Inverse Probability Weighting for Time Dependent Data

Ashley Naimi

In this document, we will cover methods to deal with complex longitudinal data. The example data are simulated from an algorithm developed by Jessica Young (Young et al. 2010) and later adapted by Erica Moodie (Moodie, Stephens, and Klein 2014).

We will start by loading some relevant packages in R:

```
# install and load packages
packages <- c("data.table","tidyverse","skimr",
              "here","ggthemes","extrafont","survival",
              "sandwich","lmtest","survminer")

for (package in packages) {
  if (!require(package, character.only=T, quietly=T)) {
    install.packages(package, repos='http://lib.stat.cmu.edu/R/CRAN')
  }
}

for (package in packages) {
  library(package, character.only=T)
}

font_import(pattern = 'Arial')
```

```
## Importing fonts may take a few minutes, depending on the number of fonts and the speed of the system
## Continue? [y/n]
```

```
# make fancy pants figures
thm <- theme_tufte() +
  theme(
    text = element_text(family="Arial",size=12),
    legend.position = "top",
    legend.background = element_rect(fill = "transparent", colour = NA),
    legend.key = element_rect(fill = "transparent", colour = NA)
  )
theme_set(thm)

# options to load and examine data
## use one of these lines
D <- read_csv(here("data","example_dat.csv"))
# D <- read_csv("change path to directoty that includes data")
# D <- read.csv("change path to directoty that includes data")

D %>% print(n=16)
```

```
## # A tibble: 44,645 x 10
##       ID   int  time     X   Xm1     Z   Zm1     Y timem1 last_flag
```

```
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>     <dbl>
##  1     1     1  1         0     0     1     0     0      0         0
##  2     1     2  2         0     0     1     1     0      1         0
##  3     1     3  3         0     0     0     1     0      2         0
##  4     1     4  4         0     0     1     0     0      3         0
##  5     1     5  5         1     0     0     1     0      4         0
##  6     1     6  6         1     1     0     0     0      5         0
##  7     1     7  7         0     1     0     0     0      6         0
##  8     1     8  8         0     0     1     0     0      7         0
##  9     1     9  9         0     0     0     1     0      8         0
## 10     1    10 10         1     0     1     0     0      9         0
## 11     1    11 11         0     1     0     1     0     10         0
## 12     1    12 12         1     0     1     0     0     11         1
## 13     2     1  1         0     0     0     0     0      0         0
## 14     2     2  2         1     0     1     0     0      1         0
## 15     2     3  3         0     1     1     1     0      2         0
## 16     2     4  3.21      1     0     1     1     1      3         1
## # ... with 44,629 more rows
```

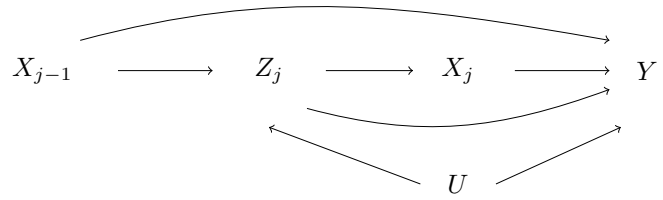
In these data, the sample consists of $N = 5,000$ individuals, each with up to 12 time-points (e.g., weeks) of follow up. The total number of person-time units is 44645.

The columns contain distinct variables: `int` is the time interval and `time` is the actual time. The only difference between these two is that, for the last observation for those who had an event, the `int` will be a integer whereas `time` will be a real number.

The variables `X` and `Z` represent the time-dependent exposure and time-dependent confounder, respectively. `Xm1`, `Zm1`, and `timem1` represent the lagged versions of `X`, `Z`, `time`. That is, for a given row, the (for example) `Xm1` value will be the same as the value of `X` in the previous row (and zero if the previous row is pre time 0). Finally, `Y` is an indicator of whether the event occurred.

Data like these (in long form), are sometimes referred to as the Andersen-Gill data structure, and are commonly encountered with survival data (Therneau and Grambsch 2000).

These data were generated from using a mechanism that can be described with the following DAG:



With these data, our interest lies in the average treatment effect. The dataset is generated such that we have a survival outcome, where `time` is the time to event, and `Y` is the event indicator. We can thus define the average treatment effect on the hazard ratio scale as:

$$\lambda_{T^{\overline{x}=1}}(t)/\lambda_{T^{\overline{x}=0}}(t)$$

which is interpreted as the hazard that would be observed if all individuals were exposed up to time $T = t$ divided by the hazard that would be observed if all individuals were unexposed up to time $T = t$. Of course, there are problems with hazard ratios (Hernán 2010). To address these, we'd typically also (or exclusively) express interest in the causal risk function (Cole et al. 2015), but we'll have to save that for another time.

IP-Weighting

Our first objective will be to use IP-weighting in these data to "erase" the arrow from Z_j to X_j, for all time points j.

To do this, we first create two propensity scores. The first will be the usual propensity score, defined as the probability of being exposed at each time point, conditional on the lagged exposure (`Xm1`), the confounder (`Z`), and the lagged confounder (`Zm1`). This will serve as the denominator of our stabilized and unstabilized weights. The second is just the probability of being exposed at each time point. This will serve as the numerator of our stabilized weights.

```
# propensity score
D$pscore <- glm(X ~ as.factor(int) + Xm1 + Z + Zm1,data=D,family=binomial(link="logit"))$fitted.values

# numerator of stabilized weights
D$p_num <- glm(X ~ as.factor(int),data=D,family=binomial(link="logit"))$fitted.values
```

Let's take a look at the dataset now that it has the propensity score:

```
D %>% print(n=16)
```

```
## # A tibble: 44,645 x 12
##       ID   int  time     X   Xm1     Z   Zm1     Y timem1 last_flag pscore p_num
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>     <dbl>  <dbl> <dbl>
##  1     1     1  1         0     0     1     0     0      0         0  0.347 0.260
##  2     1     2  2         0     0     1     1     0      1         0  0.538 0.375
##  3     1     3  3         0     0     0     1     0      2         0  0.359 0.384
##  4     1     4  4         0     0     1     0     0      3         0  0.360 0.386
##  5     1     5  5         1     0     0     1     0      4         0  0.366 0.384
##  6     1     6  6         1     1     0     0     0      5         0  0.375 0.388
##  7     1     7  7         0     1     0     0     0      6         0  0.374 0.381
##  8     1     8  8         0     0     1     0     0      7         0  0.359 0.376
##  9     1     9  9         0     0     0     1     0      8         0  0.352 0.361
## 10     1    10 10         1     0     1     0     0      9         0  0.344 0.353
## 11     1    11 11         0     1     0     1     0     10         0  0.550 0.374
## 12     1    12 12         1     0     1     0     0     11         1  0.373 0.381
## 13     2     1  1         0     0     0     0     0      0         0  0.211 0.260
## 14     2     2  2         1     0     1     0     0      1         0  0.364 0.375
## 15     2     3  3         0     1     1     1     0      2         0  0.698 0.384
## 16     2     4  3.21      1     0     1     1     1      3         1  0.534 0.386
## # ... with 44,629 more rows
```

Next, we create the stabilized and unstabilized weights. To do this, we first obtain the probability of the observed exposure. That is, in a given time point, if a person is exposed, their numerator should be `p_num` and their denominator should be `pscore`. If they are instead unexposed, their numerator should be `1 - p_num` and their denominator should be `1 - pscore`:

```
D <- D %>% group_by(ID) %>%
  mutate(num = X*p_num + (1-X)*(1-p_num),
         den = X*pscore + (1-X)*(1-pscore),
         sw = cumprod(num/den),
         w = cumprod(1/den)) %>%
  ungroup(ID)
```

```
D %>% select(ID, time, X, pscore, p_num, num, den, sw, w) %>% print(n=16)
```

```
## # A tibble: 44,645 x 9
##       ID  time     X pscore p_num   num   den    sw       w
##    <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
```

```
##  1     1  1        0  0.347 0.260 0.740 0.653 1.13     1.53
##  2     1  2        0  0.538 0.375 0.625 0.462 1.53     3.31
##  3     1  3        0  0.359 0.384 0.616 0.641 1.47     5.17
##  4     1  4        0  0.360 0.386 0.614 0.640 1.41     8.08
##  5     1  5        1  0.366 0.384 0.384 0.366 1.48    22.1
##  6     1  6        1  0.375 0.388 0.388 0.375 1.54    58.8
##  7     1  7        0  0.374 0.381 0.619 0.626 1.52    94.0
##  8     1  8        0  0.359 0.376 0.624 0.641 1.48   147.
##  9     1  9        0  0.352 0.361 0.639 0.648 1.46   226.
## 10     1 10        1  0.344 0.353 0.353 0.344 1.49   657.
## 11     1 11        0  0.550 0.374 0.626 0.450 2.08  1460.
## 12     1 12        1  0.373 0.381 0.381 0.373 2.12  3914.
## 13     2  1        0  0.211 0.260 0.740 0.789 0.938    1.27
## 14     2  2        1  0.364 0.375 0.375 0.364 0.966    3.48
## 15     2  3        0  0.698 0.384 0.616 0.302 1.97    11.5
## 16     2  3.21     1  0.534 0.386 0.386 0.534 1.42    21.6
## # ... with 44,629 more rows
```

A quick note on the difference between stabilized and unstabilized weights. In a weighted analysis, the sample size contribution of each row is increased (or decreased) by its weight. With unstabilized weights, the sample size contribution for a given row can quickly outweight the sample. For example, the unstablized weight for the last observation for ID = 1 is c(12 = 3914). This is nearly 80% of the sample size of 5,000! In contrast, the stabilized weight for the sample observation and row is c(12 = 2.12).

Because of this problem with unstablized weights, the general recommendation is to use always stabilized weights.

With our weights created, the next step is to evaluate the distribution of the weights and the propensity score. This step is important because it's one of the ways in which we can evaluate whether **positivity** holds. Recall that positivity requires that there be exposed and unexposed individuals within all confounder strata at all time points. Expressed mathematically, positivity requires that the probability of being exposed conditional on all confounders is bounded away from zero and one for all individuals over all time points:

$$0 < P(X_j = 1 \mid X_{j-1}, Z_j, Z_{j-1}) < 1, \forall j.$$

Why is this important? Suppose that for a certain exposed individual i, the probability of being exposed is zero. When we create the weights for this person, we end up with an expression that looks like this:

$$w_i = \frac{1}{P(X_j = 1 \mid X_{j-1}, Z_j, Z_{j-1})} = \frac{1}{0},$$

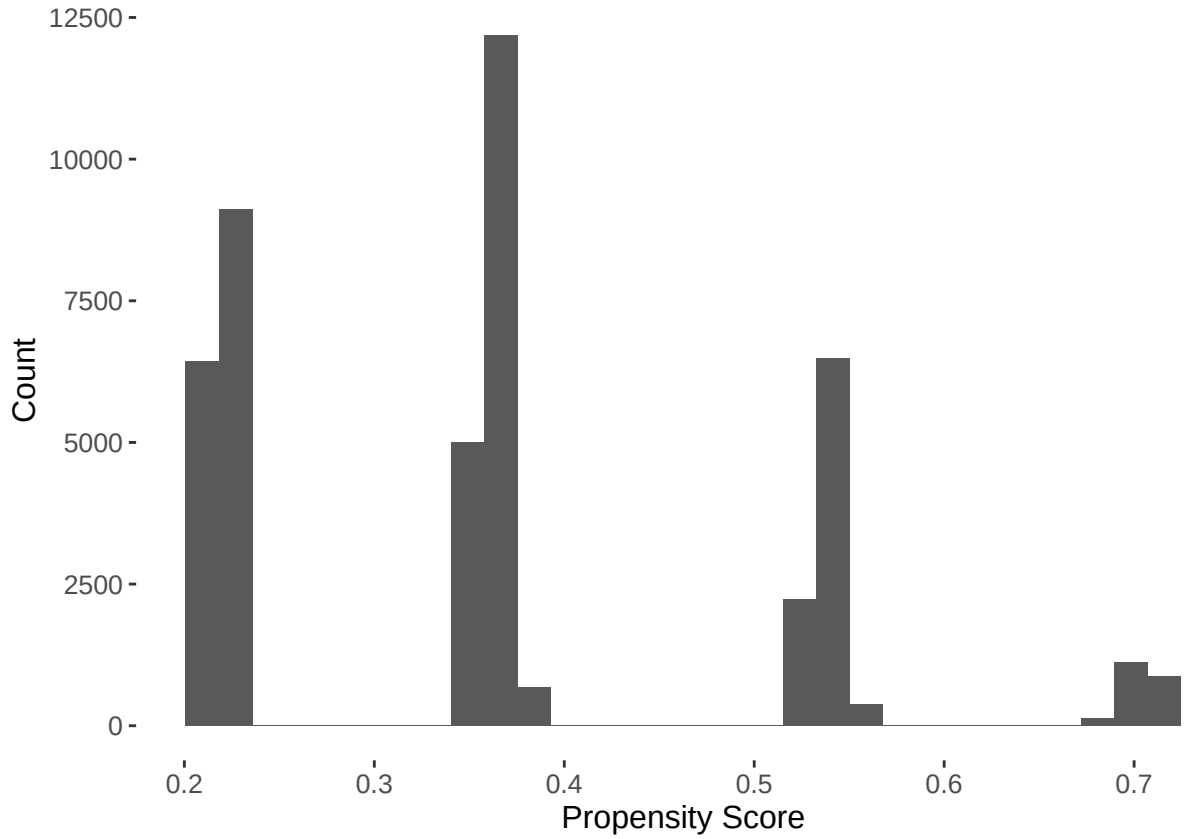
which is undefined. In fact, because of this, the conditional probability of being exposed can't even be close to zero. If, for example, this probability is 0.0001, then the weight becomes:

$$w_i = \frac{1}{P(X_j = 1 \mid X_{j-1}, Z_j, Z_{j-1})} = \frac{1}{0.0001} = 10,000.$$

This means that in a dataset like ours, with 5000 individuals, one individual's person-time contribution will be counted 10,000 times, which will have an overwhelming impact on our estimate of the causal effect.

There are a few techniques we can use to evaluate whether such problematic weights exist. As a first step, we can simply look at the distribution of the propensity score:

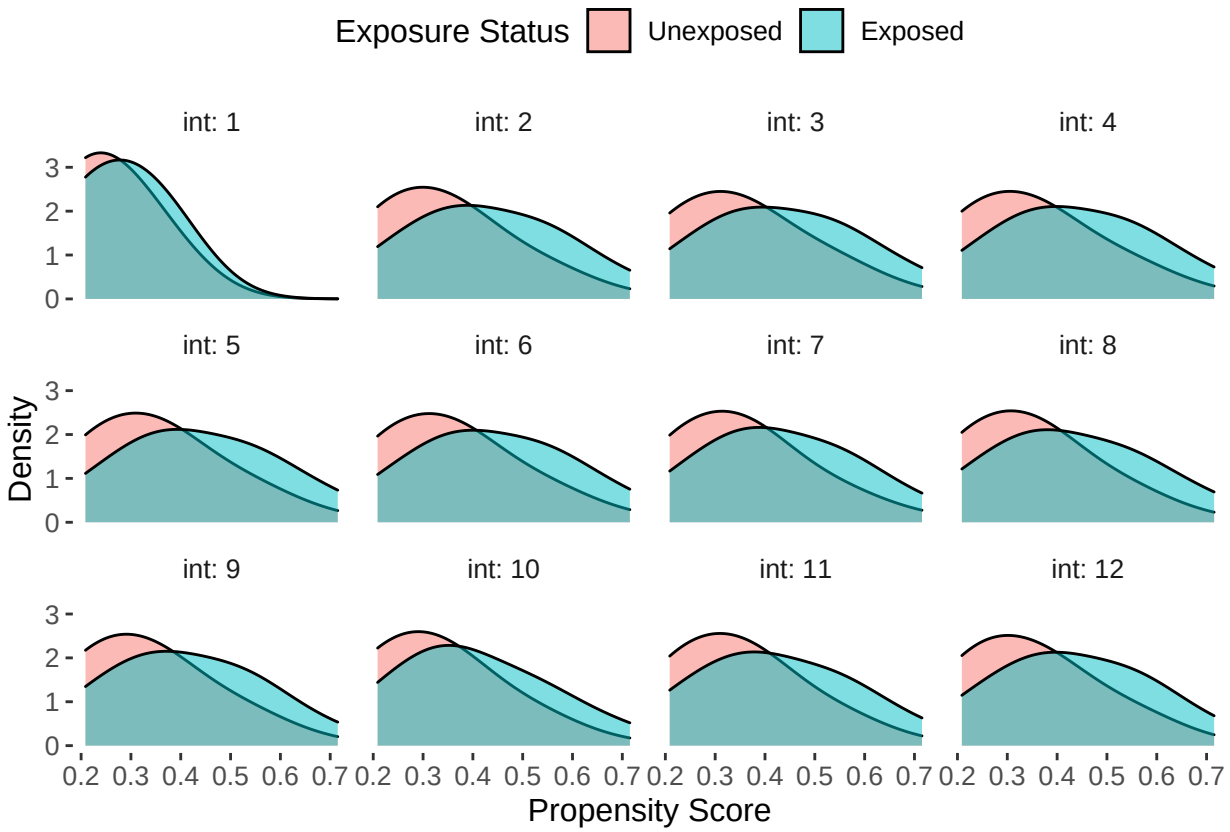
```
ggplot(D) +
  geom_histogram(aes(pscore)) +
  xlab("Propensity Score") +
  ylab("Count")
```



This doesn't reveal any immediate concerns, but there are two problems with it: 1) it is presenting propensity scores aggregated over all time points. This is a problem because the propensity score may be different at later time points than at earlier time points; 2) it is not conisdering the fact that what really matters is **propensity score overlap**, which tells us how comparable exposed and unexposed individuals are in our data.

Two address these two problems, we can create a separate figure for each time point and, within each time point, look at how the propensity score distribution in the exposed group compares to the unexposed group:

```
ggplot(D) +
  geom_density(aes(x=pscore,fill=as.factor(X)),alpha=.5,bw=.1) +
  facet_wrap(~int, labeller=label_both) +
  xlab("Propensity Score") +
  ylab("Density") +
  scale_fill_discrete(name = "Exposure Status", labels = c("Unexposed", "Exposed"))
```



This figure reveals a little more. Namely, the overlap separates slightly over time on study. But the degree of separation is not indicative of any problems.

The final strategy that we'll use to evaluate the weights is to check their distribution, specifically the mean and the max of the **stabilized weights** at each time point:

```
D %>% group_by(int) %>% summarize(meanSW = mean(sw),
                                  maxSW = max(sw))
```

```
## # A tibble: 12 x 3
##      int meanSW maxSW
##  * <dbl>  <dbl> <dbl>
##  1     1  0.998  1.23
##  2     2  0.994  1.69
##  3     3  0.997  2.20
##  4     4  1.00   2.92
##  5     5  1.00   3.79
##  6     6  1.00   5.03
##  7     7  1.01   6.01
##  8     8  1.01   7.96
##  9     9  1.02   8.57
## 10    10  1.03  11.6
## 11    11  1.02  11.9
## 12    12  1.01  11.5
```

The mean of the stabilized weights at all time points should be 1, and the max weight should not be large. In this case "large" is not formally defined, but a good rule-of-thumb is to compare the largest weight to the sample size.

Overall, the distribution of the propensity score and weights is not suggestive of any positivity concerns.

Next, we use these weights to estimate the effect of interest. To estimate the causal hazard ratio, we can use a Cox proportional hazards regression model, or a pooled logistic regression model:

```
mod1_cox <- coxph(Surv(time,Y) ~ X + cluster(ID), data=D, ties="efron", weight=sw)

summary(mod1_cox)$coefficients
```

```
##        coef exp(coef)   se(coef)  robust se        z     Pr(>|z|)
## X 0.9400852    2.5602 0.04127268 0.05066852 18.55363 7.621932e-77
```

```
mod1_plr <- glm(Y ~ as.factor(int) + X, data=D, family = quasibinomial(link = "logit"), weights=sw)
# NB: quasibinomial is used instead of binomial to avoid a warning message. Both should yield the same

coeftest(mod1_plr, vcov = vcovCL(mod1_plr, cluster=D$ID, type = "HC1"))[13,]
```

```
##     Estimate   Std. Error      z value     Pr(>|z|)
## 1.037200e+00 5.209158e-02 1.991109e+01 3.261625e-88
```

The Cox PH model yields a hazard ratio estimate of 2.56 and the pooled logistic model yields a hazard ratio estimate of 2.82. To get appropriate standard errors for these estimates, we have to use the robust standard error estimator. This is accomplished using the `cluster(ID)` argument in the Cox model, and the `coeftest` function for the pooled logistic model. The robust standard error is 0.05 for the Cox model, and 0.05 for the pooled logistic model.

Lab Questions

1) You may want to use the code-snippet below to help you answer this. In the dataset `D`, how many individuals were actually exposed at all time-points over their follow-up? How many people were unexposed? What does this say about the extent to which the estimand (i.e., everyone exposed at all time points versus everyone unexposed at all time points) is supported by these data?

```
new_dat <- D %>%
  group_by(ID) %>%
  mutate(cumexp = cumsum(X),
         expratio = cumexp/int) %>%
  filter(last_flag == 1)
```

2) Why do you think the maximum stabilized weight gets larger over time on study? What does this say about a study with only three follow-up time points versus 60 follow-up time points?

BONUS

3) What is the magnitude of the association between Z_j and X_j in the unweighted data? What happens to the magnitude of this association in the weighted data? (report coefficients and appropriate standard errors)

g Computation

In our next example, we'll look at using g computation to analyze the same data. The best approach is to start with a table documenting all relevant variables that need to be modeled. This is basically every node in the DAG, ranked according to their causal order:

Order	Dependent Variable	Independent Variable
3	Y_j	$X_j, Z_j X_{j-1}$
2	X_j	Z_j, X_{j-1}
1	Z_j	X_{j-1}

To begin our exploration of g computation, let's start with fitting the models we'll need:

```
D <- D %>% select(ID,int,time,Y,X,Z,Xm1,Zm1,last_flag)

# model for the outcome
modY <- glm(Y ~ as.factor(int) + X + Z + Xm1 + Zm1, data=D, family=binomial("logit"))

# model for the exposure
modX <- glm(X ~ as.factor(int) + Z + Xm1 + Zm1, data=D, family=binomial("logit"))

# model for the confounder
modZ <- glm(Z ~ as.factor(int) + Xm1 + Zm1, data=D, family=binomial("logit"))
```

Note that the above model for the outcome cannot be used to quantify the effect of X on Y, because it includes Z, which opens a collider bias path as shown in the DAG. But we can use all three models together to get an estimate of the average of Y under different scenarios for X.

To give some intuition behind how the g computation procedure will work, let's take the first observation for the first individual in the dataset:

```
obs <- D[1,]

obs
```

```
## # A tibble: 1 x 9
##      ID   int  time     Y     X     Z   Xm1   Zm1 last_flag
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>     <dbl>
## 1     1     1     1     0     0     1     0     0         0
```

We can use `modZ`, `modX`, and `modY` to predict the second observation for this first individual, like this:

```
dZp <- data.table(Xm1=obs$X, Zm1=obs$Z, int=as.factor(2))
Zp <- as.numeric(predict(modZ,newdata=dZp,type="response")>runif(1))

dXp <- data.table(Xm1=obs$X, Zm1=obs$Z, Z=Zp, int=as.factor(2))
Xp <- as.numeric(predict(modX,newdata=dXp,type="response")>runif(1))

dYp <- data.table(Xm1=obs$X, Zm1=obs$Z, Z=Zp, X=Xp, int=as.factor(2))
Yp <- as.numeric(predict(modZ,newdata=dZp,type="response")>runif(1))
```

If we combined these predicted second time-point data with the first time-point data, we would get a dataset that looks like this:

```
rbind(obs,data.frame(ID=1, int=2, time=2, Y=Yp, X=Xp, Z=Zp, Xm1=obs$X, Zm1=obs$Z, last_flag=0))
```

```
## # A tibble: 2 x 9
##      ID   int  time     Y     X     Z   Xm1   Zm1 last_flag
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>     <dbl>
## 1     1     1     1     0     0     1     0     0         0
## 2     1     2     2     1     1     1     0     1         0
```

Note that all the relevant variables in the second time-point were obtained from model predictions. Using these models predictions, we could continue the process until we reached the end of follow-up (time point 12 in our case), or until the event occurred ($Y = 1$). While it may not be clear yet, this procedure is the core of g computation.

Let's build this small example into a bigger working illustration g computation. First, we'll select the first observation for each person in the dataset. This should lead to a dataset with 5,000 observations that reflect the first measurement for each variable.

```r
set.seed(123)

dat <- D %>% filter(int==1)
dat$id <- 1:nrow(dat)

pgf<-function(ii, mc_data, lngth, exposure = NULL){
    d <- mc_data
    d <- d[d$id==ii,]

    lngth <- lngth
    Zp <- Xp <- Yp <- mm <- numeric()
    mm[1] <- j <- 1
    id <- d$id

    Zp <- d$Z

    if (is.null(exposure)) {
      Xp[1] <- d$X
    } else{
      Xp[1] <- exposure
    }

    Yp[1] <- d$Y

    for (j in 2:lngth) {
      #cat("Iteration",j,"for observation",ii,"from Monte Carlo Data",'\n')
      if (Yp[j - 1]==0) {
        Xl=Xp[j-1];Zl=Zp[j-1]

        #cat("Generating Z",'\n')
        dZp <- data.table(Xm1=Xl, Zm1=Zl,int=as.factor(j))
        Zp[j] <- as.numeric(predict(modZ,newdata=dZp,type="response")>runif(1))

        #cat("Generating X",'\n')
        dXp <- data.table(Z=Zp[j], Zm1=Zl, Xm1=Xp[j-1], int=as.factor(j))
         if (is.null(exposure)) {
             Xp[j] <- as.numeric(predict(modX,newdata=dXp,type="response")>runif(1))
          } else{
             Xp[j] <- exposure
          }

        #cat("Generating Y",'\n')
        dYp <- data.table(X=Xp[j], Z=Zp[j], Xm1=Xp[j-1], Zm1=Zl, int=as.factor(j))
        Yp[j] <- as.numeric(predict(modY,newdata=dYp,type="response")>runif(1))

      } else {
        break
      }
      mm[j] <- j
    }
    gdat <- data.table(id,mm,Zp,Xp,Yp)
    gdat$last<-as.numeric(!(gdat$Yp==0)|gdat$mm==lngth)
    return(gdat)
```

```
  }

gComp_dat <- lapply(1:nrow(dat), function(x) pgf(x, mc_data=dat, lngth=12, exposure = NULL))
gComp_dat <- do.call(rbind,gComp_dat)

head(gComp_dat,20)
```

```
##     id mm Zp Xp Yp last
##  1:  1  1  1  0  0    0
##  2:  1  2  1  0  0    0
##  3:  1  3  0  0  0    0
##  4:  1  4  0  0  0    0
##  5:  1  5  0  0  0    0
##  6:  1  6  0  0  0    0
##  7:  1  7  0  0  0    0
##  8:  1  8  0  0  0    0
##  9:  1  9  0  0  0    0
## 10:  1 10  0  0  0    0
## 11:  1 11  0  0  0    0
## 12:  1 12  0  0  0    1
## 13:  2  1  0  0  0    0
## 14:  2  2  0  1  0    0
## 15:  2  3  0  1  0    0
## 16:  2  4  0  1  0    0
## 17:  2  5  0  1  0    0
## 18:  2  6  1  1  0    0
## 19:  2  7  1  0  1    1
## 20:  3  1  0  0  0    0
```

Let's take a look at some summary statistics from the `gComp_dat`, and compare the same statistics in the original data:

```
D %>% filter(last_flag==1) %>% summarize(meanY=mean(Y))
```

```
## # A tibble: 1 x 1
##   meanY
##   <dbl>
## 1 0.502
```

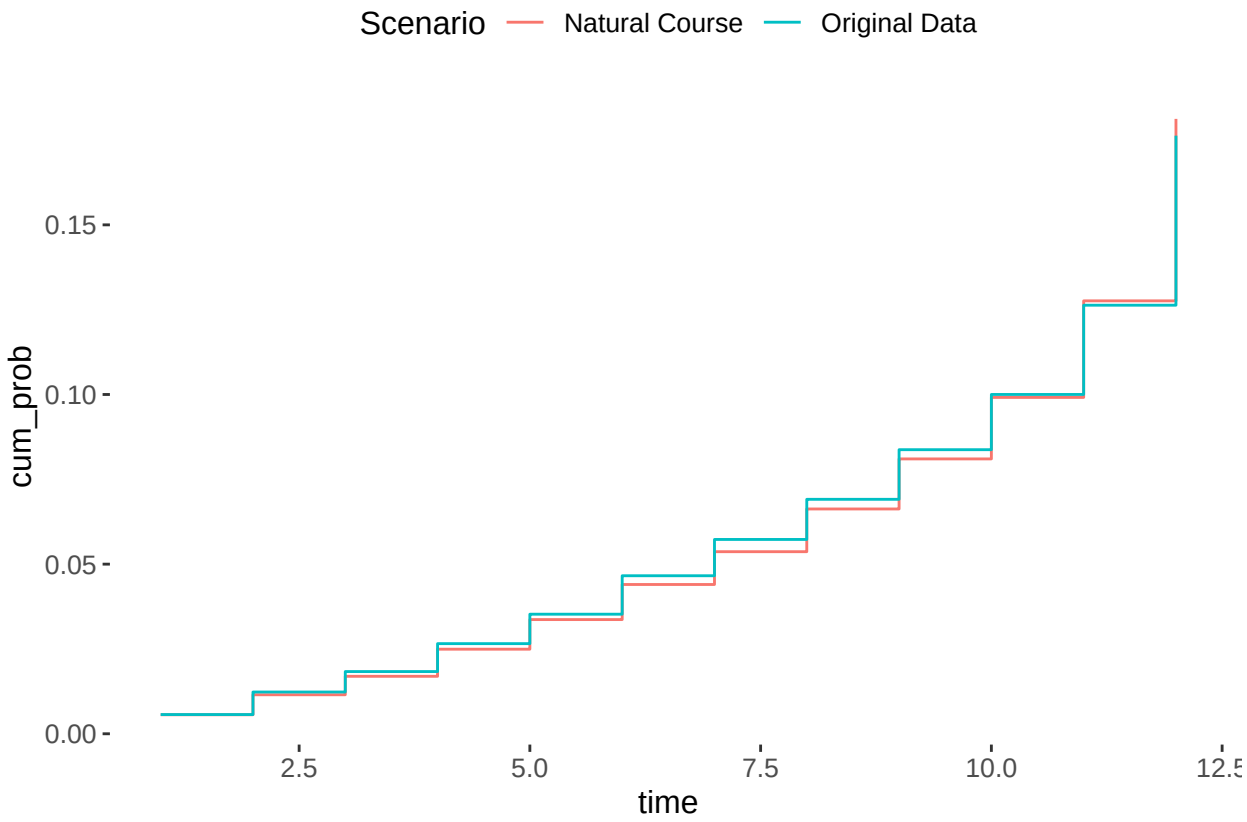
```
gComp_dat %>% filter(last==1) %>% summarize(meanY=mean(Yp))
```

```
##    meanY
## 1 0.4994
```

```
fitD <- coxph(Surv(int, Y) ~ 1, data = D)
expfitD <- survfit(fitD)

plot_dat0 <- data.frame(cum_prob=1-expfitD$surv, time=expfitD$time, Scenario="Original Data")

fitG <- coxph(Surv(mm, Yp) ~ 1, data = gComp_dat)
expfitG <- survfit(fitG)

plot_dat1 <- data.frame(cum_prob=1-expfitG$surv, time=expfitG$time, Scenario="Natural Course")

plot_dat <- rbind(plot_dat0,plot_dat1)
```

Let's plot the cumulative probabilities of the event over time in the original data, and in the data we obtained via g computation:

```
ggplot(plot_dat) + geom_step(aes(x=time,y=cum_prob,group=Scenario,color=Scenario))
```



This figure shows us that g computation is able to reproduce what we observe in the data. This is an important validation step, known as the natural course comparison. If the CDFs in the above figure did not overlap well, it would signal to us that we had work to do with our models, such as adding or taking away flexibility, adding interactions, etc.

Now let's use g computation to estimate the effect we're interested in. We do this by setting the exposure value to 1 and then to 0, instead of predicting the exposure value from the exposure model:

```
set.seed(123)

gComp_dat0 <- lapply(1:nrow(dat), function(x) pgf(x, mc_data=dat, lngth=12, exposure = 0))
gComp_dat0 <- do.call(rbind,gComp_dat0)

head(gComp_dat0,20)
```

```
##     id mm Zp Xp Yp last
##  1:  1  1  1  0  0    0
##  2:  1  2  1  0  0    0
##  3:  1  3  1  0  0    0
##  4:  1  4  0  0  0    0
##  5:  1  5  0  0  0    0
##  6:  1  6  0  0  0    0
##  7:  1  7  0  0  0    0
##  8:  1  8  0  0  0    0
##  9:  1  9  1  0  0    0
```

```
## 10:  1 10  1  0  1    1
## 11:  2  1  0  0  0    0
## 12:  2  2  1  0  0    0
## 13:  2  3  0  0  0    0
## 14:  2  4  0  0  0    0
## 15:  2  5  0  0  0    0
## 16:  2  6  0  0  0    0
## 17:  2  7  1  0  0    0
## 18:  2  8  0  0  0    0
## 19:  2  9  0  0  0    0
## 20:  2 10  1  0  0    0
```

```
gComp_dat1 <- lapply(1:nrow(dat), function(x) pgf(x, mc_data=dat, lngth=12, exposure = 1))
gComp_dat1 <- do.call(rbind,gComp_dat1)

head(gComp_dat1,20)
```

```
##     id mm Zp Xp Yp last
##  1:  1  1  1  1  0    0
##  2:  1  2  1  1  0    0
##  3:  1  3  0  1  0    0
##  4:  1  4  0  1  1    1
##  5:  2  1  0  1  0    0
##  6:  2  2  0  1  0    0
##  7:  2  3  0  1  0    0
##  8:  2  4  1  1  0    0
##  9:  2  5  0  1  0    0
## 10:  2  6  0  1  0    0
## 11:  2  7  0  1  0    0
## 12:  2  8  0  1  0    0
## 13:  2  9  0  1  0    0
## 14:  2 10  1  1  0    0
## 15:  2 11  1  1  1    1
## 16:  3  1  0  1  0    0
## 17:  3  2  0  1  0    0
## 18:  3  3  0  1  0    0
## 19:  3  4  0  1  0    0
## 20:  3  5  0  1  0    0
```

We can estimate the HR by combining `gComp_dat0` and `gComp_dat1` into a single dataset, with an exposure value:

```
gComp_dat01 <- rbind(gComp_dat0,gComp_dat1) %>% filter(last==1)

gComp_dat01
```

```
##          id mm Zp Xp Yp last
##     1:    1 10  1  0  1    1
##     2:    2 12  0  0  0    1
##     3:    3 12  0  0  0    1
##     4:    4  7  0  0  1    1
##     5:    5 12  1  0  0    1
##    ---
##  9996: 4996  5  0  1  1    1
##  9997: 4997 12  0  1  0    1
##  9998: 4998  7  0  1  1    1
##  9999: 4999 12  0  1  0    1
```

```
## 10000: 5000 12  0  1  0    1
```

```
cox_gComp <- coxph(Surv(mm,Yp) ~ Xp, data=gComp_dat01, ties="efron")

summary(cox_gComp)$coefficients
```

```
##         coef exp(coef)   se(coef)        z      Pr(>|z|)
## Xp 0.9284197  2.530507 0.02958785 31.37841 3.987163e-216
```

Running the Cox model in the data created by the g computation algorithm yields a hazard ratio of 2.53, which is very close to the estimates obtained from the marginal structural models above, as well as the true HR of 2.5.

To get confidence intervals for this point estimate, we have no choice but to bootstrap. This would require resampling the individuals in the original dataset, re-running the models used to predict in the g computation algorithm, re-run the `pgf` function, and then the Cox model to obtain a bootstrapped estimate. Running this procedure 500 times would give us a distribution of 500 point estimates that we could use to compute the standard deviation, which could be interpreted as the standard error of our original estimate.

There are several final considerations we should raise about g computation. First, in the above code, we obtained cumulative distribution functions and hazard ratios. This is one of the points of versatility with g computation. It's relatively easy to obtain the estimand on any scale we'd like (risk, odds, hazard, differences or ratios).

As a final note, one might legitimately ask why g computation works. After all, the regression model for Y includes Z, and the estimate for X in that model is thus subject to collider bias.

One can see why g computation works via single world intervention graphs (SWIGs) (Richardson and Robins 2013). The graph below shows why. The key feature is that g computation incorporates a model for Z in the estimation process. In doing so, the effect of X on Z is accounted for.

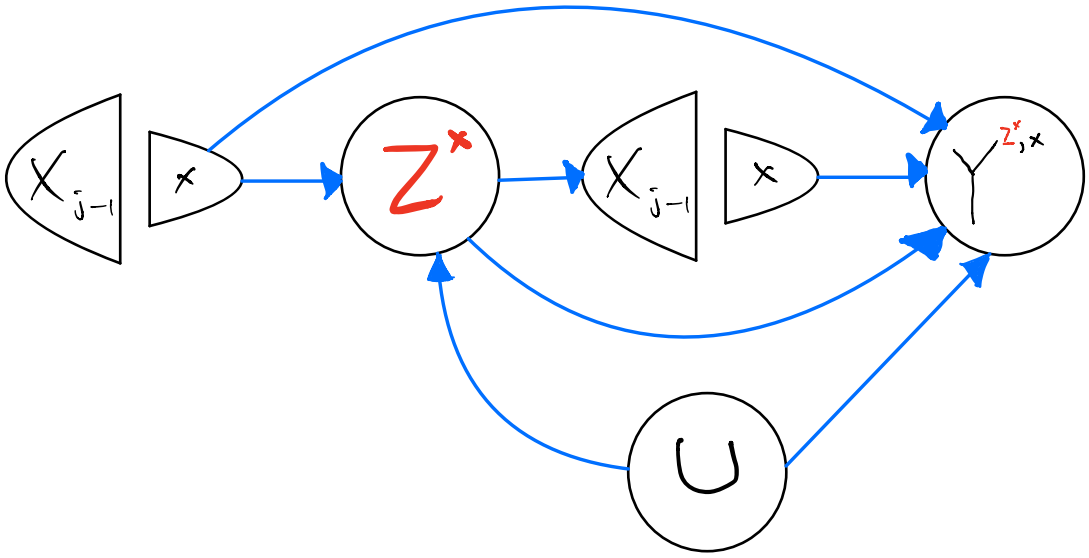


Figure 1: Single World Intervention Graph demonstrating why the g computation algorithms works in the presence of variables that are both time-varying confounders and mediators, and can also induce collider bias if adjusted for.

References

Cole, Stephen R., Michael G. Hudgens, M. Alan Brookhart, and Daniel Westreich. 2015. "Risk." *Am J Epidemiol* 181 (4): 246–50.

Hernán, M. A. 2010. "The Hazards of Hazard Ratios." *Epidemiol* 21: 13–15.

Moodie, Erica E. M., David A. Stephens, and Marina B. Klein. 2014. “A Marginal Structural Model for Multiple-Outcome Survival Data: Assessing the Impact of Injection Drug Use on Several Causes of Death in the Canadian Co-Infection Cohort.” *Stat Med* 33 (8): 1409–25.

Richardson, Thomas S, and James M Robins. 2013. “Single World Intervention Graphs (SWIGs): A Unification of the Counterfactual and Graphical Approaches to Causality.” Number 128. http://www.csss.washington.edu/Papers/wp128.pdf. Accessed Aug 26th,: Center for Statistics; the Social Sciences, University of Washington.

Therneau, Terry M., and Patricia M. Grambsch. 2000. *Modeling Survival Data : Extending the Cox Model*. New York: Springer.

Young, J. G., M. A. Hernán, S. Picciotto, and J. M. Robins. 2010. “Relation Between Three Classes of Structural Models for the Effect of a Time-Varying Exposure on Survival.” *Lifetime Data Anal* 16 (1): 71–84.