StanCon 2018: Causal inference with the g-formula in Stan

Leah Comment

January 12, 2018

Causal inference

Introduction and the frequentist g-formula

In the Rubin Causal Model, causal inference relies on contrasts of counterfactuals, also called potential outcomes (Imbens and Rubin 2015). Say we have an outcome of interest Y, and changing the distribution of some treatment or exposure A causally affects the distribution of Y. The directed acyclic graph depicting this scenario is shown in Figure 1.

Figure 1: The simplest causal model

$$A \longrightarrow Y$$

We call the value that Y_a would take the *potential outcome* under the regime $A = a$. If A is binary, $\mathbb{E}[Y_1 - Y_0]$ is the population average treatment effect (ATE) of changing A from 0 to 1 for every individual in the population. From a data set of size n, we might choose to model this with a logistic regression:

$$\text{logit}\left(P(Y_i = 1|A_i)\right) = \alpha_0 + \alpha_A A_i$$

This corresponds to a model for the potential outcomes Y_a for $a \in \{0, 1\}$:

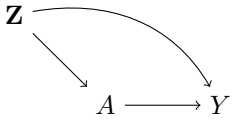
$$\mathbb{E}[Y_a] = \text{logit}^{-1}\left(\alpha_0 + \alpha_A a\right)$$

Then

$$\widehat{ATE} = \widehat{\mathbb{E}}[Y_1 - Y_0] = \text{logit}^{-1}\left(\hat{\alpha}_0 + \hat{\alpha}_A\right) - \text{logit}^{-1}\left(\hat{\alpha}_0\right)$$

Suppose we have measured counfounders $\mathbf{Z}$ and the true causal diagram looks like the one in Figure 2 below.

Figure 2: A simple causal model with exposure-outcome confounders



In the case of confounding by a vector of covariates $\mathbf{Z}$, we can add the confounders to the regression model, in which case the estimated average treatment effect depends on the distribution of $\mathbf{Z}$.

$$\widehat{ATE} = \widehat{\mathbb{E}}[Y_1 - Y_0] = \frac{1}{n}\sum_{i=1}^{n}\left[\text{logit}^{-1}\left(\hat{\alpha}_0 + \hat{\alpha}_A + \hat{\boldsymbol{\alpha}}_Z'\mathbf{Z}_i\right) - \text{logit}^{-1}\left(\hat{\alpha}_0 + \hat{\boldsymbol{\alpha}}_Z'\mathbf{Z}_i\right)\right]$$

Written another way, we can view this as a standardization over the distribution of $\mathbf{Z}$. In epidemiology, this standardization procedure is known as the g-formula. For discrete $\mathbf{Z}$, this formula can be written as a

$$\widehat{ATE} = \sum_{z}\left(\hat{P}(Y = 1|A = 1, \mathbf{Z} = \mathbf{z}) - \hat{P}(Y = 0|A = 1, \mathbf{Z} = \mathbf{z})\right)\hat{P}(\mathbf{Z} = \mathbf{z})$$

The Bayesian g-formula

The Bayesian analog to the g-formula formulates the distribution of the counterfactual Y_a as a posterior predictive value, integrating over the parameters $\boldsymbol{\theta}$ as well as the confounder distribution. A more complete explanation of the Bayesian g-formula can be found in Keil et al (Keil et al. 2015).

$$p(y_a|o) = \int\int p(\tilde{y}|a, \tilde{\mathbf{z}}, \theta)p(\tilde{\mathbf{z}}|\boldsymbol{\theta})p(\boldsymbol{\theta}|o)d\boldsymbol{\theta}d\tilde{\mathbf{z}}$$

A Bayesian estimate of the average treatment effect for a binary treatement A would be

$$\widehat{ATE} = \int\int \tilde{y}\left[p(\tilde{y}|a=1, \tilde{\mathbf{z}}, \theta) - p(\tilde{y}|a=0, \tilde{\mathbf{z}}, \theta)\right] p(\tilde{\mathbf{z}}|\boldsymbol{\theta})p(\boldsymbol{\theta}|o)d\boldsymbol{\theta}d\tilde{\mathbf{z}}$$

To perform the integration for $\boldsymbol{\theta}$, posterior draws of α_0, α_A, and $\boldsymbol{\alpha}_Z$ may be substituted for frequentist point estimates. To account for uncertainty regarding the distribution of the confounders, one could use a classical or Bayesian bootstrap. Using the classical bootstrap, n new values of $\mathbf{Z}$ would be sampled with replacement from the observed $\mathbf{Z}$ distribution during iteration b of the Markov Chain Monte Carlo. Denoting these resampled values as $\mathbf{Z}^{(1,b)}, \ldots, \mathbf{Z}^{(n,b)}$ and the parameter draws as $\alpha_0^{(b)}$, $\alpha_A^{(b)}$, and $\boldsymbol{\alpha}_0^{(b)}$, we can obtain a posterior draw of the ATE as

$$ATE^{(b)} = \frac{1}{n}\sum_{i=1}^{n}\left[\text{logit}^{-1}\left(\alpha_0^{(b)} + \alpha_A^{(b)} + \boldsymbol{\alpha}^{(b)\prime}\mathbf{Z}^{(i,b)}\right) - \text{logit}^{-1}\left(\alpha_0^{(b)} + \boldsymbol{\alpha}^{(b)\prime}\mathbf{Z}^{(i,b)}\right)\right]$$

Posterior summaries of the ATE can be obtained by taking the mean or quantiles of the $ATE^{(b)}$.

Simulating some data

```
# Simulate simple binary data with confounders situation
simulate_simple <- function(n) {
  Z1 <- rbinom(n = n, size = 1, prob = 0.3)
  Z2 <- rbinom(n = n, size = 1, prob = plogis(0 + 0.2*Z1))
  A  <- rbinom(n = n, size = 1, prob = plogis(-1 + 0.7*Z1 + 0.8*Z2))
  Y  <- rbinom(n = n, size = 1, prob = plogis(-0.5 + 1*Z1 + 0.7*Z2 + 1.3*A))
  return(data.frame(Z1, Z2, A, Y))
}

# Simulate a data set
set.seed(456)
simple_df <- simulate_simple(n = 5000)

# Package data for Stan
stan_dat <- list(N = nrow(simple_df),
                 P = 3,
                 X = cbind(1, simple_df$Z1, simple_df$Z2),
                 A = simple_df$A,
                 Y = simple_df$Y)
```

The frequentist point estimate of the ATE in this data set would be 0.27.

```
# Calculate frequentist ATE
ffit1 <- glm(Y ~ 1 + Z1 + Z2 + A, family = binomial(link = "logit"), data = simple_df)
fcoef <- coef(ffit1)
```

```r
fATE <- mean(plogis(cbind(1, simple_df$Z1, simple_df$Z2, 1) %*% fcoef) -
             plogis(cbind(1, simple_df$Z1, simple_df$Z2, 0) %*% fcoef))
print(fATE)
```

```
## [1] 0.2701615
```

A demonstration of the Bayesian g-formula in Stan

The Stan code to obtain the ATE is shown below. One could adopt Stan's default flat priors for the elements of $\boldsymbol{\theta} = (\boldsymbol{\alpha}, \alpha_A)$. Instead, we choose to place greater prior mass on the range of plausible values for coefficients on the log-odds scale. For binary covariates, independent $\mathcal{N}(0, 2.5)$ priors on all coefficients effectively rule out implausibly strong effects (e.g., log-odds ratios of 3, which would correspond to the enormous odds ratio of e^3 or ≈ 20). With non-rare outcomes, the same prior can be chosen for the intercept of the regression model since log-odds of ± 3 correspond to reference level event probabilities of ≈ 0.05 or 0.95.

```
data {
  // number of observations
  int<lower=0> N;
  // number of columns in design matrix excluding A
  int<lower=0> P;
  // design matrix, excluding treatment A
  matrix[N, P] X;
  // observed treatment
  vector[N] A;
  // outcome
  int<lower=0,upper=1> Y[N];
}

transformed data {
  // make vector of 1/N for (classical) bootstrapping
  real one = 1;
  vector[N] boot_probs = rep_vector(one/N, N);
}

parameters {
  // regression coefficients
  vector[P + 1] alpha;
}

transformed parameters {
  vector[P] alphaZ = head(alpha, P);
  real alphaA = alpha[P + 1];
}

model {
  // priors for regression coefficients
  for (p in 1:(P + 1)) {
    alpha[p] ~ normal(0, 2.5);
  }

  // likelihood
  Y ~ bernoulli_logit(X * alphaZ + A * alphaA);
}
```

```
generated quantities {
  // weights for the bootstrap
  int<lower=0> counts[N] = multinomial_rng(boot_probs, N);

  // calculate ATE in the bootstrapped sample
  real ATE = 0;
  vector[N] Y_a1;
  vector[N] Y_a0;
  for (n in 1:N) {
    // sample Ya where a = 1 and a = 0
    Y_a1[n] = bernoulli_logit_rng(X[n] * alphaZ + alphaA);
    Y_a0[n] = bernoulli_logit_rng(X[n] * alphaZ);

    // add contribution of this observation to the bootstrapped ATE
    ATE = ATE + (counts[n] * (Y_a1[n] - Y_a0[n]))/N;
  }
}
```

```
# Fit model
suppressPackageStartupMessages(library("rstan"))
rstan_options(auto_write = TRUE, mc.cores = parallel::detectCores())
fit1 <- stan(file = "simple_mc.stan", data = stan_dat)
```

```
## trying deprecated constructor; please alert package maintainer
```

```
##
## SAMPLING FOR MODEL 'simple_mc' NOW (CHAIN 1).
##
## Gradient evaluation took 0.001179 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 11.79 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 6.61749 seconds (Warm-up)
##                6.21123 seconds (Sampling)
##                12.8287 seconds (Total)
##
##
## SAMPLING FOR MODEL 'simple_mc' NOW (CHAIN 2).
##
## Gradient evaluation took 0.000642 seconds
```

```
## 1000 transitions using 10 leapfrog steps per transition would take 6.42 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 7.14205 seconds (Warm-up)
##                6.15781 seconds (Sampling)
##                13.2999 seconds (Total)
##
##
## SAMPLING FOR MODEL 'simple_mc' NOW (CHAIN 3).
##
## Gradient evaluation took 0.000494 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 4.94 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 7.15839 seconds (Warm-up)
##                6.55694 seconds (Sampling)
##                13.7153 seconds (Total)
##
##
## SAMPLING FOR MODEL 'simple_mc' NOW (CHAIN 4).
##
## Gradient evaluation took 0.000497 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 4.97 seconds.
## Adjust your expectations accordingly!
##
##
```

```
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 7.04265 seconds (Warm-up)
##                7.88858 seconds (Sampling)
##                14.9312 seconds (Total)
```

```
# Regression coefficients not very different from frequentist model
rbind(coef(ffit1), summary(fit1, pars = "alpha")[["summary"]][,"mean"])
```

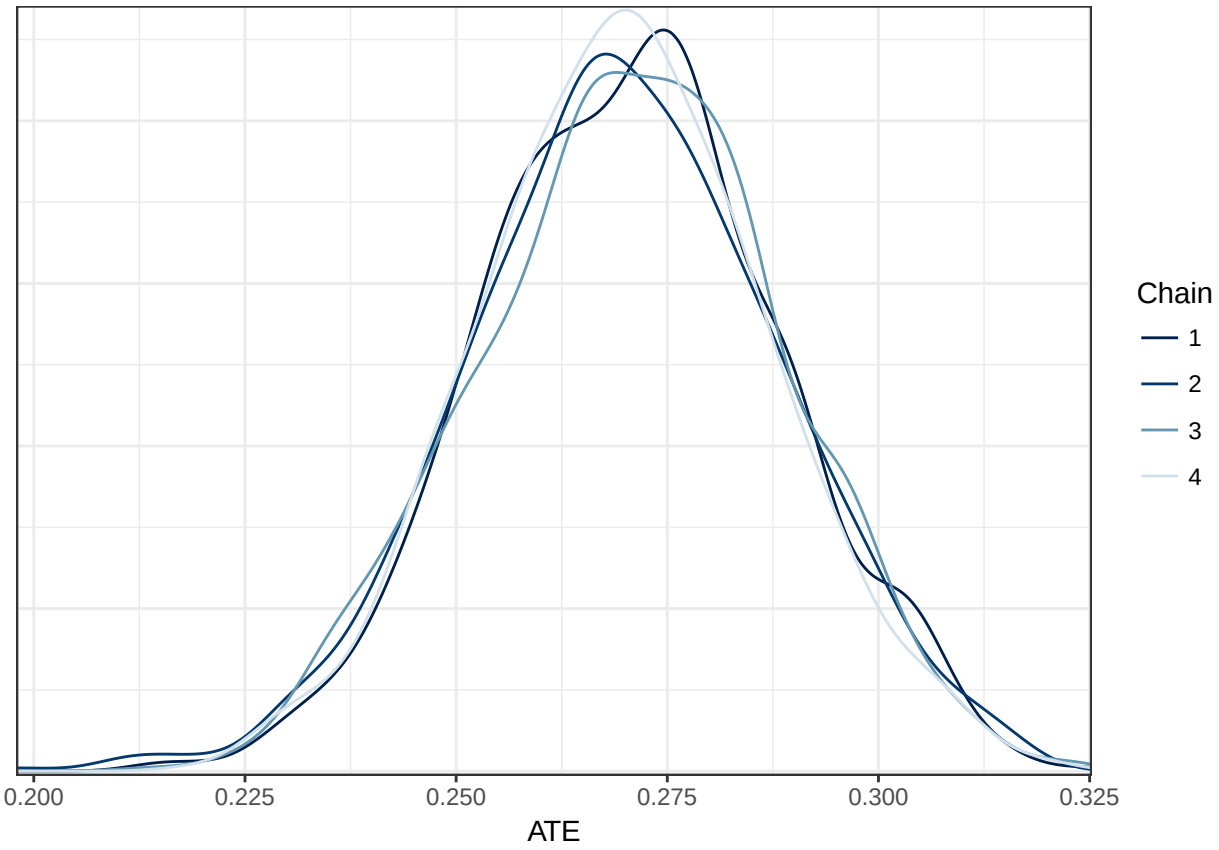
```
##      (Intercept)       Z1        Z2        A
## [1,]  -0.5773249 1.027699 0.7624320 1.358859
## [2,]  -0.5750731 1.025909 0.7611783 1.359530
```

```
# Posterior summary of ATE
summary(fit1, pars = "ATE")[["summary"]]
```

```
##         mean      se_mean         sd     2.5%    25%    50%    75%
## ATE 0.270241 0.0002969695 0.01771151 0.235395 0.2582 0.2704 0.2822
##        97.5%    n_eff      Rhat
## ATE 0.304605 3557.032 0.9998491
```

```
# Posterior mean
bATE <- summary(fit1, pars = "ATE")[["summary"]][,"mean"]

# Posterior density of ATE by chain
library("bayesplot")
library("ggplot2")
theme_set(theme_bw())
mcmc_dens_overlay(As.mcmc.list(fit1), pars = "ATE")
```

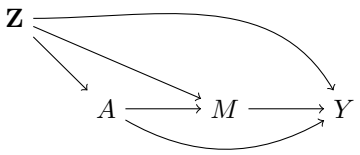


The posterior mean ATE is 0.270241, which is compatible with the frequentist estimate of 0.2701615.

Another demonstration: the g-formula applied to mediation

Mediators are intermediate variables on the causal path between the exposure and the outcome. Vanderweele provides a comprehensive overview of mediation analysis in his book (VanderWeele 2015). Figure 3 shows a basic causal diagram with an exposure A, a mediator M, outcome Y, and baseline confounders $\mathbf{Z}$.

Figure 3: Basic mediation model with exposure-mediator and exposure-outcome confounders



One potential question in causal inference is the degree to which an effect would remain if we somehow intervened upon part of the downstream causal pathway. This estimand is typically referred to as a natural direct effect (NDE). We simulate data from this scenario below using logistic link models for binary M and Y, and two binary baseline confounders $\mathbf{Z} = (Z_1, Z_2)$:

```
# Function to simulate data for mediation
simulate_simple_med <- function(n) {
  Z1 <- rbinom(n = n, size = 1, prob = 0.3)
  Z2 <- rbinom(n = n, size = 1, prob = plogis(0 + 0.2*Z1))
  A  <- rbinom(n = n, size = 1, prob = plogis(-1 + 0.7*Z1 + 0.8*Z2))
```

```
  M  <- rbinom(n = n, size = 1, prob = plogis(-2 + 0.3*Z1 + 0.5*Z2 + 1*A))
  Y  <- rbinom(n = n, size = 1, prob = plogis(-2 + 0.4*Z1 + 0.3*Z2 + 0.9*A + 0.9*M))
  return(data.frame(Z1, Z2, A, M, Y))
}

med_df <- simulate_simple_med(n = 5000)

# Package data for Stan
# Last 6 arguments specify weakly informative priors on coefficients
med_dat <- list(N = nrow(med_df),
                P = 3,
                X = cbind(1, med_df$Z1, med_df$Z2),
                A = med_df$A,
                M = med_df$M,
                Y = med_df$Y,
                alpha_m = rep(0, 5),
                beta_m = rep(0, 4),
                alpha_vcv = 2.5 * diag(5),
                beta_vcv = 2.5 * diag(4))
```

The mediator gets added to the outcome model:

$$\text{logit}\left(P(Y_i = 1|A_i, M_i, \mathbf{Z}_i)\right) = \alpha_0 + \boldsymbol{\alpha}_Z' \mathbf{Z}_i + \alpha_M M_i + \alpha_A A_i$$

In addition to the Y model, we also adopt a logistic model for M.

$$\text{logit}\left(P(M_i = 1|A_i, \mathbf{Z}_i)\right) = \beta_0 + \boldsymbol{\beta}_Z' \mathbf{Z}_i + \beta_A A_i$$

These two models can be estimated simultaneously with Stan. Using the resampling of $\mathbf{Z}$ as described earlier, we can draw samples from the distributions of the counterfactuals M_a for $a \in \{0, 1\}$. At the b^{th} MCMC iteration and for $i = 1, \ldots, n$,

$$M_a^{(i,b)} \sim \text{Bernoulli}\left(\text{logit}^{-1}\left(\beta_0^{(b)} + \boldsymbol{\beta}_Z^{(b)} \mathbf{Z}^{(i,b)} + \beta_A a\right)\right)$$

The outcome counterfactuals Y_{aM_a} and $Y_{aM_{a^*}}$ represent the potential outcome values under regime $A = a$ when the mediator M is set to the value it would naturally take under either a or a^*. For example, we may be interested in the hypothetical outcomes if the population were exposed (i.e., all $A = 1$) while the path through M is somehow disabled (i.e., $M = M_{a=0}$). This contrast is sometimes referred to as a natural direct effect because it captures the effect of A on Y that is "direct" – that is, the effect *not* through the mediator.

$$M_a^{(i,b)} \sim \text{Bernoulli}\left(\text{logit}^{-1}\left(\beta_0^{(b)} + \boldsymbol{\beta}_Z^{(b)} \mathbf{Z}^{(i,b)} + \beta_A a\right)\right)$$

Stan code to fit these models is shown below.

```
data {
  // number of observations
  int<lower=0> N;
  // number of columns in design matrix excluding A (and M)
  int<lower=0> P;
  // design matrix, excluding treatment A
  matrix[N, P] X;
  // observed treatment
  vector[N] A;
```

```
  // observed mediator
  int<lower=0,upper=1> M[N];
  // outcome
  int<lower=0,upper=1> Y[N];
  // mean of regression priors
  vector[P + 2] alpha_m;
  vector[P + 1] beta_m;
  // variance-covariance of regression priors
  cov_matrix[P + 2] alpha_vcv;
  cov_matrix[P + 1] beta_vcv;
}

transformed data {
  // make vector of 1/N for (classical) bootstrapping
  real one = 1;
  vector[N] boot_probs = rep_vector(one/N, N);

  // make vector version of M
  vector[N] Mv = to_vector(M);
}

parameters {
  // regression coefficients (outcome model)
  vector[P + 2] alpha;

  // regression coefficients (mediator model)
  vector[P + 1] beta;
}

transformed parameters {
  // partial M coefficient parameters
  vector[P] betaZ = head(beta, P);
  real betaA = beta[P + 1];

  // partial Y coefficient parameters
  vector[P] alphaZ = head(alpha, P);
  real alphaA = alpha[P + 1];
  real alphaM = alpha[P + 2];
}

model {
  // priors on causal coefficients weakly informative for binary exposure
  alpha ~ multi_normal(alpha_m, alpha_vcv);
  beta ~ multi_normal(beta_m, beta_vcv);

  // likelihoods
  M ~ bernoulli_logit(X * betaZ + A * betaA);
  Y ~ bernoulli_logit(X * alphaZ + A * alphaA + Mv * alphaM);
}

generated quantities {
  // weights for the bootstrap
  int<lower=0> counts[N] = multinomial_rng(boot_probs, N);
```

```
  // calculate NDE in the bootstrapped sample
  real NDE = 0;
  vector[N] M_a0;
  vector[N] Y_a1Ma0;
  vector[N] Y_a0Ma0;
  for (n in 1:N) {
    // sample Ma where a = 0
    M_a0[n] = bernoulli_logit_rng(X[n] * betaZ);

    // sample Y_(a=1, M=M_0) and Y_(a=0, M=M_0)
    Y_a1Ma0[n] = bernoulli_logit_rng(X[n] * alphaZ + M_a0[n] * alphaM + alphaA);
    Y_a0Ma0[n] = bernoulli_logit_rng(X[n] * alphaZ + M_a0[n] * alphaM);

    // add contribution of this observation to the bootstrapped NDE
    NDE = NDE + (counts[n] * (Y_a1Ma0[n] - Y_a0Ma0[n]))/N;
  }
}
```

```
library("rstan")
library("bayesplot")
fit2 <- stan(file = "mediation_mc.stan", data = med_dat)
```

```
## trying deprecated constructor; please alert package maintainer
```

```
##
## SAMPLING FOR MODEL 'mediation_mc' NOW (CHAIN 1).
##
## Gradient evaluation took 0.001815 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 18.15 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 21.6369 seconds (Warm-up)
##                22.5983 seconds (Sampling)
##                44.2352 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mediation_mc' NOW (CHAIN 2).
##
## Gradient evaluation took 0.001216 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 12.16 seconds.
```

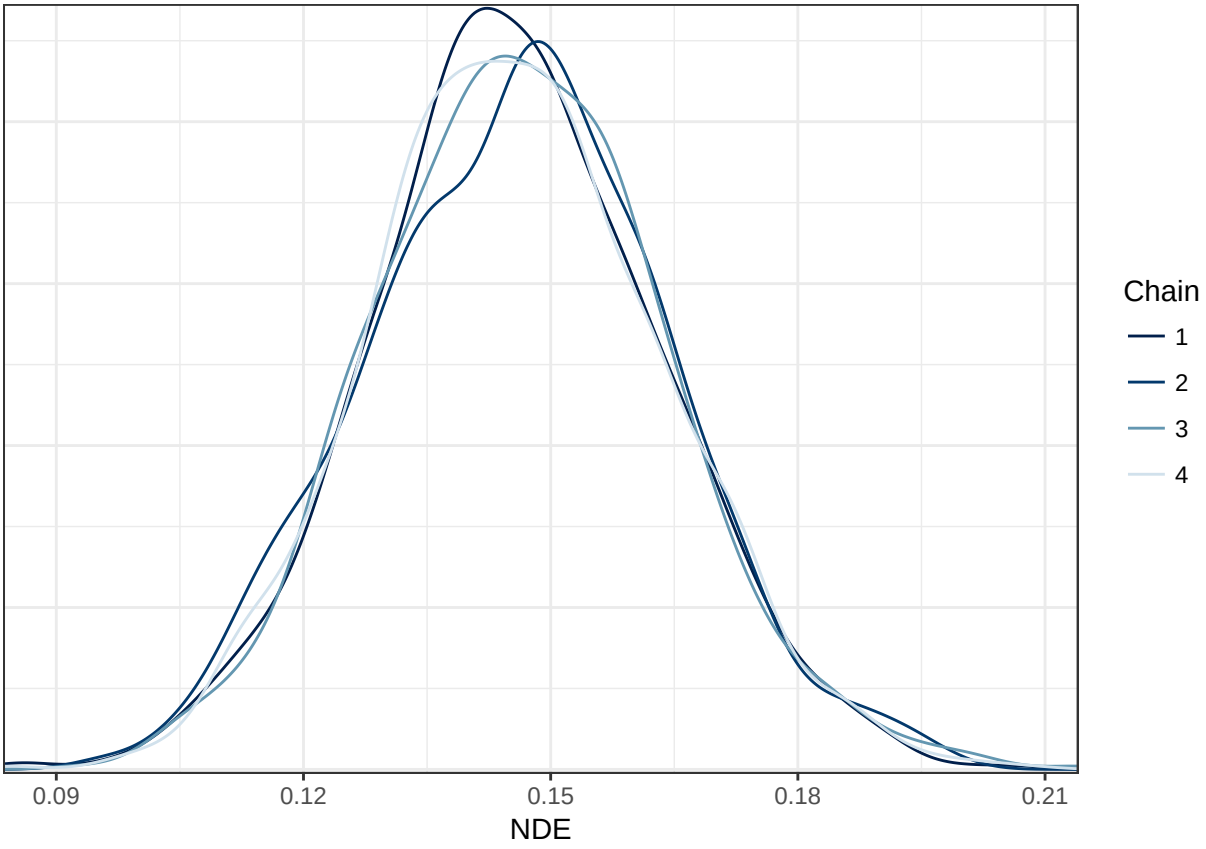
```
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 21.2848 seconds (Warm-up)
##                23.4931 seconds (Sampling)
##                44.7779 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mediation_mc' NOW (CHAIN 3).
##
## Gradient evaluation took 0.001534 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 15.34 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 22.4109 seconds (Warm-up)
##                24.1178 seconds (Sampling)
##                46.5287 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mediation_mc' NOW (CHAIN 4).
##
## Gradient evaluation took 0.001508 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 15.08 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
```

```
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 19.1758 seconds (Warm-up)
##                22.0621 seconds (Sampling)
##                41.2379 seconds (Total)
```

```
# Posterior summary of NDE
summary(extract(fit2)[["NDE"]])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0836  0.1340  0.1458  0.1460  0.1578  0.2140
```

```
# Posterior density for NDE
mcmc_dens_overlay(As.mcmc.list(fit2), pars = "NDE")
```

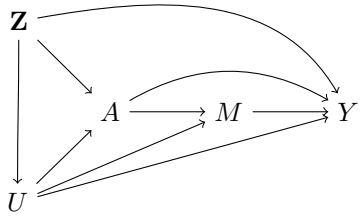


Application: data integration in unmeasured confounding

While the above models demonstrate application of the parametric g-formula for mediation in a Bayesian framework, they offer little advantage over existing frequentist methods. However, this need not be true. Suppose that one variable of $\mathbf{Z}$ is not measured in the analysis data set but is measured in a smaller, secondary data source. Denoting this variable with U, we can fit maximum likelihood regression models for U, M, and Y in the secondary data set. The point estimates and variance-covariance matrices from the maximum likelihood fits can serve as the mean and variance of multivariate normal priors.

We can simulate data using the same `R` function as the previous mediation example, renaming the simulated confounder Z_2 as U because it is unmeasured. (In our simulated data, $\mathbf{Z}$ will no longer be a vector of covariates, but we keep the more general notation since $\mathbf{Z}$ may be vector-valued.) The specific causal structure posited is shown in Figure 4. Importantly, we are not addressing the special case of exposure-induced mediator-outcome confounding, i.e., where U is caused by A.

Figure 4: Mediation with one type of unmeasured confounder



We add a model for U and add a term for U in the M and Y regressions, giving model equations:

$$\text{logit}\left(P(U_i = 1|\mathbf{Z}_i, A_i)\right) = \gamma_0 + \boldsymbol{\gamma}_Z'\mathbf{Z}_i$$

$$\text{logit}\left(P(M_i = 1|A_i, M_i, U_i, \mathbf{Z}_i)\right) = \beta_0 + \boldsymbol{\beta}_Z'\mathbf{Z}_i + \beta_U U_i + \beta_A A_i$$

$$\text{logit}\left(P(Y_i = 1|A_i, M_i, U_i, \mathbf{Z}_i)\right) = \alpha_0 + \boldsymbol{\alpha}_Z'\mathbf{Z}_i + \alpha_U U_i + \alpha_A A_i + \alpha_M M_i$$

Let $\boldsymbol{\alpha} = (\alpha_0, \boldsymbol{\alpha}_Z, \alpha_U, \alpha_A, \alpha_M)'$, $\boldsymbol{\beta} = (\beta_0, \boldsymbol{\beta}_Z, \beta_U, \beta_A)'$, and $\boldsymbol{\gamma} = (\gamma_0, \boldsymbol{\gamma}_Z)'$. Denote the full regression coefficient parameter vector $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$ by $\boldsymbol{\theta}$. Had U been observed, the joint data likelihood for n observations would be

$$\prod_{i=1}^{n} f(y_i|\boldsymbol{\alpha}, \mathbf{z}_i, a_i, m_i, u_i) f(m_i|\boldsymbol{\beta}, \mathbf{z}_i, a_i, u_i) f(u_i|\boldsymbol{\gamma}, \mathbf{z}_i)$$

Since U is binary, this can be rewritten as

$$\prod_{i=1}^{n} f(y_i|\boldsymbol{\alpha}, \mathbf{z}_i, a_i, m_i, u_i) f(m_i|\boldsymbol{\beta}, \mathbf{z}_i, a_i, u_i) P(U_i = u_i|\boldsymbol{\gamma}, \mathbf{z}_i)$$

To marginalize over the unobserved U_i, we sum over $u = 0$ and $u = 1$ to obtain:

$$\prod_{i=1}^{n} \left[\sum_{u=0}^{1} f(y_i|\boldsymbol{\alpha}, \mathbf{z}_i, a_i, m_i, u_i = u) f(m_i|\boldsymbol{\beta}, \mathbf{z}_i, a_i, u_i = u) P(U_i = u|\boldsymbol{\gamma}, \mathbf{z}_i) \right]$$

With binary M and Y, this becomes

$$\prod_{i=1}^{n} \left[\sum_{u=0}^{1} \left(\pi_i^Y(u)\right)^{y_i} \left(1 - \pi_i^Y(u)\right)^{1-y_i} \left(\pi_i^M(u)\right)^{m_i} \left(1 - \pi_i^M(u)\right)^{1-m_i} P(U_i = u|\boldsymbol{\gamma}, \mathbf{z}_i, a_i) \right]$$

where $\pi_i^Y(u) = P(Y_i = 1|\boldsymbol{\alpha}, \mathbf{Z}_i, A_i, M_i, U_i = u)$ and $\pi_i^M(u) = P(M_i = 1|\boldsymbol{\beta}, \mathbf{Z}_i, A_i, U_i = u)$.

Informative prior distributions are derived from the maximum likelihood point estimates and variance-covariance matrices from the secondary data source. For example, the prior for $\boldsymbol{\gamma} = (\gamma_0, \boldsymbol{\gamma}_Z)'$ would be

$$\boldsymbol{\gamma} \sim \mathcal{MVN}\left(\hat{\boldsymbol{\gamma}}_{MLE}, \widehat{\boldsymbol{\Sigma}}_{MLE}\right)$$

where $\hat{\boldsymbol{\gamma}}_{MLE}$ and $\widehat{\boldsymbol{\Sigma}}_{MLE}$ are the frequentist point estimate and variance-covariance matrix from the maximum likelihood estimation (MLE). Analogous priors are adopted for $\boldsymbol{\beta} = (\beta_0, \boldsymbol{\beta}_Z, \beta_U, \beta_A)'$ and $\boldsymbol{\alpha} = (\alpha_0, \boldsymbol{\alpha}_Z, \alpha_U, \alpha_A, \alpha_M)'$.

```
# Simulate small and big mediation data sets from same data generating parameters
small_df <- simulate_simple_med(n = 200)
big_df   <- simulate_simple_med(n = 5000)

# Rename Z2 to U (because it is unmeasured)
names(small_df)[names(small_df) == "Z2"] <- "U"
names(big_df)[names(big_df) == "Z2"] <- "U"

# Frequentist model fits for prior information
fitU <- glm(U ~ Z1, data = small_df)
fitM <- glm(M ~ Z1 + U + A, data = small_df)
fitY <- glm(Y ~ Z1 + U + A + M, data = small_df)

# Prior means for coefficients
gamma_m <- unname(coef(fitU))
beta_m  <- unname(coef(fitM))
alpha_m <- unname(coef(fitY))

# Prior variance-covariance matrices
gamma_vcv <- unname(vcov(fitU))
beta_vcv  <- unname(vcov(fitM))
alpha_vcv <- unname(vcov(fitY))

# Package data for Stan
medU_dat <- list(N = nrow(big_df),
                 P = 2,
                 X = cbind(1, big_df$Z1),
                 A = big_df$A,
                 M = big_df$M,
                 Y = big_df$Y,
                 alpha_m = alpha_m,
                 beta_m = beta_m,
                 gamma_m = gamma_m,
                 alpha_vcv = alpha_vcv,
                 beta_vcv = beta_vcv,
                 gamma_vcv = gamma_vcv)
```

The Stan code to fit these models is shown below.

```
data {
  // number of observations
  int<lower=0> N;
  // number of columns in design matrix excluding A
```

```
  int<lower=0> P;
  // design matrix, excluding A, M, U
  matrix[N, P] X;
  // observed treatment
  vector[N] A;
  // observed mediator
  int<lower=0,upper=1> M[N];
  // outcome
  int<lower=0,upper=1> Y[N];
  // mean of regression priors
  vector[P + 3] alpha_m;
  vector[P + 2] beta_m;
  vector[P] gamma_m;
  // variance-covariance of regression priors
  cov_matrix[P + 3] alpha_vcv;
  cov_matrix[P + 2] beta_vcv;
  cov_matrix[P] gamma_vcv;
}

transformed data {
  // vectors of ones and zeros
  vector[N] all_ones = rep_vector(1, N);
  vector[N] all_zeros = rep_vector(0, N);

  // make vector of 1/N for (classical) bootstrapping
  real one = 1;
  vector[N] boot_probs = rep_vector(one/N, N);

  // make vector version of M
  vector[N] Mv = to_vector(M);
}

parameters {
  // regression coefficients (confounder model)
  vector[P] gamma;

  // regression coefficients (outcome model)
  vector[P + 3] alpha;

  // regression coefficients (mediator model)
  vector[P + 2] beta;
}

transformed parameters {
  // P(U = 1) for mixture weights
  vector[N] pU1 = inv_logit(X * gamma);

  // partial M coefficient parameters
  vector[P] betaZ = head(beta, P);
  real betaU = beta[P + 1];
  real betaA = beta[P + 2];

  // partial Y coefficient parameters
```

```
  vector[P] alphaZ = head(alpha, P);
  real alphaU = alpha[P + 1];
  real alphaA = alpha[P + 2];
  real alphaM = alpha[P + 3];
}

model {
  // linear predictors
  // U regression
  vector[N] eta_u;
  // M regression, if U = 0
  vector[N] eta_mu0;
  // Y regression, if U = 0
  vector[N] eta_yu0;

  // log-likelihood contributions for U = 0 and U = 1 cases
  real ll_0;
  real ll_1;

  // calculate linear predictors for U = 0 case
  // will selectively add on betaU and alphaU as needed
  eta_u = X * gamma;
  eta_mu0 = X * betaZ + A * betaA;
  eta_yu0 = X * alphaZ + A * alphaA + Mv * alphaM;

  // informative priors
  alpha ~ multi_normal(alpha_m, alpha_vcv);
  beta  ~ multi_normal(beta_m, beta_vcv);
  gamma ~ multi_normal(gamma_m, gamma_vcv);

  // likelihood
  for (n in 1:N) {
    // contribution if U = 0
    ll_0 = log_inv_logit(eta_yu0[n]) * Y[n] +
           log1m_inv_logit(eta_yu0[n]) * (1 - Y[n]) +
           log_inv_logit(eta_mu0[n]) * M[n] +
           log1m_inv_logit(eta_mu0[n]) * (1 - M[n]) +
           log1m_inv_logit(eta_u[n]);

    // contribution if U = 1
    ll_1 = log_inv_logit(eta_yu0[n] + alphaU) * Y[n] +
           log1m_inv_logit(eta_yu0[n] + alphaU) * (1 - Y[n]) +
           log_inv_logit(eta_mu0[n] + betaU) * M[n] +
           log1m_inv_logit(eta_mu0[n] + betaU) * (1 - M[n]) +
           log_inv_logit(eta_u[n]);

    // contribution is summation over U possibilities
    target += log_sum_exp(ll_0, ll_1);
  }
}

generated quantities {
  // weights for the bootstrap
```

```
  int<lower=0> counts[N] = multinomial_rng(boot_probs, N);

  // calculate NDE in the bootstrapped sample
  real NDE = 0;
  vector[N] U;
  vector[N] M_a0;
  vector[N] Y_a1Ma0;
  vector[N] Y_a0Ma0;
  for (n in 1:N) {
    // sample U
    U[n] = bernoulli_logit_rng(pU1[n]);

    // sample M_a where a = 0
    M_a0[n] = bernoulli_logit_rng(X[n] * betaZ + U[n] * betaU);

    // sample Y_(a=0, M=M_0) and Y_(a=1, M=M_0)
    Y_a0Ma0[n] = bernoulli_logit_rng(X[n] * alphaZ + M_a0[n] * alphaM +
                                     U[n] * alphaU);
    Y_a1Ma0[n] = bernoulli_logit_rng(X[n] * alphaZ + M_a0[n] * alphaM + alphaA +
                                     U[n] * alphaU);

    // add contribution of this observation to the bootstrapped NDE
    NDE = NDE + (counts[n] * (Y_a1Ma0[n] - Y_a0Ma0[n]))/N;
  }
}
```

```
# Fit mediation model model with unmeasured confounder
fit3 <- stan(file = "mediation_unmeasured_mc.stan", data = medU_dat)
```

```
## trying deprecated constructor; please alert package maintainer
```

```
##
## SAMPLING FOR MODEL 'mediation_unmeasured_mc' NOW (CHAIN 1).
##
## Gradient evaluation took 0.009969 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 99.69 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 92.0842 seconds (Warm-up)
##                97.9077 seconds (Sampling)
##                189.992 seconds (Total)
```

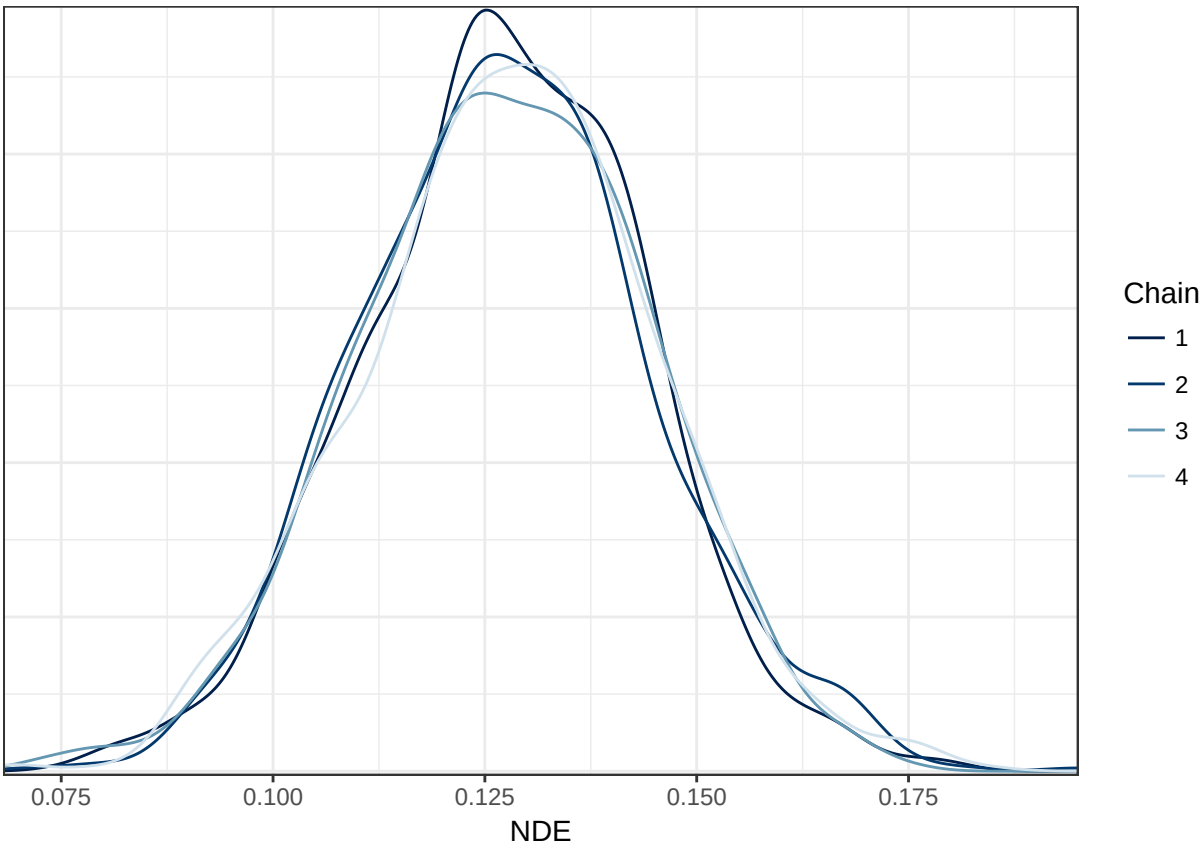
```
##
##
## SAMPLING FOR MODEL 'mediation_unmeasured_mc' NOW (CHAIN 2).
##
## Gradient evaluation took 0.005515 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 55.15 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 94.4244 seconds (Warm-up)
##                97.9423 seconds (Sampling)
##                192.367 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mediation_unmeasured_mc' NOW (CHAIN 3).
##
## Gradient evaluation took 0.00785 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 78.5 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 92.8346 seconds (Warm-up)
##                98.8082 seconds (Sampling)
##                191.643 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mediation_unmeasured_mc' NOW (CHAIN 4).
##
```

```
## Gradient evaluation took 0.009184 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 91.84 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 2000 [  0%]  (Warmup)
## Iteration:  200 / 2000 [ 10%]  (Warmup)
## Iteration:  400 / 2000 [ 20%]  (Warmup)
## Iteration:  600 / 2000 [ 30%]  (Warmup)
## Iteration:  800 / 2000 [ 40%]  (Warmup)
## Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Iteration: 2000 / 2000 [100%]  (Sampling)
##
##  Elapsed Time: 90.7697 seconds (Warm-up)
##                96.9376 seconds (Sampling)
##                187.707 seconds (Total)
```

```
# Posterior summary of NDE
summary(extract(fit3)[["NDE"]])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0682  0.1164  0.1276  0.1275  0.1388  0.1950
```

```
# Posterior density for NDE
mcmc_dens_overlay(As.mcmc.list(fit3), pars = "NDE")
```



Data integration: informative priors vs. combining data sets

The above analysis could have been performed two ways. The first is as a classical missing data problem. The analysis could proceed by combining the two datasets, taking the traditional Bayesian approach to missing data with the observations coming from the original main data set where U is unmeasured. The second approach is the one shown here, where information from the external data enters through informative priors on the regression coefficients.

The first approach might be preferable if the data sets are thought to be samples from the same underlying population, as with a validation data set. In that case, the supplemental data simply adds more information about the distribution of $\mathbf{Z}$. Combining the data sets also allows for non-normal priors, unlike with MLEs. However, there are two downsides to this approach. First, the missing data way requires access to the actual external data set. Obtaining the external data may not be possible due to data privacy reasons. Sharing the point estimates and variance-covariance matrices from a maximum likelihood model would not require data use agreements the same way that sharing data would. Secondly, the interpretation of the target population is cleaner without full data integration. If the underlying populations are thought to differ with respect to their confounder distributions, a combined data source has a mixture of those confounder distributions which depends on the number of observations in each data set, which is somewhat arbitrary. In this case, sampling baseline confounder vectors exclusively from the main data source is more interpretable.

Summary

Stan's `generated quantities` block offers a straightforward way to apply the g-formula to Bayesian models, including models for mediation. Incorporating prior information from another data source may partially

address problems due to unmeasured confounding and improve the communication of uncertainty to decision makers.

References

Imbens, G.W., and D.B. Rubin. 2015. *Causal Inference in Statistics, Social, and Biomedical Sciences*. Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction. Cambridge University Press.

Keil, Alexander P, Eric J Daza, Stephanie M Engel, Jessie P Buckley, and Jessie K Edwards. 2015. "A Bayesian Approach to the G-Formula." *Statistical Methods in Medical Research*. SAGE Publications Sage UK: London, England.

VanderWeele, T. 2015. *Explanation in Causal Inference: Methods for Mediation and Interaction*. Oxford University Press. https://books.google.com/books?id=K6cgBgAAQBAJ.