct covariate-adaptive randomization for clinical trials. To generate a sequence of treatment assignments using a specific covariate-adaptive randomization procedure, users can either provide complete covariate data or specify a data-generating mechanism for the covariates. The former is helpful when the baseline covariate information is fully available prior to the onset of randomization, or it is of interest to apply a covariate-adaptive randomization procedure on existing data from an earlier study. The latter is used to generate a randomization sequence by only specifying the underlying distribution of the covariates, which is typically at the planning stage of a trial when the covariates are yet to be collected. It also serves as the basis for use of the Monte Carlo method by generating multiple randomization sequences to evaluate the properties of a randomization procedure.

In addition, the package is equipped with a command-line user interface to deal with sequentially accrued covariate data. It is thus designed to facilitate the real-world applications of covariate-adaptive randomization, because patients are usually enrolled in a clinical trial over a period of time. This interactive feature reflects an important logistical distinction between adaptive randomization and other simpler randomization methods, such as complete randomization and restricted randomization. For example, in contrast to restricted randomization procedures for which the entire randomization can be generated in advance, in this package the randomization of each patient is not available until the patient's covariate profile is obtained. Therefore, the interactive interface is especially suited for practical use, and provides a basic infrastructure for more sophisticated interactive web response systems (IWRS).

The usage of these three ways to generate a treatment assignment sequence by covariate-adaptive randomization is illustrated, respectively, in the following sections. The current version supports a total of six covariate-adaptive randomization methods, as outlined in Section 2.1.

Generating randomization sequence with complete covariate data

As listed in Table 1, the package **carat** contains six functions to generate a randomization sequence based on user-supplied covariate data. The input arguments of these functions comprise the name of the dataset containing the covariate information, and the design parameters required for the corresponding randomization procedure. Note that the parameters are randomization-specific and are different for each procedure: `omega` and `weight` specify the different weights for Hu and Hu's general covariate-adaptive randomization procedure and Pocock and Simon's minimization, respectively. The biased coin probability `p` is also required for these two methods, as well as the stratified biased coin design. The block size for the stratified permuted block randomization is specified by `bsize`, and the design parameter `a` is a parameter used solely for covariate-adjusted biased coin design. The detailed meanings of these function options are listed as follows.

`data`: A data frame. A row of the data frame corresponds to the covariate profile of a patient.

`omega`: A vector of weights at the overall, within-stratum, and within-covariate-margin levels. It is required that at least one element is larger than 0. If `omega = NULL` (default), the overall, within-stratum, and within-covariate-margin imbalances are weighted with the

proportions `0.2, 0.3`, and `0.5/cov_num` for each covariate-margin, respectively, where `cov_num` is the number of covariates of interest.

`weight`: A vector of weights for within-covariate-margin imbalances. It is required that at least one element is larger than 0. If `weight = NULL` (default), the within-covariate-margin imbalances are weighted with an equal proportion, `1/cov_num`, for each covariate-margin.

`p`: The biased coin probability. `p` should be larger than 1/2 and less than 1. The default is `0.85`.

`bsize`: The block size for stratified randomization. It is required to be a multiple of 2. The default is `4`.

`a`: A design parameter governing the degree of randomness. The default is `3`.

For covariate data to be used in the randomization functions, they should be stored in a data frame in which each row contains an individual's covariate profile and each column corresponds to a covariate. We use the provided dataset as an example.

```
R> data("pats", package = "carat")
R> head(pats, 10)
```

```
   gender employment.status  income marriage.status
1  Female             full. <= 0.5w       unmarried
2  Female             full.   >= 1w         married
3    Male            unemp. <= 0.5w       unmarried
4    Male             part.  0.5~1w         married
5  Female             full.   >= 1w       unmarried
6  Female            unemp.   >= 1w        divorced
7  Female             part.   >= 1w       unmarried
8    Male            unemp. <= 0.5w         married
9    Male             part.   >= 1w        divorced
10 Female            unemp.   >= 1w         married
```

For the purpose of illustration, we use the function `PocSimMIN()` to assign these 1000 patients with Pocock and Simon's minimization to one of the two treatments.

```
R> PS_RealD <- PocSimMIN(data = pats, weight = c(1, 2, 1, 1), p = 0.85)
```

An overview of the result is displayed with `print(PS_RealD)` or simply `PS_RealD`.

```
R> PS_RealD
```

```
        Pocock and Simon's Procedure with Two Arms

Data:  Real
group = A B
```

```
Sample size = 1000
cov_num = 4
considered covariates:   gender  employment.status
                         income  marriage.status
level_num = 2 3 3 3

the first three patients' covariate-profiles and assignments:
     covariate1 covariate2 covariate3 covariate4 assignment
pat1          1          1          1          1          A
pat2          1          1          2          2          B
pat3          2          2          1          1          A

Mean absolute imbalances at overall, within-strt., and
within-cov.-margin levels:
           overall       within-strt. within-cov.-margin
             2.000              2.593              1.091

Remark-Index:
1 -- gender
          1 <--> Female; 2 <--> Male
2 -- employment.status
          1 <--> full.; 2 <--> unemp.; 3 <--> part.
3 -- income
          1 <--> <= 0.5w; 2 <--> >= 1w; 3 <--> 0.5~1w
4 -- marriage.status
          1 <--> unmarried; 2 <--> married; 3 <--> divorced
```

The output printed in class '`carandom`' contains a short summary of design information, such as the number of patients (`Sample size`), the number of covariates (`cov_num`), and the number of levels within each covariate (`level_num`). In this example, there are four covariates, which are "gender", "employment.status", "income", and "marriage.status", and with two, three, three and three levels, respectively. The covariate levels have been converted to numeric values for concise outputs, and the convention rules are given in the `Remark-Index` section at the bottom of the output. The function also returns the covariate profiles of the first three patients and their assignments, and a rough analysis of the imbalances at different levels. Here, the mean absolute imbalances at the within-stratum and within-covariate-margin levels are calculated by taking the average of the absolute differences over all the strata and margins, respectively. The detailed imbalances of each stratum and margin are stored in the component of `Imb` in the output list. The whole assignment sequence is not output by default, but users can access it through the generic accessor.

```
R> PS_RealD$assignments[1:10]

[1] "A" "B" "A" "A" "B" "A" "B" "B" "B" "A"
```

The format of the output for other randomization functions is almost identical to the above output. However, within-stratum imbalances are additionally summarized, according to the stratum size, when the number of strata is large relative to the number of patients.

Generating randomization sequence with a covariate data-generating mechanism

The package **carat** also supports generation of assignment sequences with covariate-adaptive randomization by providing a data-generating mechanism for the covariates. This feature is achieved by the functions `Design.sim()`, where `Design` is one of the function names listed in the main body of Table 2. Thus, instead of inputting a covariate data set, users specify the number of patients (`n`), the number of covariates (`cov_num`), and the number of levels within each covariate (`level_num`), so that the function can generate the values of covariates used for randomization. Note that all of these functions assume independence between the covariates. Therefore no correlation between covariates needs to be specified. The input arguments for data generation are detailed as follows, while the design parameters are the same as those in Section 3.2 under the header "generating randomization sequence with complete covariate data", and thus are omitted here.

`n`: The number of patients. The default is `1000`.

`cov_num`: The number of covariates. The default is `2`.

`level_num`: A vector of level numbers for each covariate. Hence the length of `level_num` should be equal to the number of covariates. The default is `c(2, 2)`.

`pr`: A vector of probabilities. Under the assumption of independence between covariates, `pr` is a vector containing probabilities for each level of each covariate. The length of `pr` should correspond to the number of all levels, and the sum of the probabilities for each margin should be 1. The default is `rep(0.5, 4)`, which corresponds to `cov_num = 2`, and `level_num = c(2, 2)`.

We next consider an example of using Hu and Hu's randomization method, where there are three covariates with 2, 5, and 10 levels, respectively. We set `pr` to be `c(rep(0.5, 2), rep(0.2, 5), rep(0.1, 10))`, and `omega` to be the default value.

```
R> HuHuCAR.sim(n = 1000, cov_num = 3, level_num = c(2, 5, 10),
+    pr = c(rep(0.5, 2), rep(0.2, 5), rep(0.1, 10)), p = 0.85)

        Hu and Hu's General CAR

Data:  Simulated
group = A B
Sample size = 1000
cov_num = 3
level_num = 2 5 10

the first three patients' covariate-profiles and assignments:
     covariate1 covariate2 covariate3 assignment
pat1          1          1          3          B
pat2          2          5          3          A
pat3          2          4         10          A

Mean absolute imbalances at overall, within-strt., and
```

```
within-cov.-margin levels:
           overall       within-strt. within-cov.-margin
             0.000              1.300              1.176
```

The output has a similar format as that of `PocSimMIN()`. From the output, it is clearly seen that a total of 1000 patients are involved, and three covariates are considered with two, five, and ten levels, respectively. The default weights are used in this example, and all of the covariate-margins are treated equally. More details can be accessed by the generic accessor `$`. The result is that 500 patients are assigned to group `A`, and also 500 patients are assigned to group `B`.

Command-line interface

In addition to the above functions that generate all treatment assignments at the same time, we also developed a command-line interface (CLI) to enable use in real-world applications where treatment assignments may be allocated one by one by users. Thus, the purpose of the CLI is to deal with scenarios where the covariate information of patients is collected gradually during the process of enrollment. The CLI is invoked by the user-interface functions `Design.ui()`, where `Design` corresponds to one of the supported randomization procedures in the package. To run these functions, the options below must be specified, and a folder will be automatically created to store all of the information related to the randomization process, such as patients' covariates and treatment assignments.

`path`: The path in which a folder used to store variables will be created.

`folder`: Name of the folder. If default, a folder named "Design" will be created, where "Design" corresponds to the function name `Design.ui()`.

Before entering the first patient's information, users must follow the CLI instructions to initialize the randomization process, which may involve designating labels of covariates and design parameters. For the subsequent patients, only the covariate profiles must be provided, and the CLI will output the treatment assignment immediately. The backend database of covariates and assignments will be automatically updated. A flowchart of the process of covariate-adaptive randomization using the CLI is displayed in Figure 1. An example of running the `HuHuCAR.ui()` is given below, with the corresponding outputs.

```
R> HH.ui <- HuHuCAR.ui(path = getwd(), folder = "HuHuCAR")
```

Due to space limitations, the detailed process of interactions between the user and the CLI are presented in Appendix A. The typical output of a patient, which is printed in class '`carseq`', is as follows.

```
R> HH.ui

        Hu and Hu's General CAR

group = A B
```

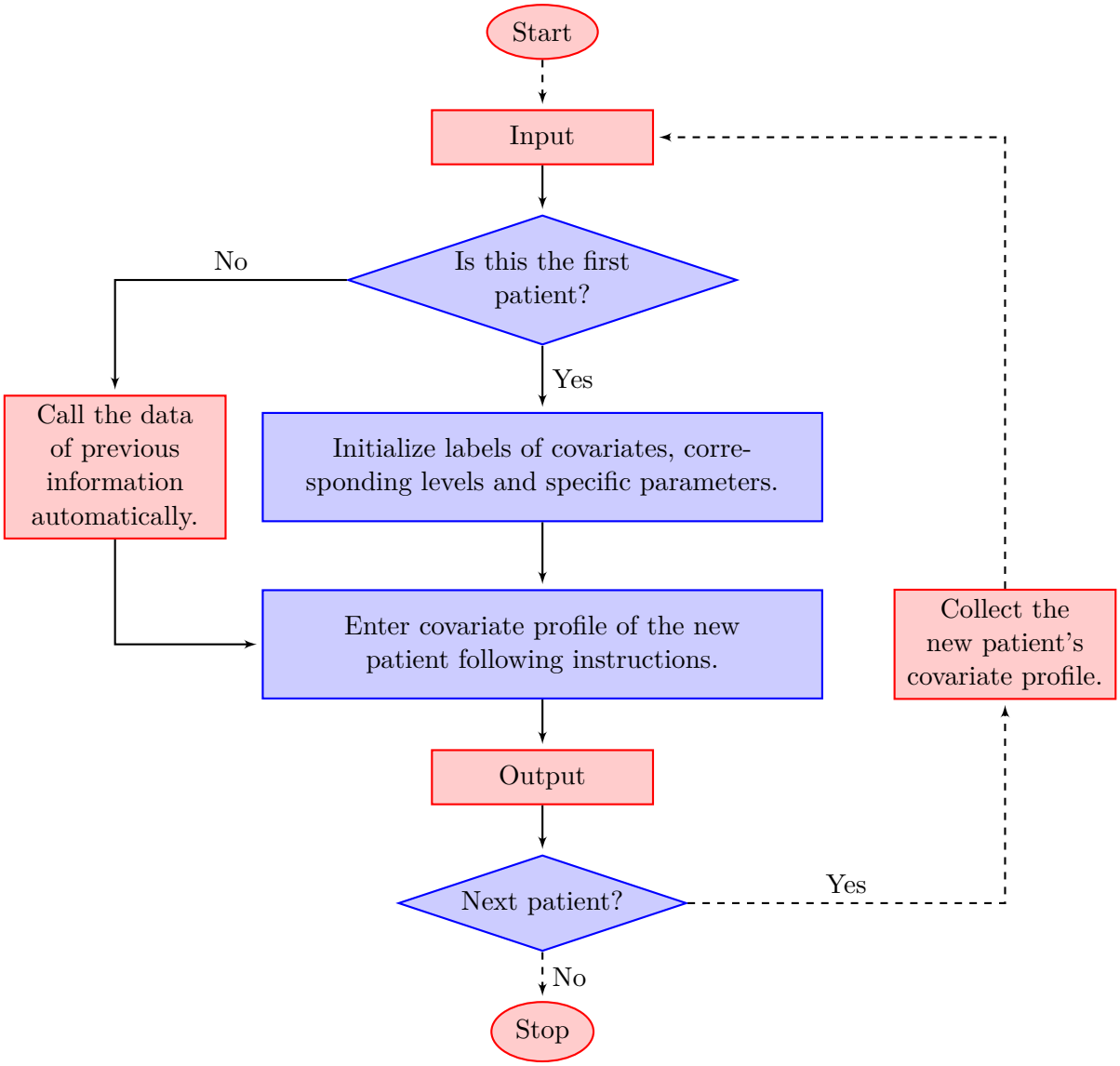

Figure 1: A general process of command-line user-interface functions.

```
Stamps for covariates:    1--sex; 2--age; 3--pills
Stamps for levels of each covariate:
  1 -- sex
         1 <--> male; 2 <--> female
  2 -- age
         1 <--> 0-30; 2 <--> 30-50; 3 <--> >=51
  3 -- pills
         1 <--> 0; 2 <--> 1-3; 3 <--> 3-5; 4 <--> >=6
covariate profile:  2 2 4

assignment:  B
```

From the output, we can easily see the patient's covariate profile and the resulting treatment assignment: This female patient, aged between 30 and 50, and addicted heavily to pills (`pills` ≥ 6), is assigned to treatment `B`.

3.3. Evaluation and comparison

In the package **carat**, the Monte Carlo-based evaluation of a covariate-adaptive randomization procedure is implemented through functions `evalRand()` and `evalRand.sim()`, which correspond to the cases of complete covariate data and specified data-generating mechanism, respectively. In addition to the design parameters used in randomization functions, some extra input arguments are required, as listed below.

`N`: The iteration number. The default is 500.

`Replace`: A bool. If `Replace = FALSE`, the function does clinical trial design for `N` iterations for one group of patients. If `Replace = TRUE`, the function does clinical trial design for `N` iterations for `N` different groups of patients. This is only applicable for `evalRand.sim`.

`method`: The randomization procedure to be evaluated. This package provides assessment for `"HuHuCAR"`, `"PocSimMIN"`, `"StrBCD"`, `"StrPBR"`, `"AdjBCD"`, and `"DoptBCD"`.

`...`: Arguments to be passed to `method`. These arguments depend on the randomization method and the following arguments are accepted:

`omega`: Only applicable for the method of `"HuHuCAR"`.

`weight`: Only applicable for the method of `"PocSimMIN"`.

`p`, only applicable for the methods of `"HuHuCAR"`, `"PocSimMIN"` and `"StrBCD"`.

`bsize`: Only applicable for the method of `"StrPBR"`.

`a`: Only applicable for the method of `"AdjBCD"`.

In the following example, we use the function `evalRand.sim()` to evaluate Hu and Hu's method through the covariate data-generating mechanism, and we need to specify the `omega` and `p` in "`...`" for this method.

```
R> n <- 1000
R> N <- 500
R> p <- 0.85
R> cov_num <- 3
R> level_num <- c(2, 5, 2)
R> pr <- c(rep(0.5, 2), rep(0.2, 5), rep(0.5, 2))
R> omega <- c(1, 2, rep(1, cov_num))
R> evalR.HH_simD <- evalRand.sim(n, N, TRUE, cov_num, level_num,
+    pr, "HuHuCAR", omega, p)
R> evalR.HH_simD

        Hu and Hu's General CAR
```

```
call:
 evalRand.sim(method = HuHuCAR)

group = A B
Sample size = 1000
iteration = 500
cov_num = 3
level_num = 2 5 2
Data type:  Simulated
Data generation mode:  TRUE

assignments of the first 3 iterations for the first 7 patients :
      pat1 pat2 pat3 pat4 pat5 pat6 pat7
iter1    A    B    B    B    A    B    A
iter2    A    B    B    B    A    A    A
iter3    A    B    A    B    B    A    A

Evaluation by imbalances:
absolute overall imbalances:
     max 95% quan   median     mean
    4.00     2.00     0.00     1.02

absolute within-strt. imbalances for the first 3 strata:
                max 95% quan median  mean
stratum1(1,1,1)   4        2      1 0.988
stratum2(1,1,2)   6        3      1 1.010
stratum3(1,2,1)   5        3      1 0.998

absolute within-cov.-margin imbalances for 3 margins:
            max 95% quan median mean
margin(1;1)   5        3      1 1.24
margin(2;1)   5        3      1 1.21
margin(3;1)   5        3      1 1.19
```

The output is printed in class '`careval`' and illustrates the information, as well as a brief analysis of the designing study, such as sample size (`Sample size`), number of iterations (`iteration`) and imbalances (`Evaluation by imbalances`). The strata are sorted in increasing order based on the values of the first covariate, followed by the second, and so on through all the covariates under consideration. The detailed labels of each stratum are stored in the component of `All strata` in the output list. In this example, a dataset of 1000 patients with three covariates with 2, 5, and 2 levels, respectively, is generated to evaluate the procedure with 500 iterations. Users can evaluate the goodness of Hu and Hu's randomization procedure through analysis of the absolute imbalances.

For comparison of two or more randomization procedures, `compRand()`, should be used. The inputs are objects of the class '`careval`'. We compare all of the included randomization procedures through a data-generating mechanism.

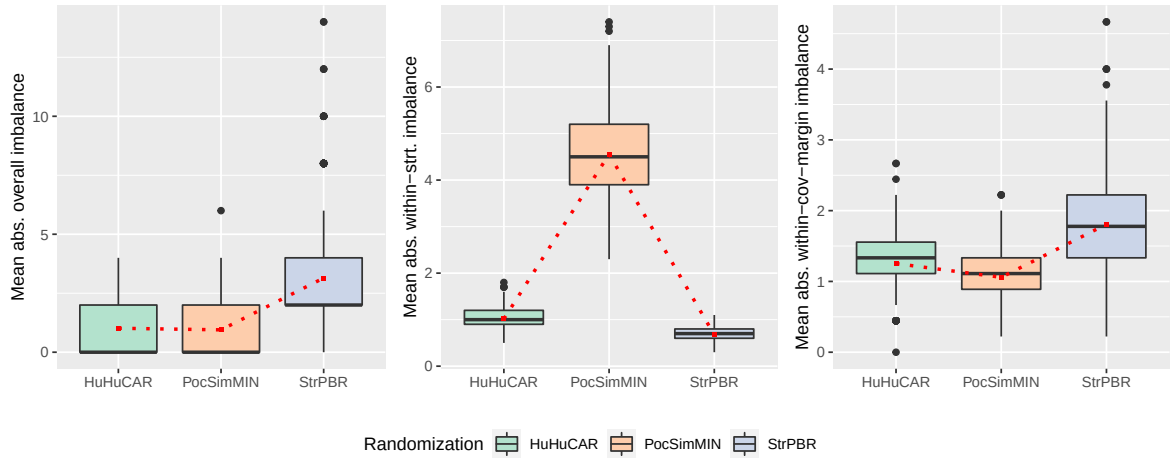

Figure 2: Boxplots of overall, within-stratum, and within-covariate-margin imbalances for the Hu and Hu's randomization (HuHuCAR), the Pocock and Simon's minimization (PocSimMIN), and the stratified permuted block randomization (StrPBR).

```
R> weight <- rep(1, cov_num)
R> bsize <- 4
R> a <- 3
R> evalR.PS_simD <- evalRand.sim(n, N, TRUE, cov_num, level_num, pr,
+    "PocSimMIN", weight, p)
R> evalR.STR_simD <- evalRand.sim(n, N, TRUE, cov_num, level_num, pr,
+    "StrPBR", bsize)
R> compRand(evalR.HH_simD, evalR.PS_simD, evalR.STR_simD)
```

Detailed outputs are displayed in Appendix B. The comparison of mean absolute imbalances at the overall, within-stratum, and within-covariate-margin levels are visualized in Figure 2. Among the mean absolute imbalances at all aspects, the within-stratum and within-covariate-margin ones are calculated by taking the average over all strata and covariate-margins.

3.4. Hypothesis testing

The package **carat** offers three different testing functions that correspond to the three tests described in Section 2.3, i.e., the bootstrap t test, the corrected t test, and the randomization test. These functions are developed to determine the differences in treatment effects based on the data from a covariate-adaptive clinical trial. Similar to the randomization functions, the testing functions can be used for different types of data, whether it is user-provided or generated by a specified mechanism. For the latter usage, the patients' outcomes also need to be generated, for which the package provides a data generation function. We next introduce the data generation process, and then use the generated data to illustrate the usage of the testing functions.

Data generation

The function `getData()` is implemented to create a dataset, comprising covariate profiles,

assignments, and outcomes, based on the user-specified randomization procedure and data-generating mechanism. It currently supports the generation of continuous and binary outcomes based on linear and logit models. For details, see Section 2.4. The required inputs, other than those listed in the randomization part, are:

`type`: A data-generating method. Optional input: `"linear"` or `"logit"`.

`beta`: A vector of coefficients of covariates. The length of `beta` must correspond to the sum of all covariates' levels.

`mu1, mu2`: Main effects of treatment 1 and treatment 2.

`sigma`: The error variance for the linear model. The default is 1. This should be a positive value and is only used when `type = linear`.

For an illustration of the tests, we first generate patient data with the function `getData()`, in which the underlying model is specified as the logistic model with binary outcomes, and the randomization method is the stratified biased coin design. Here, we consider the case of 2×2 strata for 100 patients, the logit link function, and the biased coin probability `p = 0.85`.

```
R> dataS <- getData(n = 100, cov_num = 2, level_num = c(2, 2),
+    pr = rep(0.5, 4), type = "logit", beta = c(0.1, 0.2, 0.4, 0.8),
+    mu1 = 0, mu2 = 0, method = "StrBCD", p = 0.85)
R> dataS[, 1:4]
```

The first four columns of generated data are displayed below.

```
           X1 X2 X3 X4
covariate1  2  1  1  1
covariate2  2  2  2  1
assignment  1  1  2  2
outcome     1  1  1  0
```

The reason we focus on continuous or binary outcomes in the current version of **carat** is in part because inferences are better studied for these two types of outcomes. Our software can potentially be extended to incorporate other types of outcomes, such as time-to-event data or count data.

Bootstrap t test

The implementation of the bootstrap t test depends on the process of data collection, so the randomization method and the corresponding parameters must be specified, and must be exactly the same as those used in the data simulation or the real experiment. Other inputs of the bootstrap t test are:

`data`: A data frame. It consists of patients' covariate profiles, treatment assignments, and outcomes.

`B`: An integer. It is the number of bootstrap samples. The default is `200`.

`conf`: Confidence level of the interval. The default is `0.95`.

We now perform the bootstrap t test on the previously generated dataset named `dataS`. The null hypothesis is that there is no difference in treatment effects between groups. This is indeed true. The default values are used for both `B` and `conf`, such that the inputs are omitted. Note that the randomization method `method = "StrBCD"` and the biased coin probability `p = 0.85` are the same as those used in the data generation process.

```
R> boot.test(data = dataS, method = "StrBCD", p = 0.85)

        Bootstrap t-test

data:  dataS
t = 1.1093, p-value = 0.2673
95 percent confidence interval:
 -0.07057956  0.25465319
sample estimates:
difference for treatment effect
                     0.09203681
```

The result of the bootstrap t test conveys four messages. First, it prints the name of the data: `dataS`. Second, the calculated t statistic is 1.1093, and the corresponding p value is 0.2673. Thus, we cannot reject the null hypothesis under the 0.05 significance level. Third, the 95% confidence interval is $[-0.07, 0.25]$. Fourth, the estimated difference is around 0.09.

Corrected t test

Here we generate patients' profiles with a linear model, and obtain assignments under Hu and Hu's randomization, which satisfies the conditions to apply the corrected t test. We then consider the case where the difference in treatment effect is 2.0. As the implementation of the corrected t test does not rely on randomization methods, the required inputs are much simple; only the data and the level of confidence interval are required:

```
R> dataH <- getData(n = 100, cov_num = 2, level_num = c(2, 2),
+    pr = rep(0.5, 4), type = "linear", beta = c(0.1, 0.2, 0.4, 0.8),
+    mu1 = 2.0, mu2 = 0, sigma = 1, method = "HuHuCAR",
+    omega = c(0.1, 0.1, 0.4, 0.4), p = 0.85)
R> corr.test(data = dataH, conf = 0.95)

        Corrected t-test

data:  dataH
t = 6.4183, p-value = 1.378e-10
95 percent confidence interval:
 0.9754009 1.8330159
sample estimates:
difference for treatment effect
                       1.404208
```

The output of the corrected t test contains the same information as that of the bootstrap t test. In this simulation, the calculated value of the t statistic is 6.4183, and the corresponding p value is almost 0, so the null hypothesis is rejected under the 0.05 significance level. The estimated difference is approximately 1.40, which is close to the real value.

Randomization test

For the randomization test, the p value is determined by the location of observed statistics over the whole set of replications, so we also provide a histogram as an intuitive illustration. The inputs that must be specified are:

`Reps`: An integer. It is the number of randomized replications used in the randomization test. The default is `200`.

`binwidth`: The number of bins for each bar in the histogram. The default is `30`.

We consider the covariate-adjusted biased coin design, and use the default values for `Reps` and `binwidth`.

```
R> dataA <- getData(n = 100, cov_num = 2, level_num = c(2, 2),
+    pr = rep(0.5, 4), type = "linear", beta = c(0.1, 0.2, 0.4, 0.8),
+    mu1 = 0, mu2 = 0, method = "AdjBCD", a = 3)
R> rand.test(data = dataA, method = "AdjBCD", a = 3)

        Randomization test

data:  dataA
p-value = 0.32
95 percent confidence interval:
-0.1522571           0.3134017
sample estimates:
difference for treatment effect
                      -0.103843
```

The estimated difference is approximately -0.10, and the calculated p value is 0.32, so the null hypothesis is not rejected. The histogram represents the distribution of sample statistics, and the red dotted line indicates the observed statistic.

3.5. Power analysis

In the planning stage of a covariate-adaptive randomized clinical trial, one of the main tasks is to ensure that the study is adequately powered to detect a meaningful clinical benefit. To fulfill this purpose, the package **carat** is equipped with two easy-to-use power analysis functions: `evalPower()` is used for calculating the power of a single test under a particular covariate-adaptive randomization procedure, while `compPower()` is used to compare the powers between various tests or randomization procedures, or combinations thereof. All of the aforementioned randomization and testing methods are supported. Corresponding plots are also optional for better illustration and intuitive understanding. We describe these two functions in the following two sections.

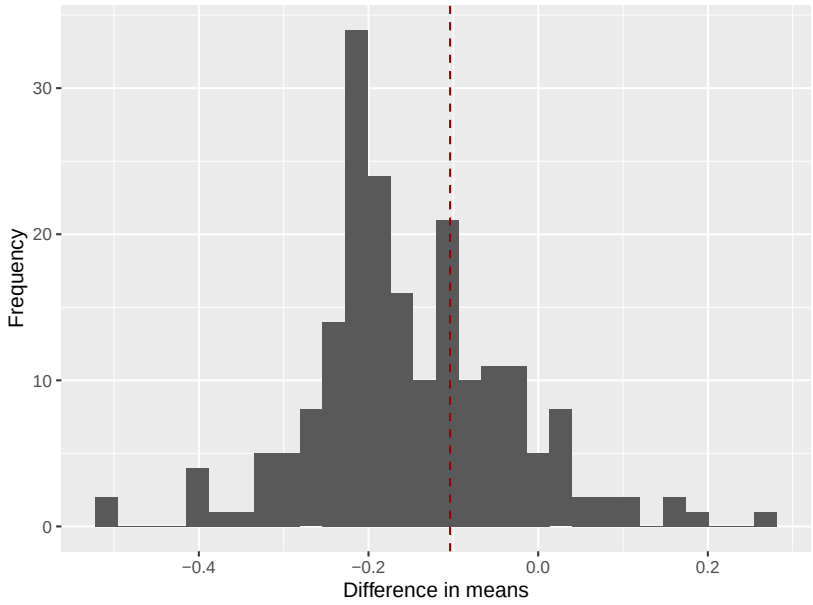

Figure 3: Histogram of the randomization test under the covariate-adjusted biased coin design.

Power calculation

The function `evalPower()` allows users to simulate the power (and also type I error), under either a single value of the difference in treatment effects or under a sequence of values to better understand the trend. The calculations are performed using the Monte Carlo method, and we provide parallel computation to accelerate the algorithm. For the randomization test and the bootstrap t test, the number of cores to be used must be specified. The algorithm of corrected t test is much faster than the other two tests, so parallel computation is not provided. In addition to the arguments used in the functions `evalRand()`, `evalRand.sim()`, and `getData()`, some extra input arguments used for power calculation are as follows.

`di`: A value or a vector of values of difference in treatment effects. The default value is a sequence from `0` to `0.5` with increments of `0.1`. The value(s) forms the horizontal axis of the plot.

`Iternum`: An integer. It is the number of iterations required for power calculation.

`sl`: The significance level. If the p value returned by the test is less than `sl`, the null hypothesis will be rejected. The default value is `0.05`.

`test`: A character string specifying the alternative tests used to verify hypothesis, must be one of `"boot.test"`, `"corr.test"` or `"rand.test"`, which are the bootstrap t test, the corrected t test, and the randomization test, respectively. The arguments associated with the testing function can be specified; otherwise, the default value will be used.

`plot`: A bool. It indicates whether to plot or not. Optional input: `TRUE` or `FALSE`.

`nthreads`: The number of threads to be used in parallel computation. This is needed only under `rand.test` and `boot.test`. The default is `1`.

We calculate the power and type I error for the randomization test under the stratified permuted block design, with significance level $\alpha = 0.05$, with 100 patients, and 1000 iterations. The block size 4 is used for the stratified permuted block design (`bsize = 4`), and 200 replications are used for the randomization test (`Reps = 200`):

```
R> evalPower(n = 100, cov_num = 3, level_num = rep(2, 3), pr = rep(0.5, 6),
+    type = "linear", beta = c(0.1, 0.2, 0.1, 0.2, 0.2, 0.4), sigma = 1,
+    di = seq(0, 0.8, 0.1), Iternum = 1000, sl = 0.05, method = "StrPBR",
+    bsize = 4, test = "rand.test", Reps = 200, plot = FALSE, nthreads = 1)

$Powers
  diff value    se
1  0.0 0.052 0.007
2  0.1 0.097 0.009
3  0.2 0.162 0.012
4  0.3 0.270 0.014
5  0.4 0.494 0.016
6  0.5 0.689 0.015
7  0.6 0.810 0.012
8  0.7 0.908 0.009
9  0.8 0.975 0.005

$Time
[1] "Execute time: 3.96 mins"
```

The type I error is 0.052, which is obtained with `diff = 0.0`. It is close to the nominal level of 0.05. As the difference in treatment effects increases, the power of the randomization test also increases, as expected. The standard errors of the powers are reported to quantify the precision of the power estimate. The formula for the standard error of power is

$$\texttt{se} = \sqrt{\texttt{value}(1 - \texttt{value})/\texttt{Iternum}},$$

where `value` is the estimated power, and `Iternum` is the number of iterations. This formula stems from the fact that each iteration can be thought of as a Bernoulli trial. The total execution time is 3.96 minutes, and this may vary from computer to computer. Computations throughout this paper except for Table 4 were all performed on a machine with an Intel Core i7-9750H CPU 2.60GHz processor and 16GB RAM.

Power comparison

Based on `evalPower()`, we can compare power between different tests under different randomization procedures. In consideration of extendability, the power from other test functions can also be compared, as long as they all have the required format. Inputs needed for `compPower()` are:

`powers`: A list. Each argument consists of the power generated by `evalPower()` in this package or by other sources. The length of each argument must match.

`diffs`: A vector. It contains values of group treatment effect differences. The length of this argument and the length of each argument of `powers` must match.

`testname`: A vector. Each element is the name of test and the randomization method used. For example, when applying `rand.test` and `corr.test` under `HuHuCAR`, it can be `c("HH.rand","HH.corr")`. The length of this argument must match the length of `diffs`.

Here, we calculate the power of the bootstrap t test and the corrected t test under Hu and Hu's randomization method, and the bootstrap t test under Atkinson's D_A-optimal biased coin design. The simple two-sample t test is also performed for comparison.

```
R> HHbtp <- evalPower(n = 100, cov_num = 2, level_num = c(2, 2),
+    pr = rep(0.5, 4), type = "linear", beta = c(1, 2, 2, 4),
+    sigma = 1, di = seq(0, 1.5, 0.3), Iternum = 1000, sl = 0.05,
+    method = "HuHuCAR", omega = c(0.1, 0.1, 0.4, 0.4), p = 0.85,
+    test = "boot.test", B = 200, plot = FALSE, nthreads = 1)
R> HHctp <- evalPower(n = 100, cov_num = 2, level_num = c(2, 2),
+    pr = rep(0.5, 4), type = "linear", beta = c(1, 2, 2, 4),
+    sigma = 1, di = seq(0, 1.5, 0.3), Iternum = 1000, sl = 0.05,
+    method = "HuHuCAR", omega = c(0.1, 0.1, 0.4, 0.4), p = 0.85,
+    test = "corr.test", plot = FALSE)
R> Doptbtp <- evalPower(n = 100, cov_num = 2, level_num = c(2, 2),
+    pr = rep(0.5, 4), type = "linear", beta = c(1, 2, 2, 4),
+    sigma = 1, di = seq(0, 1.5, 0.3), Iternum = 1000, sl = 0.05,
+    method = "DoptBCD", test = "boot.test", B = 200,
+    plot = FALSE, nthreads = 1)
R> di <- seq(0, 1.5, 0.3)
R> Iternum <- 1000
R> Hpvs <- matrix(0, nrow = length(di), ncol = Iternum)
R> Dpvs <- matrix(0, nrow = length(di), ncol = Iternum)
R> for (i in 1:length(di)) {
+    for (j in 1:Iternum) {
+       dataDt <- getData(n = 100, cov_num = 2, level_num = c(2, 2),
+           pr = rep(0.5, 4), type = "linear", beta = c(1, 2, 2, 4),
+           mu1 = di[i], mu2 = 0, sigma = 1, method = "DoptBCD")
+       dataHt <- getData(n = 100, cov_num = 2, level_num = c(2, 2),
+           pr = rep(0.5, 4), type = "linear", beta = c(1, 2, 2, 4),
+           mu1 = di[i], mu2 = 0, sigma = 1, method = "HuHuCAR",
+           omega = c(0.1, 0.1, 0.4, 0.4), p = 0.85)
+       dataDt <- data.frame(t(dataDt))
+       dataHt <- data.frame(t(dataHt))
+       ocD1 <- subset(dataDt, assignment == 1, select = outcome)
+       ocD2 <- subset(dataDt, assignment == 2, select = outcome)
+       ocH1 <- subset(dataHt, assignment == 1, select = outcome)
+       ocH2 <- subset(dataHt, assignment == 2, select = outcome)
+       reD <- t.test(ocD1, ocD2)
+       reH <- t.test(ocH1, ocH2)
```

```
+       Dpvs[i, j] <- (reD$p.value < 0.05)
+       Hpvs[i, j] <- (reH$p.value < 0.05)
+    }
+ }
R> powerD <- data.frame(diff = di, value = apply(Dpvs, 1, sum) /
+    Iternum, se = round(sqrt((apply(Dpvs, 1, sum) / Iternum) *
+    (1 - apply(Dpvs, 1, sum) / Iternum) / Iternum), 3))
R> powerH <- data.frame(diff = di, value = apply(Hpvs, 1, sum) /
+    Iternum, se = round(sqrt((apply(Hpvs, 1, sum) / Iternum) *
+    (1 - apply(Hpvs, 1, sum) / Iternum) / Iternum), 3))
R> Doptp <- list(Powers = powerD)
R> HHtp <- list(Powers = powerH)
```

Then, we combine the results above as input for `compPower()`.

```
R> powers_compare <- list(HHbtp, HHctp, HHtp, Doptbtp, Doptp)
R> testname_compare <- c("HH.boot", "HH.corr", "HH.simple",
+    "Dopt.boot", "Dopt.simple")
R> compPower(powers_compare, di, testname_compare)
```

```
$powers
                        0          0.3          0.6
HH.boot     0.055(0.007) 0.332(0.015) 0.845(0.011)
HH.corr     0.058(0.007) 0.323(0.015) 0.856(0.011)
HH.simple   0.002(0.001) 0.077(0.008) 0.492(0.016)
Dopt.boot   0.058(0.007) 0.293(0.014) 0.774(0.013)
Dopt.simple 0.006(0.002) 0.091(0.009)  0.51(0.016)
                      0.9          1.2  1.5
HH.boot     0.992(0.003)         1(0) 1(0)
HH.corr     0.997(0.002) 0.999(0.001) 1(0)
HH.simple   0.924(0.008) 0.999(0.001) 1(0)
Dopt.boot   0.981(0.004) 0.999(0.001) 1(0)
Dopt.simple 0.918(0.009) 0.993(0.003) 1(0)

$plot
```

From these outputs, we observe that under both covariate-adaptive randomization methods the corrected t test and the bootstrap t test are able to preserve the type I error rate. The simple two-sample t test tends to be conservative, in terms of a smaller type I error, than the nominal level. Importantly, we observe from Figure 4 that between these three tests, the corrected t test has the highest power, followed by that of the bootstrap t test, while the simple t test has the lowest power. From the perspective of the randomization method, Hu and Hu's method increased the power of the bootstrap t test, compared to Atkinson's D_A optimal biased coin design, which can be attributed to the more balanced covariates induced by Hu and Hu's randomization method.

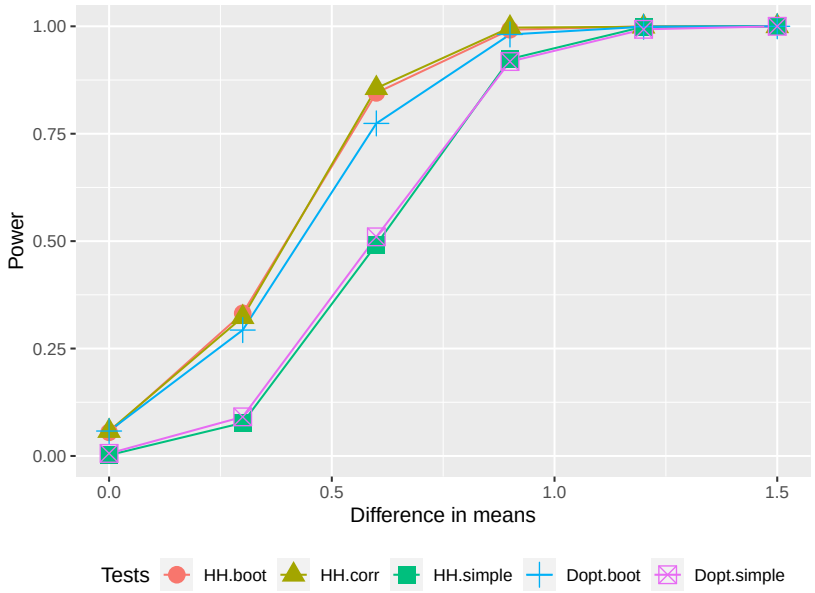

Figure 4: Power comparison between tests, i.e., the bootstrap t test (boot), the simple two-sample t test (simple) and the corrected t test (corr) under Hu and Hu's randomization method (HH) and Atkinson's D_A-optimal biased coin design (Dopt), with different values of difference in means.

4. Application of carat

The nefazodone CBASP trial (Keller *et al.* 2000) is a randomized clinical trial to contrast the effects of three alternative treatments for chronic depression. The trial randomized 681

Characteristic	Nefazodone ($\boldsymbol{N = 226}$)	Psychotherapy and Nefazodone ($\boldsymbol{N = 227}$)	All patients ($\boldsymbol{N = 453}$)
Gender (%)			
Female	64.2	69.2	66.7
Male	35.8	30.8	33.3
Race (%)			
White	87.2	92.5	89.8
Black or African American	3.1	3.5	3.3
American Indian or Alaska Native	6.6	1.8	4.2
Asian	1.3	2.2	1.8
Native Hawaiian or Pacific Islander	1.8	0.0	0.9
Obsessive compulsive disorder (%)			
Sub-threshold	2.7	2.6	2.6
Otherwise	97.3	97.4	97.4

Table 3: The characteristics of the patients for the chosen treatments in the original clinical trial.

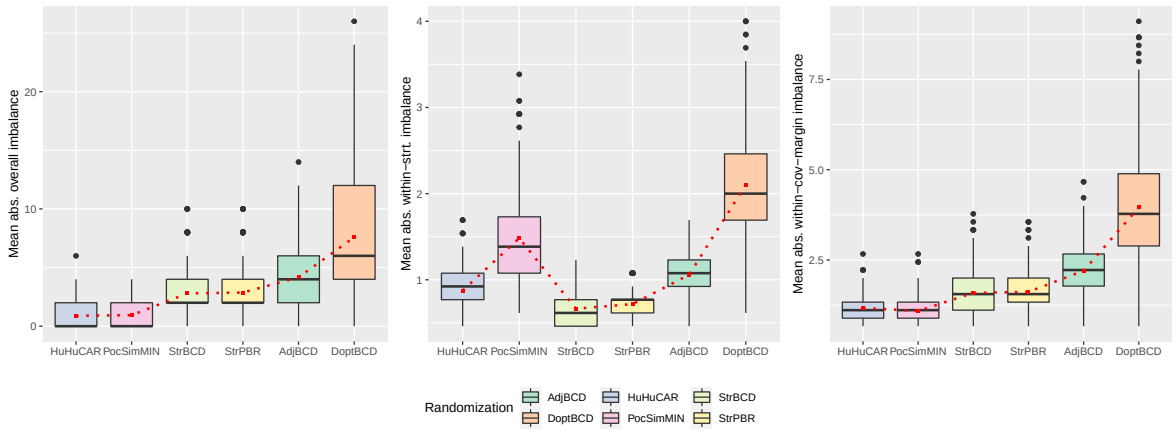

Figure 5: Boxplots of overall, within-stratum, and within-covariate-margin imbalances for the Hu and Hu's randomization (HuHuCAR), Pocock and Simon's minimization (PocSimMIN), stratified biased coin design (StrBCD), stratified permuted block randomization (StrPBR), covariate-adjusted biased coin design (AdjBCD) and Atkinson's D_A-optimal biased coin design (DoptBCD) applied to nefazodone CBASP data.

patients to nefazodone, CBASP, or combinations thereof. We choose this nefazodone CBASP data as an enlightening example and only consider two treatments, i.e., nefazodone and the combination. The characteristics of the patients for the chosen treatments in the original clinical trial are displayed in Table 3. We consider three covariates of strong association, namely, race, gender, and obsessive compulsive disorder, with five, two, and two levels, respectively. Hereafter, we only consider 440 patients, excluding 13 patients with missing outcomes.

4.1. Randomization

To select the well-behaved randomization for the nefazodone CBASP data, we need to evaluate and compare different randomization procedures. Based on the published recommendations, the biased coin probability, the block size, and the design parameter are set as follows.

- *Weights:*
  - `omega = c(1, 2, rep(1, 3))` for Hu and Hu's randomization.
  - `weight = rep(1, 3)` for Pocock and Simon's minimization.
- *Specific parameters:*
  - `p = 0.85` for Hu and Hu's randomization, Pocock and Simon's minimization, and stratified biased coin design.
  - `bsize = 4` for stratified permuted block randomization.
  - `a = 3` for covariate-adjusted biased coin design.
- *Iteration number:* `N = 500`.

Detailed outputs are displayed in Appendix C, and the comparison is illustrated in Figure 5. These results show that all the considered covariate-adaptive randomization methods perform satisfactorily in achieving balances, although the covariate-adjusted biased coin design

and Atkinson's D_A-optimal biased coin design are somewhat suboptimal. For the covariate-adjusted biased coin design, very small values of `a`, such as `a` < 2, can substantially increase the likelihood of imbalances in covariates in our experience, and thus are not recommended. Moreover, large values of `a` can make the procedure too deterministic, thus increasing the selection bias. In this example, we set the value of `a` to be 3 to ensure a good trade-off between balance and selection bias. Although the Atkinson's D_A-optimal biased coin design attains a balanced allocation under the assumed homogeneous linear models, its performance of covariate balance is still inferior to the covariate-adaptive randomization methods that directly deal with covariate imbalances, either within-stratum or within-covariate-margin. Therefore, we do not recommend the Atkinson's D_A-optimal biased coin design if the primary goal of randomization is to balance covariates.

To achieve general balance, we apply Hu and Hu's randomization method to the dataset based on the aforementioned comparison analysis.

```
        Hu and Hu's General CAR

Data:  Real
group = A B
Sample size = 440
cov_num = 3
considered covariates:   ObsCompul  GENDER  RACE
level_num = 2 2 5

the first three patients' covariate-profiles and assignments:
     covariate1 covariate2 covariate3 assignment
pat1          1          1          1          A
pat2          1          1          1          B
pat3          1          1          1          A

Mean absolute imbalances at overall, within-strt., and
within-cov.-margin levels:
           overall       within-strt. within-cov.-margin
            0.0000             0.9231             1.5556

Remark-Index:
1 -- ObsCompul
          1 <--> Otherwise; 2 <--> Sub-threshold
2 -- GENDER
          1 <--> Female; 2 <--> Male
3 -- RACE
          1 <--> White; 2 <--> Black or African American;
          3 <--> American Indian or Alaska Native;
          4 <--> Asian; 5 <--> Native Hawaiian or Pacific Islander
```

The detailed analysis of output is similar to that in Section 3.2 under the header "generating randomization sequence with complete covariate data". As the output outlined, half of the patients are assigned to each of the two treatments, respectively, because the overall absolute mean is 0.0000.

4.2. Hypothesis testing

Power analysis is essential in determining the proper sample size required to detect a meaningful treatment effect in clinical trials. Here we use one of the most commonly used randomization methods in practice – Pocock and Simon's minimization – to illustrate an example of sample size determination by calculating the power of the corrected t test with different sample sizes.

In the sample size determination, we first use the selected variables above and the treatment indicator to fit a linear model using the function `lm()` in the R package **stats**. Then, we use three loops, each of which corresponds to a key step in the sample size determination process. The outermost loop over i corresponds to different sample sizes, the second loop over j corresponds to different treatment effects, and the innermost loop over k contains `Iternum = 1000` times of data generation, hypothesis testing, and p value calculation.

In the innermost loop, for a given sample size and treatment effect, we randomly sample the profiles of patients with replacement from the original data with equal probability. Then, we perform Pocock and Simon's minimization on sampled profiles using the function `PocSimMIN()` in **carat** and obtain new assignments. Next, we simulate new outcomes using the function `predict()` under the given value of treatment effect (determined by the loop over j). Finally, we perform the corrected t test using the function `corr.test()` in **carat** on the new data to obtain the corresponding p value.

For hypothesis testing, we assume that the null hypothesis is that the treatment effect equals zero, and set significance level at 0.05. After obtaining the above p values, we calculate the empirical probabilities of the rejection of the null hypothesis as estimated powers. The detailed code for sample size determination is displayed as follows.

```
R> Iternum <- 1000
R> sslm4 <- lm(FinalHAMD ~ A2 + ObsCompul + GENDER + RACE, data = data)
R> nop <- seq(50, 300, 50)
R> diffs <- seq(0, 11, by = 1.5)
R> nop.name <- rep("", length(nop))
R> PSnop <- list()
R> pvals <- matrix(0, length(diffs), Iternum)
R> for (i in 1:length(nop)) {
+   for (j in 1:length(diffs)) {
+       sslm4$coefficients[2] <- diffs[j]
+       for (k in 1:Iternum) {
+           PSind <- sample(1:nrow(cont_subset), nop[i],
+                           replace = TRUE)
+           data_temp <- data[PSind, ]
+           data_combine <- rbind(PSCovA, predict(sslm4, data_temp) +
+               rnorm(nrow(data_temp), sd = summary(sslm4)$sigma))
+           pvals[j, k] <- (corr.test(data_combine)$p.value < 0.05)
+       }
+   }
+   result.temp <- data.frame(diff = diffs,
+      value = apply(pvals, 1, sum) / Iternum,
+      se = round(sqrt((apply(pvals, 1, sum) / Iternum) *
```

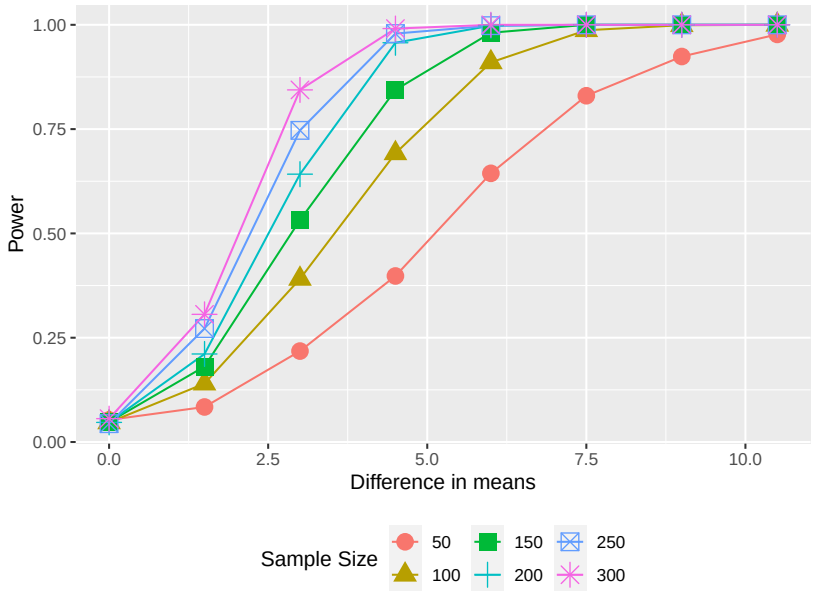

Figure 6: Power comparison for different sample sizes under Pocock and Simon's minimization with different values of difference in means.

```
+          (1 - apply(pvals, 1, sum) / Iternum) / Iternum), 3))
+   eval(parse(text = paste("PSnop[[", i, "]]",
+      "=list(Powers = result.temp)", sep = "")))
+   nop.name[i] <- nop[i]
+ }
R> compPower(PSnop, diffs = diffs, nop.name)

$powers
                0           1.5             3           4.5
50  0.053(0.007) 0.084(0.009) 0.218(0.013) 0.398(0.015)
100 0.047(0.007)  0.14(0.011) 0.391(0.015) 0.692(0.015)
150 0.048(0.007) 0.181(0.012) 0.532(0.016) 0.844(0.011)
200 0.047(0.007) 0.211(0.013) 0.642(0.015) 0.957(0.006)
250 0.044(0.006) 0.272(0.014) 0.747(0.014) 0.979(0.005)
300 0.056(0.007) 0.306(0.015) 0.844(0.011) 0.991(0.003)
                6           7.5             9          10.5
50  0.644(0.015)  0.83(0.012) 0.924(0.008) 0.977(0.005)
100  0.91(0.009) 0.987(0.004) 0.999(0.001)         1(0)
150 0.981(0.004)         1(0)         1(0)         1(0)
200 0.998(0.001)         1(0)         1(0)         1(0)
250 0.998(0.001)         1(0)         1(0)         1(0)
300         1(0)         1(0)         1(0)         1(0)

$plot
```

The simulated power of the corrected test is presented in Figure 6. For a fixed sample size,

the power first increases rapidly as the difference in treatment effect increases, and then it slows down and levels off. In this case, there is a trade-off between the power and the number of enrolled patients in a clinical trial. For example, a sample size of 200 is sufficient to give over 95% power to detect a treatment effect difference of 4.5, based on Figure 6.

5. Computational details

The primary purpose of this section is to discuss the implementation of modern computation tools, such as **Rcpp** and **OpenMP**, to deal with the most computationally intensive tasks in the development of **carat** and illustrate the substantial benefits gained by these tools. It worths noting that the run-time in this section may not be reproducible exactly due to the different performances from computers to computers or from servers to servers.

We first emphasize that the covariate-adaptive randomization and the associated tests cause lots of challenges in computation due to the following reasons. First, for covariate-adaptive randomization, the randomization sequence has to be updated sequentially, which is more complicated than the non-adaptive methods where the entire randomization sequence can be generated at once. Second, the bootstrap test and the randomization test depend on replicating the randomization procedure hundreds of times for a single test. Third, the Monte Carlo method implemented for power calculation adds another layer of complexity on top of the previous concerns.

These issues inevitably cause nested loops, which generally slow the performance of R. To overcome this drawback of R, we extensively use **Rcpp** in our package, which provides an efficient way of combining R and C++ to significantly accelerate the speed of the algorithms. To illustrate the improvement delivered with C++, we compare the performance of the function `DoptBCD()` in **carat** with that of the function `Atkinson()` fully coded with R (Ma *et al.* 2020). The setting is as follows:

- The number of covariates of interest is two with two levels for each covariate.
- Sample size `n` traverses from 200 to 2500 with increments of 200.

The run-time of the two functions is presented in Figure 7. They were evaluated using the function `microbenchmark()` in R, and we set the number of evaluation times to be 20. It is clear that the function `DoptBCD()` coded with **Rcpp** is much faster than the function `Atkinson()` coded fully with R. We also observe from Figure 7 that the difference in run-time tends to be more noticeable as the sample size increases. These findings are as expected because the run-time of compiled loops is generally less than that when using an interpreted language, such as R (Eddelbuettel *et al.* 2023).

In addition to **Rcpp**, another computational feature of the package is parallel computing performed by **OpenMP**. Thus, using **OpenMP**, the Monte Carlo tasks can be parallelized and executed in multiple threads. Combined with **Rcpp**, this can markedly reduce the computational time of the power analysis functions, which are the most computationally intensive part in the package, as discussed at the beginning of this section. The gain of computational efficiency is illustrated in the following example, in which power calculations based on different test functions are compared under the following settings:

- The number of patients `n = 10`.

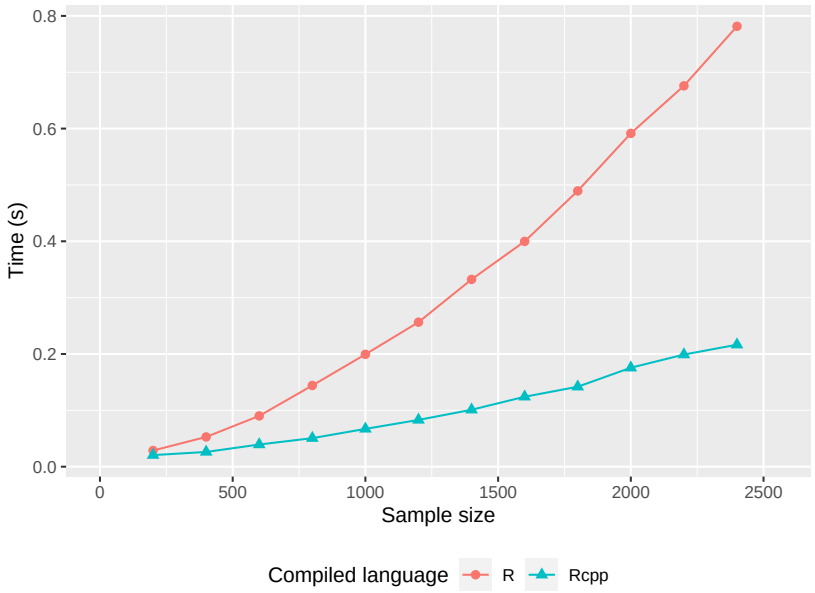

Figure 7: Comparison of the run-time (s) for R and **Rcpp** under Atkinson's D_A-optimal biased coin design with various sample sizes.

- The number of covariates `cov_num = 4`, and each covariate has two levels.
- The probability of one covariate falling into each level is equivalent.
- The number of replications in tests is 10.
- Data is generated by a linear model with the coefficient `beta = c(0.1, 0.4, 0.3, 0.1, 0.3, 0.2, 0.1, 0.2)`, the group treatment effect difference is 0.1, and the variance of the error term is 1.
- The randomization method is Hu and Hu's method with `omega = c(0.1, 0.1, 0.2, 0.2, 0.2, 0.2)`, and the bias coin probability `p = 0.85`.
- The number of Monte-Carlo samples `Iternum = 25600`.

The run-time of test functions based on different computing tools is displayed in Table 4. The run-time in this table was generated on a 1456 core high-performance computing (HPC) cluster, comprising 55 computing nodes (including 4 four-way computing nodes and 50 dual-way computing nodes), with 4608 GB RAM in total. These were also evaluated through the function `microbenchmark()` in R, and the number of evaluation times was set to be 10. From the first two rows, the improvement caused solely by **Rcpp** is already significant. Then, we compare two of the most commonly used parallel computation tools: **OpenMP** in **Rcpp**, and **doSNOW** (Daniel, Microsoft Corporation, and Weston 2022) in R. The execution time of **doSNOW** markedly exceeds that of **OpenMP**. Moreover, the run-time is further reduced by enabling **OpenMP** with multiple cores. For example, the run-time of the function in **carat** with 16 cores is only approximately 0.056% of the function solely coded by R, and the execution time will be even less with more cores.

Computing tools	25% Quantile	Mean	Median	75% Quantile
R only	23325.86	23434.64	23420.90	23542.22
Rcpp	21.99	22.18	22.16	22.30
Rcpp + **OpenMP** (2 cores)	14.99	15.00	15.03	15.18
Rcpp + **OpenMP** (4 cores)	14.28	14.31	14.33	14.37
Rcpp + **OpenMP** (8 cores)	13.67	13.46	13.74	13.75
Rcpp + **OpenMP** (16 cores)	13.03	13.17	13.18	13.28
Rcpp + **doSNOW**	2174.15	2177.60	2177.36	2181.38
Rcpp + **doSNOW** (2 cores)	1659.96	1666.32	1662.31	1663.37
Rcpp + **doSNOW** (4 cores)	5651.76	6818.12	7226.78	8096.69
Rcpp + **doSNOW** (8 cores)	8856.10	9283.81	9197.05	9849.45
Rcpp + **doSNOW** (16 cores)	8995.08	9145.53	9235.94	9378.86

Table 4: Comparison of run-times (in minutes) of different computing tools for power calculation, under the covariate-adjusted biased coin design and the randomization test.

Unfortunately, we were recently informed that **OpenMP** will not be supported in R 4.0.0 on macOS because of the system compiler. As a temporary solution, we have removed the parallel computation in the Comprehensive R Archive Network (CRAN) release of **carat**, but the parallel version is still available via GitHub `https://github.com/yexiaoqingruc/caratOMP`. We will pay close attention to the subsequent development of **OpenMP** in R and make the required modifications.

As a byproduct of the computational efficiency of **carat**, the asymptotic properties of a covariate-adaptive randomization procedure can be easily verified by gradually increasing the sample size. Here, a simulation of the 2×2 case (i.e., two covariates with two levels for each covariate) is conducted for all the randomization procedures included in the package with different sample sizes. The settings are as follows:

- Sample sizes are 200, 400, 600, ..., 1800, 2000.
- 1000 iterations are imposed on each case with each fixed sample size.
- Specific parameters for different procedures: `omega = c(1, 2, rep(1, 2))`, `weight = rep(1, 2)`, `p = 0.85`, `bsize = 4` and `a = 3.0`.

Standard deviations for all of the imbalances of different levels have been achieved. For simplicity, only the overall imbalance and the results corresponding to covariate-margin (1; 1) and strata (1, 1) are reported in Figure 8. It can be seen that the standard deviations of the three levels stabilize as the sample size increases under Hu and Hu's general randomization, the stratified biased coin design, the stratified permuted block randomization, and the covariate-adjusted biased coin design. This result indicates that these imbalances are bounded in probability. The standard deviations at the within-stratum level also increase with the sample size under Pocock and Simon's minimization, although the overall and within-covariate-margin imbalances tend to be stable with various sample sizes. As for Atkinson's D_A-optimal biased coin design, the standard deviations of the imbalances at all the levels increase with the sample size, whereas it outperforms the complete randomization. All of these

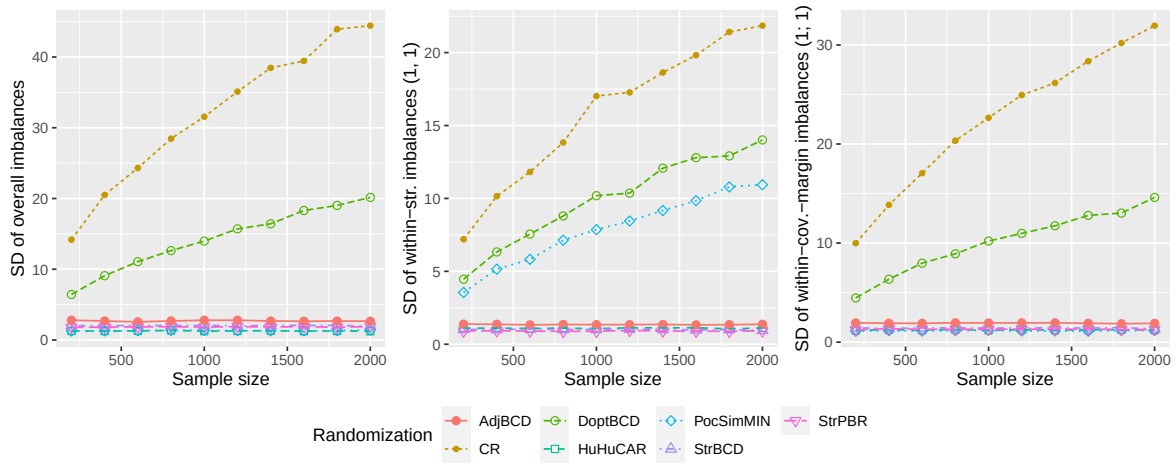

Figure 8: Standard deviations (SD) of imbalances at the overall, within-stratum, and within-covariate-margin levels under various randomization procedures and sample sizes. Here, "CR" represents the complete randomization.

findings are consistent with results previously reported in the literature (e.g., Atkinson 1982; Baldi Antognini and Zagoraiou 2011; Hu and Hu 2012), which partly validates the accuracy of the algorithms in the package **carat**.

6. Summary and discussion

In this paper, we have presented the package **carat** to facilitate the design and analysis of covariate-adaptive clinical trials. The highlights of **carat** are as follows. (1) It provides the most comprehensive spectrum of covariate-adaptive randomization and inference methods, and incorporates recent developments in the field. It currently supports six randomization procedures and three hypothesis tests. (2) It contains evaluation and comparison tools to enable users to easily assess the performances of the randomization and testing functions under different scenarios. (3) A command-line interface is implemented for interactive allocations when patients are enrolled sequentially. (4) The package provides power analysis tools to assist investigators in performing power calculations for the determination of sample size at the planning stage of a covariate-adaptive clinical trial. (5) C++ code is used throughout the package to deal with computationally intensive tasks, and parallelization by **OpenMP** is also available to fully utilize the power of multi-core processors.

To the best of our knowledge, the current version of **carat** is the most comprehensive software tool for covariate-adaptive randomization and associated statistical inference. Moreover, several extensions can be made to further improve the package. Extensions may be desired to cover the cases of multi-arm clinical trials, or additional endpoint types such as time-to-event data or count data. More assessment criteria may be added to enable determination of the superiority and suitability of a covariate-adaptive randomization procedure, such as selection bias or predictability. In addition, a more user-friendly interface, such as **Shiny** (Chang *et al.* 2022) or other web-based applications, can be developed for an online interactive allocation system.

Acknowledgments

This work was supported by the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China (grant number 20XNA023).

References

Atkinson AC (1999). "Optimum Biased-Coin Designs for Sequential Treatment Allocation with Covariate Information." *Statistics in Medicine*, **18**(14), 1741–1752. doi:10.1002/(sici)1097-0258(19990730)18:14<1741::aid-sim210>3.0.co;2-f.

Atkinson AC (1982). "Optimum Biased Coin Designs for Sequential Clinical Trials with Prognostic Factors." *Biometrika*, **69**(1), 61–67. doi:10.1093/biomet/69.1.61.

Baden LR, El Sahly HM, Essink B, Kotloff K, Frey S, Novak R, Diemert D, Spector SA, Rouphael N, Creech CB, McGettigan J, Khetan S, Segall N, Solis J, Brosz A, Fierro C, Schwartz H, Neuzil K, Corey L, Gilbert P, Janes H, Follmann D, Marovich M, Mascola J, Polakowski L, Ledgerwood J, Graham BS, Bennett H, Pajon R, Knightly C, Leav B, Deng W, Zhou H, Han S, Ivarsson M, Miller J, Zaks T (2021). "Efficacy and Safety of the mRNA-1273 SARS-CoV-2 Vaccine." *New England Journal of Medicine*, **384**(5), 403–416. doi:10.1056/NEJMoa2035389.

Baldi Antognini A, Zagoraiou M (2011). "The Covariate-Adaptive Biased Coin Design for Balancing Clinical Trials in the Presence of Prognostic Factors." *Biometrika*, **98**(3), 519–535. doi:10.1093/biomet/asr021.

Begg CB, Kalish LA (1984). "Treatment Allocation for Nonlinear Models in Clinical Trials: The Logistic Model." *Biometrics*, **40**(2), 409–420. doi:10.2307/2531394.

Bland M (2004). ***Clinstat**: Simple Statistical Software*. URL https://www-users.york.ac.uk/~mb55/soft/soft.htm.

Cai HW, Xia JL, Gao DH, Cao XM (2010). "Implementation and Experience of a Web-Based Allocation System with Pocock and Simon's Minimization Methods." *Contemporary Clinical Trials*, **31**(6), 510–513. doi:10.1016/j.cct.2010.07.009.

Canty A, Ripley BD (2022). ***boot**: Bootstrap Functions*. R package version 1.3-28.1, URL https://CRAN.R-project.org/package=boot.

Carey V, Gentleman R (2023). ***randPack**: Randomization Routines for Clinical Trials*. Bioconductor package version 1.44.0, URL https://bioconductor.org/packages/randPack.

Chang W, Cheng J, Allaire JJ, Sievert C, Schloerke B, Xie Y, Allen J, McPherson J, Dipert A, Borges B (2022). ***shiny**: Web Application Framework for R*. R package version 1.7.4, URL https://CRAN.R-project.org/package=shiny.

Coppock A, Cooper J, Fultz N (2023). ***randomizr**: Easy-to-Use Tools for Common Forms of Random Assignment and Sampling*. R package version 0.24.0, URL https://CRAN.R-project.org/package=randomizr.

Dagum L, Menon R (1998). "**OpenMP**: An Industry-Standard API for Shared-Memory Programming." *IEEE Computational Science and Engineering*, **5**(1), 46–55. doi:10.1109/99.660313.

Dallal GE (2003). *Randomization Plans: Never the Same Thing Twice!* URL http://www.randomization.com/.

Daniel F, Microsoft Corporation, Weston S (2022). ***doSNOW**: Foreach Parallel Adaptor for the **snow** Package*. R package version 1.0.20, URL https://CRAN.R-project.org/package=doSNOW.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Eddelbuettel D, François R, Allaire JJ, Ushey K, Kou Q, Russell N, Bates D, Chambers J (2023). ***Rcpp**: Seamless R and C++ Integration*. R package version 1.0.10, URL https://CRAN.R-project.org/package=Rcpp.

Eddelbuettel D, Sanderson C (2014). "**RcppArmadillo**: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics & Data Analysis*, **71**, 1054 –1063. doi:10.1016/j.csda.2013.02.005.

Efron B (1971). "Forcing a Sequential Experiment to Be Balanced." *Biometrika*, **58**(3), 403–417. doi:10.1093/biomet/58.3.403.

Evans S, Day S, Royston P (1995). ***Minim**: Minimisation Program for Allocating Patients to Treatments in Clinical Trials*. Department of Clinical Epidemiology, The London Medical College, URL https://www-users.york.ac.uk/~mb55/guide/minim.htm.

Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge Y, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini AJ, Sawitzki G, Smith C, Smyth G, Tierney L, Yang JYH, Zhang J (2004). "**Bioconductor**: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology*, **5**(10), R80. doi:10.1186/gb-2004-5-10-r80.

GraphPad Software Inc (2017). *GraphPad **QuickCalcs***. URL http://www.graphpad.com/quickcalcs.

Greg S (2020). ***blockrand**: Randomization for Block Random Clinical Trials*. R package version 1.5, URL https://CRAN.R-project.org/package=blockrand.

Hothorn T, Hornik K, Van de Wiel MA, Zeileis A (2006). "A Lego System for Conditional Inference." *The American Statistician*, **60**(3), 257–263. doi:10.1198/000313006X118430.

Hu Y, Hu F (2012). "Asymptotic Properties of Covariate-Adaptive Randomization." *The Annals of Statistics*, **40**(3), 1794–1815. doi:10.1214/12-aos983.

Kalish LA, Begg CB (1985). "Treatment Allocation Methods in Clinical Trials: A Review." *Statistics in Medicine*, **4**(2), 129–144. doi:10.1002/sim.4780040204.

Keller MB, McCullough JP, Klein DN, Arnow B, Dunner DL, Gelenberg AJ, Markowitz JC, Nemeroff CB, Russell JM, Thase ME, Trivedi MH, Blalock JA, Borian FE, Jody DN, DeBattista C, Koran LM, Schatzberg AF, Fawcett J, Hirschfeld RMA, Keitner G, Miller I, Kocsis JH, Kornstein SG, Manber R, Ninan PT, Rothbaum B, Rush AJ, Vivian D, Zajecka J (2000). "A Comparison of Nefazodone, the Cognitive Behavioral-Analysis System of Psychotherapy, and Their Combination for the Treatment of Chronic Depression." *New England Journal of Medicine*, **342**(20), 1462–1470. doi:10.1056/nejm200005183422001.

Kundt G (2009). "Comparative Evaluation of Balancing Properties of Stratified Randomization Procedures." *Methods of Information in Medicine*, **48**, 129–134. doi:10.3414/me0538.

Lin Y, Zhu M, Su Z (2015). "The Pursuit of Balance: An Overview of Covariate-Adaptive Randomization Techniques in Clinical Trials." *Contemporary Clinical Trials*, **45**, 21–25. doi:10.1016/j.cct.2015.07.011.

Ma W, Hu F, Zhang L (2015). "Testing Hypotheses of Covariate-Adaptive Randomized Clinical Trials." *Journal of the American Statistical Association*, **110**(510), 669–680. doi:10.1080/01621459.2014.922469.

Ma W, Qin Y, Li Y, Hu F (2020). "Statistical Inference for Covariate-Adaptive Randomization Procedures." *Journal of the American Statistical Association*, **115**(531), 1488–1497. doi:10.1080/01621459.2019.1635483.

Metropolis N, Ulam S (1949). "The Monte Carlo Method." *Journal of the American Statistical Association*, **44**(247), 335–341. doi:10.1080/01621459.1949.10483310.

Pocock SJ, Simon R (1975). "Sequential Treatment Assignment with Balancing for Prognostic Factors in the Controlled Clinical Trial." *Biometrics*, **31**(1), 103–115. doi:10.2307/2529712.

R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rosenberger WF, Lachin JM (2015). *Randomization in Clinical Trials: Theory and Practice*. John Wiley & Sons, New Jersey.

Rosenberger WF, Sverdlov O (2008). "Handling Covariates in the Design of Clinical Trials." *Statistical Science*, **23**(3), 404–419. doi:10.1214/08-sts269.

Rosenberger WF, Uschner D, Wang Y (2019). "Randomization: The Forgotten Component of the Randomized Clinical Trial." *Statistics in Medicine*, **38**(1), 1–12. doi:10.1002/sim.7901.

Ryan P (2018). "**RALLOC**: Stata Module to Design Randomized Controlled Trials." URL https://EconPapers.repec.org/RePEc:boc:bocode:s319901.

Saghaei M (2004). "Random Allocation Software for Parallel Group Randomized Trials." *BMC Medical Research Methodology*, **4**(1), 1–6. doi:10.1186/1471-2288-4-26.

Saghaei M, Saghaei S (2011). "Implementation of an Open-Source Customizable Minimization Program for Allocation of Patients to Parallel Groups in Clinical Trials." *Journal of Biomedical Science and Engineering*, **4**(11), 734–739. doi:10.4236/jbise.2011.411090.

Schindler D, Uschner D, Hilgers RD, Heussen N (2023). ***randomizeR**: Randomization for Clinical Trials*. R package version 3.0.1, URL https://CRAN.R-project.org/package=randomizeR.

Shao J, Yu X (2013). "Validity of Tests under Covariate-Adaptive Biased Coin Randomization and Generalized Linear Models." *Biometrics*, **69**(4), 960–969. doi:10.1111/biom.12062.

Shao J, Yu X, Zhong B (2010). "A Theory for Testing Hypotheses under Covariate-Adaptive Randomization." *Biometrika*, **97**(2), 347–360. doi:10.1093/biomet/asq014.

Smith RL (1984). "Sequential Treatment Allocation Using Biased Coin Designs." *Journal of the Royal Statistical Society B*, **46**(3), 519–543. doi:10.1111/j.2517-6161.1984.tb01323.x.

Stroustrup B (2013). *The C++ Programming Language*. 4th edition. Addison-Wesley.

Taves DR (1974). "Minimization: A New Method of Assigning Patients to Treatment and Control Groups." *Clinical Pharmacology and Therapeutics*, **15**(5), 443–453. doi:10.1002/cpt1974155443.

Tu F, Ye X, Ma W, Hu F (2023). ***carat**: Covariate-Adaptive Randomization for Clinical Trials*. R package version 2.2.1, URL https://CRAN.R-project.org/package=carat.

Wang Y, Zhang D, Du G, Du R, Zhao J, Jin Y, Fu S, Gao L, Cheng Z, Lu Q, Hu Y, Luo G, Wang K, Lu Y, Li H, Wang S, Ruan S, Yang C, Mei C, Wang Y, Ding D, Wu F, Tang X, Ye X, Ye Y, Liu B, Yang J, Yin W, Wang A, Fan G, Zhou F, Liu Z, Gu X, Xu J, Shang L, Zhang Y, Cao L, Guo T, Wan Y, Qin H, Jiang Y, Jaki T, Hayden FG, Horby PW, Cao B, Wang C (2020). "Remdesivir in Adults with Severe COVID-19: A Randomised, Double-Blind, Placebo-Controlled, Multicentre Trial." *The Lancet*, **395**(10236), 1569–1578. doi:10.1016/s0140-6736(20)31022-9.

Wickham H (2016). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag, New York.

Yu J, Lai D (2019). "Sequential Monitoring of Covariate Adaptive Randomized Clinical Trials with Sample Size Re-Estimation." *Contemporary Clinical Trials*, **87**. doi:10.1016/j.cct.2019.105874.

Zelen M (1974). "The Randomization and Stratification of Patients to Clinical Trials." *Journal of Chronic Diseases*, **27**(7), 365–375. doi:10.1016/0021-9681(74)90015-0.

Zhu H, Hu F (2019). "Sequential Monitoring of Covariate-Adaptive Randomized Clinical Trials." *Statistica Sinica*, **29**(1), 265–282. doi:10.5705/ss.202016.0330.

A. Process of command-line interactive functions

```
Is this the first patient?
Enter T or F: T
Please enter the involved covariates:
  Please enter a new covariate:
    Notice: If no more covariates to be entered,
            please PRESS Enter directly
New Covariate: sex
  Please enter a new covariate:
    Notice: If no more covariates to be entered,
            please PRESS Enter directly
New Covariate: age
  Please enter a new covariate:
    Notice: If no more covariates to be entered,
            please PRESS Enter directly
New Covariate: pills
  Please enter a new covariate:
    Notice: If no more covariates to be entered,
            please PRESS Enter directly
New Covariate:
According to your input, all covariates are stamped to be


         sex -- 1
         age -- 2
         pills -- 3



Continue or not?
 'n' -- stop running  input 'y' or PRESS Enter -- reenter or save

Enter y or n:
Reenter involved covariates?
Enter y or n:


Please enter LEVELs for each covariate:
  Enter the new LEVEL for covariate -- sex:
    Notice: If no more level to be entered for sex,
            please PRESS Enter directly
New level: male
  Enter the new LEVEL for covariate -- sex:
    Notice: If no more level to be entered for sex,
            please PRESS Enter directly
```

```
New level: female
  Enter the new LEVEL for covariate -- sex:
    Notice: If no more level to be entered for sex,
            please PRESS Enter directly
New level:
According to you input, levels for covariate -- sex

1--sex

        male--1
        female--2


Reenter LEVELs for covariate -- sex or not?
Enter y or n:
  Enter the new LEVEL for covariate -- age:
    Notice: If no more level to be entered for age,
            please PRESS Enter directly
New level: 0-30
  Enter the new LEVEL for covariate -- age:
    Notice: If no more level to be entered for age,
            please PRESS Enter directly
New level: 30-50
  Enter the new LEVEL for covariate -- age:
    Notice: If no more level to be entered for age,
            please PRESS Enter directly
New level: >=51
  Enter the new LEVEL for covariate -- age:
    Notice: If no more level to be entered for age,
            please PRESS Enter directly
New level:
According to you input, levels for covariate -- age

2--age

        0-30--1
        30-50--2
        >=51--3


Reenter LEVELs for covariate -- age or not?
Enter y or n:
  Enter the new LEVEL for covariate -- pills:
    Notice: If no more level to be entered for pills,
            please PRESS Enter directly
New level: 0
  Enter the new LEVEL for covariate -- pills:
```

```
    Notice: If no more level to be entered for pills,
            please PRESS Enter directly
New level: 1-3
  Enter the new LEVEL for covariate -- pills:
    Notice: If no more level to be entered for pills,
            please PRESS Enter directly
New level: 3-5
  Enter the new LEVEL for covariate -- pills:
    Notice: If no more level to be entered for pills,
            please PRESS Enter directly
New level: >=6
  Enter the new LEVEL for covariate -- pills:
    Notice: If no more level to be entered for pills,
            please PRESS Enter directly
New level:
According to you input, levels for covariate -- pills

3--pills

        0--1
        1-3--2
        3-5--3
        >=6--4


Reenter LEVELs for covariate -- pills or not?
Enter y or n:
Please allocate WEIGHTs to each aspects:
  Notice: larger the absolute value you enter, stronger tendency
            to obtain balance on the corresponding aspect.
Enter the weight for the OVERALL aspect: 1
Enter the weight for the WITHIN-STRATUM aspect: 2
Enter the weight for the MARGIN -- sex : 1
Enter the weight for the MARGIN -- age : 2
Enter the weight for the MARGIN -- pills : 1
Weights for each aspects are:


        OVERALL--0.142857142857143
        WITHIN-STRT.--0.285714285714286

        sex--0.142857142857143
        age--0.285714285714286
        pills--0.142857142857143


Reenter weights or not?
```

```
Enter y or n:
Please enter the biased coin probability (0-1):
Enter the probability: 0.85
 Please enter COVARIATE PROFILE of the coming patients:
Please enter the level of covariate---sex:
Enter the level: female
Please enter the level of covariate---age:
Enter the level: 30-50
Please enter the level of covariate---pills:
Enter the level: >=6
COVARIATE PROFILE of the coming patient is:



        sex -- female

        age -- 30-50

        pills -- >=6



Reenter COVARIATE PROFILE or not?
Enter y or n:
```

B. Detailed output of `compRand()`

```
Comparison:
Randomization = HuHuCAR, PocSimMIN, StrPBR
Data Type:  Simulated
Data generation:  TRUE
group = A B
Sample size = 1000
iteration = 500
cov_num = 3
level_num = 2 5 2

Mean absolute imbalances at overall, within-strt. and
within-cov.-margin levels:
Overall:
          max 95%-quan median  mean
HuHuCAR     4        2      0 1.016
PocSimMIN   6        2      0 0.948
StrPBR     14        8      2 3.144
```

```
Within-strt.:
            max 95%-quan median  mean
HuHuCAR    4.95      2.8      1 1.016
PocSimMIN 18.25     11.2      4 4.556
StrPBR     2.00      2.0      1 0.673

Within-cov.-margin:
           max 95%-quan median mean
HuHuCAR   5.44     3.00   1.00 1.25
PocSimMIN 4.67     3.00   1.00 1.06
StrPBR    7.44     4.44   1.44 1.81
```

C. Detailed output of `compRand()` for nefazodone CBASP data

```
Comparison:
Randomization = HuHuCAR, PocSimMIN, StrBCD, StrPBR, AdjBCD, DoptBCD
Data Type:  Real
group = A B
Sample size = 440
iteration = 500
cov_num = 3
level_num = 2 2 5

Mean absolute imbalances at overall, within-strt. and
within-cov.-margin levels:
Overall:
          max 95%-quan median  mean
HuHuCAR     6        2      0 0.892
PocSimMIN   4        2      0 0.940
StrBCD     10        8      2 2.796
StrPBR     10        8      2 2.872
AdjBCD     14       10      4 4.168
DoptBCD    26       18      6 7.584

Within-strt.:
           max 95%-quan median  mean
HuHuCAR   3.38     2.00  0.462 0.874
PocSimMIN 5.23     3.39  1.385 1.483
StrBCD    2.92     1.54  0.462 0.664
StrPBR    1.23     1.23  0.462 0.718
AdjBCD    2.77     1.85  1.462 1.059
DoptBCD   7.38     4.77  2.000 2.104

Within-cov.-margin:
            max 95%-quan median mean
```

```
HuHuCAR    4.67     2.67  0.667 1.17
PocSimMIN  4.89     2.67  0.667 1.09
StrBCD     6.44     4.00  1.333 1.59
StrPBR     4.89     3.78  1.333 1.62
AdjBCD     7.56     5.11  2.000 2.20
DoptBCD   14.22     9.34  3.333 3.95
```

Affiliation:

Feifang Hu
Department of Statistics
George Washington University
801 22nd St. NW, 7th Floor
Washington, DC 20052, United States of America
Telephone: 804-310-0383
E-mail: feifang@gwu.edu
URL: https://statistics.columbian.gwu.edu/feifang-hu

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/
September 2023, Volume 107, Issue 2 *Submitted:* 2020-11-02
doi:10.18637/jss.v107.i02 *Accepted:* 2023-01-18