Causal Inference Final Project

CBPS and Entropy Balancing - A simulation Study

Chansoo Song

November 2018

Introduction:

Matching:

In an observational study setting where the confounding covariates are known and measured, we may use mathching methods to ensure that there is sufficient overlap and balance on these covariates. Then, we can estimate the treatement effect using a simple difference in means or regression methods.

Overlap is important because we want to make sure that for each treated or control subject in the study, there exists an empirical counterfactual (this criteria varies depending on the estimand of interest, i.e. to estimate the ATT it is sufficient to have empirical counterfactuals for just the treated subjects in the study). Imbalance on the covariates forces us to rely more on the correct functional form of the model.

There are many different matching methods, but the driving principle is to identify observations that are "most similar", based on some distance metric. Methods include K-nearest-neighbor, caliper-matching, kernel-matching, Mahalanobis matching, Genetic Matching, Optimal Matching.

Propensity Scores:

A propensity score is a one-number summary of the covariates. Rosenbaum and Rubin (1983) define the propensity score for participant i as the conditional probability of treatment assignment ($Z_i = 1$) given a vector of observed covariates: $e(X_i) = Pr(Z_i = 1|X)$. The most common traditional approaches to estimating the propensity score are logistic regression and probit regression.

If strong ignorability holds after conditioning on the propensity score, that is:

$$y_0, y_1 \perp Z_i|e(X_i),\ 0 < e(X_i) < 1$$

Then we may obtain an unbiased estimate of the treatment effect by either matching or weighting using just the propensity score instead of the vector of covariates.

In practice, a key challenge in the application of propensity scores for matching is that the propensity score is unknown and must be estimated. To make matters worse, slight misspecification of the propensity score model can lead to substantial biases in treatmenet effects. As such, researchers may iteratively re-estimate the propensity score model and subsequently check the resulting covariate balance. Imai et al (2008) calls this the 'propensity score tautology': the estimated propensity score is appropriate if it balances covariates.

In this simulation study, I analyze two approaches that seek to bypass this 'propensity score tautology': Covariate Balancing Propensity Score and Entropy Balancing. Each method obviates the need for iteratively re-estimating the propensity score model and checking balance on the covariate moments. That is, a single model is used to estimate both the treatment assignment mechanism and the covariate balancing weights.

Covariate Balancing Propensity Score (CBPS):

The CBPS exploits the dual characteristics of the propensity score as a covariate balancing score and the conditional probability of treatment assignment (Imai and Ratkovic (2012)).

First, consider a commonly used model for estimating propensity scores: logistic regression:

$$e_B(X_i) = \frac{exp(X_i^T \beta)}{1 + exp(X_i^T \beta)}$$

We typically estimate the unknown parameters by maximum likelihood, i.e. maximizing the log-likelihood function:

$$\hat{\beta}_{MLE} = \sum_{i=1}^{N} Z_i \ log\{e_B(X_i) \ + (1 - Z_i) \ log\{1 - e_B(X_i)\}$$

Differentiating with respect to β, we get:

$$\frac{1}{N} \sum_{i=1}^{N} \frac{Z_i \ e'_B(X_i)}{e_B(X_i)} - \frac{(1 - Z_i) \ e'_B(X_i)}{1 - e_B(X_i)}$$

Imai and Ratkovic emphasize that the equation above can also be interpreted as the condition that balances a particular function of covariates, i.e. the first derivative of $e_B(X_i)$. Then, they operationalize the covariate balancing property by using inverse propensity score weighting:

$$E\left\{\frac{Z_i \ e'_B(\tilde{X}_i)}{e_B(X_i)} - \frac{(1 - Z_i) \ e'_B(\tilde{X}_i)}{1 - e_B(X_i)}\right\} = 0$$

where $\tilde{X}_i = f(X_i)$, a function of X_i specified by the researcher. Setting $tildeX_i = e'_B(X_i)$ gives more weights to covariates that are predictive of treatmeent assignment according to the logistic regression propensity score model. But so long as the expectaion exists, the equation must hold for any choice of f(.). For example, setting $\tilde{X}_i = X_i$ ensures the first moment of each covariate is balanced. Setting $\tilde{X}_i = (X_i^T X_i^{2T})^T$ ensures the first and second moment of each covariate is balanced.

Entropy Balancing:

Entropy balancing similarly involves a reweighting scheme that directly incorporates covariate balance into the weight function that is applied to the sample units (Heinmueller 2015). To do this, entropy balancing searches for a set of weights that satisfies the balance constraints, while trying to keep the distribution of weights as uniform as possible (i.e. minimizing the divergence of distribution of weights from a uniform distribution). Thus, entropy balancing (1) allows us to obtain a high degree of covariate balance (using balance constraints that can involve the first, second, and possibly higher moments of the covariate distributions as well as interactions). And (2) allows for a more flexible reweighting scheme that seeks to retain as much information as possible. For example, nearest neighbor matching may discard subjects that are not matched (i.e. set weight equal to 0).

Consider the reweighting scheme to estimate the Average Treatement Effect on the Treated (ATT). We would want to estimate the counterfactual mean by:

$$E[\widehat{Y(0)|Z} = 1] = \frac{\sum_{i|Z=0} Y_i w_i}{\sum_{i|Z=0} w_i}$$

where w_i is a weight for each control unit.

The weights are chosen by the following reweighting scheme:

$$H(w) = \sum_{i|D=0} h(w_i)$$

where $h(.)$ is a distance metric and $c_{ri}(X_i) = m_r$ describes a set of R balance constraints imposed on the covariate moments of the reweighted control group.

Minimize $H(w)$ subject to the balance and normalizing constraints:

$$\sum_{i|D=0} w_i c_{ri}(X_i) = m_r$$

with $r \in 1, ..., r$

$$\sum_{i|Z=0} w_i = 1$$

and $w_i \geq 0$ for all i such that $T = 0$.

Simulation and DGP Method:

In this section, I examine whether the CBPS or Entropy Balancing methods improve upon the performance of a baseline approach to both (1) achieving balanced covariates and (2) estimating the treatment effect. The baseline approach estimates include (1) propensity scores using logistic regression and matches using 1-1 matching with replacement and (2) mahalanobis matching.

I consider eight data generating processes:

(1) Standard Normally Distributed Covariates: The pre-treatment covariates $X_i = (X_{i1}, X_{i2}, X_{i3}, X_{i4})$ are four independent and identically distributed random variables following a standard normal distribution. The true propensity score model is a logistic regression whose linear predictor is a linear transform of the pre-treatment covariates.

(2) Standard Normally Distributed Covariates (non-linear propensity score model): The pre-treatment covariates are the same as Simulation #1; however, the true propensity score model is a logistic regression whose linear predictor are non-linear transforms of the pre-treatment covariates: $X_i^* = (-exp(X_{i1}/2), -X_{i2}/(1 + exp(X_{i1})), X_{i3}, -sqrt(X_{i4}^2))$

(3) Normally Distributed Covariates with Unequal Variances Same as (1) except the variances of the pre-treatment covariates are no longer equal. That is, the covariates are no longer identically standard normal, but have variances: (0.5, 0.9, 1.1, and 1.2), respectively.

(4) Normally Distributed Covariates with Unequal Variances (non-linear propensity score model): Same as (2) except the variances of the pre-treatment covariates are no longer equal. That is, the covariates are no longer identically standard normal, but have variances: (0.5, 0.9, 1.1, and 1.2), respectively.

(5) Standard Normally Distributed Covariates + 3 count covariates: The pre-treatment covariates $X_i = (X_{i1}, X_{i2}, X_{i3}, X_{i4}, X_{i5}, X_{i6}, X_{i7})$ consist of four independent and identically distributed random variables following a standard normal distribution, a random variable following a poisson distribution with $\lambda = 1$, the negative values of a random variable following a binomial distribution with $n = 3$ and $p = 0.8$, and a random variable following a chi-squared distribution with $df = 1$.

(6) Standard Normally Distributed Covariates + 3 count covariates (non-linear propensity score model): The pre-treatment covariates are the same as Simulation (5); however, the true propensity score model is a logistic regression whose linear predictor are non-linear transforms of the pre-treatment covariates: $X_i^* = (0.5 * exp(X_{i1}/2),\ X_{i2}/(1 + exp(X_{i1})),\ -.2 * X_{i3}^2,\ X_{i1} * X_{i4},\ -0.4 * sqrt(X_{i5} - X_{i6}),\ 0.2 * (X_{i1} + 1.2 * X_{i6})^2,\ 0.5 * X_{i7})$

(7) Normally Distributed Covariates with unequal variances + 3 count covariates: Same as (5) except the variances of the pre-treatment covariates are no longer equal. That is, the covariates are no longer identically standard normal, but have variances: (0.5, 0.9, 1.1, and 1.2), respectively.

(8) Normally Distributed Covariates with unequal variances + 3 count covariates (non-linear propensity score model): Same as (5) except the variances of the pre-treatment covariates are no longer equal. That is, the covariates are no longer identically standard normal, but have variances: (0.5, 0.9, 1.1, and 1.2), respectively.

In all eight simulations, the true outcome model is a linear regression with the pre-treatment covariates as predictors.

Simulations:

Set Up and Initialize:

```
# Samle size
n = 2000
n_sims = 100

# Models
model_names = c('Baseline Logistic', 'Baseline Mahalanobis',
                'CBPS - Over', 'CBPS - Just',
                'EB - 1', 'EB - 2')
dgp_names = c('std normally dist covs',
              'std normally dist covs (non-linear ps model)',
              'normally dist covs with unequal variances',
              'normally dist covs with unequal variances (non-linear ps model)',
              'std normally dist covs with 3 count covariates',
              'std normally dist covs with 3 count covariates (non-linear ps model)',
              'normally dist covs with unequal var and 3 count covs',
              'normally dist covs with unequal var and 3 count covs (non-linear ps model)')
wts = c('wt','wt_mh',
        'wt.cbps_over','wt.cbps_just',
        'wt.eb','wt.eb2')

# Inverse Logit Function
inv.logit = function (x) {
  y = 1/(1+exp(-x))
  return(y)
}


df = list()
SATE = list()
confounders = list()
for(i in 1:4) confounders[[i]] = paste('z',seq(1,4),sep='')
for(i in 5:8) confounders[[i]] = paste('z',seq(1,7),sep='')
```

DGP Function (Pre-Treatment Covariates):

```
# Function to Generate Pre-Treatment Covariates
dgp = function(n){
  Z_ = list()
  X_ = list()

  # Pre-Treatment covs
  z1 = rnorm(n,  0, 1)
```

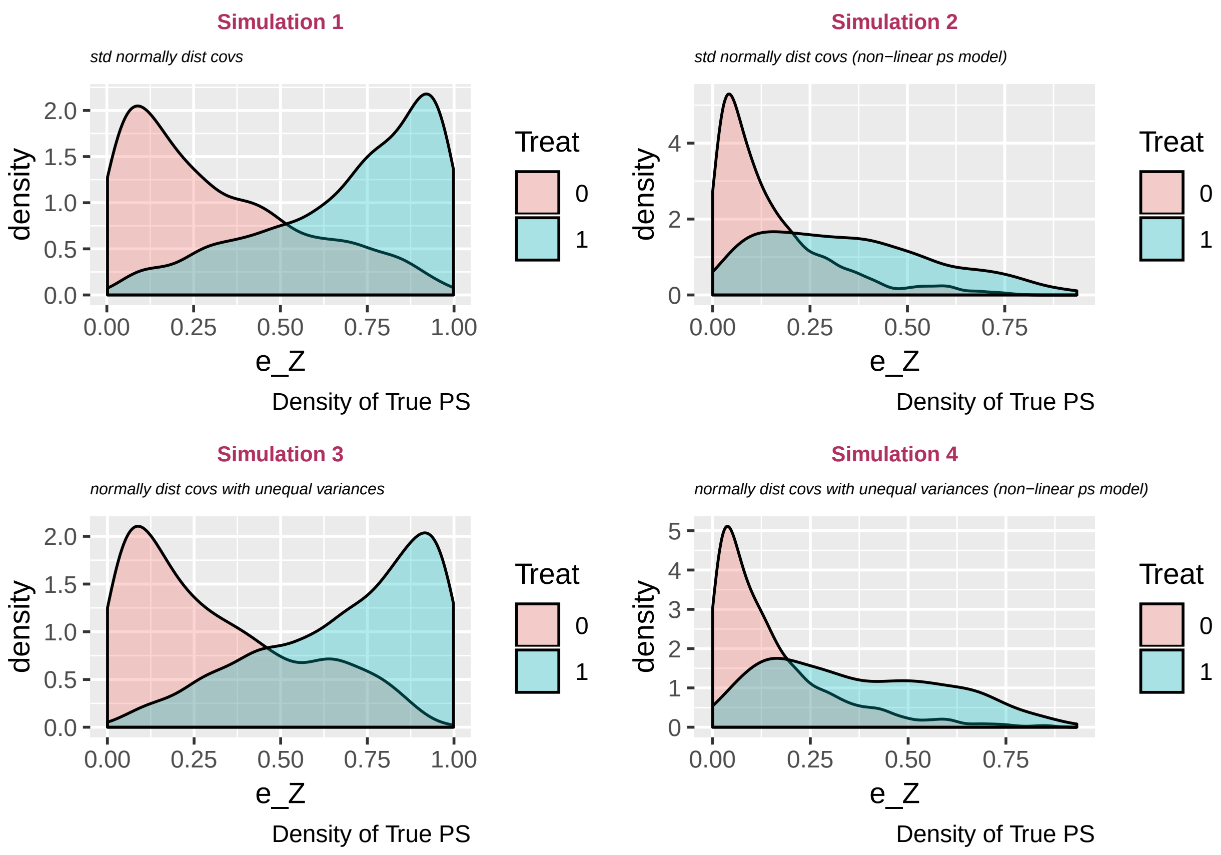
```r
z2 = rnorm(n,  0, 1)
z3 = rnorm(n,  0, 1)
z4 = rnorm(n,  0, 1)

# DGP 1: std normally dist covs
Z_[[1]] = cbind(z1, z2, z3, z4)
X_[[1]] = z1 + z2 + z3 + z4

# DGP 2: std normally dist covs (non-linear ps model)
Z_[[2]] = cbind(z1, z2, z3, z4)
X_[[2]] = -exp(z1/2) - z2/(1+exp(z1)) + z3 - sqrt(z4^2)

# Pre-Treatment covs
z1 = rnorm(n,  0, 0.5)
z2 = rnorm(n,  0, 0.9)
z3 = rnorm(n,  0, 1.1)
z4 = rnorm(n,  0, 1.2)

# DGP 3: normally dist covs with unequal variances
Z_[[3]] = cbind(z1, z2, z3, z4)
X_[[3]] = z1 + z2 + z3 + z4

# DGP 4: normally dist covs with unequal variances (non-linear ps model)
Z_[[4]] = cbind(z1, z2, z3, z4)
X_[[4]] = -exp(z1/2) - z2/(1+exp(z1)) + z3 - sqrt(z4^2)

# Pre-Treatment covs
z1 = rnorm(n,  0, 1)
z2 = rnorm(n,  0, 1)
z3 = rnorm(n,  0, 1)
z4 = rnorm(n,  0, 1)
z5 = rpois(n, 1)
z6 = -rbinom(n, 3, 0.8)
z7 = rchisq(n, df=1)

# DGP 5: std normally dist covs with 3 count covs
Z_[[5]] = cbind(z1, z2, z3, z4, z5, z6, z7)
X_[[5]] = z1 + z2 + z3 + z4 + z5 + z6 + z7

# DGP 6: std normally dist covs with 3 count covs (non-linear ps model)
Z_[[6]] = cbind(z1, z2, z3, z4, z5, z6, z7)
X_[[6]] = 0.5*exp(z1/2) + z2/(1+exp(z1)) + -.2*z3^2 + z1*z4 -
          0.4*sqrt(z5-z6) - 0.2*(z1+1.2*z6)^2 + 0.5*z7

# Pre-Treatment covs
z1 = rnorm(n,  0, 0.5)
z2 = rnorm(n,  0, 0.9)
z3 = rnorm(n,  0, 1)
z4 = rnorm(n,  0, 1.1)
z5 = rpois(n, 1)
z6 = -rbinom(n, 2, 0.8)
z7 = rchisq(n, df=1)
```

```
  # DGP 7: normally dist covs with unequal vars and 3 count covs
  Z_[[7]] = cbind(z1, z2, z3, z4, z5, z6, z7)
  X_[[7]] = z1 + z2 + z3 + z4 + z5 + z6 + z7

  # DGP 8: normally dist covs with unequal vars and 3 count covs (non-linear ps model)
  Z_[[8]] = cbind(z1, z2, z3, z4, z5, z6, z7)
  X_[[8]] = 0.5*exp(z1/2) + z2/(1+exp(z1)) -.2*z3^2 + z1*z4 -
            0.4*sqrt(z5-z6) - 0.2*(z1+1.2*z6)^2 + 0.5*z7

  return(list(Z_,X_))
}
```

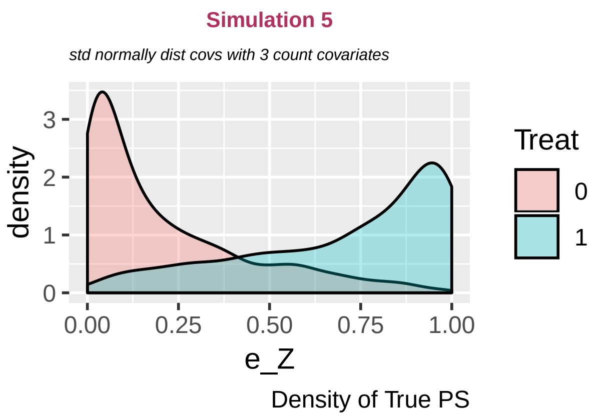
Generate Researcher Dataset Function:

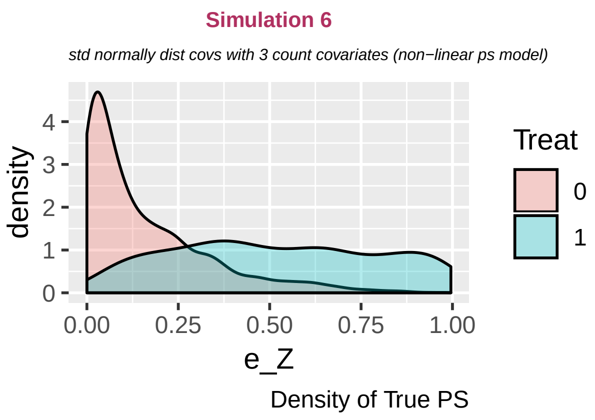
```
# Function to simulate DGP given pre-treatment covariates
sim_dgp = function(Z_=Z_[[1]], X_=X_[[1]], sim_no=1, n,
                   B_=c(0.4,0.5,0.3,0.9), nlo=FALSE, dgp_name='dgp name here'){

  # Propensity Score
  e_Z = inv.logit(X_)

  # Generate treatment vector
  # treat = as.integer(e_Z>runif(n,0,1))
  treat = rbinom(n,1,prob=e_Z)

  # Treatment Effect
  eff = 4

  if(nlo==FALSE){
    Z_2 = Z_
  }

  if(nlo==TRUE){
    Z_2 = Z_
    Z_2[1] = Z_[1]^2
    Z_2[2] = Z_[2]*Z_[1]
    Z_2[3] = Z_[3]^2
    Z_2[4] = sqrt(Z_[4]^2)
  }

  # Generate Potential Outcomes
  y_0 = Z_2 %*% B_ + rnorm(n,0,1)
  y_1 = Z_2 %*% B_ + eff + rnorm(n,0,1)
  y = y_0*(1-treat) + y_1*treat


  # Generate Researcher Dataset
  df.temp = as.data.frame(cbind(y,y_1,y_0, e_Z, treat, Z_))
  colnames(df.temp)[1:3] = c('y','y_1','y_0')

  return(df.temp)
}
```

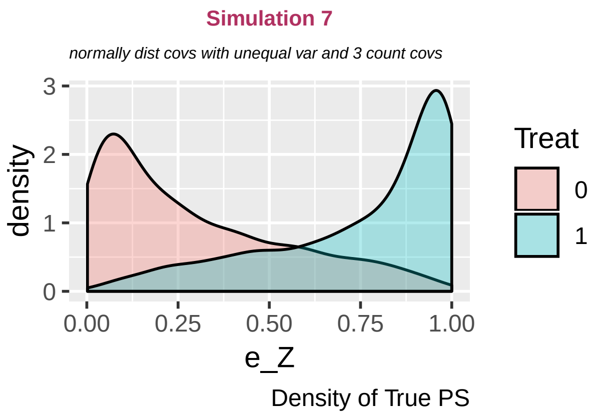
```r
dgp_data = dgp(n)
Z_ = dgp_data[[1]]
X_ = dgp_data[[2]]

b_4 = c(0.4,0.5,0.3,0.9)
b_6 = c(0.4,0.5,0.3,0.9,0.1,1,0.3)
B_ = list(b_4,b_4,b_4,b_4,b_6,b_6,b_6,b_6)


for(k in 1:8){
  df[[k]] = sim_dgp(Z=Z_[[k]], X=X_[[k]], sim_no=k,
                    n=n, B_=B_[[k]], dgp_name=dgp_names[k])
  SATE[[k]] = mean(df[[k]]$y_1 - df[[k]]$y_0)

}

# True Propensity Score Density
true_prs_density = list()
for(k in 1:8){
  true_prs_density[[k]] = ggplot(df[[k]], aes(x=e_Z, group=treat, fill = as.factor(treat))) +
    geom_density(alpha=.3) +
    theme_gray() +
    labs(title = paste("Simulation",k),
         subtitle = dgp_names[k],
         caption = "Density of True PS") +
    labs(fill = "Treat") +
    theme(plot.title=element_text(size=8, hjust=0.5, face="bold", colour="maroon")) +
    theme(plot.subtitle=element_text(size=6, face="italic", color="black"))
}

grid.arrange(grobs=true_prs_density[1:4], ncol=2, nrow = 2)
```

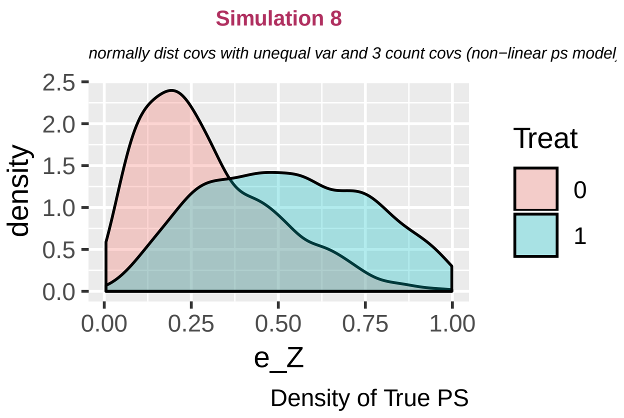


```
grid.arrange(grobs=true_prs_density[5:8], ncol=2, nrow = 2)
```









Matching Methods

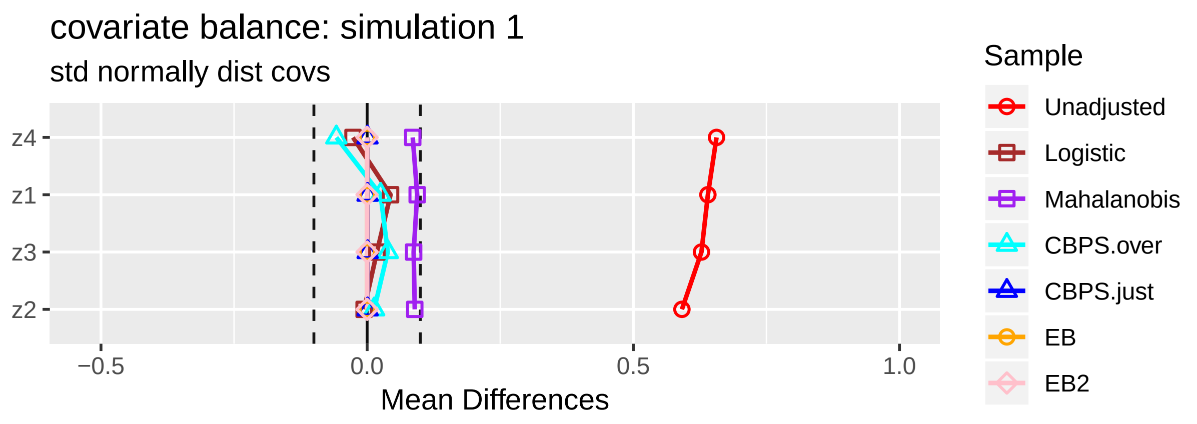
1. Baseline: Propensity Score using Logistic Regression:
    a. Logistic Regression to estimate propensity scores
    b. 1-1 nearest neighbor matching using propensity score
    c. linear regression using propensity score and covariates
2. Baseline: Mahalanobis Matching [Insert explanation of Mahalanobis]
3. Covariate Balancing Propensity Score
    a. Over-identified: combines the propensity score and covariate balancing conditions
4. Covariate Balancing Propensity Score
    b. Just-identified: only contains covariate blancing conditions
5. Entropy Balancing

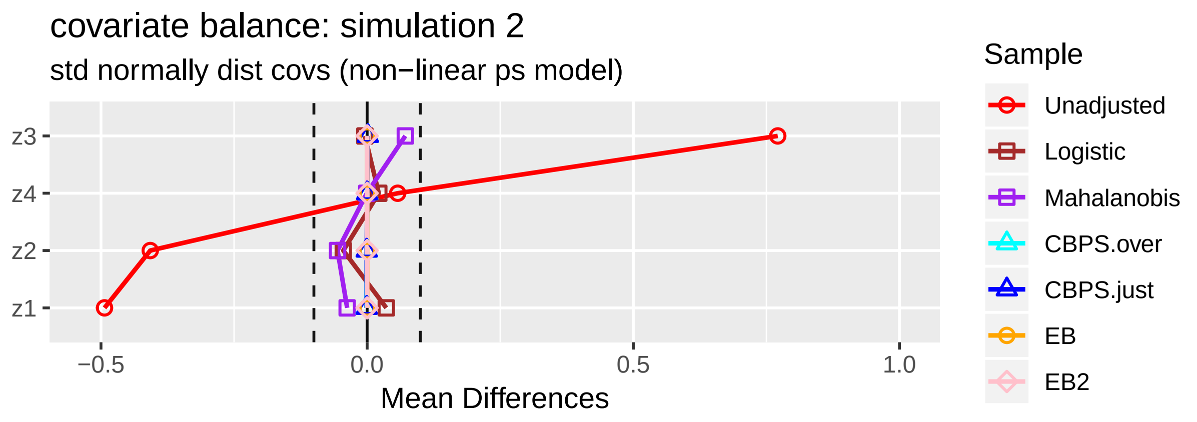
Estimate Weights

```
mod_match = mod_mahalo = cbps.over = cbps.just = eb.out = eb.out2 = list()

for(k in 1:8){
  formula = as.formula(paste('treat ~ ',paste(confounders[[k]],collapse=' + '),sep=''))

  #### (1)

  # Logistic Regression, 1-1 Matching
  mod_match[[k]] = matchit(formula,
                           data = df[[k]][,c('treat',confounders[[k]])],
                           method = 'nearest',
                           distance = 'logit',
                           replace = TRUE)
  df[[k]]$wt = mod_match[[k]]$weights

  #### (2)

  # Mahalanobis Distance Matching
  mod_mahalo[[k]] = matchit(formula,
                            data = df[[k]][,c('treat',confounders[[k]])],
                            method = "nearest",
                            distance = "mahalanobis",
                            replace = TRUE)
  df[[k]]$wt_mh = mod_mahalo[[k]]$weights

  #### (3)

  # CBPS Weights (OVER)
  cbps.over[[k]]= weightit(formula,
                           data = df[[k]][,c('treat',confounders[[k]])],
                           method = "cbps",
                           estimand = "ATT",
                           over=TRUE)
  df[[k]]$wt.cbps_over = cbps.over[[k]]$weights

  #### (4)

  # CBPS Weights (JUST)
  cbps.just[[k]]= weightit(formula,
                           data = df[[k]][,c('treat',confounders[[k]])],
                           method = "cbps",
                           estimand = "ATT",
                           over=FALSE)
  df[[k]]$wt.cbps_just = cbps.just[[k]]$weights

  #### (5)
  # Entropy Balancing Weights 1
  eb.out[[k]] = ebalance(Treatment = df[[k]]$treat,
                         X = df[[k]][,confounders[[k]]])
  df[[k]]$wt.eb = 1
  df[[k]][df[[k]]$treat == 0,'wt.eb'] = eb.out[[k]]$w

  #### (6)
  # Entropy Balancing Weights 2
  eb.out2[[k]] = ebalance(Treatment = df[[k]]$treat,
                          X = cbind(df[[k]][,confounders[[k]]],
                                    df[[k]][,confounders[[k]]]^2))
  df[[k]]$wt.eb2 = 1
  df[[k]][df[[k]]$treat == 0,'wt.eb2'] = eb.out2[[k]]$w
}
```

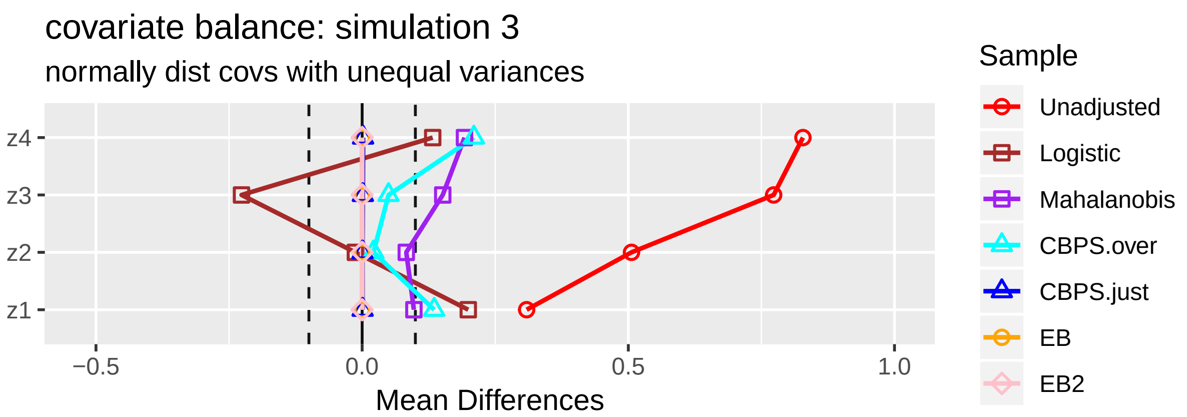
Examine Overlap of Propensity Score and Covariates (Before Matching)

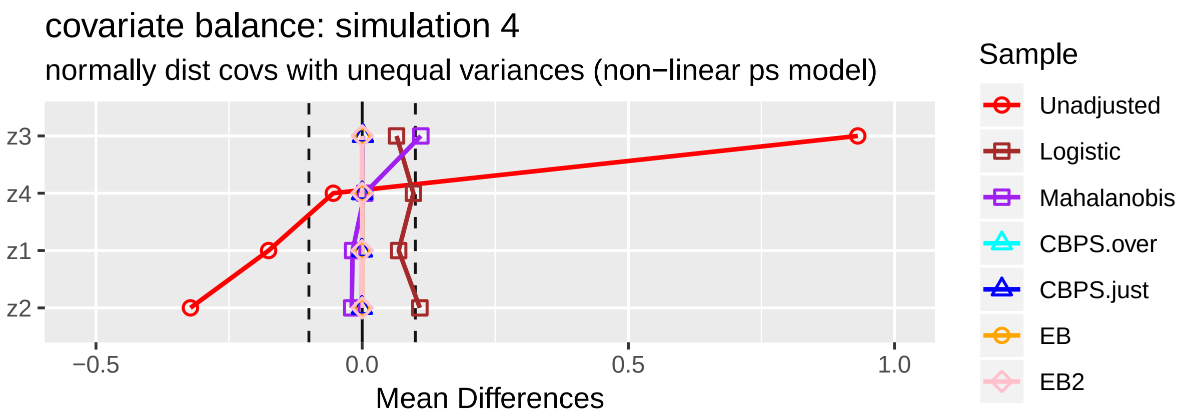
```r
love.plots=list()

for(k in 1:8){
  formula = as.formula(paste('treat ~ ',paste(confounders[[k]],collapse=' + '),sep=''))
  love.plots[[k]] = love.plot(bal.tab(formula,
                          data = df[[k]][,c('treat',confounders[[k]])],
                          weights = data.frame(Logistic = get.w(mod_match[[k]]),
                                               Mahalanobis = get.w(mod_mahalo[[k]]),
                                               CBPS.over = get.w(cbps.over[[k]]),
                                               CBPS.just = get.w(cbps.just[[k]]),
                                               EB = df[[k]]$wt.eb,
                                               EB2 = df[[k]]$wt.eb2),
                          m.threshold = 0.1),
                  var.order = "unadjusted",
                  abs = FALSE, colors = c("red", "brown", "purple", "cyan",
                                          "blue", "orange", "pink"),
                  shapes = c("circle open", "square open",
                             "square open", "triangle open",
                             "triangle open", "circle open", "diamond open"),
                  stat = c('mean.diffs'),
                  line = TRUE) +
          theme_gray() +
          labs(title=paste('covariate balance: simulation',k),
               subtitle=dgp_names[k]) +
    xlim(-0.52,1)
}

grid.arrange(grobs=love.plots[1:2], ncol=1, nrow = 2)
```



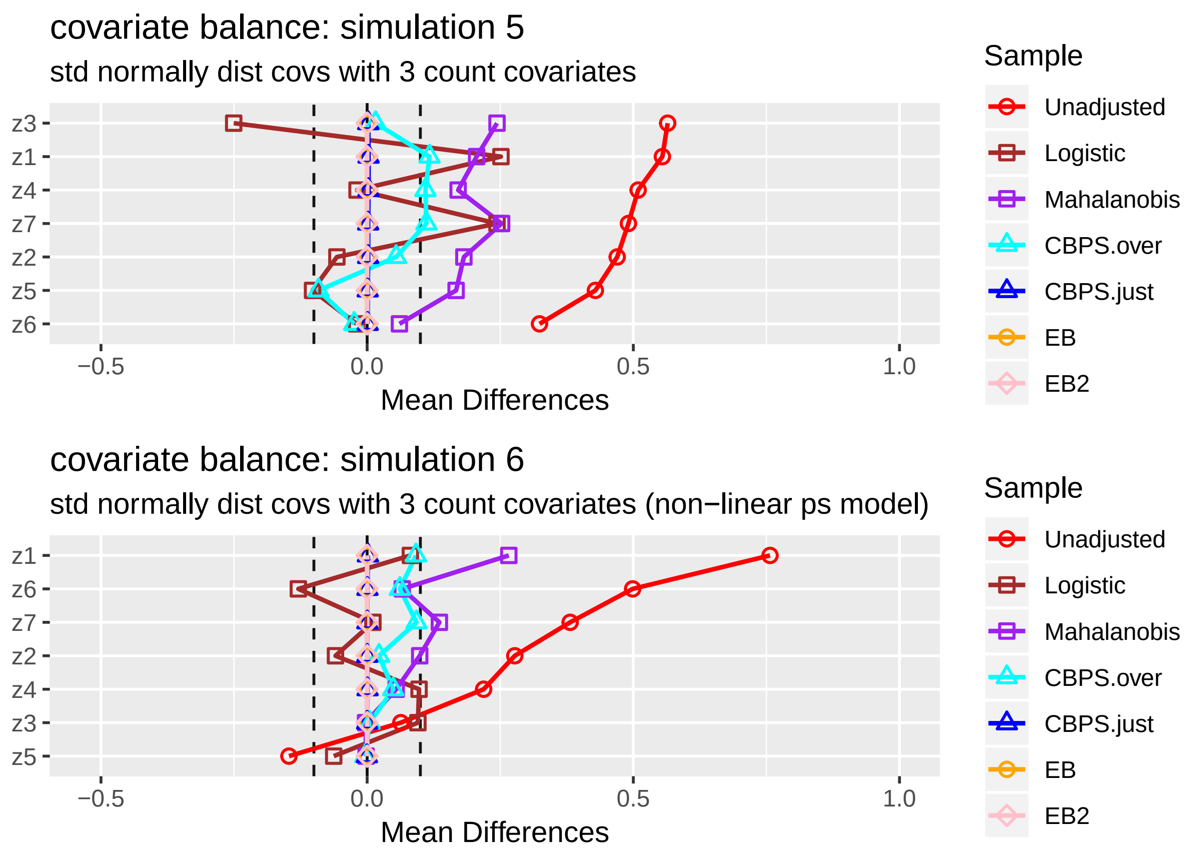


```r
grid.arrange(grobs=love.plots[3:4], ncol=1, nrow = 2)
```

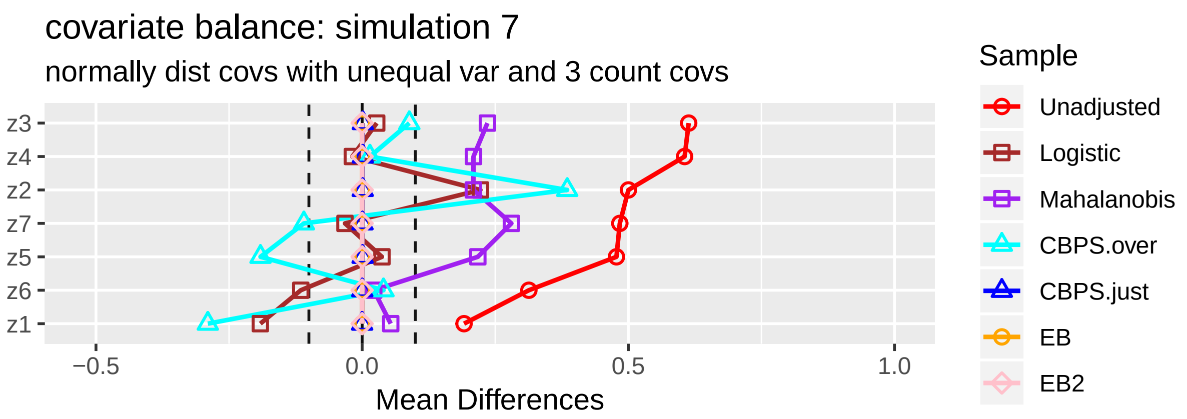


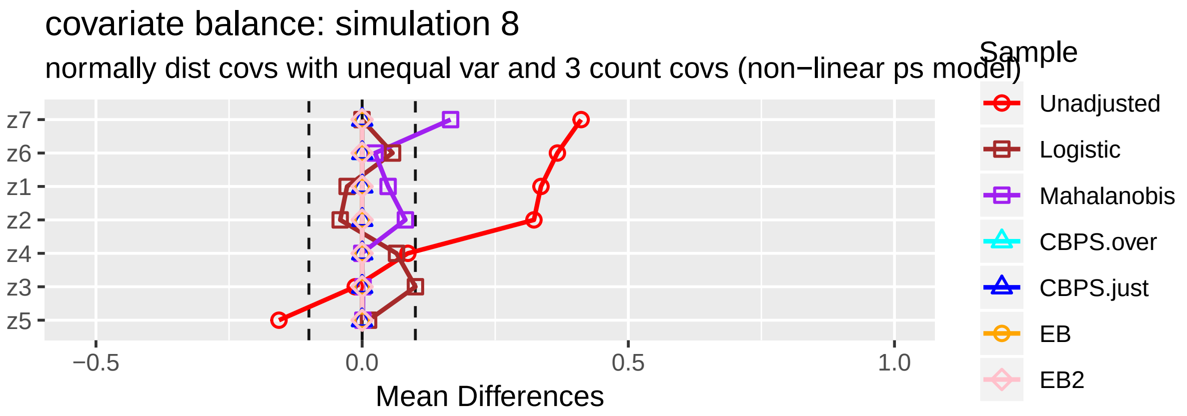


```
grid.arrange(grobs=love.plots[5:6], ncol=1, nrow = 2)
```



```r
grid.arrange(grobs=love.plots[7:8], ncol=1, nrow = 2)
```

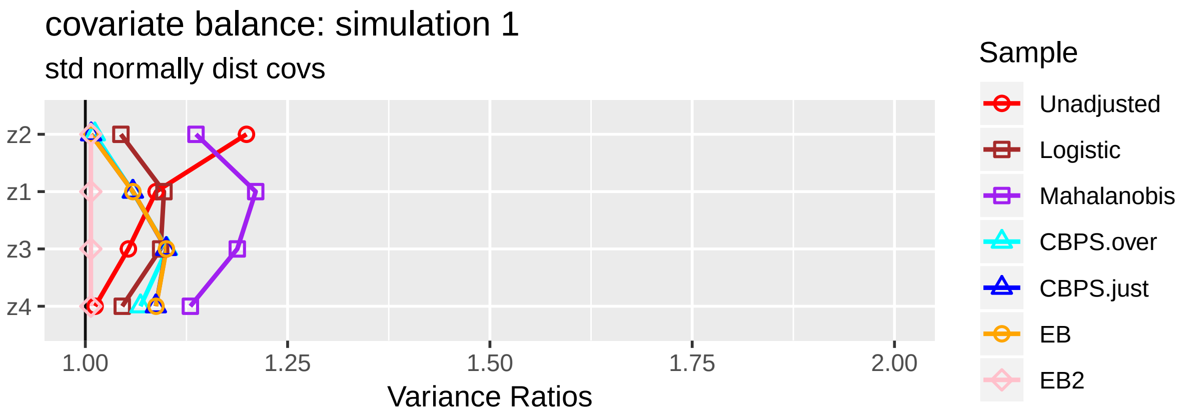


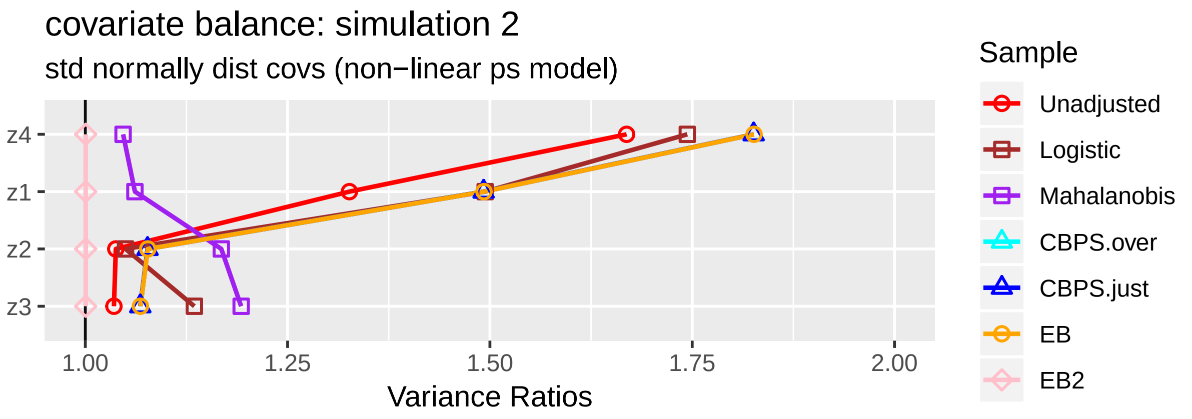


Analysis of Mean Differences:

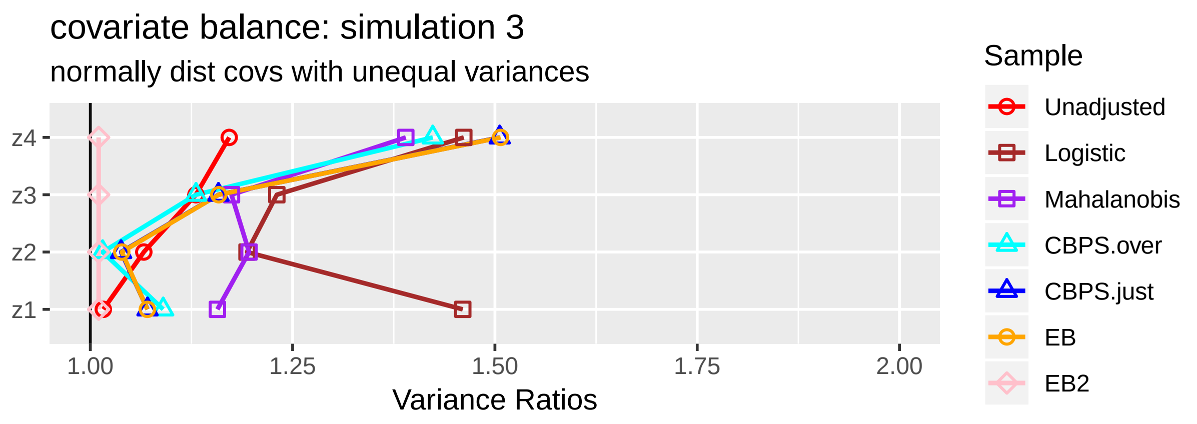
- Entropy balancing shows the best performance with respect to balancing covariate means. For all 8 simulations, the standardized mean difference is approximately zero for all covariates.
- The CBPS just-identified model similarly achieves balanced covariate means, except for simulation #7, where discrete variables are included and the continuous variables have unequal variance.
- The CBPS over-identified model performs significantly better in simulations where the true propensity score model is non-linear and worse in simulations where the true propensity score model is linear. In fact, for 3 of 4 simulations with a non-linear true propensity score model, the CBPS over-identified model achieves a 0 mean difference for all covariates. When the true propensity score model is linear, the covariates' mean differences are very similar to the covariates' mean differences under the logistic model. Remember that the over-identified model combines the propensity score and covariate balancing conditions whereas the just-identified model only contains covariate balancing conditions. It seems likely that when the true propensity score model is linear in the covariates, the propensity score condition dominates the covariate balancing conditions, so the CBPS over-identified model's performance resembles the logistic baseline model's results. For the simulations with a non-linear propensity score model, the propensity score condition no longer dominates, so the CBPS over-identified model's performance resembles the just-identified model's results.
- The logistic regression and mahalanobis matching methods show strong performance in simulation 1, where the pre-treatment covariates have standard normal distributions. Performance appears to weaken when variances are not equal and when the true propensity score model is non-linear.

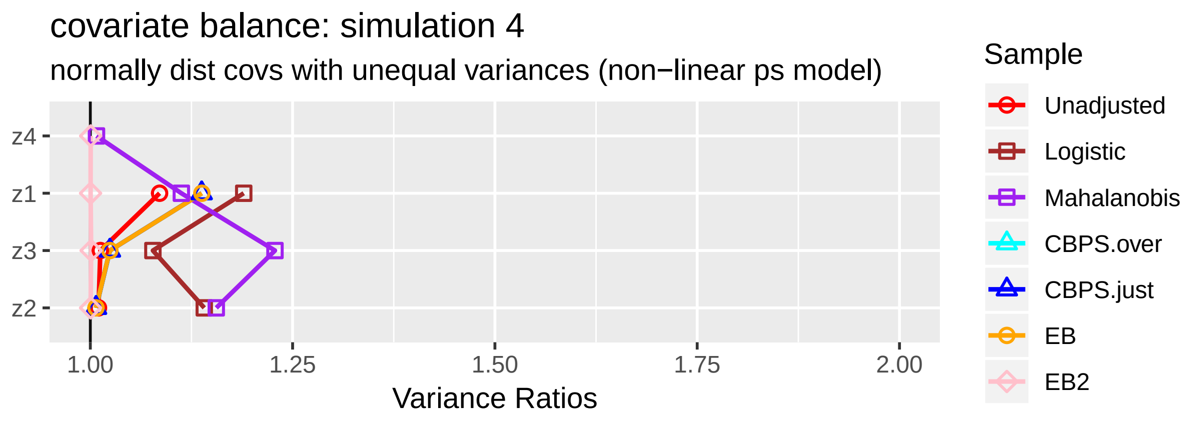
```r
love.plots.var=list()

for(k in 1:8){
  formula = as.formula(paste('treat ~ ',paste(confounders[[k]],collapse=' + '),sep=''))
  love.plots.var[[k]] = love.plot(bal.tab(formula,
                            data = df[[k]][,c('treat',confounders[[k]])],
                            weights = data.frame(Logistic = get.w(mod_match[[k]]),
                                                 Mahalanobis = get.w(mod_mahalo[[k]]),
                                                 CBPS.over = get.w(cbps.over[[k]]),
                                                 CBPS.just = get.w(cbps.just[[k]]),
                                                 EB = df[[k]]$wt.eb,
                                                 EB2 = df[[k]]$wt.eb2),
                            m.threshold = 0.1),
                    var.order = "unadjusted",
                    abs = FALSE, colors = c("red", "brown", "purple",
                                            "cyan", "blue", "orange", "pink"),
                    shapes = c("circle open", "square open",
                               "square open", "triangle open",
                               "triangle open", "circle open", "diamond open"),
                    stat = c('variance.ratios'),
                    line = TRUE) +
            theme_gray() +
            labs(title=paste('covariate balance: simulation',k),
                 subtitle=dgp_names[k])+
  xlim(1,2)
}

grid.arrange(grobs=love.plots.var[1:2], ncol=1, nrow = 2)
```



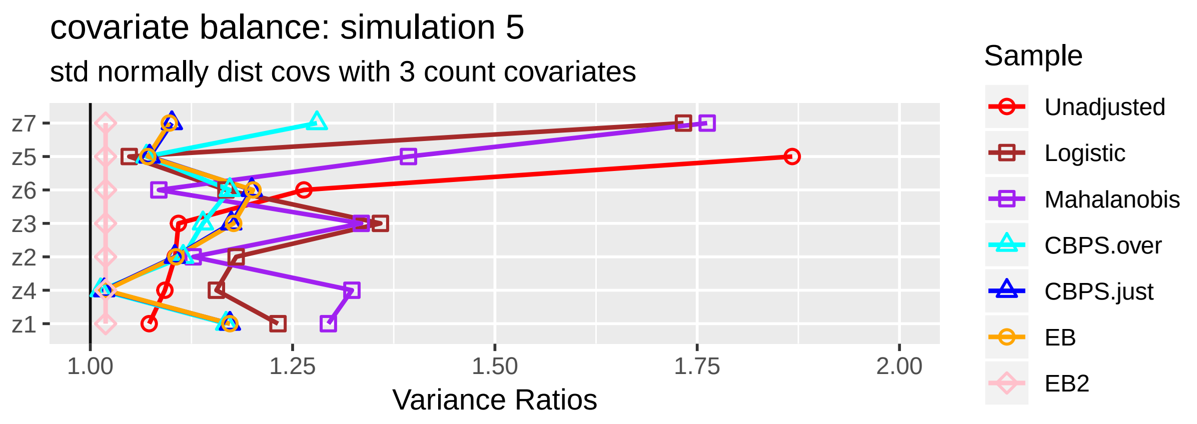


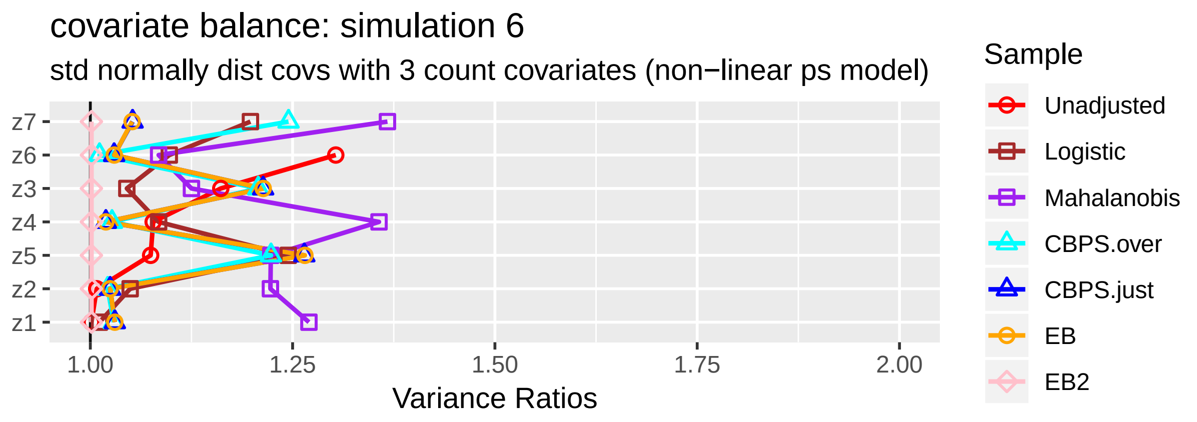
```
grid.arrange(grobs=love.plots.var[3:4], ncol=1, nrow = 2)
```



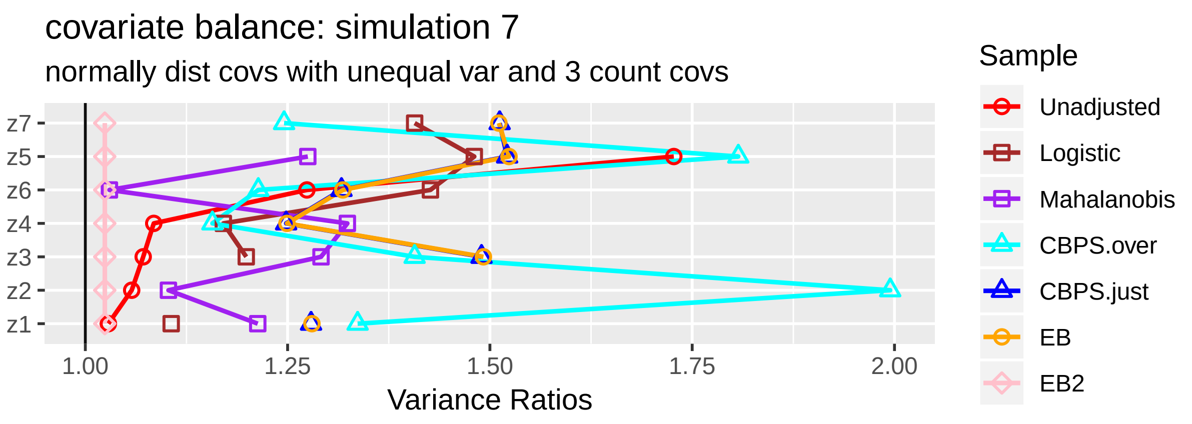


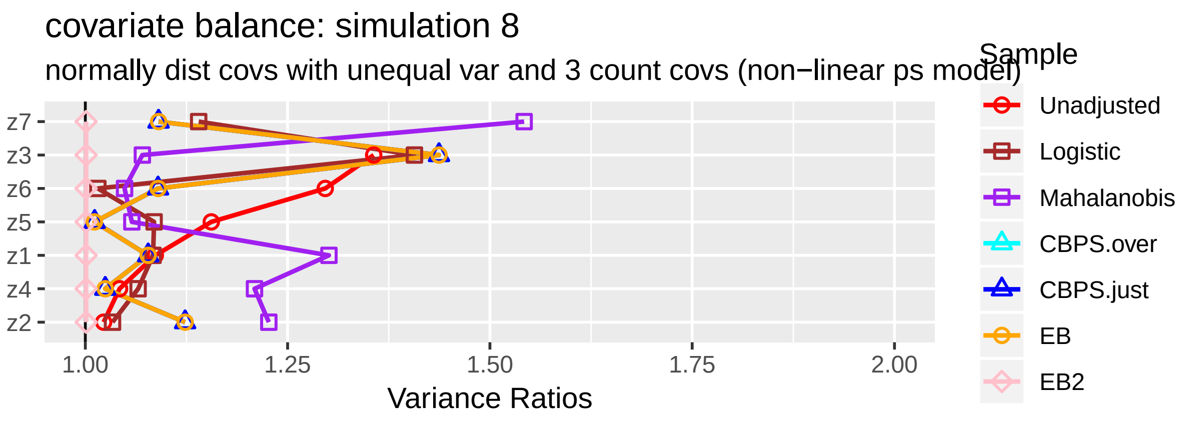
```r
grid.arrange(grobs=love.plots.var[5:6], ncol=1, nrow = 2)
```





```r
grid.arrange(grobs=love.plots.var[7:8], ncol=1, nrow = 2)
```





Analysis of Variance Ratios:

- The Entropy Balancing model where I have set both first and second moment conditions (EB 2) is the only model that consistently achieves variance ratios = 1. The other models' ability to obtain similar variances of matched samples across treatment groups is rather sporadic.
- Simulations 3/4 and simulations 7/8 are not discernibly different from simulations 1/2 and simulations 5/6. This suggests that unequal variances in the pre-treatment covariates does not impact variance ratios of matched samples.

Sampling Distributions of Estimated Treatment Effect

The block of code below was run on HPC clusters to save time, hence eval=FALSE. I load results of models below for analysis.

```r
for(k in 1:8){

  # For Debugging
  print(k)

  # Initialize Matrices to Store Results for Mean Difference and Linear Regression
  md_results  = matrix(nrow=n_sims,
                       ncol=length(model_names),
                       dimnames = list(1:n_sims,model_names))

  lin_results = matrix(nrow=n_sims,
                       ncol=length(model_names),
                       dimnames = list(1:n_sims,model_names))
```

```r
df.temp = df[[k]]

for(i in 1:n_sims){

  formula = as.formula(paste('treat ~ ',paste(confounders[[k]],collapse=' + '),sep=''))

  # Generate treatment vector
  df.temp$treat = as.integer(runif(n) <= df.temp$e_Z)
  df.temp$y = df.temp$y_0*(1-df.temp$treat) + df.temp$y_1*df.temp$treat

  # Logistic Regression, 1-1 Matching
  mod_match = matchit(formula,
                      data = df.temp,
                      method = 'nearest',
                      distance = 'logit',
                      replace = TRUE)
  df.temp$wt = mod_match$weights

  # Mahalanobis Distance Matching
  mod_mahalo = matchit(formula,
                       data = df.temp,
                       method = "nearest",
                       distance = "mahalanobis",
                       replace = TRUE)
  df.temp$wt_mh = mod_mahalo$weights

  # CBPS Weights (OVER)
  cbps.over = weightit(formula,
                       data = df.temp,
                       method = "cbps",
                       estimand = "ATT",
                       over=TRUE)
  df.temp$wt.cbps_over = cbps.over$weights

  # CBPS Weights (JUST)
  cbps.just = weightit(formula,
                       data = df.temp,
                       method = "cbps",
                       estimand = "ATT",
                       over=FALSE)
  df.temp$wt.cbps_just = cbps.just$weights

  # Entropy Balancing Weights 1
  eb.out = ebalance(Treatment = df.temp$treat,
                    X = df.temp[,confounders[[k]]])
  df.temp$wt.eb = 1
  df.temp[df.temp$treat == 0,'wt.eb'] = eb.out$w

  # Entropy Balancing Weights 2
  eb.out2 = ebalance(Treatment = df.temp$treat,
                     X = cbind(df.temp[,confounders[[k]]],
                               df.temp[,confounders[[k]]]^2))
  df.temp$wt.eb2 = 1
```

```
    df.temp[df.temp$treat == 0,'wt.eb2'] = eb.out2$w

    # Weighted Mean Differences
    ##############################

    trt=df.temp$treat==1
    ctrl=df.temp$treat==0

    df.trt = df.temp[trt,]
    df.ctrl = df.temp[ctrl,]

    for(j in 1:length(wts)){
      md_results[i,j] = weighted.mean(df.trt$y, df.trt[,wts[j]]) -
        weighted.mean(df.ctrl$y, df.ctrl[,wts[j]])
    }

    # Linear Regression
    ##############################

    for(j in 1:length(wts)){
      lm.mod = lm(y ~ .,
                  data = df.temp[,c('y','treat',confounders[[k]])],
                  weights = df.temp[,wts[j]])
      lin_results[i,j] = summary(lm.mod)$coefficients['treat','Estimate']
    }


  }

  save(lin_results, md_results, SATE,
       file = paste('sim',k,'_n',n,'_nsims',n_sims,'_nlo',nlo,'.RDATA',sep=''))
}
```

Analysis of Simulations:

```
lin_mse = lin_bias = matrix(nrow=8,
                            ncol=length(model_names),
                            dimnames = list(paste('sim',1:8),model_names))

md_mse = md_bias = matrix(nrow=8,
                            ncol=length(model_names),
                            dimnames = list(paste('sim',1:8),model_names))


for(k in 1:8){
  load(paste('sim_results/sim',k,'_n2000_nsims200_nloFALSE.RDATA',sep=''))
  lin_bias[k,] = (SATE[[k]] - apply(lin_results,2,mean)) / SATE[[k]]
  lin_mse[k,] = apply((SATE[[k]]-lin_results)^2,2,mean)
  md_bias[k,] = (SATE[[k]] - apply(md_results,2,mean)) / SATE[[k]]
  md_mse[k,] = apply((SATE[[k]]-md_results)^2,2,mean)
}

kable(round(lin_bias,4),
      caption='Linear Regression: Bias')
```

Table 1: Linear Regression: Bias

	Baseline Logistic	Baseline Mahalanobis	CBPS - Over	CBPS - Just	EB - 1	EB - 2
sim 1	0.0064	0.0110	0.0076	0.0075	0.0075	0.0085
sim 2	-0.0004	0.0049	0.0012	0.0012	0.0012	0.0018
sim 3	0.0002	-0.0014	-0.0001	0.0001	0.0001	0.0008
sim 4	-0.0006	-0.0077	-0.0031	-0.0031	-0.0031	-0.0091
sim 5	-0.0019	0.0032	-0.0032	-0.0045	-0.0047	-0.0059
sim 6	-0.0094	-0.0073	-0.0092	-0.0092	-0.0092	-0.0109
sim 7	-0.0054	0.0071	-0.0026	-0.0050	-0.0052	-0.0053
sim 8	0.0086	0.0020	0.0091	0.0095	0.0095	0.0056

```
kable(sqrt(round(lin_mse,4)),
      caption='Linear Regression: RMSE')
```

Table 2: Linear Regression: RMSE

	Baseline Logistic	Baseline Mahalanobis	CBPS - Over	CBPS - Just	EB - 1	EB - 2
sim 1	0.1224745	0.0888819	0.0964365	0.1004988	0.1004988	0.1014889
sim 2	0.0774597	0.0761577	0.0500000	0.0500000	0.0500000	0.0556776
sim 3	0.1170470	0.0734847	0.0830662	0.0866025	0.0866025	0.0894427
sim 4	0.0812404	0.0800000	0.0556776	0.0556776	0.0556776	0.0692820
sim 5	0.1565248	0.0663325	0.1183216	0.1371131	0.1392839	0.1417745
sim 6	0.0848528	0.0728011	0.0632456	0.0632456	0.0632456	0.0670820
sim 7	0.1705872	0.0728011	0.1200000	0.1421267	0.1435270	0.1714643
sim 8	0.0806226	0.0591608	0.0591608	0.0608276	0.0608276	0.0624500

```
# kable(round(md_bias,4),
#       caption='Weighted Mean: Bias')
# kable(sqrt(round(md_mse,4)),
#       caption='Weighted Mean: RMSE')
```

```
lin_mse = lin_bias = matrix(nrow=8,
                            ncol=length(model_names),
                            dimnames = list(paste('sim',1:8),model_names))

md_mse = md_bias = matrix(nrow=8,
                            ncol=length(model_names),
                            dimnames = list(paste('sim',1:8),model_names))


for(k in 1:8){
  load(paste('sim_results/sim',k,'_n2000_nsims200_nloTRUE.RDATA',sep=''))
  lin_bias[k,] = (SATE[[k]] - apply(lin_results,2,mean)) / SATE[[k]]
  lin_mse[k,] = apply((SATE[[k]]-lin_results)^2,2,mean)
  md_bias[k,] = (SATE[[k]] - apply(md_results,2,mean)) / SATE[[k]]
  md_mse[k,] = apply((SATE[[k]]-md_results)^2,2,mean)
}
```

```r
kable(round(lin_bias,4),
      caption='Non-Linear PS Model -- Linear Regression: Bias')
```

Table 3: Non-Linear PS Model – Linear Regression: Bias

	Baseline Logistic	Baseline Mahalanobis	CBPS - Over	CBPS - Just	EB - 1	EB - 2
sim 1	0.0064	0.0110	0.0076	0.0075	0.0075	0.0085
sim 2	-0.0004	0.0049	0.0012	0.0012	0.0012	0.0018
sim 3	0.0002	-0.0014	-0.0001	0.0001	0.0001	0.0008
sim 4	-0.0006	-0.0077	-0.0031	-0.0031	-0.0031	-0.0091
sim 5	-0.0019	0.0032	-0.0032	-0.0045	-0.0047	-0.0059
sim 6	-0.0094	-0.0073	-0.0092	-0.0092	-0.0092	-0.0109
sim 7	0.0064	0.0047	0.0073	0.0092	0.0096	0.0085
sim 8	0.0119	0.0142	0.0107	0.0111	0.0111	0.0122

```r
kable(sqrt(round(lin_mse,4)),
      caption='Non-Linear PS Model -- Linear Regression: MSE')
```

Table 4: Non-Linear PS Model – Linear Regression: MSE

	Baseline Logistic	Baseline Mahalanobis	CBPS - Over	CBPS - Just	EB - 1	EB - 2
sim 1	0.1224745	0.0888819	0.0964365	0.1004988	0.1004988	0.1014889
sim 2	0.0774597	0.0761577	0.0500000	0.0500000	0.0500000	0.0556776
sim 3	0.1170470	0.0734847	0.0830662	0.0866025	0.0866025	0.0894427
sim 4	0.0812404	0.0800000	0.0556776	0.0556776	0.0556776	0.0692820
sim 5	0.1565248	0.0663325	0.1183216	0.1371131	0.1392839	0.1417745
sim 6	0.0848528	0.0728011	0.0632456	0.0632456	0.0632456	0.0670820
sim 7	0.2002498	0.0741620	0.1371131	0.1679286	0.1711724	0.1852026
sim 8	0.0787401	0.0800000	0.0624500	0.0648074	0.0648074	0.0734847

```r
# kable(round(md_bias,4),
#       caption='Weighted Mean: Bias')
# kable(sqrt(round(md_mse,4)),
#       caption='Weighted Mean: MSE')
```

Application to IHDP Data:

In this section, I apply the matching and reweighting methods to evaluate the Infant Health and Development Program (IHDP).

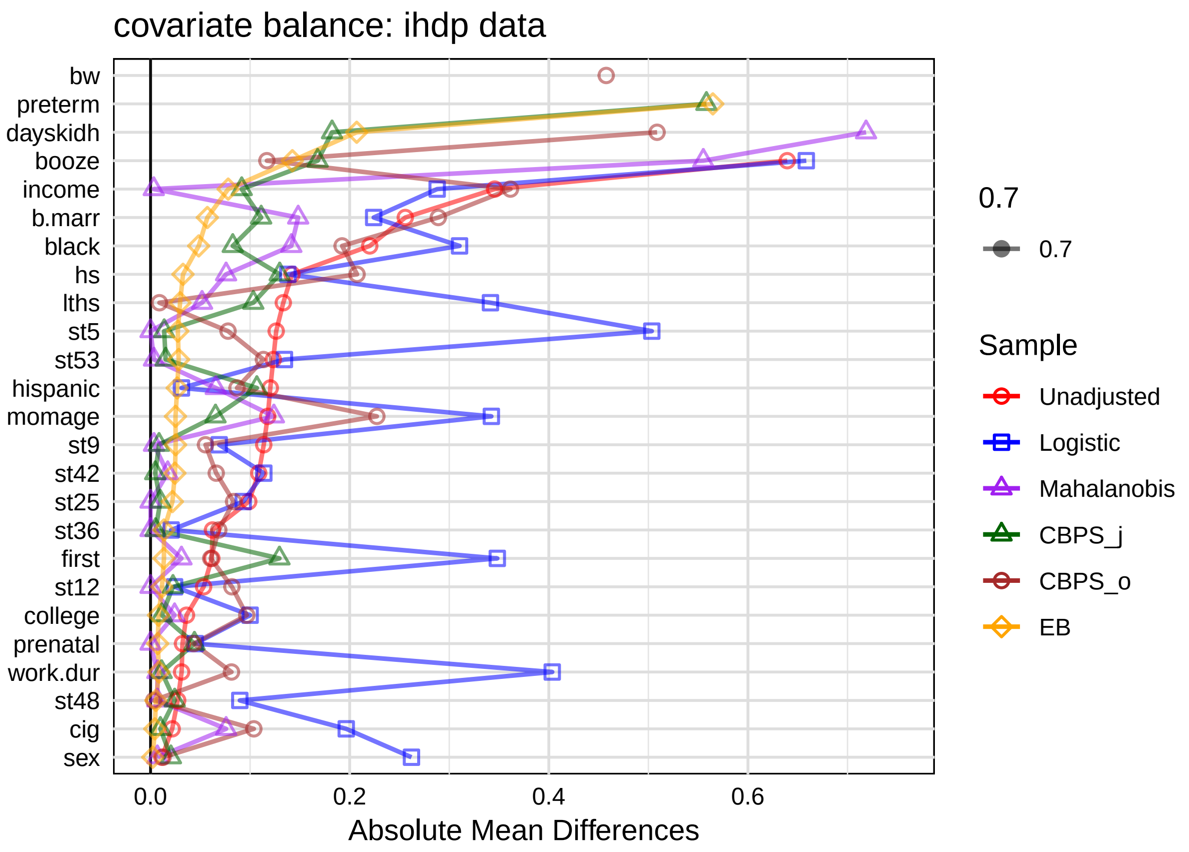
The dataset for this section is from homework 4. The dataset contains personal details about approximately 4500 children born in the 1980s and their mothers. Of these 4500, 290 received the treatment: IHDP. IHDP provides special services for the children who receive treatment, such as high-quality child care in the second and third years of life. Treatment was not randomly assigned, but rather, to children who were born (1) prematurely, (2) with low birth weight (1500-2500 grams), and (3) lived in one of the eight cities where the intervention took place. The outcome of interest is a test score conducted at age 3 (similar to an IQ measure).

```r
load('data/hw4.Rdata')
df.ihdp = hw4
```

```r
col_names = colnames(df.ihdp)
cols_to_keep = col_names[!col_names %in% c('treat','ppvtr.36','white',
                                           'ltcoll','bwg','momed','st99')]
ihdp.formula = formula(paste('treat ~ ',paste(cols_to_keep,collapse = ' + ')))

# Logistic Regression, 1-1 Matching
ihdp.log = matchit(ihdp.formula,
                   data = df.ihdp,
                   method = 'nearest',
                   distance = 'logit',
                   replace = TRUE)
df.ihdp$wt = ihdp.log$weights

# Mahalanobis Distance Matching
ihdp.mh = matchit(ihdp.formula,
                  data = df.ihdp,
                  method = "nearest",
                  distance = "mahalanobis",
                  replace = TRUE)
df.ihdp$wt_mh = ihdp.mh$weights

# CBPS Weights (OVER)
ihdb.cbps_o = weightit(ihdp.formula,
                       data = df.ihdp,
                       method = "cbps",
                       estimand = "ATT",
                       over=TRUE)
df.ihdp$wt.cbps_over = ihdb.cbps_o$weights

# CBPS Weights (JUST)
ihdb.cbps_j = weightit(ihdp.formula,
                       data = df.ihdp,
                       method = "cbps",
                       estimand = "ATT",
                       over=FALSE)
df.ihdp$wt.cbps_just = ihdb.cbps_j$weights

# Entropy Balancing Weights
ihdp.eb = ebalance(Treatment = df.ihdp$treat,
                   X = df.ihdp[,cols_to_keep])
df.ihdp$wt.eb = 1
df.ihdp[df.ihdp$treat == 0,'wt.eb'] = ihdp.eb$w
```

```r
love.plot(bal.tab(covs = df.ihdp[,cols_to_keep],
                  treat = df.ihdp$treat,
                  data = df.ihdp[,c('treat',cols_to_keep)],
                  weights = data.frame(Logistic = get.w(ihdp.log),
                                       Mahalanobis = get.w(ihdp.mh),
                                       CBPS_j = get.w(ihdb.cbps_j),
                                       CBPS_o = get.w(ihdb.cbps_o),
                                       EB = df.ihdp$wt.eb)),
          var.order = "unadjusted",
          abs = TRUE,
          line = TRUE,
          colors = c("red", "blue", "purple",
                     "darkgreen","brown", "orange"),
          shapes = c("circle open", "square open",
                     "triangle open","triangle open",
                     "circle open", "diamond open")) +
  xlim(0,0.75) +
  labs(title=paste('covariate balance: ihdp data')) +
  aes(alpha=0.7)
```

```
## Assuming "weighting". If not, specify with an argument to method.

## Note: estimand and s.d.denom not specified; assuming ATT and treated.
```



Simular to the results from the previous simulations, the entropy balancing model achieves the best balance on the covariates.

```
# Linear Regression
#################################
wts2 = c('wt','wt_mh','wt.cbps_over','wt.cbps_just','wt.eb')
ihdp_lin_b = ihdp_lin_se = c()
for(j in 1:length(wts2)){
  lm.mod = lm(ppvtr.36 ~ .,
              data = df.ihdp[,c('ppvtr.36','treat',cols_to_keep)],
              weights = df.ihdp[,wts2[j]])
  ihdp_lin_b[j] = summary(lm.mod)$coefficients['treat','Estimate']
  ihdp_lin_se[j] = summary(lm.mod)$coefficients['treat','Std. Error']
}
```

```
ihdp_results = rbind(ihdp_lin_b,ihdp_lin_se)
colnames(ihdp_results) = model_names[1:5]
kable(ihdp_results)
```

	Baseline Logistic	Baseline Mahalanobis	CBPS - Over	CBPS - Just	EB - 1
ihdp_lin_b	10.400352	5.994024	7.4173014	15.5415627	9.5612430
ihdp_lin_se	3.445996	2.649219	0.7729403	0.9819042	0.6288457

Conclusion and Limitations: