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

Leah Comment

9/7/2017 (updated 1/02/2018)

Causal inference

In the Rubin Causal Model, causal inference relies on contrasts of counterfactuals, also called potential outcomes. 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

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 $a \in \{0, 1\}$:

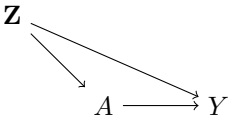
$$\mathbb{E}[Y_a] = \text{logit}^{-1}\left(\alpha_0 + \alpha_A a\right)$$

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 Z. In epidemiology, this standardization procedure is known as the g-formula. For discrete $\mathbf{Z}$, this formula can be written as a

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

In the Bayesian framework, 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)
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;
  }
}
```

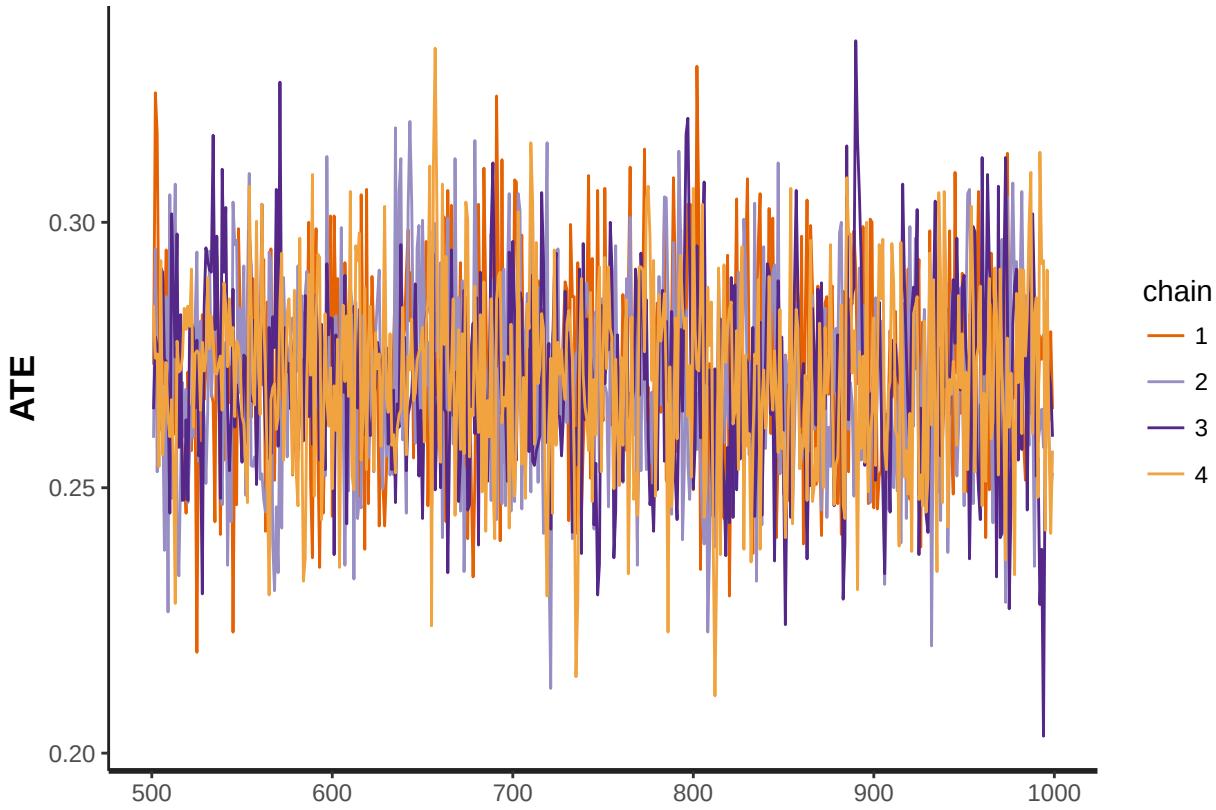
```
# Fit model
suppressPackageStartupMessages(library("rstan"))
rstan_options(auto_write = TRUE, mc.cores = parallel::detectCores())
fit1 <- stan(file = "simple_mc.stan", data = stan_dat, iter = 1000)
```

```
##
## SAMPLING FOR MODEL 'simple_mc' NOW (CHAIN 1).
##
## Gradient evaluation took 0.000777 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 7.77 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 3.79987 seconds (Warm-up)
##                3.07652 seconds (Sampling)
##                6.87639 seconds (Total)
##
##
## SAMPLING FOR MODEL 'simple_mc' NOW (CHAIN 2).
##
## Gradient evaluation took 0.00047 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 4.7 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
```

```
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 3.4946 seconds (Warm-up)
##                3.25374 seconds (Sampling)
##                6.74834 seconds (Total)
##
##
## SAMPLING FOR MODEL 'simple_mc' NOW (CHAIN 3).
##
## Gradient evaluation took 0.000477 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 4.77 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 3.38397 seconds (Warm-up)
##                3.48641 seconds (Sampling)
##                6.87038 seconds (Total)
##
##
## SAMPLING FOR MODEL 'simple_mc' NOW (CHAIN 4).
##
## Gradient evaluation took 0.00047 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 4.7 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 3.40528 seconds (Warm-up)
##                3.15963 seconds (Sampling)
```

```
##                 6.5649 seconds (Total)
```

```
# Traceplot of ATE
traceplot(fit1, pars=c("ATE"))
```




```
# Posterior summary of ATE
summary(extract(fit1)[["ATE"]])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2032  0.2584  0.2716  0.2710  0.2834  0.3342
```

```
# Posterior mean
bATE <- mean(extract(fit1)[["ATE"]])
```

The posterior mean ATE is 0.2710415, which is compatible with the frequentist estimate of 0.2701615.

TODO(LCOMM): marginalizing over a continuous Z instead of bootstrapping

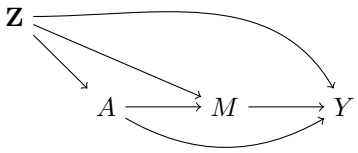
Another demonstration: the g-formula applied to mediation

Mediators are intermediate variables on the causal path between the exposure and the outcome. Figure 3 shows a causal diagram with an exposure A, a mediator M, outcome Y, and baseline confounders $\mathbf{Z}$.

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:

```
# Function to simulate data for mediation
simulate_simple_med <- function(n) {
  Z1 <- rbinom(n = n, size = 1, prob = 0.3)
```

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




```
  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;
  }
}
```

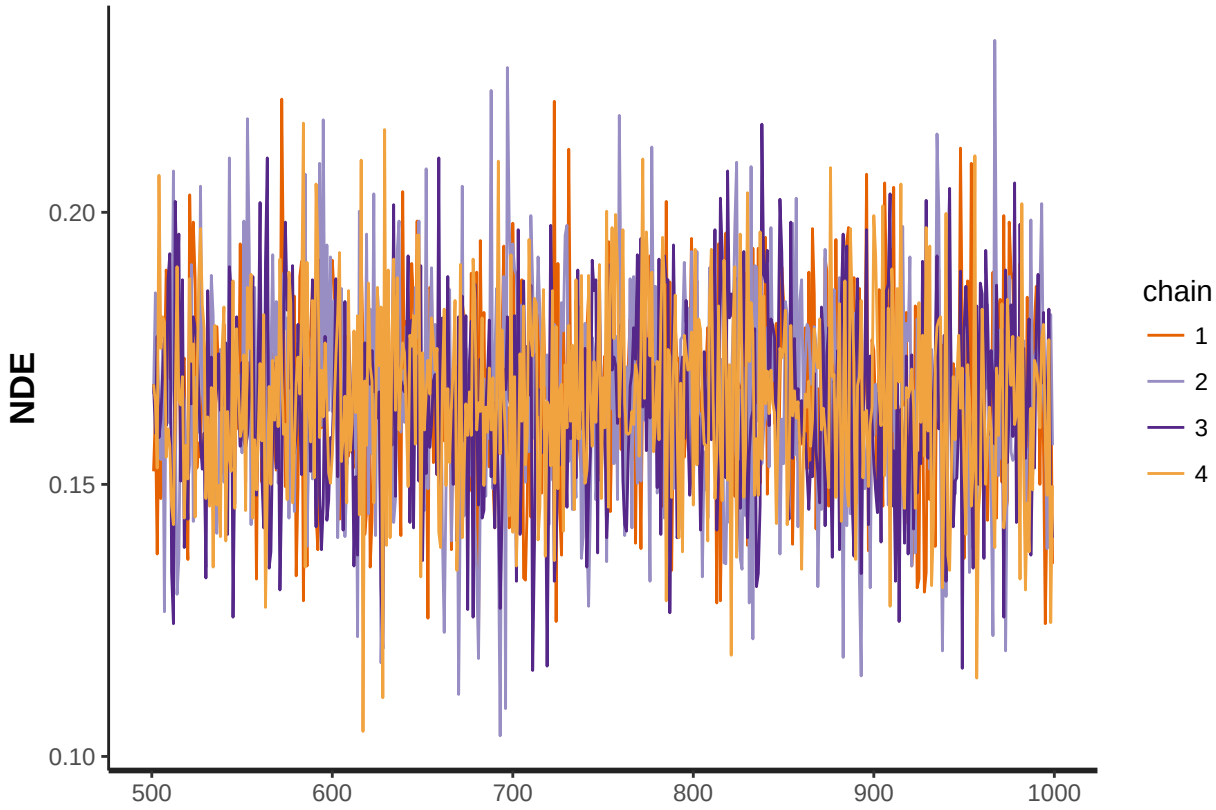
```
library("rstan")
fit2 <- stan(file = "mediation_mc.stan", data = med_dat, iter = 1000)
```

```
##
## SAMPLING FOR MODEL 'mediation_mc' NOW (CHAIN 1).
##
## Gradient evaluation took 0.001924 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 19.24 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 11.2402 seconds (Warm-up)
##                10.7638 seconds (Sampling)
##                22.004 seconds (Total)
```

```
##
##
## SAMPLING FOR MODEL 'mediation_mc' NOW (CHAIN 2).
##
## Gradient evaluation took 0.001192 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 11.92 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 11.3765 seconds (Warm-up)
##                10.1833 seconds (Sampling)
##                21.5598 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mediation_mc' NOW (CHAIN 3).
##
## Gradient evaluation took 0.00112 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 11.2 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 9.86717 seconds (Warm-up)
##                9.78958 seconds (Sampling)
##                19.6568 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mediation_mc' NOW (CHAIN 4).
##
```

```
## Gradient evaluation took 0.001274 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 12.74 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 10.7245 seconds (Warm-up)
##                10.8768 seconds (Sampling)
##                21.6013 seconds (Total)
```

```
# Traceplot of NDE
traceplot(fit2, pars=c("NDE"))
```




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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
##  0.1038  0.1536  0.1656  0.1658  0.1779  0.2316
```

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 function as the previous mediation example, renaming Z_2 to 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.) We also add a model for U, 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. Analogous priors are adopted for $\boldsymbol{\beta} = (\beta_0, \boldsymbol{\beta}_Z, \beta_U, \beta_A)'$ and $\boldsymbol{\alpha} = (\alpha_0, \boldsymbol{\alpha}_Z, \alpha_U, \alpha_A, \alpha_M)'$.

```r
# 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;
  }
}
```

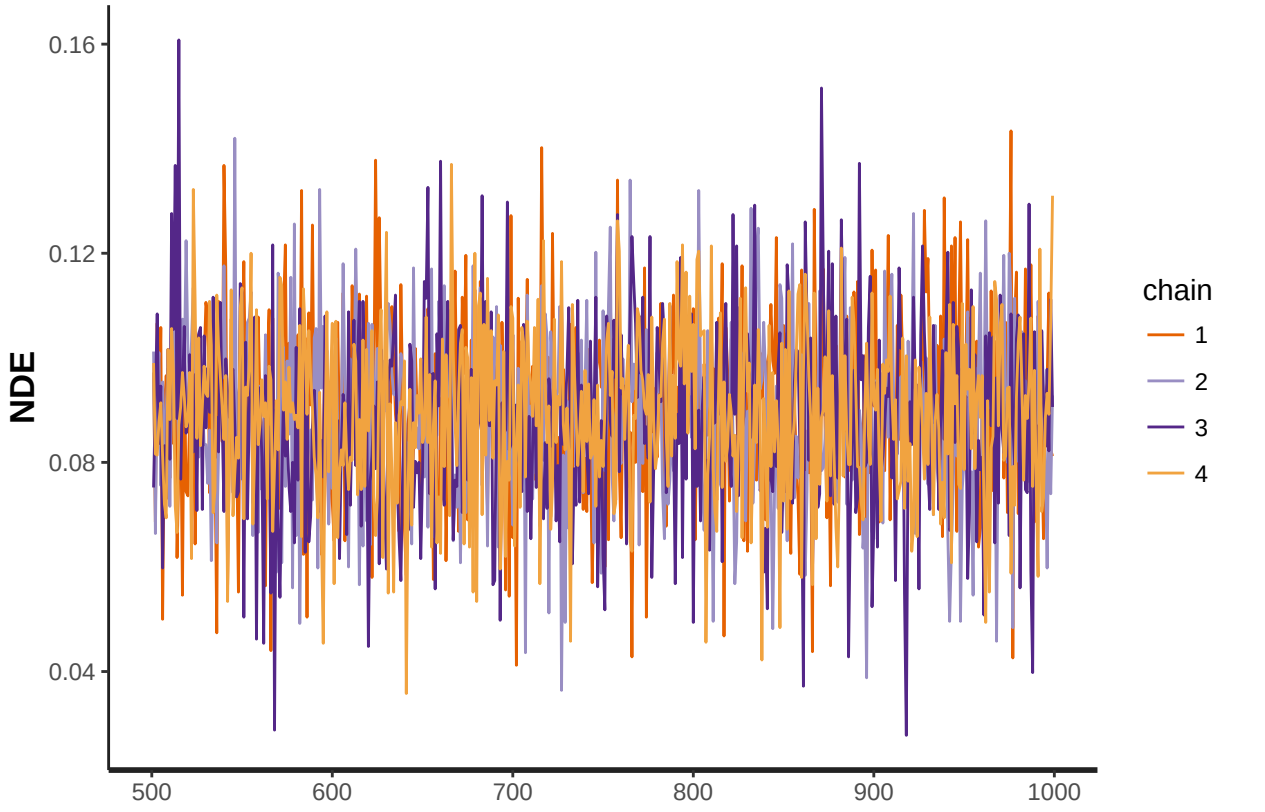
```
# Fit mediation model model with unmeasured confounder
fit3 <- stan(file = "mediation_unmeasured_mc.stan", data = medU_dat, iter = 1000)
```

```
##
## SAMPLING FOR MODEL 'mediation_unmeasured_mc' NOW (CHAIN 1).
##
## Gradient evaluation took 0.009688 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 96.88 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 49.4934 seconds (Warm-up)
##                48.9195 seconds (Sampling)
##                98.4129 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mediation_unmeasured_mc' NOW (CHAIN 2).
##
## Gradient evaluation took 0.005109 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 51.09 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
```

```
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 47.9601 seconds (Warm-up)
##                45.2688 seconds (Sampling)
##                93.2289 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mediation_unmeasured_mc' NOW (CHAIN 3).
##
## Gradient evaluation took 0.005615 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 56.15 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 47.7438 seconds (Warm-up)
##                45.1189 seconds (Sampling)
##                92.8627 seconds (Total)
##
##
## SAMPLING FOR MODEL 'mediation_unmeasured_mc' NOW (CHAIN 4).
##
## Gradient evaluation took 0.005135 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 51.35 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:   1 / 1000 [  0%]  (Warmup)
## Iteration: 100 / 1000 [ 10%]  (Warmup)
## Iteration: 200 / 1000 [ 20%]  (Warmup)
## Iteration: 300 / 1000 [ 30%]  (Warmup)
## Iteration: 400 / 1000 [ 40%]  (Warmup)
## Iteration: 500 / 1000 [ 50%]  (Warmup)
## Iteration: 501 / 1000 [ 50%]  (Sampling)
## Iteration: 600 / 1000 [ 60%]  (Sampling)
```

```
## Iteration: 700 / 1000 [ 70%]  (Sampling)
## Iteration: 800 / 1000 [ 80%]  (Sampling)
## Iteration: 900 / 1000 [ 90%]  (Sampling)
## Iteration: 1000 / 1000 [100%]  (Sampling)
##
##  Elapsed Time: 50.8624 seconds (Warm-up)
##                44.0352 seconds (Sampling)
##                94.8975 seconds (Total)
```

```
# Traceplot of NDE
traceplot(fit3, pars=c("NDE"))
```




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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.02780 0.07820 0.08940 0.08916 0.10045 0.16080
```

TODO(LCOMM): Add data privacy rationale for separation of stages.

TODO(LCOMM): Add part about extensions with combined data set to allow for non-conjugate priors (code up example?).

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.

TODO(LCOMM): Figure out where/how to add references.