Multiple Causal Discoveries with Sensitivity Analyses using NHANES Data — A Report and User Manual

Statistics 921, Dylan Small, Final Paper
Adam Kapelner and Joshua Magarick
Department of Statistics, The Wharton School of the University of Pennsylvania

December 21, 2011

Abstract

We present an open-source research tool, `HANESinator`, that investigates causal relationships among measurements in the National Health and Nutrition Examination Survey (NHANES). The platform allows specification of multiple measurements that are considered the "response", multiple measurements that are considered the "binary treatment", as well as a multitude of measurements that are considered "control" variables. After specification, the program outputs the effect size(s), Bonferroni-corrected p-value(s) for their significance, as well as sensitivity analyses, which are corrected for multiple comparisons. We introduce the data in section 1, explain the methods in section 2, demonstrate the system in section 3, conclude in section 4, and present the code and an end-user guide in appendix A.

1 The NHANES Data

1.1 Background

In 1956, the United States was concerned that they had no way of obtaining general health readings on its population. Surveys up to that time were provincial either in a local geographical region, or if they were widespread, they were limited to specific questions. Congress passed the National Health Survey Act which authorized an annual survey. In 1971 this effort was expanded to create the National Health and Nutrition Examination Survey (NHANES), "a program of studies designed to assess the health and nutritional status of adults and children in the United States", which was run once every five years. In 1999, NHANES became annual; every year about 10,000 Americans are included. The sampling is stratified to oversample the elderly and minorities.

NHANES is interested in demographic, socioeconomic, dietary, and health-related measurements. The first three categories are queried via questionnaires. The last category in recent years involves sophisticated in-depth physical examination (e.g. audiometry, body-fat caliper measurements, X-ray absorption of the femur, a retinal imaging battery, dental information tooth-by-tooth, etc) as well as comprehensive blood and urine tests in a laboratory

(e.g. lead and arsenic levels, pesticides in urine, sexually transmitted disease testing, thyroid profile, glucose and insulin, etc).

NHANES data mining has been used in ways that affect our everyday lives. The "growth charts" graphics for children's height we see in the doctor's offices, the national mandate to add Folate and Iron to cereal, the nationwide efforts to reduce cholesterol have all been influenced by trends seen in the NHANES data.

We hope to provide a data mining platform that allows researchers to uncover not only associations but *causal* relationships.

1.2 The data used in this platform

The 2007 dataset was arbitrarily chosen for use in this platform. We include about 2400 measurements per individual which is all data save the detailed dietary questionnaires where every food type and supplement was queried.[1]

Note that there is a large amount of missing data; many of the covariates are spotty and some are all blank with the exception of a few individuals.

2 Methods for one Causal Investigation

In short, we match individuals who received the treatment to individuals who received the control using estimated propensity scores and we calculate effect size, significance, and conduct a sensitivity analysis using this matched set.

2.1 Data Selection

The first step is to select *one* response variable, R_i. The platform developed employs theory that assumes the response to be continuous interval data.[2] The next step will be to choose *one* treatment variable, Z_i which must be binary[3] (we will use the convention of $Z_i = 1$ indicates the individual was "treated" and $Z_i = 0$ indicates no treatment). The final step is to select a host of control variables $X_{.i} \triangleq X_{i1}, X_{i2}, \ldots, X_{ip}$ which can be any type of data.[4] We refer to the full data matrix as $\boldsymbol{X} = [Z, X_1, \ldots, X_{p^+}]$ where p^+ indicates a number possibly larger that p due to the hydrating of dummy variables.

Our platform does not handle missing data. Therefore, if there are any missing observations in any of the treatment, response, or controls, the entire record will be *discarded* during the analysis. Let n_C denote the number of control records in the full design matrix where $Z_i = 0$ after cleaning out the missing measurements and let n_T denote the analagous number of treatment records.

[1] Creating measurements from the dietary portion of the data would be a tedious process consisting of cross-linking the variable codes to their names, and then coding new covariates of interest by hand.

[2] It will also work if the response is binary or ordinal categorical, although you will have to think carefully about how to interpret the effect sizes, significance, and sensitivity metrics.

[3] We allow the user to specify a function which will code a non-binary variable into 0's and 1's.

[4] If a control variable is categorical nominal, dummy variables will be automatically generated. Categorical ordinal variables can be kept as-is which has the added benefit of reduction of computation time, or chosen to be coded as dummy variables as well.

Note that this epistemologically limts what our final conclusions can be. If indeed NHANES is a true sample of the American population as claimed, then using our platform[5] we are allowed to infer causal results for the American population. However, since we discarded any individual with missing data, we are now biasing our sampling, possibly unfairly so. Therefore, the *only inference we can make* with this platform is inference for the "type" of person that has a "full record" in the NHANES data for these specific treatment, response, and control variables specified.

2.2 Matching

In a randomized study, we would expect balance among the covariates. Since the NHANES data is observational by definition, we will expect selection bias because people "opt in" or out of a treatment for many other unmeasured reasons. To mitigate such bias, we attempt to match individuals from the control group to individuals from the treatment group as best as possible. Let $\lambda(X_{i\cdot})$ be the likelihood the ith individual receives the treatment based on his or her control measurements; this likelihood is known as the "propensity score." Since we know who received treatment, Z_i, we can match two records which share the same propensity score:

$$(i,j) \text{ as a matched pair means that: } \quad Z_i = 1, \quad Z_j = 0 \quad \text{and} \quad \lambda(X_{i\cdot}) = \lambda(X_{j\cdot})$$

It can be shown that individuals that share the same propensity score share the same distribution of the control variables. Thus, on average, treated individuals and control individuals matched in pairs exhibit the same property as treated individuals and control individuals in a randomized experimental design (assuming that we haven't left out any useful covariates, a point we will discuss in the next section).

We model the estimated propensity scores, $\hat{\lambda}(X_{\cdot j})$'s, as follows:

$$\lambda(X_{\cdot j}) = \mathbb{P}(Z_i = 1 | X_{i\cdot}) \approx \hat{\lambda}(X_{\cdot i}) = \frac{\exp(\beta_0 + \beta_1 X_{i1} + \ldots + \beta_{p^+} X_{ip^+})}{1 + \exp(\beta_0 + \beta_1 X_{i1} + \ldots + \beta_{p^+} X_{ip^+})}$$

where $\beta_0, \beta_1, \ldots, \beta_{p^+}$ are estimated using the maximum likelihood machinery of a vanilla logistic regression.

After estimates of the scores are obtained, we create $n^* = \min\{n_C, n_T\}$ matches by solving the minimum cost flow problem where estimated propensity score differences are considered the "cost."[6]

After matching, we can then compute the paired wilcoxon rank sum test to test the significance level of the median difference between response in the matched treated and control group. We also compute a p_{val} for the 2-sample paired t-test for difference in means, and report a p_{val} for the treatment effect in an ordinary least squares regression.

[5] after correcting for the NHANES stratified sampling (the over and under sampling of the elderly and minorities)

[6] The library in `R` called `optmatch` is a convenient way to employ this type of pair matching using the latest algorithms.

2.3 Sensitivity Analysis

What if there is some unmeasured covariate that we did not account for? It is plausible that such a measurement can induce "hidden" selection bias. If we were able to measure it, then we could find ourselves in the paradoxical situation of our effect becoming insignificant, or worse, the effect direction could flip. If we find a particularly strong association, only a particularly "strong" hidden covariate can be powerful enough to inflict such a paradox. By "powerful enough" we mean power to move an individual into or out of the treatment group with a high probability.

We formalize this notion as follows. Assume two individuals are equal on all observed covariates *i.e.* $X_{j\cdot} = X_{k\cdot}$ and note that the odds ratio of the jth unit belonging to the treatment would be $\frac{\lambda(X_{j\cdot})}{1-\lambda(X_{j\cdot})}$ and the odds ratio of the kth unit belonging to the treatment would be the analagous fraction indexed by k. If their propensities to belong to the treated group are not equal *i.e.* $\lambda(X_{j\cdot}) \neq \lambda(X_{k\cdot})$, we can be sure there is some unobserved covariate, call it X_H, that is responsible for inducing this hidden bias. This covariate would have to be strong enough to move the ratio of the two odds ratios. We define the sensitivity metric Γ below:

$$\frac{\dfrac{\lambda(X_{j\cdot})}{1-\lambda(X_{j\cdot})}}{\dfrac{\lambda(X_{k\cdot})}{1-\lambda(X_{k\cdot})}} \leq \Gamma \quad \text{and} \quad \frac{\dfrac{\lambda(X_{j\cdot})}{1-\lambda(X_{k\cdot})}}{\dfrac{\lambda(X_{j\cdot})}{1-\lambda(X_{j\cdot})}} \leq \Gamma$$

This metric measures to what degree X_H can manipulate $\lambda(X_{j\cdot})$ to be different from $\lambda(X_{k\cdot})$. At $\Gamma = 1$, the propensity scores are forced to be equal; thereby the study is free of hidden bias. A $\Gamma = 2$ would allow X_H to move one of the two equal records into the treatment at double the odds ratio.

In practice, we find the largest Γ for which there is a nonzero effect by a naïve grid search. Assuming an additive effect, we find, for every Γ, a range of effect sizes that we would not reject at a $(1-\alpha)$ level. These are computed using a normal approximation of the bounding random variables to the rank sum statistic given in lecture notes. We proceed, computing confidence intervals until one contains zero.

2.4 Adjusting results for multiple comparisons

To adjust for p_{val}'s we simply use a Bonferroni correction. This approach is overly-conservative since we should be able to exploit some symmetry among the comparisons due to the shared control variables. Given the large number of variables present in NHANES, a better approach should be found for future work.

To create adjusted sensitivity intervals, we again employ the naïve approach and, for a desired level of significance α compute $1-\alpha$ sensitivity intervals for each Γ in our grid. At the present, we are not sure if this approach is valid but we did not want to underestimate our sensitivity to hidden bias and it was done due to the obvious analogies to confidence intervals.

3 Examples of Analyses

We were originally inspired to look into the NHANES data to find measurements that associate with sleep. We searched Google Scholar for "sleep" AND "NHANES" and of the first 100 results, we found 13 relevant papers where 10 use sleep as covariate to predict something else. None of the studies located used methodology found in section 2. Now, we can prove certain lifestyles *cause* a good (or bad) night's sleep.

In this section, we demonstrate the platform using the response variable of average number of hours of sleep on the treatments "ever been told you had a thyroid problem?" and "do you do vigorous work activity?" We include about 100 control measurements spanning the gamut of the NHANES pantheon.

```
Testing 2 comparison(s)

comparison #1:
treatment: MCQ160M (Ever told you had a thyroid problem)
response: SLD010H (How much sleep do you get (hours)?)
num controls obs = 2591, num treatment obs = 268 (total n = 2859)

effect size is
  -0.0149
Bonf-adj wilcox test pval / 2-samp t-test / OLS t-test pval =
  1.7814 / 1.8205 / 1.7795
Sensitivity Analysis, 95% Gamma =
  1

comparison #2:
treatment: PAQ605 (Vigorous work activity)
response: SLD010H (How much sleep do you get (hours)?)
num controls obs = 2257, num treatment obs = 602 (total n = 2859)

effect size is
  -0.2608
Bonf-adj wilcox test pval / 2-samp t-test / OLS t-test pval =
  0.0026 / 0.0033 / 0.0016
Sensitivity Analysis, 95% Gamma =
  1.11
```

In the first analysis, we conclude that ever having a thyroid problem has no causal effect on average sleep duration. In the second analysis, we conclude that vigorous work activity reduced average sleep duration by about 15 minutes, but the result is not robust to hidden bias as evidenced by $\Gamma = 1.16$; thereby we cannot claim the relationship is causative.

4 Our System and Concluding Remarks

We developed `HANESinator`, a platform that can investigate causal effects in the 2007 NHANES data by specifying any possible binary treatment, and any possible continuous response. The output consists of effect size, significance level, and a sensitivity analysis to test for a covariate that may introduce hidden bias. This is a veritably powerful research tool which is freely availably via open-source and fully documented.

The platform can be improved. Specifically, we would like to work on implementing the following features. Some of these may require theoretical advances:

1. Use propensity score caliper full matching via rank-based Mahalanobis distance.
2. Output balance diagnostics after matching using the cross-match test.
3. Using records with missing data which will allow inference of the causal results to the general American population.
4. Bonferroni is overly conservative for both the p_{val} and Γ calculations. We would like to develop theory to compute less realistic multiple comparisons.
5. Generalize to use different years of NHANES data or better yet, combine years of NHANES data together in a sensible way.

Acknowledgments

We would like to extend a big thanks to Professor Dylan Small for not only teaching the course in observational studies, but for inspiring us to do this project, and his patient mentoring. We would also like to thank our classmates for helpful discussions and brainstorming.

A `R` Code and User Guide to Running Analyses

This code has been open sourced under GPL and is publically available on github.[7] This section will explain the functionality of each script..

A.1 Code for the end-user

The file shown below is the "main portal." Here you set your working directory, the multiple response variables, and the multiple treatments of which you would like to test a causal relationship. You specify variables by NHANES filename and then by the unique NHANES variable keycode. You can alter the response variable by choosing to exclude a subset of the values. You can also choose a non-binary treatment and specify a function that will map the values to $\{0, 1\}$.

[7] You can access it via `https://github.com/kapelner/HANESinator`

As detailed in the methods section, we remove any NHANES record which has missing data in *any* of the treatment(s), response, or controls. Therefore we strongly encourage the user to select treatments, response, and controls that retain a large sample size.

By sourcing this file alone, the entire analysis will be run and results will be printed to the screen. The code also generates a variable, `causal_results` which caches the results.

../R/00_set_resp_trts_and_run_NHANES_analysis.R

```
#HANESinator -- a platform to find causal results in America's NHANES data
#Copyright (C) 2011  Adam Kapelner & Joshua Magarick
#
#This program is free software: you can redistribute it and/or modify
#it under the terms of the GNU General Public License as published by
#the Free Software Foundation, either version 3 of the License, or
#(at your option) any later version.
#
#This program is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#GNU General Public License for more details.
#
#You should have received a copy of the GNU General Public License
#along with this program.  If not, see <http://www.gnu.org/licenses/>.



#your working directory (PLEASE LEAVE COMMENTED OUT)
#setwd("C:\\Users\\kapelner\\desktop\\Dropbox\\Stat921_KM_presentation")
#setwd("C:\\Users\\magarick\\Dropbox\\Stat921_KM_presentation")

#First pick a set of response variables.
ResponseVarFilenames = c(
  "Questionnaire__sleep_disorders.xpt")
# "Questionnaire__sleep_disorders.xpt")
ResponseVarCodes = c(
  "SLD010H")
# "SLD020M")

#these functions allow you to alter the response by rejecting a subset of the
    data
reject_functions = list()
reject_functions[["SLD010H"]] = function(y){
  y < 12
}

#Now pick a set of treatment variables you would like to investigate
#causal relationships with the responses.
TreatmentVarFilenames = c(
  #"Examination__body_measures.xpt",
```

```
  "Questionnaire__medical_conditions.xpt",
  "Questionnaire__physical_activity.xpt")
# "Questionnaire__smoking_household_smokers.xpt") #"Questionnaire__air_
    quality.xpt"
TreatmentVarCodes = c(
  #"BMXBMI",
  "MCQ160M",
  "PAQ605")
# "SMD410") #"PAQ685"

#these functions you to take a continuous data vector and
#transform it to a binary 0/1 treatment variable
trt_coding_functions = list()
trt_coding_functions[["BMXBMI"]] = function(x_z){
  as.numeric(x_z >= 35)
}
trt_coding_functions[["MCQ160M"]] = function(x_z){
  as.numeric(x_z == 1)
}
trt_coding_functions[["PAQ605"]] = function(x_z){
  as.numeric(x_z == 1)
}

#the number of comparisons is the number of responses x number of treatments
num_comparisons = length(TreatmentVarCodes) * length(ResponseVarCodes
    )
#load up the data
source("R\\01_load_NHANES_data.R")
#initialize the object that will allow the results to persist after the script
    is written
causal_results = as.data.frame(matrix(NA, nrow = num_comparisons,
    ncol = 8))
colnames(causal_results) = c("response", "treatment", "num_controls"
    , "effect size (mean)", "Bonf wilcox pval", "Bonf 2-samp t-test
    pval", "Bonf OLS t-test pval", "gamma")
cat(paste("Testing ", num_comparisons, " comparison(s)\n", sep = "")
    )
#begin the loops to do the comparisons, first by response
for (i_resp in 1 : length(ResponseVarCodes)){
  ResponseVarFilename = ResponseVarFilenames[i_resp]
  ResponseVarCode = ResponseVarCodes[i_resp]
  #then by treatment on this response
  for (i_trt in 1 : length(TreatmentVarCodes)){
    comparison_num = (i_resp - 1) * length(TreatmentVarCodes) + i_trt
    #pull out the treatment variable we are examining at the moment
    TreatmentVarFilename = TreatmentVarFilenames[i_trt]
    TreatmentVarCode = TreatmentVarCodes[i_trt]
```

```
    #run the analysis for one comparison
    source("R\\02_set_controls_and_params.R")
    source("R\\03_create_info_about_covariates.R")
    source("R\\04_create_design_matrix.R")
    source("R\\05_prop_score_and_matching.R")
    source("R\\06_gamma_sensitivity.R")
    #save the results in an object
    causal_results[comparison_num, ] = c(ResponseVarCode,
        TreatmentVarCode,
        num_control_vars,
        round(avg_diff, 3), #round(mwwtest$estimate, 3), #the sample
            median effect (not as important)
        round(mwwtest_pval, 3),
        round(ttest_pval, 3),
        round(lin_regr_pval, 3),
        round(Gamma_min, 3))
  }
}
```

In the following file, you define the covariates you wish to control for in your analysis. You specify by NHANES filename and then code. You also have to manually tell the program which of the covariates should be treated as categorical (so the program can create the necessary dummy variables). Further in the file, you will manually specify which values of certain variables (treatments, responses, and controls) should be ignored. It is normal to turn control variables on/off for certain tests. This can be done effortlessly by via commenting.

You can also specify the verbosity of the output. If turn on `PRINT_CONTROLS`, the program will print the name of each control and exactly the effective sample size after the individuals with missing data are discarded. If turn on `PRINT_COV_CHARACTERISTICS`, the program will print before *and* after matching the table of covariate means, standard deviations for both the treatment and control, standardized differences, with testing and p_{val}'s for the null of no difference.

../R/02_set_controls_and_params.R

```
PRINT_CONTROLS = FALSE
PRINT_COV_CHARACTERISTICS = FALSE
VERBOSE_GAMMA_COMPUTATIONS = FALSE

#we now want a list of variables that we potentially want to control and match
    in our study
control_variables = list()

control_variables[["Demographics__all.xpt"]] =
    c(
      "RIAGENDR",
      "RIDAGEYR",
      "DMDMARTL",
      "RIDRETH1",
```

```
      "DMDFMSIZ",
      "INDHHIN2")
#control_variables[["Examination__body_measures.xpt"]] =
#    c(
#      "BMXWT",
#      "BMXHT",
#      "BMXARML",
#      "BMXWAIST",
#      "BMXTRI")
control_variables[["Examination__vision.xpt"]] =
    c("VIQ220")
control_variables[["Laboratory__blood_cadmium_and_lead.xpt"]] =
    c("LBXBCD",
      "LBXBPB")
control_variables[["Laboratory__blood_total_mercury_and_blood_
   inorganic_mercury.xpt"]] =
    c("LBXTHG")
control_variables[["Laboratory__complete_blood_count.xpt"]] =
    c("LBXWBCSI",
      "LBXLYPCT",
      "LBXRBCSI",
      "LBXMCVSI",
      "LBXMCHSI",
      "LBXPLTSI",
      "LBXMPSI")
control_variables[["Laboratory__HDL_cholesterol.xpt"]] =
    c("LBDHDD")
control_variables[["Laboratory__rbc_folate_and_serum_folate.xpt"]]
   =
    c("LBDRBF")
control_variables[["Laboratory__serum_cotinine_and_urinary_total_
   NNAL.xpt"]] =
    c("LBXCOT",
      "URXNAL")
control_variables[["Laboratory__standard_biochemistry_profile.xpt"
   ]] =
    c("LBXSAL",
      "LBXSATSI",
      "LBXSAPSI",
      "LBXSBU",
      "LBXSCA",
      "LBXSCH",
      "LBXSC3SI",
      "LBXSGTSI",
      "LBXSGL",
      "LBXSIR",
      "LBXSPH",
```

```
      "LBXSTB",
      "LBXSTP",
      "LBXSTR",
      "LBXSUA",
      "LBXSNASI",
      "LBXSKSI",
      "LBXSCLSI",
      "LBXSOSSI",
      "LBXSGB")
#control_variables[["Laboratory__thyroid_profile.xpt"]] =
#   c("LBXATG",
#     "LBXT3F",
#     "LBXT4F",
#     "LBXTGN",
#     "LBXTSH1",
#     "LBXTPO",
#     "LBXTT3",
#     "LBXTT4")
control_variables[["Laboratory__urinary_iodine.xpt"]] =
    c("URXUIO")
control_variables[["Laboratory__urinary_nitrate_perchlorate_and_
   thiocyanate.xpt"]] =
    c("URXUP8",
      "URXNO3",
      "URXSCN")
#control_variables[["Laboratory__urinary_specific_gravity.xpt"]] =
#   c("SSUSG") ##########for some reason this one don't work??
control_variables[["Questionnaire__air_quality.xpt"]] =
    c("PAQ685")
control_variables[["Questionnaire__audiometry.xpt"]] =
    c("AUQ131") #model this as continuous even though it's orginal
       categorical
control_variables[["Examination__oral_health.xpt"]] =
    c("OHXDECAY",
      "OHXREST",
      "OHXSEAL")
control_variables[["Questionnaire__blood_pressure_and_cholesterol.
   xpt"]] =
    c("BPQ020",
      "BPQ052",
      "BPQ057",
      "BPQ060")
control_variables[["Questionnaire__bowel_health.xpt"]] =
    c("BHD050")
control_variables[["Questionnaire__consumer_behavior.xpt"]] =
    c("CBD010",
      "CBQ020",
```

```
      "CBQ030",
      "CBQ040",
      "CBQ050",
      "CBQ060",
      "CBD070",
      "CBD090",
      "CBD110",
      "CBD120",
      "CBD130",
      "CBQ140",
      "CBD150",
      "CBD160",
      "CBD170")
# model a lot of these as continuous even though they're orginal categorical
control_variables[["Questionnaire__diabetes.xpt"]] =
    c("DIQ010")
control_variables[["Questionnaire__health_insurance.xpt"]] =
    c("HIQ011")
control_variables[["Questionnaire__hospital_utilization_and_access_
   to_care.xpt"]] =
    c("HUQ090")
control_variables[["Questionnaire__income.xpt"]] =
    c("IND235")
control_variables[["Questionnaire__medical_conditions.xpt"]] =
    c("MCQ010", "MCQ160M")
control_variables[["Questionnaire__physical_functioning.xpt"]] =
    c("PFQ090")
control_variables[["Questionnaire__respiratory_health.xpt"]] =
    c("RDQ070")
control_variables[["Questionnaire__smoking_cigarette_use.xpt"]] =
    c("SMQ020")
control_variables[["Questionnaire__smoking_household_smokers.xpt"]]
    =
    c("SMD410")
control_variables[["Questionnaire__vision.xpt"]] =
    c("VIQ031")


################finished looking for covariates up to row 1296 / 2257 in cov_
   master_list
#cov_master_list[1296:2257, ]

control_variables_that_are_categorical =
    c("RIAGENDR",
      "RIDRETH1", "DMDMARTL", "INDHHIN2",
      "OHXDECAY", "OHXREST", "OHXSEAL",
      "VIQ220",
```

```
      "BPQ020", "BPQ052", "BPQ057", "BPQ060",
      "CBQ020",
      "CBQ060",
      "DIQ010",
      "HIQ011",
      "HUQ090",
      "MCQ010",
      "MCQ160M",
      "PFQ090",
      "RDQ070",
      "SMQ020",
      "SMD410",
      "VIQ031",
      "PAQ685")

#all the variable values that need to be removed from the trt, resp, controls,
#for any reason or no reason
illegal_variable_values = list()
#only mark numerical values (all NA's are removed automatically)
illegal_variable_values[["DMDMARTL"]] = c(77, 99)
illegal_variable_values[["OHXDECAY"]] = c(9)
illegal_variable_values[["OHXREST"]] = c(9)
illegal_variable_values[["OHXSEAL"]] = c(9)
illegal_variable_values[["VIQ220"]] = c(9)
illegal_variable_values[["PAQ685"]] = c(3, 7, 9)
illegal_variable_values[["AUQ131"]] = c(7, 9)
illegal_variable_values[["BHD050"]] = c(777, 999)
illegal_variable_values[["BPQ020"]] = c(7, 9)
illegal_variable_values[["BPQ052"]] = c(7, 9)
illegal_variable_values[["BPQ057"]] = c(7, 9)
illegal_variable_values[["BPQ060"]] = c(7, 9)
illegal_variable_values[["CBD010"]] = c(7, 9)
illegal_variable_values[["CBQ020"]] = c(77, 99)
illegal_variable_values[["CBQ030"]] = c(77, 99)
illegal_variable_values[["CBQ040"]] = c(77, 99)
illegal_variable_values[["CBQ050"]] = c(77, 99)
illegal_variable_values[["CBQ060"]] = c(77, 99)
illegal_variable_values[["CBQ070"]] = c(777777, 999999)
illegal_variable_values[["CBD090"]] = c(777777, 999999)
illegal_variable_values[["CBD110"]] = c(777777, 999999)
illegal_variable_values[["CBD120"]] = c(777777, 999999)
illegal_variable_values[["CBD130"]] = c(777777, 999999)
illegal_variable_values[["CBQ140"]] = c(77, 99)
illegal_variable_values[["CBD150"]] = c(77777, 99999)
illegal_variable_values[["CBD160"]] = c(777, 999)
illegal_variable_values[["CBD170"]] = c(77777, 99999)
illegal_variable_values[["DIQ010"]] = c(7, 9)
```

```
illegal_variable_values[["HIQ011"]] = c(7, 9)
illegal_variable_values[["HUQ090"]] = c(7, 9)
illegal_variable_values[["MCQ010"]] = c(7, 9)
illegal_variable_values[["MCQ160M"]] = c(7, 9)
illegal_variable_values[["PFQ090"]] = c(7, 9)
illegal_variable_values[["RDQ070"]] = c(7, 9)
illegal_variable_values[["SMQ020"]] = c(7, 9)
illegal_variable_values[["SMD410"]] = c(7, 9)
illegal_variable_values[["VIQ031"]] = c(7, 9)
illegal_variable_values[["DBQ915"]] = c(7, 9)
illegal_variable_values[["ENQ010"]] = c(7, 9)
illegal_variable_values[["ENQ020"]] = c(7, 9)
illegal_variable_values[["DMDYRSUS"]] = c(77, 99)



#code here to check if trt, resp, controls are unique
if (TreatmentVarCode %in% control_variables[[TreatmentVarFilename]]){
# cat("WARNING: The treatment var is also a covariate. It has been removed
   from the controls.\n")
  index = which(control_variables[[TreatmentVarFilename]] ==
     TreatmentVarCode)
  control_variables[[TreatmentVarFilename]] = control_variables[[
     TreatmentVarFilename]][-index]
}
if (ResponseVarCode %in% control_variables[[ResponseVarFilename]]){
# cat("WARNING: The treatment var is also a covariate. It has been removed
   from the controls.\n")
  index = which(control_variables[[ResponseVarFilename]] ==
     ResponseVarCode)
  control_variables[[ResponseVarFilename]] = control_variables[[
     ResponseVarFilename]][-index]
}
if (TreatmentVarCode %in% control_variables_that_are_categorical){
# cat("WARNING: The treatment var is listed as a control variable that is
   categorical.\n")
  index = which(control_variables_that_are_categorical ==
     TreatmentVarCode)
  control_variables_that_are_categorical = control_variables_that_
     are_categorical[-index]
}
if (ResponseVarCode %in% control_variables_that_are_categorical){
# cat("WARNING: The response var is listed as a control variable that is
   categorical.\n")
  index = which(control_variables_that_are_categorical ==
     ResponseVarCode)
  control_variables_that_are_categorical = control_variables_that_
```

```
    are_categorical[-index]
}
cat("\n")
```

A.2 Code for the developer only

The user's job is now finished and we move on to the computation. We recommend not touching this portion of the codebase unless you wish to alter the methodology found in section 2.

The following file will load the raw data from the SAS files downloaded from the official NHANES website.

../R/01_load_NHANES_data.R

```
graphics.off()

#all referenced libraries
library(foreign)
library(optmatch)

#first of all: load all the names of the data tables in
FileListToUse = "R\\xpt_files_no_dietary.csv"
xpt_file_list = t(read.csv(FileListToUse, header = F))[1, ]


#truncate to something reasonable ONLY IF USING DIETARY DATA
#otherwise, leave this line commented out:
#xpt_file_list = xpt_file_list[1 : 10]

#Load ALL the data in, yes that's all the data in ALL the files
ALL_DATA = list()
for (xpt_file in xpt_file_list){
# print(paste("importing...", xpt_file, "..."))
  ALL_DATA[[xpt_file]] = read.xport(paste("NHanes_Data\\", xpt_file
      , sep = ""))
}


#get list of dimensions for each
dim_table = matrix(NA, ncol = 3, nrow = length(xpt_file_list))
for (i in 1 : length(xpt_file_list)){
  dim_table[i, ] = c(xpt_file_list[i], dim(ALL_DATA[[xpt_file_list
      [i]]]))
}
dim_table = as.data.frame(dim_table)
colnames(dim_table) = c("filename", "n", "p")
dim_table$n= as.numeric(as.character(dim_table$n))
```

```
dim_table$p = as.numeric(as.character(dim_table$p))
```

This code will then select out the rows of the raw data that contain information about the response, treatment and control variables. If the row has any missing data on those variables, it will be ignored.

../R/03_create_info_about_covariates.R

```
#important global scope constant that indicates the record locator for a
    surveyed person
ID = "SEQN"

#first handle recoding of the treatments
trt_coding_function = trt_coding_functions[[TreatmentVarCode]]
if (!is.null(trt_coding_function)){
  ALL_DATA[[TreatmentVarFilename]][, TreatmentVarCode] = trt_coding_
      function(ALL_DATA[[TreatmentVarFilename]][, TreatmentVarCode])
}

#now we need the seq numbers of our response variable
response_data = ALL_DATA[[ResponseVarFilename]]
#get our full y vector and name it that
y = response_data[, ResponseVarCode]
#now we reject data that we don't want
rej_function = reject_functions[[ResponseVarCode]]
if (!is.null(rej_function)){
  response_data = response_data[rej_function(y), ]
}
#now get our ID #'s for the people who gave a response variable
id_nos = response_data[, ID]
num_id_nos = length(id_nos)

#We pull out the rows of each data frame that share an ID with sleep data
ALLsub = list()
for (xpt_file in xpt_file_list){
  X = ALL_DATA[[xpt_file]]
  ALLsub[[xpt_file]] = X[X$SEQN %in% id_nos, ]
}

#get list of dimensions for each of the data frames that have overlap
dim_table_sub = matrix(NA, ncol = 3, nrow = length(xpt_file_list)
    )
for (i in 1 : length(xpt_file_list)){
  dim_table_sub[i, ] = c(xpt_file_list[i], dim(ALLsub[[xpt_file_
      list[i]]]))
}
dim_table_sub = as.data.frame(dim_table_sub)
colnames(dim_table_sub) = c("filename", "n", "p")
dim_table_sub$n= as.numeric(as.character(dim_table_sub$n))
```

```
dim_table_sub$p= as.numeric(as.character(dim_table_sub$p))
#how many covariates do we have?
p = sum(dim_table_sub$p)

#now we need a hash from covariate code => covariate short descriptions
cov_code_to_description = list()
raw_cov_desc = read.csv("R\\CovariateDescriptions.csv")
for (i in 1 : nrow(raw_cov_desc)){
  cov_code_to_description[[as.character(raw_cov_desc[i, 1])]] = as
     .character(raw_cov_desc[i, 2])
}

#now, we need to loop through each of the data tables and get the number of
#records per covariate that is not missing data
cov_master_list = matrix(NA, ncol = 4, nrow = 0)
for (xpt_file in xpt_file_list){
  Xsub = ALLsub[[xpt_file]]
  name_cov_and_num_rows = matrix(NA, ncol = 4, nrow = 0)
  for (cov_code in colnames(Xsub)){
    if (cov_code != ID){
      num_rows = sum(!is.na(Xsub[, cov_code]))
      cov_desc = cov_code_to_description[[cov_code]]
      cov_desc = ifelse(is.null(cov_desc), "", cov_desc)
      name_cov_and_num_rows = rbind(name_cov_and_num_rows, c(xpt_
         file, cov_code, cov_desc, num_rows))
    }
  }
  cov_master_list = rbind(cov_master_list, name_cov_and_num_rows)
}
cov_master_list = as.data.frame(cov_master_list)
colnames(cov_master_list) = c("filename", "cov code", "cov
   description", "num_obs_non_null")
cov_master_list$num_obs_non_null = as.numeric(levels(cov_master_
   list$num_obs_non_null)[cov_master_list$num_obs_non_null])

#kill inconsistencies
cov_master_list = cov_master_list[cov_master_list$num_obs_non_null
    <= num_id_nos, ]

#how many non-missing points are there by covariate?
#hist(cov_master_list$num_obs_non_null, br = 100, xlab = "Response Rate",
   main = "Covariates by Response Rate in NHANES 2007/8")

#loookup names
#cov_master_list[cov_master_list[, "cov code"] == "PAD680", ]

num_control_vars = 0
```

```
for (filename in names(control_variables)){
  num_control_vars = num_control_vars + length(control_variables[[
      filename]])
}


#want table of variables used (just subtable of master list)
control_var_descriptions = matrix(NA, ncol = 4, nrow = 0)
for (filename in names(control_variables)){
  covariates = control_variables[[filename]]
  for (cov in covariates){
    control_var_descriptions = rbind(control_var_descriptions, cov
        _master_list[cov_master_list[, "cov code"] == cov, ])
  }
}
control_var_descs_ordrd = control_var_descriptions[order(control_
    var_descriptions[, "cov code"]), ]


treatment_var_description = cov_master_list[cov_master_list[, "cov
    code"] == TreatmentVarCode, ]
response_var_description = cov_master_list[cov_master_list[, "
    cov code"] == ResponseVarCode, ]

cat(paste("comparison #", comparison_num, ":\ntreatment: ",
    TreatmentVarCode, " (", treatment_var_description[, "cov
    description"] ,")\nresponse: ", ResponseVarCode, " (", response_
    var_description[, "cov description"] ,")\n", sep = ""))
if (PRINT_CONTROLS){
  cat(paste("controlled by", num_control_vars, "covariates:\n"))
}
```

Using the rows selected in the previous code, we now construct a design matrix. We make sure that we convert categorical variables to dummy indicator columns and we make sure to delete observations with illegal values specified by the user.

../R/04_create_design_matrix.R

```
#do an intersection of all the control variables that are not NULL,
#and the treatment var that is not null, then tally the IDs left
#and of course see how many there are
column_selections = list()
#do control vars
for (filename in names(control_variables)){
  covariates = control_variables[[filename]]
  current_data = ALLsub[[filename]]
  for (cov in covariates){
    #save the data that is not missing
```

```
    column_selections[[cov]] = current_data[!is.na(current_data[,
       cov]), ID]
    if (PRINT_CONTROLS){
      #get the description of this covariate
      description = cov_master_list[cov_master_list[, "cov code"]
         == cov, ][, "cov description"]
      #get the num obs's left
      appended_list = c()
      for (record_list in column_selections){
        appended_list = append(appended_list, record_list)
      }
      num_left_over = sum(table(appended_list) >= length(column_
         selections))
      #is this a categorical var?
      categorical_var = cov %in% control_variables_that_are_
         categorical
      #print to screen
      cat(paste(ifelse(categorical_var, paste("{", cov, "}", sep =
         ""), cov), " (", description, ") n_left = ", num_left_over,
          "\n", sep = ""))
    }
  }
}
#do treatment var
covariates = control_variables[[TreatmentVarFilename]]
current_data = ALLsub[[TreatmentVarFilename]]
column_selections[[TreatmentVarCode]] = current_data[!is.na(current_
   data[, TreatmentVarCode]), ID]


appended_list = c()
for (record_list in column_selections){
  appended_list = append(appended_list, record_list)
}

#we want to get a list of the record IDs for unique people that we'll use in
   our
#master data matrix
record_ids_to_use = table(appended_list) >= num_control_vars
record_ids_to_use = names(record_ids_to_use[record_ids_to_use])
record_ids_to_use = sort(record_ids_to_use)

num_records_left = length(record_ids_to_use)


##want actual dataframe now with treatment as FIRST cov and response as SECOND
    cov and controls as all other columns
```

```
#so what are our dimensions??
#n will be the num records left and p will be the number of control vars + 2
Xm = matrix(NA, nrow = num_records_left, ncol = num_control_vars +
    2)
rownames(Xm) = record_ids_to_use

#the first column is the treatment
trtXsub = ALLsub[[TreatmentVarFilename]]
#get records that match
trtXsub = trtXsub[trtXsub[, ID] %in% record_ids_to_use, ]
trtXsub = trtXsub[order(trtXsub[, ID]), ]
#now since the record ids are sorted and we're sure they're all there, so sort
    this matrix by record
Xm[, 1] = trtXsub[, TreatmentVarCode]

#the second column is the response
resXsub = ALLsub[[ResponseVarFilename]]
#get records that match
resXsub = resXsub[resXsub[, ID] %in% record_ids_to_use, ]
resXsub = resXsub[order(resXsub[, ID]), ]
#now since the record ids are sorted and we're sure they're all there, so sort
    this matrix by record
Xm[, 2] = resXsub[, ResponseVarCode]

#now, the rest are controls
count = 1
for (filename in names(control_variables)){
# print(filename)
  covariates = control_variables[[filename]]
  conXsub = ALLsub[[filename]]
  conXsub = conXsub[conXsub[, ID] %in% record_ids_to_use, ]
  conXsub = conXsub[order(conXsub[, ID]), ]
  for (cov in covariates){
#   print(paste("  ", cov))
    Xm[, count + 2] = conXsub[, cov]
    count = count + 1
  }
}


#now convert to data frame and set column names
Xm = as.data.frame(Xm)
colnames(Xm)[1] = TreatmentVarCode
colnames(Xm)[2] = ResponseVarCode
colnames(Xm)[3 : ncol(Xm)] = as.character(control_var_descriptions
    [, "cov code"])
```

```
#kill NA's even though they SHOULDN'T exist in the first place
nrow_before_killing_nas = nrow(Xm)
Xm = Xm[rowSums(is.na(Xm)) == 0, ]
nrow_after_killing_nas = nrow(Xm)
#cat(paste("deleted", (nrow_before_killing_nas - nrow_after_killing_nas), "
    rows that had missing data\n"))

###NOW we need to kill rows that have value that we don't care to analyze
nrow_before_killing_illegals = nrow(Xm)
for (cov in names(illegal_variable_values)){
  if (cov %in% colnames(Xm)){
    Xm = Xm[!(Xm[, cov] %in% illegal_variable_values[[cov]]), ]
  }
}
num_records_left = nrow(Xm)
if (PRINT_CONTROLS){
  cat(paste("deleted", (nrow_before_killing_illegals - num_records_
      left), "rows that had illegal values\n"))
}

treatment_col = paste("treatment", TreatmentVarCode, sep = "_")
response_col = paste("response", ResponseVarCode, sep = "_")
colnames(Xm)[1] = treatment_col
colnames(Xm)[2] = response_col


### now, we need to convert the categorical variables to dummies
for (covar in control_variables_that_are_categorical){
# print(covar)
  #pull out column
  categorial_var = Xm[, covar]
  #now we need to pull out the categories
  categories = sort(names(table(categorial_var)))
  #now as usual we hack off the last category
  categories = categories[1 : (length(categories) - 1)]
  #now we need to make a new matrix of 0's with ONLY the orthogonal contrasts
  Xm_cat = matrix(0, nrow = num_records_left, ncol = length(
      categories))
  for (i in 1 : num_records_left){
    x = as.character(categorial_var[i])
    Xm_cat[i, which(categories == x)] = 1
  }

  #now convert to a dataframe
  Xm_cat = as.data.frame(Xm_cat)
  colnames(Xm_cat) = paste(covar, categories, sep = "_")
```

```
  #now kill the covariate in the original matrix
  Xm[, covar] = NULL

  #now add on the covariates
  Xm = cbind(Xm, Xm_cat)
}

#now we need to code treatment as 0 / 1
# if (max(Xm[,treatment_col]) ==2 ){
#   Xm[, treatment_col] = Xm[, treatment_col] - 1
# }
```

We now estimate propensity scores, run the matching algorithm, compute effect sizes and Bonferroni-corrected significance levels to investigate whether or not there is a causal relationship.

../R/05_prop_score_and_matching.R

```
#we can see what the result would have been if this was a randomized study
#mod = lm(formula(paste(response_col, "~ .")), Xm)
#summary(mod)
#t.test(Xm[Xm[,treatment_col] == 1, response_col], Xm[Xm[,treatment_col] ==
    0, response_col])
#wilcox.test(Xm[Xm[,treatment_col] == 1, response_col], Xm[Xm[,treatment_col]
    == 0, response_col], conf.int = TRUE)

#Get balance
treatment_cov_matrix <- Xm[Xm[,treatment_col] == 1, c(-1,-2)]
controls_cov_matrix <- Xm[Xm[,treatment_col] == 0, c(-1,-2)]
cat(paste("num controls obs = ", nrow(controls_cov_matrix), ", num
    treatment obs = ", nrow(treatment_cov_matrix), " (total n = ",
   (nrow(controls_cov_matrix) + nrow(treatment_cov_matrix)), ")\n\
   n", sep = ""))


#Get outcomes for each group
control_outcomes <- Xm[Xm[,treatment_col] == 0,2]
treatment_outcomes <- Xm[Xm[,treatment_col] == 1,2]

compute_cov_stats = function(treatment_cov_matrix, controls_cov_
    matrix){
  p = ncol(treatment_cov_matrix)
  n_c = nrow(controls_cov_matrix)
  n_t = nrow(treatment_cov_matrix)
  diff_mat = matrix(NA, nrow = p, ncol = 7)
  colnames(diff_mat) = c("trt_mean", "cont_mean", "trt_sd", "cont_
     sd", "std_sd_diff", "t", "pval")
  rownames(diff_mat) = colnames(treatment_cov_matrix)
```

```
  diff_mat[, "trt_mean"] = mean(treatment_cov_matrix)
  diff_mat[, "cont_mean"] = mean(controls_cov_matrix)
  diff_mat[, "trt_sd"] = sd(treatment_cov_matrix)
  diff_mat[, "cont_sd"] = sd(controls_cov_matrix)

  diff_mat[, "std_sd_diff"] = abs(diff_mat[, "trt_mean"] - diff_
     mat[, "cont_mean"]) / sqrt((diff_mat[, "trt_sd"]^2 + diff_mat
     [, "cont_sd"]^2) / 2)
  est_var_diff = diff_mat[, "trt_sd"]^2 / n_t + diff_mat[, "cont_
     sd"]^2 / n_c
  diff_mat[, "t"] = (diff_mat[, "trt_mean"] - diff_mat[, "cont_
     mean"]) / sqrt(est_var_diff)
  df = min(n_c, n_t) #don't bother with WelchSatterthwaite equation...
     who cares
  diff_mat[, "pval"] = 2 * pt(abs(diff_mat[, "t"]), df, lower.tail
     = FALSE)
  #if the std diff is NaN that means there's perfect separation or equal means
      between trt and control e.g. on a binary variable
  #all 1's are in the trt and all 0's are in the control or all are 1's or 0's
      in both categories.
  #We want to bring this to the users attention. Since
  #the output is sorted by pval, let's make sure it appears on top with a flag
      if the mean is different, otherwise on the bottom:
  diff_mat[, "pval"] = ifelse(is.nan(diff_mat[, "pval"]) & diff_
     mat[, "trt_mean"] == diff_mat[, "cont_mean"], Inf, diff_mat[,
     "pval"])
  diff_mat[, "pval"] = ifelse(is.nan(diff_mat[, "pval"]) & diff_
     mat[, "trt_mean"] != diff_mat[, "cont_mean"], -Inf, diff_mat[,
      "pval"])
  #return the rounded matrix
  round(diff_mat, 2)
}

diff_mat = compute_cov_stats(treatment_cov_matrix, controls_cov_
   matrix)
diff_mat = diff_mat[order(diff_mat[, "pval"]), ]

if (PRINT_COV_CHARACTERISTICS){
  cat("\nCovariate characteristics between treatment and control
     before matching (sorted by magnitude of difference)\n")
  print(diff_mat)
}


model_formula = formula(paste(paste(treatment_col, "~"), paste(
   names(treatment_cov_matrix), "", collapse = " + ")))
prop.model <- glm(model_formula, data = Xm, family = "binomial")
```

```
treatment_scores <- predict(prop.model, newdata=treatment_cov_
    matrix, type = "response")
control_scores <- predict(prop.model, newdata=controls_cov_matrix
    , type = "response")

#boxplot(treatment_scores, control_scores)

naive_dist <- abs(outer(control_scores, treatment_scores, FUN = "-")
    )

#Transpose if we have more controls than treatments
if (length(control_scores) > length(treatment_scores)) {
  naive_dist <- t(naive_dist)
}
matches <- pairmatch(naive_dist)
matches_matrix <- outer(matches, levels(matches), FUN = "==")

matches_matrix <- ifelse(is.na(matches_matrix), FALSE, matches_
    matrix)

Xm_trt_matched = matrix(NA, ncol = ncol(Xm), nrow = 0)
Xm_control_matched = matrix(NA, ncol = ncol(Xm), nrow = 0)

for (j in 1 : ncol(matches_matrix)){
  match = names(matches_matrix[matches_matrix[, j] == TRUE, j])
  if (match[1] %in% rownames(controls_cov_matrix)){
    #we're in a control
    Xm_control_matched = rbind(Xm_control_matched, Xm[match[1], ])
    Xm_trt_matched = rbind(Xm_trt_matched, Xm[match[2], ])
  }
  else {
    #we're in a trt
    Xm_control_matched = rbind(Xm_control_matched, Xm[match[2], ])
    Xm_trt_matched = rbind(Xm_trt_matched, Xm[match[1], ])
  }
}

Xm_trt_matched = as.data.frame(Xm_trt_matched)
Xm_control_matched = as.data.frame(Xm_control_matched)
#calculate std diffs after matching
diff_mat = compute_cov_stats(Xm_trt_matched, Xm_control_matched)
diff_mat = diff_mat[3 : nrow(diff_mat), ]
diff_mat = diff_mat[order(diff_mat[, "pval"]), ]

if (PRINT_COV_CHARACTERISTICS){
  cat("\n\nCovariate characteristics between treatment and control
```

```
    *after* matching (sorted by magnitude of difference)\n")
  print(diff_mat)
}

#now we have our matches, let's run the same tests above on them
avg_diff = mean(Xm_trt_matched[, response_col] - Xm_control_
    matched[, response_col])
ttest = t.test(Xm_trt_matched[, response_col], Xm_control_matched[,
    response_col], paired = TRUE)
mwwtest = wilcox.test(Xm_trt_matched[, response_col], Xm_control_
    matched[, response_col], paired = TRUE, conf.int = TRUE)
lin_mod = lm(formula(paste(response_col, "~ .")), rbind(Xm_trt_
    matched, Xm_control_matched))

#make sure to Bonferroni correct for multiple comparisons
ttest_pval = ttest$p.value * num_comparisons
mwwtest_pval = mwwtest$p.value * num_comparisons
lin_regr_pval = coef(summary(lin_mod))[, 4][2] * num_comparisons #the
    trt is always the second row

#print results to screen
cat(paste("\neffect size is\n  ", round(avg_diff, 4), "\nBonf-adj
    wilcox test pval / 2-samp t-test / OLS t-test pval =\n  ", round
    (mwwtest_pval, 4), " / ", round(ttest_pval, 4), " / ", round(lin_
    regr_pval, 4), "\n", sep = ""))
```

Lastly, we run a sensitivity analysis on this comparison using the Bonferroni-corrected Γ metric.

../R/06_gamma_sensitivity.R

```
sens_int_bounds <- function(treat, control, G){
  return(
    function(beta){
      effect_diffs <- (treat-beta) - control;
      abs_ranks <- rank(abs(effect_diffs))
      pos_diffs <- effect_diffs > 0
      neg_diffs <- effect_diffs < 0
      test_stat <- sum(pos_diffs * abs_ranks)
      rank_sum <- sum((effect_diffs != 0) * abs_ranks)
      E_plus <- rank_sum * G / (1 + G)
      E_minus <- rank_sum / (1 + G)
      SD <- sqrt(sum((effect_diffs != 0) * abs_ranks^2) * G / (1 + G
          )^2)
      c((test_stat - E_plus) / SD, (test_stat - E_minus) / SD)
    }
  )
}
```

```
alpha_ci <- function(betagrid, treat, control, Gamma, alpha, n_
    tests){
  D_plus_minus <- sapply(betagrid, sens_int_bounds(treat, control,
      Gamma))
  alpha_corrected <- alpha / 2 / n_tests
  z <- qnorm(alpha_corrected)
  upper <- max(betagrid[D_plus_minus[2,]>=z])
  lower <- min(betagrid[D_plus_minus[1,]<=-z])
  c(lower,upper)
}

betagrid <- seq(-abs(avg_diff), abs(avg_diff), 2*abs(avg_diff)/
    1000)
Gammas <- seq(1,5,0.01)

Gamma_min <- 1 #Smallest Gamma for which we have no effect

for (G in Gammas){
  CI <- alpha_ci(betagrid, Xm_trt_matched[, response_col], Xm_
      control_matched[, response_col], G, 0.05, num_comparisons)
  if (VERBOSE_GAMMA_COMPUTATIONS){
    cat(sprintf("%f   SI for Gamma = %f: (%f, %f)\n", 95, G, CI[1],
        CI[2]))
  }
  if (0 >= CI[1] * CI[2] || (CI[1] == min(betagrid) && CI[2] == max(
      betagrid)) ){
    Gamma_min <- G
    break
  }
}



#mult_comp_adj_gamma = 1
#print sensitivity result to screen
cat(paste("Sensitivity Analysis, 95% Gamma =\n  ", Gamma_min, "\n\
    n\n", sep = ""))
```