Web appendix for "Causal mediation analysis with two mediators: A comprehensive guide to estimating total and natural effects across various multiple mediators setups"

Jesse Gervais[1], Geneviève Lefebvre[1], and Erica E.M. Moodie[2]

[1]Université du Québec à Montréal

[2]McGill University

Contents

Variable description

To illustrate the process of conducting a causal mediation analysis with two mediators, we used data from the International Dating Violence Study (Strauss, 2011) conducted between 2001 and 2006. The initial sample size consisted of $17,404$ participants, of whom $8,896$ were excluded because they had not been involved in a romantic relationship for at least one year, leaving a sample of $8,508$ individuals. An additional $2,284$ participants were excluded because they were not Canadian, American, or European, resulting in a final sample size of $n = 6,224$.

Childhood maltreatment was assessed using the following question: "When I was less than 12 years old, I was spanked or hit a lot by my mother or father." Responses ranged from 1 ("Strongly Disagree") to 4 ("Strongly Agree"). The variable has been recoded to vary from 0 to 3 when treating childhood maltreatment as a continuous variable. A binary indicator was created to indicate exposure to childhood maltreatment; the indicator was equal to 1 if participants answered "Agree" or "Strongly Agree", and 0 otherwise.

Post-traumatic stress symptoms were measured as the mean of eight items, such as "I've been terrified by things that have happened to me" and "I try not to think about terrible things that happened to me." Responses ranged from 1 ("Strongly Disagree") to 4 ("Strongly Agree"). The Cronbach's alpha based on our final sample was $\alpha = 0.74$.

Alcohol use was assessed using four items, including "I sometimes drink enough to feel really high or drunk" and "Sometimes I can't remember what happened the night before because of drinking." Responses ranged from 1 ("Strongly Disagree") to 4 ("Strongly Agree"). For instructional purposes, a binary indicator of problematic alcohol use was created. It was equal to 1 ("Problematic alcohol use") if participants answered "Strongly Agree" to any of these items, and 0 ("No problematic alcohol use") otherwise.

Religious involvement was measured as the mean of two items: "I attend a church, synagogue, or mosque once a month or more" and "I rarely have anything to do with religious activities". The answers ranged from 1 ("Strongly Disagree") to 4 ("Strongly

Agree"). The second item was reverse coded so that a higher score indicates higher religious involvement. The Cronbach's alpha based on our final sample was $\alpha = 0.80$.

Family violence approval was assessed as the mean of four items such as "I can think of a situation when I would approve of a husband slapping a wife's face" and "It is sometimes necessary for parents to slap a teen who talks back or is getting into trouble". The answers ranged from 1 ("Strongly Disagree") to 4 ("Strongly Agree"). The Cronbach's alpha based on our final sample was $\alpha = 0.65$.

Severe physical IPV perpetration was evaluated using seven questions asking about the frequency of events such as "I burned or scalded my partner on purpose" and "I used a knife or gun on my partner." Response options were: 1 ("Once in the past year"), 2 ("Twice in the past year"), 3 ("3-5 times in the past year"), 4 ("6-10 times in the past year"), 5 ("11-20 times in the past year"), 6 ("More than 20 times in the past year"), 7 ("Not in the past year, but it did happen before"), and 8 ("This has never happened"). A binary indicator was created to indicate severe physical IPV perpetration in past year. It was equal to 1 if participants answered 1 through 6 to at least one of the items, and 0 otherwise.

Scale of effects

In the paper, the conditional total and natural effects have been defined on the difference scale as

$$\begin{aligned} NIE^{x}(\boldsymbol{c}) &= \mathbb{E}[Y(x, M(x))|\boldsymbol{C} = \boldsymbol{c}] - \mathbb{E}[Y(x, M(x^*))|\boldsymbol{C} = \boldsymbol{c}], \\ NDE^{x^*}(\boldsymbol{c}) &= \mathbb{E}[Y(x, M(x^*))|\boldsymbol{C} = \boldsymbol{c}] - \mathbb{E}[Y(x^*, M(x^*))|\boldsymbol{C} = \boldsymbol{c}], \\ TE(\boldsymbol{c}) &= \mathbb{E}[Y(x)|\boldsymbol{C} = \boldsymbol{c}] - \mathbb{E}[Y(x^*)|\boldsymbol{C} = \boldsymbol{c}]. \end{aligned}$$

These quantities can also be defined as

$$\begin{aligned} OR^{NIE^{x}(\boldsymbol{c})} &= \frac{\mathbb{E}[Y(x, M(x))|\boldsymbol{C}=\boldsymbol{c}]}{1-\mathbb{E}[Y(x, M(x))|\boldsymbol{C}=\boldsymbol{c}]}\frac{1-\mathbb{E}[Y(x, M(x^*))|\boldsymbol{C}=\boldsymbol{c}]}{\mathbb{E}[Y(x, M(x^*))|\boldsymbol{C}=\boldsymbol{c}]}, \\ OR^{NDE^{x^*}(\boldsymbol{c})} &= \frac{\mathbb{E}[Y(x, M(x^*))|\boldsymbol{C}=\boldsymbol{c}]}{1-\mathbb{E}[Y(x, M(x^*))|\boldsymbol{C}=\boldsymbol{c}]}\frac{1-\mathbb{E}[Y(x^*, M(x^*))|\boldsymbol{C}=\boldsymbol{c}]}{\mathbb{E}[Y(x^*, M(x^*))|\boldsymbol{C}=\boldsymbol{c}]}, \\ OR^{TE(\boldsymbol{c})} &= \frac{\mathbb{E}[Y(x)|\boldsymbol{C}=\boldsymbol{c}]}{1-\mathbb{E}[Y(x)|\boldsymbol{C}=\boldsymbol{c}]}\frac{1-\mathbb{E}[Y(x^*)|\boldsymbol{C}=\boldsymbol{c}]}{\mathbb{E}[Y(x^*)|\boldsymbol{C}=\boldsymbol{c}]}, \end{aligned}$$

on the odds ratio (OR) scale as and

$$\begin{aligned} RR^{NIE^{x}(\boldsymbol{c})} &= \frac{\mathbb{E}[Y(x, M(x))|\boldsymbol{C}=\boldsymbol{c}]}{\mathbb{E}[Y(x, M(x^*))|\boldsymbol{C}=\boldsymbol{c}]}, \\ RR^{NDE^{x^*}(\boldsymbol{c})} &= \frac{\mathbb{E}[Y(x, M(x^*))|\boldsymbol{C}=\boldsymbol{c}]}{\mathbb{E}[Y(x^*, M(x^*))|\boldsymbol{C}=\boldsymbol{c}]}, \\ RR^{TE(\boldsymbol{c})} &= \frac{\mathbb{E}[Y(x)|\boldsymbol{C}=\boldsymbol{c}]}{\mathbb{E}[Y(x^*)|\boldsymbol{C}=\boldsymbol{c}]}, \end{aligned}$$

on the risk ratio (RR) scale (Samoilenko & Lefebvre, 2021). All the other conditional natural direct and indirect effects discussed in the paper can be likewise defined.

There are a few possible estimands for the marginal total and natural odds ratio and risk ratio. The first marginal estimand, targeted by the imputation, extended-imputation, and inverse probability weighted approaches, applies the definition of the odds ratio to the marginal counterfactual expected value (Lange et al., 2012, 2014; StataCorp, 2023); that is, marginalization is performed for the quantities in the numerator and denominator of the odds (risk) separately before taking the ratio. More precisely, we define the marginal total and natural effects as

$$\begin{aligned} OR^{NIE^{x}} &= \frac{\mathbb{E}[Y(x, M(x))]}{1-\mathbb{E}[Y(x, M(x))]}\frac{1-\mathbb{E}[Y(x, M(x^*))]}{\mathbb{E}[Y(x, M(x^*))]}, \\ OR^{NDE^{x^*}} &= \frac{\mathbb{E}[Y(x, M(x^*))]}{1-\mathbb{E}[Y(x, M(x^*))]}\frac{1-\mathbb{E}[Y(x^*, M(x^*))]}{\mathbb{E}[Y(x^*, M(x^*))]}, \\ OR^{TE} &= \frac{\mathbb{E}[Y(x)]}{1-\mathbb{E}[Y(x)]}\frac{1-\mathbb{E}[Y(x^*)]}{\mathbb{E}[Y(x^*)]}, \end{aligned}$$

on the odds ratio scale and as

$$\begin{aligned} RR^{NIE^x} &= \frac{\mathbb{E}[Y(x, M(x))]}{\mathbb{E}[Y(x, M(x^*))]}, \\ RR^{NDE^{x^*}} &= \frac{\mathbb{E}[Y(x, M(x^*))]}{\mathbb{E}[Y(x^*, M(x^*))]}, \\ RR^{TE} &= \frac{\mathbb{E}[Y(x)]}{\mathbb{E}[Y(x^*)]}. \end{aligned}$$

on the risk ratio scale. To the best of our understanding, the extended quasi-Bayesian Monte Carlo implemented in the package takes the alternative approach of marginalizing the odds (risk) ratio over the distribution of the mediators and the covariates $\boldsymbol{C}$ (for similar estimands, see Jun & Lee, 2023; Karlson et al., 2023), rather than marginalizing the components of the ratio as shown above.

Normal-based bootstrap confidence intervals

In this section, we show how to report normal-based rather than percentile-based bootstrap confidence intervals for total and natural effects.

Joint effect

In the main article, we provide `R` code in the **Application** subsection of the ***Imputation Approach*** section to compute joint natural indirect effect and natural direct effect in relation to two mediators using the imputation approach. We also show how to quantify estimate uncertainties using percentile-based bootstrap confidence intervals using the `medflex` package. This package also permits calculation of normal-based bootstrap confidence intervals using the option `type="norm"` (default) in the `confint` function.

```
expDat <- neImpute(y ~ factor(x)*m1*m2 + c1 + c2 + c3 + c4 + c5 +
        c6 + c7 + c8, family="binomial", nMed=2, data=dat)

fitCond <- neModel(y ~ x0*x1 + c1 + c2 + c3 + c4 + c5 +
        c6 + c7 + c8, family="binomial", expData=expDat,
        se="boot", nBoot=1000)
```

```
effectCond <- neEffdecomp(fitCond)

R <- exp(cbind(coef(effectCond), confint(effectCond, type="norm")))
colnames(R) <- c("OR", "95% LCL", "95% UCL")
R

##                              OR  95% LCL  95% UCL
## pure direct effect    1.595635 1.315847 1.927468
## total direct effect   1.508696 1.255230 1.814102
## pure indirect effect  1.219527 1.157864 1.284905
## total indirect effect 1.153081 1.068710 1.249860
## total effect          1.839896 1.540048 2.199781
```

Path-specific effects: causally dependent mediators

In the **Application** subsection of the ***Extended-imputation approach*** section of the main paper, we show how to compute natural direct effect and path-specific effects when the two mediators are causally dependent. We also provide `R` code to produce percentile-based bootstrap confidence intervals for conditional effects when the weights are computed based on M_2. The code that follows is identical to what is presented in the paper, but outputs normal-based rather than percentile-based bootstrap confidence intervals.

```
library(boot)
bootNEM <- function(data, index) {
bootDat <- data[index, ]

bootExpDat <- data.frame(replicate = rep(1:4, times = nrow(bootDat)),
        dat[rep(bootDat$id, each = 4),], x0 = NA, x1 = NA, x2 = NA)

bootExpDat2 <- within(bootExpDat, {
        x0 <- ifelse(replicate %in% c(2,4), x, 1-x)
        x1 <- x
        x2 <- ifelse(replicate %in% c(3,4), x, 1-x)
})
```

```
fitm2 <- glm(m2 ~ x*m1 + c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8,
        family="binomial", data=bootDat)

num2 <- with(bootExpDat2,
        dbinom(m2, 1, predict(fitm2, newdata=within(bootExpDat2, x <- x2),
        type="response")))

den2 <- with(bootExpDat2,
        dbinom(m2, 1, predict(fitm2, newdata=within(bootExpDat2, x <- x1),
        type="response")))

bootExpDat2$w2_c <- num2/den2

fity <- glm(y ~ x*m1*m2 + c1 + c2 + c3 + c4 + c5 +
                c6 + c7 + c8, family="binomial", data=bootDat)

bootExpDat2$y <- predict(fity, newdata=within(bootExpDat2, x <- x0),
        type="response")

fitNEM2_conditional  <- glm(y ~ x0*x1*x2 + c1 + c2 + c3 + c4 + c5 +
        c6 + c7 + c8, family="binomial", weight=w2_c, data=bootExpDat2)

NDE_m_c2    <- sum(coef(fitNEM2_conditional)[2])
NIE_m1y_c2  <- sum(coef(fitNEM2_conditional)[c(3,13,15,16)])
NIE_m2_y_c2 <- sum(coef(fitNEM2_conditional)[c(4,14)])
NIE_m_c2    <- NIE_m1y_c2 + NIE_m2_y_c2
TE_c2       <- NIE_m_c2 + NDE_m_c2

return(c(NDE_m_c2, NIE_m1y_c2, NIE_m2_y_c2, NIE_m_c2, TE_c2))
}

bootCI <- boot(data = dat, statistic = bootNEM, R = 1000)
R <- exp(t(sapply(1:5, function(x) c(bootCI$t0[x],
        boot.ci(bootCI, conf = 0.95, type = "norm", index = x)$norm[2:3]))))
colnames(R) = c("OR", "95 %LCL", "95 %UCL")
rownames(R) = c("NDE_m", "NIE_m1y", "NIE_m2_y", "NIE_m", "TE")
```

```
R

##                OR   95 %LCL  95 %UCL
## NDE_m    1.595640 1.3158477 1.927479
## NIE_m1y  1.154991 1.0737019 1.248231
## NIE_m2_y 0.998283 0.9806676 1.016158
## NIE_m    1.153008 1.0686547 1.249754
## TE       1.839785 1.5399439 2.199643
```

The difference in this code relative to that in the main paper is that the line of code `exp(t(sapply(1:5 function(x) c(bootCI$t0[x],boot.ci(bootCI, conf = 0.95, type = "perc", index = x)$perc[4:5]))))` is changed to `exp(t(sapply(1:5, function(x) c(bootCI$t0[x], boot.ci(bootCI, conf = 0.95, type = "norm", index = x)$norm[2:3]))))`.

Path-specific effects: independent mediators

In the **Application** subsection of the ***Inverse probability weighted approach*** section of the main article, we provide `R` code to implement the inverse probability weighted approach to conduct a mediation analysis when the mediators are independent. We also illustrate how percentile-based bootstrap confidence intervals can be computed in the conditional case. The following code replicates the example given in the paper but instead produces normal-based bootstrap confidence intervals by changing the line `exp(t(sapply(1:6, function(x) c(bootCI$t0[x], boot.ci(bootCI, conf = 0.95, type = "perc", index = x)$perc[4:5]))))` to `exp(t(sapply(1:6, function(x) c(bootCI$t0[x], boot.ci(bootCI, conf = 0.95, type = "norm", index = x)$norm[2:3]))))`.

```
library(boot)
bootNEM <- function(data, index) {
bootDat <- data[index, ]

bootExpDat <- data.frame(replicate = rep(1:4, times = nrow(bootDat)),
```

```
        dat[rep(bootDat$id, each = 4),], x0 = NA, x1 = NA, x2 = NA)

bootExpDat <- within(bootExpDat, {
        x0 <- x
        x1 <- ifelse(replicate %in% c(2,4), x, 1-x)
        x2 <- ifelse(replicate %in% c(3,4), x, 1-x)
})

fitm1 <- lm(m1 ~ x + c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8, data=bootDat)

num1 <- with(bootExpDat, dnorm(x=m1,
        mean=predict(fitm1, newdata=within(bootExpDat, x <- x1)),
        sd=summary(fitm1)$sigma))

den1 <- with(bootExpDat, dnorm(x=m1,
        mean=predict(fitm1, newdata=within(bootExpDat, x <- x0)),
        sd=summary(fitm1)$sigma))

w1   <- num1/den1

fitm2 <- lm(m2 ~ x + c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8, data=bootDat)

num2 <- with(bootExpDat, dnorm(x=m2,
        mean=predict(fitm2, newdata=within(bootExpDat, x <- x2)),
        sd=summary(fitm2)$sigma))

den2 <- with(bootExpDat, dnorm(x=m2,
        mean=predict(fitm2, newdata=within(bootExpDat, x <- x0)),
        sd=summary(fitm2)$sigma))

w2   <- num2/den2

bootExpDat$w_c <- w1*w2

fitNEM_conditional <- glm(y ~ x0*x1*x2 + c1 + c2 + c3 + c4 + c5 +
        c6 + c7 + c8, family="binomial", weight=w_c, data=bootExpDat)
```

```
NDE_m_c          <- sum(coef(fitNEM_conditional)[2])
NIE_m1_y_noint_c <- sum(coef(fitNEM_conditional)[c(3,13)])
NIE_m2_y_noint_c <- sum(coef(fitNEM_conditional)[c(4,14)])
MI_c             <- sum(coef(fitNEM_conditional)[c(15,16)])
NIE_m_c          <- NIE_m1_y_noint_c + NIE_m2_y_noint_c + MI_c
TE_c             <- NIE_m_c + NDE_m_c
return(c(NDE_m_c, NIE_m1_y_noint_c, NIE_m2_y_noint_c, MI_c, NIE_m_c, TE_c))
}

bootCI <- boot(data = dat, statistic = bootNEM, R = 1000)
R <- exp(t(sapply(1:6, function(x) c(bootCI$t0[x],
        boot.ci(bootCI, conf = 0.95, type = "norm", index = x)$norm[2:3]))))
colnames(R) = c("OR", "95 %LCL", "95 %UCL")
rownames(R) = c("NDE_m", "NIE_m1_y_noint", "NIE_m2_y_noint", "MI", "NIE_m",
        "TE")
R

##                       OR   95 %LCL   95 %UCL
## NDE_m          1.6859431 1.3873055 2.0487015
## NIE_m1_y_noint 1.1476278 1.0618011 1.2426827
## NIE_m2_y_noint 0.9454592 0.9067205 0.9847881
## MI             1.0114370 0.9932087 1.0299828
## NIE_m          1.0974449 1.0061858 1.1978762
## TE             1.8502296 1.5473562 2.2138624
```

Continuous exposure: R code

In this section, we replicate the four snippets of code presented in the main article to conduct causal mediation analyses using the bootstrapping procedure, but now treating the exposure as a continuous variable. More precisely, we reproduce the R code for the conditional joint effects (***Imputation approach***), conditional path-specific effects when the mediators are causally dependent and the weights are based on M_2 (***Extended-imputation approach***), conditional path-specific effects when the mediators are independent (***Inverse probability weighted approach***), and marginal path-specific

effects when the mediators are non-causally dependent (***Extended quasi-Bayesian Monte Carlo approach***). We assess the total and natural effects when the exposure is compared at level 0 versus 0.7 (close to the average).

Joint effect

The `medflex` package can readily accommodate a continuous exposure. The code for the `neImpute` and `neModel` functions is identical to that for a binary exposure. However, by default, the `medflex` package evaluates the total and natural effects assuming the exposure is compared at level 0 versus 1. To estimate the total and natural effects for exposure values 0 and 0.7, we can use the `neLht` function.

The main technical difference between the binary and continuous exposure cases lies in how the data set is extended. Recall that in the binary case, each individual is duplicated once, resulting in each individual being present twice in the newly created data set. In contrast, for the continuous exposure case, by the default, the `neImpute` function duplicates each individual four times, resulting in a total of five replications for each individual. The variable x^* represents the original value of the exposure, while x corresponds to five quantiles from a normal distribution characterized by the mean and standard deviation of the exposure conditional on the covariates. Referring to the inverse probability weighted approach, Lange et al. (2014) recommend estimating conditional, instead of marginal, total and natural effects, as the marginal estimators can be unstable when the exposure is continuous. Our view is that this recommendation should apply to imputation and extended-imputation as well.

```
expDat <- neImpute(y ~ x_cont*m1*m2 + c1 + c2 + c3 + c4 + c5 +
        c6 + c7 + c8, family="binomial", nMed=2, data=dat)

fit    <- neModel(y ~ x_cont0*x_cont1 + c1 + c2 + c3 + c4 + c5 +
        c6 + c7 + c8, family="binomial", expData = expDat,
        se="boot", nBoot=1000)
```

```
Eff   <- neLht(fit, linfct = c("0.7*x_cont0  = 0",
        "(x_cont1 + 0.7*x_cont0:x_cont1)*0.7 = 0",
        "0.7*x_cont0 + (x_cont1 + 0.7*x_cont0:x_cont1)*0.7 = 0"))

R     <- exp(cbind(coef(Eff), confint(Eff, type="perc")))
rownames(R) <- c("NDE", "NIE", "TE")
colnames(R) <- c("OR", "95 %LCL", "95 %UCL")

R

##           OR  95 %LCL  95 %UCL
## NDE 1.228289 1.155281 1.303017
## NIE 1.072419 1.054703 1.090841
## TE  1.317241 1.238993 1.397943
```

Extended-imputation approach: weights based on M_2

In this section, we replicate the extended-imputation approach with weights calculated based on M_2 for a continuous exposure. The code is similar to that presented in the paper for a binary exposure, but with two major differences. First, the data set extension must be adapted to account for the continuous nature of the exposure. More precisely, if the weights are calculated based on M_2, x^* corresponds to the original value of the exposure. For x and x^{**} separately, the values alternate among J options: one is the original value of the exposure, and the remaining $J-1$ are quantiles from a normal distribution with parameters set to the mean and standard deviation of the exposure conditional on the covariates. For x, the sequence of J options is repeated J times, while each value of the J options is repeated J times for x^{**}. As a result, each row is now present a total of J^2 times in the newly created data set. According to Steen et al. (2017), J should be at least three, and little is gained when J exceeds five. Additionally, the quantiles should be equally spaced and large enough to cover the distribution range effectively (Steen et al., 2017). In this example, we choose $J = 3$, with the quantiles 0.1 and 0.9. The second difference is that the formula for the total and natural effects must be

adapted to evaluate these effects when the exposure is contrasted at the level 0 versus 0.7.

```
bootNEM <- function(data, index) {
bootDat <- data[index, ]

bootDat$tempID <- 1:nrow(bootDat)

fitX <- lm(x_cont ~ c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8,
        data=bootDat)

J <- 3

q <- cbind(sapply(c(0.1, 0.9), FUN = qnorm,
        mean = predict(fitX, type = "response"),
        sd = summary(fitX)$sigma), bootDat$x_cont)

bootExpDat <- bootDat[rep(bootDat$tempID, each = J^2), ]
bootExpDat <- bootExpDat[order(bootExpDat$tempID),]
bootExpDat$x0 <- c(apply(q, 1, rep, J))
bootExpDat$x1 <- bootExpDat$x_cont
bootExpDat$x2 <- c(apply(q, 1, rep, each=J))

fitm2 <- glm(m2 ~ x_cont*m1 + c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8,
        family="binomial", data=bootDat)

num2 <- with(bootExpDat,
        dbinom(m2, 1, predict(fitm2,
        newdata=within(bootExpDat, x_cont <- x2),
        type="response")))

den2 <- with(bootExpDat,
        dbinom(m2, 1, predict(fitm2,
        newdata=within(bootExpDat, x_cont <- x1),
        type="response")))

bootExpDat$w <- num2/den2
```

```
fity <- glm(y ~ x_cont*m1*m2 + c1 + c2 + c3 + c4 + c5 +
        c6 + c7 + c8, family="binomial", data=bootDat)

bootExpDat$y <- predict(fity,
        newdata=within(bootExpDat, x_cont <- x0),
        type="response")

fitNEM <- glm(y ~ x0*x1*x2 + c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8,
        family="binomial", weight=w, data=bootExpDat)

mycoef <- coef(fitNEM)

NDE_m    <- 0.7*mycoef[2]
NIE_m1y  <- 0.7*(mycoef[3] + 0.7*(mycoef[13] + mycoef[15] + 0.7*mycoef[16]))
NIE_m2_y <- 0.7*(mycoef[4] + 0.7*mycoef[14])
NIE_m    <- NIE_m1y + NIE_m2_y
TE       <- NIE_m + NDE_m

return(c(NDE_m, NIE_m1y, NIE_m2_y, NIE_m, TE))
}

bootCI <- boot(data = dat, statistic = bootNEM, R = 1000)
R = exp(t(sapply(1:5, function(x) c(bootCI$t0[x],
        boot.ci(bootCI, conf = 0.95, type = "perc", index = x)$perc[4:5]))))
colnames(R) <- c("OR", "95 %LCL", "95 %UCL")
R

##             OR   95 %LCL  95 %UCL
## [1,] 1.2285228 1.1546880 1.301850
## [2,] 1.0739944 1.0571940 1.093312
## [3,] 0.9996901 0.9947725 1.003331
## [4,] 1.0736616 1.0556639 1.093574
## [5,] 1.3190177 1.2425711 1.398906
```

Inverse probability weighted approach

Lange et al. (2014) proposed a procedure to estimate total and natural effects with multiple mediators when the exposure is continuous. However, no code was provided for implementing this procedure, and we were unable to find applied studies using the IPW estimator in this context. It is our understanding that, as in the binary exposure scenario, x corresponds to the observed value of the exposure. For x^* and x^{**}, J random draws are taken from the (conditional) distribution of the observed values of the exposure. For x^*, the entire sequence of the J simulated values is repeated J times, while each of the simulated J values is repeated J times for x^{**}. Therefore, each row is now present J^2 times in the extended data set, as x^* and x^{**} are assigned every possible pair of values obtained by considering the $J \times J$ possible pairs of values from the random draws. Lange et al. (2014) recommend setting J to a minimum of five.

```
bootNEM <- function(data, index) {
bootDat <- data[index, ]

bootDat$tempID <- 1:nrow(bootDat)

fitX <- lm(x_cont ~ c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8,
data=bootDat)

J <- 5
r <- replicate(J, rnorm(nrow(bootDat),
        mean=predict(fitX, type="response"),
        sd = summary(fitX)$sigma))

bootExpDat <- bootDat[rep(bootDat$tempID, each = J^2), ]
bootExpDat <- bootExpDat[order(bootExpDat$tempID),]
bootExpDat$x0 <- bootExpDat$x_cont
bootExpDat$x1 <- c(apply(r, 1, rep, J))
bootExpDat$x2 <- c(apply(r, 1, rep, each=J))
```

```
fitm1 <- lm(m1 ~ x_cont + c1 + c2 + c3 + c4 + c5 +
c6 + c7 + c8, data=bootDat)

num1 <- with(bootExpDat, dnorm(x=m1,
mean=predict(fitm1, newdata=within(bootExpDat, x_cont <- x1)),
sd=summary(fitm1)$sigma))

den1 <- with(bootExpDat, dnorm(x=m1,
mean=predict(fitm1, newdata=within(bootExpDat, x_cont <- x0)),
sd=summary(fitm1)$sigma))

w1   <- num1/den1

fitm2 <- lm(m2 ~ x_cont + c1 + c2 + c3 + c4 + c5 +
c6 + c7 + c8, data=bootDat)

num2 <- with(bootExpDat, dnorm(x=m2,
mean=predict(fitm2, newdata=within(bootExpDat, x_cont <- x2)),
sd=summary(fitm2)$sigma))

den2 <- with(bootExpDat, dnorm(x=m2,
mean=predict(fitm2, newdata=within(bootExpDat, x_cont <- x0)),
sd=summary(fitm2)$sigma))

w2   <- num2/den2

bootExpDat$w <- w1*w2


fitNEM <- glm(y ~ x0*x1*x2 + c1 + c2 + c3 + c4 + c5 +
c6 + c7 + c8, family="binomial", weight=w, data=bootExpDat)

mycoef <- coef(fitNEM)

NDE_m     <- 0.7*mycoef[2]
NIE_m1_y  <- 0.7*(mycoef[3] + 0.7*mycoef[13])
```

```
NIE_m2_y  <- 0.7*(mycoef[4] + 0.7*mycoef[14])
MI        <- 0.7*(0.7*mycoef[15] + 0.7^2*mycoef[16])
NIE_m     <- NIE_m1_y + NIE_m2_y + MI
TE        <- NIE_m + NDE_m

return(c(NDE_m, NIE_m1_y, NIE_m2_y, MI, NIE_m, TE))
}
bootCI <- boot(data = dat, statistic = bootNEM, R = 1000)
R = exp(t(sapply(1:6, function(x) c(bootCI$t0[x],
boot.ci(bootCI, conf = 0.95, type = "perc", index = x)$perc[4:5]))))
colnames(R) <- c("OR", "95 %LCL", "95 %UCL")

R

##              OR   95 %LCL  95 %UCL
## [1,] 1.1642089 1.1347335 1.335106
## [2,] 1.0955592 1.0372187 1.117095
## [3,] 1.0025700 0.9639195 1.026601
## [4,] 0.9901498 0.9837570 1.015802
## [5,] 1.0875556 1.0096631 1.136193
## [6,] 1.2661419 1.1867147 1.442691
```

Extended quasi-Bayesian Monte Carlo approach

The `multimediate` package can accommodate a continuous exposure. In what follows, the code is identical to that presented in the paper, but the argument `treat.value` is set to 0.7 instead of 1.

```
fitm1 <- lm(m1 ~ x_cont + c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8, data=dat)
fitm2 <- lm(m2 ~ x_cont + c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8, data=dat)
list_med <- list(fitm1, fitm2)
fity <- glm(y ~ x_cont*m1*m2 + c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8,
        data=dat, family="binomial")

fit = multimediate(lmodel.m=list_med, model.y=fity, correlated=T,
        treat="x_cont", treat.value=0.7, control.value=0, J=1000, data=dat)
```

```
summary(fit)[1:nrow(summary(fit)),6:10]

##                          . Estimation IC.inf IC.sup P.val
## 1    OR ACME.joint.treat     1.1483 1.1200 1.1764     0
## 2           OR PM(treat)     0.5526 0.4051 0.7746     0
## 3  OR ACME.joint.control     1.1458 1.1122 1.1789     0
## 4         OR PM(control)     0.5411 0.4229 0.7041     0
## 5       OR ACME.treat.m1     1.0706 1.0505 1.0915     0
## 6        OR PM(treat).m1     0.2817 0.1927 0.4118     0
## 7     OR ACME.control.m1     1.0805 1.0552 1.1074     0
## 8      OR PM(control).m1     0.3186 0.2101 0.4710     0
## 9       OR ACME.treat.m2     1.0717 1.0545 1.0903     0
## 10       OR PM(treat).m2     0.2872 0.1932 0.4179     0
## 11    OR ACME.control.m2     1.0610 1.0410 1.0834     0
## 12     OR PM(control).m2     0.2468 0.1514 0.3759     0
## 13          OR ADE.treat     1.1473 1.0634 1.2371     0
## 14        OR ADE.control     1.1453 1.0489 1.2558     0
## 15       OR Total Effect     1.3148 1.2101 1.4346     0
```

Stata code

In this section, we replicate the `R` code presented in the paper and the present Web Appendix in `Stata`. We start by demonstrating the four estimation strategies for a binary exposure, followed by code for a continuous exposure example. Before implementing the `Stata` code, we created a variable named `condition` that equals 1 if an individual meets the inclusion criteria and can therefore be sampled in the boostraping procedure, and 0 otherwise. The type (normal-based, percentile-based, etc.) of bootstrap confidence intervals can be obtained using the `norm` or `perc` option of the `estat bootstrap` command. The exponential form of the results can be derived using the `eform` option of the `estat bootstrap` command. The number of replications for the bootstrap procedure is determined using the `rep(#)` option of the prefix `bootstrap`.

Binary exposure

The following `Stata` code replicates the `R` code presented in the ***Imputation approach*** section of the main article.

Imputation approach

```
cap program drop medflex
program define medflex, eclass
version 18

* Sample to use for bootstrap
tempvar touse
gen `touse' = condition

* Exposure model for calculating marginal weights
logit x i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store pxc

* Outcome model for imputation
logit y i.x##c.m1##i.m2 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store py

preserve
* Matrix to store estimates
matrix coef = J(1, 10, .)
* Expand the data set by a factor of 2 and generate a duplication indicator
expand 2, gen(duplicate)
* Set x0 to x for one replication, and 1-x for the other replication
gen x0 = cond(duplicate==0, x, 1-x)
* Set x1 to x for all replications
gen x1 = x

* Imputation of the outcome
estimate restore py
replace x = x0
predict y_hat
```

```
* Compute weights for marginal estimands
estimate restore pxc
replace x = x1
predict pxc
replace pxc = 1-pxc if x == 0
gen w = 1/pxc

*-----Natural effect models

***Conditional
glm y_hat i.x0##i.x1 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8, fam(bin) link(logit)
matrix coef[1,1] = _b[1.x0]
matrix coef[1,2] = _b[1.x0] +  _b[1.x0#1.x1]
matrix coef[1,3] = _b[1.x1]
matrix coef[1,4] = _b[1.x1] + _b[1.x0#1.x1]
matrix coef[1,5] = coef[1,1] + coef[1,4]

***Marginal
glm y_hat i.x0##i.x1  [pweight=w], fam(bin) link(logit)
matrix coef[1,6] = _b[1.x0]
matrix coef[1,7] = _b[1.x0] + _b[1.x0#1.x1]
matrix coef[1,8] = _b[1.x1]
matrix coef[1,9] = _b[1.x1] + _b[1.x0#1.x1]
matrix coef[1,10] = coef[1,6] + coef[1,9]

restore

*-----Results

***Colname
matrix colnames coef = "PNDEc" "TNDEc" "PNIEc" "TNIEc" ///
"TEc" "PNDEm" "TNDEm" "PNIEm" "TNIEm" "TEm"

*****Return
ereturn post  coef, esample(`touse')
```

```
end

bootstrap, rep(1000) : medflex
estat bootstrap, eform norm perc
```

Extended-imputation approach: weights based on M_1

The following `Stata` code reproduces the `R` code illustrated in the ***Extended-imputation approach*** section of the main article when the weights are calculated based on M_1.

```
cap program drop extImputaton1
program define extImputaton1, eclass
version 18

* Sample to use for bootstrap
tempvar touse
gen `touse' = condition



* Exposure model for calculating marginal weights
logit x i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store pxc

* Mediator model (M1) for calculating the weights
regress m1 x i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store m1

* Outcome model for imputation
logit y i.x##c.m1##i.m2 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store py

preserve
* Matrix to store estimates
matrix coef = J(1, 10, .)
```

```
gen tempID = _n
* Expand the data set by a factor of 4 and generate a duplication indicator
expand 4, gen(duplicate)
bys tempID : gen replicate = _n
* The second and fourth replications are set to x,
*while the first and third replications are set to 1 - x
gen x0 = cond(inlist(replicate, 2, 4), x, 1-x)
* The third and fourth replications are set to x,
*while the first and second replications are set to 1 - x
gen x1 = cond(inlist(replicate, 3, 4), x, 1-x)
* Set x2 to x for all replications
gen x2 = x

* Imputation of the outcome
estimate restore py
replace x = x0
predict y_hat

* Compute weights for conditional estimands (based on M1)
estimate restore m1
replace x = x1
predict meanM1_1
gen num = normalden(m1, meanM1_1, e(rmse))
replace x = x2
predict meanM1_2
gen den = normalden(m1, meanM1_2, e(rmse))
gen w_c = num/den

* Compute weights for marginal estimands (based on M1)
estimate restore pxc
replace x = x2
predict pxc
replace pxc = 1-pxc if x == 0
gen w_m = w_c/pxc

*-----Natural effect models
```

```
***Conditional
glm y_hat i.x0##i.x1##i.x2 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8 ///
[pweight=w_c], fam(bin) link(logit)
matrix coef[1,1] = _b[1.x0]
matrix coef[1,2] = _b[1.x1] + _b[1.x0#1.x1] + _b[1.x1#1.x2] + ///
_b[1.x0#1.x1#1.x2]
matrix coef[1,3] = _b[1.x2] + _b[1.x0#1.x2]
matrix coef[1,4] = coef[1,2] + coef[1,3]
matrix coef[1,5] = coef[1,1] + coef[1,4]

***Marginal
glm y_hat i.x0##i.x1##i.x2  [pweight=w_m], fam(bin) link(logit)
matrix coef[1,6] = _b[1.x0]
matrix coef[1,7] = _b[1.x1] + _b[1.x0#1.x1] + _b[1.x1#1.x2] + ///
_b[1.x0#1.x1#1.x2]
matrix coef[1,8] = _b[1.x2] + _b[1.x0#1.x2]
matrix coef[1,9] = coef[1,7] + coef[1,8]
matrix coef[1,10] = coef[1,6] + coef[1,9]

restore

***Colname
matrix colnames coef = "NDE_m_c" "NIE_m1y_c" "NIE_m2_y_c" "NIE_m_c" ///
"TE_c" "NDE_m_m" "NIE_m1y_m" "NIE_m2_y_m" "NIE_m_m" "TE_m"

*****Return
ereturn post  coef, esample(`touse')

end

bootstrap, rep(1000) : extImputaton1
estat bootstrap, eform norm perc
```

Extended-imputation approach: weights based on M_2

The `Stata` code that follows reproduces the `R` code from the ***Extended-imputation approach*** section of the main article when the weights are calculated based on M_2.

```
cap program drop extImputaton2
program define extImputaton2, eclass
version 18

* Sample to use for bootstrap
tempvar touse
gen `touse' = condition

*  Exposure model for calculating marginal weights
logit x i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store pxc

* Mediator model (M2) for calculating marginal weights
logit m2 i.x##c.m1 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store m2

* Outcome model for imputation
logit y i.x##c.m1##i.m2 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store py

preserve
* Matrix to store estimates
matrix coef = J(1, 10, .)
gen tempID = _n
* Expand the data set by a factor of 4 and generate a duplication indicator
expand 4
bys tempID : gen replicate = _n
* The second and fourth replications are set to x,
*while the first and third replications are set to 1 - x
gen x0 = cond(inlist(replicate, 2, 4), x, 1-x)
```

```
* Set x1 to x for all replications
gen x1 = x
* The third and fourth replications are set to x,
*while the first and second replications are set to 1 - x
gen x2 = cond(inlist(replicate, 3, 4), x, 1-x)

* Imputation of the outcome
estimate restore py
replace x = x0
predict y_hat

* Compute weights for conditional estimands (based on M2)
estimate restore m2
replace x = x2
predict pm2_2
replace pm2_2 = 1-pm2_2 if m2 == 0
replace x = x1
predict pm2_1
replace pm2_1 = 1-pm2_1 if m2 == 0
gen w_c = pm2_2/pm2_1

* Compute weights for marginal estimands (based on M2)
estimate restore pxc
replace x = x1
predict pxc
replace pxc = 1-pxc if x == 0
gen w_m = w_c/pxc

*-----Natural effect models

***Conditional
glm y_hat i.x0##i.x1##i.x2 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8 ///
[pweight=w_c], fam(bin) link(logit)
matrix coef[1,1] = _b[1.x0]
matrix coef[1,2] = _b[1.x1] + _b[1.x0#1.x1] + _b[1.x1#1.x2] +  ///
_b[1.x0#1.x1#1.x2]
```

```
matrix coef[1,3] = _b[1.x2] + _b[1.x0#1.x2]
matrix coef[1,4] = coef[1,2] + coef[1,3]
matrix coef[1,5] = coef[1,1]+ coef[1,4]

***Marginal
glm y_hat i.x0##i.x1##i.x2  [pweight=w_m], fam(bin) link(logit)
matrix coef[1,6] = _b[1.x0]
matrix coef[1,7] = _b[1.x1] + _b[1.x0#1.x1] + _b[1.x1#1.x2] + ///
_b[1.x0#1.x1#1.x2]
matrix coef[1,8] = _b[1.x2] + _b[1.x0#1.x2]
matrix coef[1,9] = coef[1,7] + coef[1,8]
matrix coef[1,10] = coef[1,6] + coef[1,9]

restore

***Colname
matrix colnames coef = "NDE_m_c" "NIE_m1y_c" "NIE_m2_y_c" "NIE_m_c" ///
"TE_c" "NDE_m_m" "NIE_m1y_m" "NIE_m2_y_m" "NIE_m_m" "TE_m"

*****Return
ereturn post  coef, esample(`touse')

end

bootstrap, rep(1000) : extImputaton2
estat bootstrap, eform norm perc
```

Inverse probability weighted approach

The following `Stata` code replicates the `R` code from the ***Inverse probability weighted approach*** section of the main article.

```
cap program drop ipw
program define ipw, eclass
version 18
```

```
* Sample to use for bootstrap
tempvar touse
gen `touse' = condition

* Exposure model for calculating marginal weights
logit x i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store pxc

* Mediator model (M1) for calculating weights
regress m1 i.x i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store m1

* Mediator model (M2) for calculating weights
regress m2 i.x i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store m2

preserve
* Matrix to store estimates
matrix coef = J(1, 12, .)
gen tempID = _n
* Expand the data set by a factor of 4 and generate a duplication indicator
expand 4
bys tempID : gen replicate = _n
* Set x0 to x for all replications
gen x0 = x
* The second and fourth replications are set to x,
*while the first and third replications are set to 1 - x
gen x1 = cond(inlist(replicate, 2, 4), x, 1-x)
* The third and fourth replications are set to x,
*while the first and second replications are set to 1 - x
gen x2 = cond(inlist(replicate, 3, 4), x, 1-x)

* Computing weights based on M1
estimate restore m1
replace x = x1
predict meanM1_1
```

```
gen num1 = normalden(m1, meanM1_1, e(rmse))
replace x = x0
predict meanM1_0
gen den1 = normalden(m1, meanM1_0, e(rmse))
gen w1 = num1/den1

* Computing weights based on M2
estimate restore m2
replace x = x2
predict meanM2_1
gen num2 = normalden(m2, meanM2_1, e(rmse))
replace x = x0
predict meanM2_0
gen den2 = normalden(m2, meanM2_0, e(rmse))
gen w2 = num2/den2

* Computing conditional weights based on w1 and w2
gen w_c = w1*w2

* Computing marginal weights based on w1 and w2
estimate restore pxc
predict pxc
replace pxc = 1-pxc if x == 0
gen w_m = w_c/pxc

*-----Natural effect models

***Conditional
glm y i.x0##i.x1##i.x2 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8 ///
[pweight=w_c] , fam(bin) link(logit)
matrix coef[1,1] = _b[1.x0]
matrix coef[1,2] = _b[1.x1] + _b[1.x0#1.x1]
matrix coef[1,3] = _b[1.x2] + _b[1.x0#1.x2]
matrix coef[1,4] = _b[1.x1#1.x2] + _b[1.x0#1.x1#1.x2]
matrix coef[1,5] = coef[1,2] + coef[1,3] + coef[1,4]
matrix coef[1,6] = coef[1,1] + coef[1,5]
```

```
***Marginal
glm y i.x0##i.x1##i.x2  [pweight=w_m] , fam(bin) link(logit)
matrix coef[1,7] = _b[1.x0]
matrix coef[1,8] = _b[1.x1] + _b[1.x0#1.x1]
matrix coef[1,9] = _b[1.x2] + _b[1.x0#1.x2]
matrix coef[1,10] = _b[1.x1#1.x2] + _b[1.x0#1.x1#1.x2]
matrix coef[1,11] = coef[1,8] + coef[1,9] + coef[1,10]
matrix coef[1,12] = coef[1,7] + coef[1,11]

restore

***Colname
matrix colnames coef = "NDE_m_c" "NIE_m1_y_c" "NIE_m2_y_c" "MI_c" ///
"NIE_m_c" "TE_c"  "NDE_m_m" "NIE_m1_y_m" "NIE_m2_y_m" "MI_m" ///
"NIE_m_m" "TE_m"

*****Return
ereturn post  coef, esample(`touse')

end

bootstrap, rep(1000) : ipw
estat bootstrap, eform norm perc
```

Extended quasi-Bayesian Monte Carlo approach

The following `Stata` code mirrors the `R` code from the ***Extended quasi-Bayesian Monte Carlo approach*** section of the main article. The code has some limitations, however, as it requires important adjustments if the results are presented on the difference scale, if one of the mediators is not continuous, or if alternative models are specified for the mediators and/or the outcome. The code also necessitates to create all dummy variables and interaction terms manually, as it does not allow to use `Stata`'s prefixes and operators. The code also assumes that there are no missing data. In the following code, $xm1$ is the

interaction term between the exposure and the first mediator, $xm2$ is the interaction term between the exposure and the second mediator, $m1m2$ is the interaction term between the mediators, $xm1m2$ is the interaction term between the exposure, the first mediator, and the second mediator. The variables $c12$, $c72$ and $c82$ are the dummy variables for $c1$, $c7$, and $c8$, respectively. The first step is to install the `moremata` SSC package to enable the calculation of quantiles using the `mm_quantile` function.

```
cap ssc install moremata
```

The second step is to create two mata functions, namely the function `OR` and `multimediate`.

```
cap mata mata drop multimediate()
cap mata mata drop OR()
mata:
function OR(real matrix x, real matrix y) {
out = exp(x'):/exp(y')
return (out)
}
function multimediate(real matrix C, real matrix simParM1,
real matrix simParM2, real matrix simParY, real matrix cov,
real scalar x1, real scalar x0) {
nbrSim = rows(simParM1)
sampleSize = rows(C)
one = J(sampleSize, 1, 1)
X1  = J(sampleSize, 1, x1)
X0  = J(sampleSize, 1, x0)
out = J(nbrSim, 9, .)

/* Specifying the linear model for the mediators */
valueM_1 = X1, C, one
valueM_0 = X0, C, one
for(i=1; i<=nbrSim; i++) {
/*Simulate the mediators*/
```

```
Z = (cholesky(cov)*rnormal(sampleSize,2,0,1)')'
Z11 = Z :+ ((simParM1[i,]*valueM_1')',(simParM2[i,]*valueM_1')')
Z10 = Z :+ ((simParM1[i,]*valueM_1')',(simParM2[i,]*valueM_0')')
Z01 = Z :+ ((simParM1[i,]*valueM_0')',(simParM2[i,]*valueM_1')')
Z00 = Z :+ ((simParM1[i,]*valueM_0')',(simParM2[i,]*valueM_0')')

/*Specifying the linear model for the outcome*/
valueY_1_11 = X1, Z11[,1], Z11[,1]:*x1, Z11[,2], Z11[,2]:*x1,
Z11[,1]:*Z11[,2], Z11[,1]:*Z11[,2]:*x1, C, one
valueY_1_10 = X1, Z10[,1], Z10[,1]:*x1, Z10[,2], Z10[,2]:*x1,
Z10[,1]:*Z10[,2], Z10[,1]:*Z10[,2]:*x1, C, one
valueY_1_01 = X1, Z01[,1], Z01[,1]:*x1, Z01[,2], Z01[,2]:*x1,
Z01[,1]:*Z01[,2], Z01[,1]:*Z01[,2]:*x1, C, one
valueY_1_00 = X1, Z00[,1], Z00[,1]:*x1, Z00[,2], Z00[,2]:*x1,
Z00[,1]:*Z00[,2], Z00[,1]:*Z00[,2]:*x1, C, one
valueY_0_11 = X0, Z11[,1], Z11[,1]:*x0, Z11[,2], Z11[,2]:*x0,
Z11[,1]:*Z11[,2], Z11[,1]:*Z11[,2]:*x0, C, one
valueY_0_10 = X0, Z10[,1], Z10[,1]:*x0, Z10[,2], Z10[,2]:*x0,
Z10[,1]:*Z10[,2], Z10[,1]:*Z10[,2]:*x0, C, one
valueY_0_01 = X0, Z01[,1], Z01[,1]:*x0, Z01[,2], Z01[,2]:*x0,
Z01[,1]:*Z01[,2], Z01[,1]:*Z01[,2]:*x0, C, one
valueY_0_00 = X0, Z00[,1], Z00[,1]:*x0, Z00[,2], Z00[,2]:*x0,
Z00[,1]:*Z00[,2], Z00[,1]:*Z00[,2]:*x0, C, one

/*Compute the average of the conditional odds ratios*/
out[i, 1] = mean(OR(simParY[i,]*valueY_1_11', simParY[i,]*valueY_1_00'))
out[i, 2] = mean(OR(simParY[i,]*valueY_0_11', simParY[i,]*valueY_0_00'))
out[i, 3] = mean(OR(simParY[i,]*valueY_1_11', simParY[i,]*valueY_1_01'))
out[i, 4] = mean(OR(simParY[i,]*valueY_0_11', simParY[i,]*valueY_0_01'))
out[i, 5] = mean(OR(simParY[i,]*valueY_1_11', simParY[i,]*valueY_1_10'))
out[i, 6] = mean(OR(simParY[i,]*valueY_0_11', simParY[i,]*valueY_0_10'))
out[i, 7] = mean(OR(simParY[i,]*valueY_1_11', simParY[i,]*valueY_0_11'))
out[i, 8] = mean(OR(simParY[i,]*valueY_1_00', simParY[i,]*valueY_0_00'))
out[i, 9] = mean(OR(simParY[i,]*valueY_1_11', simParY[i,]*valueY_0_00'))
}
/*Summary of the results*/
```

```
m = mean(out)'
low = mm_quantile(out, 1, 0.025, 6)'
high = mm_quantile(out, 1, 0.975, 6)'
result = m, low, high
st_matrix("result", result)
return (result)
}
end
```

The next step is to create a matrix in `mata` that corresponds to the value of the covariates using `putmata`.

```
putmata C=(c12 c2 c3 c4 c5 c6 c72 c82), replace
```

Then regressions for the M_1, M_2, and Y are fitted. For each variable in a model, the parameters are simulated 1000 times using the Cholesky decomposition. The residuals for the mediators are also created using the postestimation command `predict`.

```
qui regress m1 x c12 c2 c3 c4 c5 c6 c72 c82
estimate store m1
cap drop r1
qui predict r1, resid
* Simulate the parameters
mata:
paramM1 = st_matrix("e(b)")
vcovM1 = st_matrix("e(V)")
simParM1 = (cholesky(vcovM1)*rnormal(1000,10,0,1)' :+ paramM1')'
end

qui regress m2 x c12 c2 c3 c4 c5 c6 c72 c82
estimate store m2
cap drop r2
qui predict r2, resid
* Simulate the parameters
mata:
```

```
paramM2 = st_matrix("e(b)")
vcovM2 = st_matrix("e(V)")
simParM2 = (cholesky(vcovM2)*rnormal(1000,10,0,1)' :+ paramM2')'
end

qui logit y x m1 xm1 m2 xm2 m1m2 xm1m2 c12 c2 c3 c4 c5 c6 c72 c82
estimate store y
* Simulate the parameters
mata:
paramY = st_matrix("e(b)")
vcovY = st_matrix("e(V)")
simParY = (cholesky(vcovY)*rnormal(1000,16,0,1)' :+ paramY')'
end
```

The user must change the arguments `n` and `p` of the `rnormal(n,p,0,1)` function so that the first element (`n`) represents the number of draws and the second (`p`) corresponds to the number of parameters of a given model. Next, the conditional covariance between the counterfactual mediators must be estimated as the covariance between the residuals of the mediators, and exported into `Stata` using the `st_matrix` function.

```
qui corr r1 r2, cov
mata: cov = st_matrix("r(C)")
```

Finally, the results can be obtained using the `multimediate` function in `mata`. The function takes as argument the matrix of covariates (`C`), the matrix of simulated parameters for M_1 (`simParM1`), the matrix of simulated parameters for M_2 (`simParM2`), the matrix of simulated parameters for Y (`simParY`), the matrix of the covariance between the counterfactual mediators (`cov`), the value of the treatment (1) and the value of the control (0). The `multimediate` function generates a `Stata` matrix called `Result` that can be used to display the results.

```
mata: sim = multimediate(C, simParM1, simParM2, simParY, cov, 1, 0)
matrix colnames result = "OR" "2.5% UL" "97.5% LL"
matrix rownames result = "ACME_joint_treat" "ACME_joint_control" ///
"ACME_treat_m1" "ACME_control_m1" "ACME_treat_m2" "ACME_control_m2"  ///
"ADE_treat" "ADE_control" "Total Effect"
matlist result
```

Continuous exposure

In what follows, we provide `Stata` code for cases where the exposure is continuous.

Imputation approach

```
cap program drop medflex_cont
program define medflex_cont, eclass
version 18

* Sample to use for bootstrap
tempvar touse
gen `touse' = condition

* Exposure model for data replications
regress x_cont i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store pxc

* Outcome model for imputation
logit y c.x_cont##c.m1##i.m2 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store py

preserve
* Matrix to store estimates
matrix coef = J(1, 3, .)

* Expand the data set by a factor of 5
*and generate a duplication indicator (using quantiles)
gen tempID = _n
expand 5
```

```
bys tempID : gen replicate = _n
* Find the quantiles
gen D = 0.05 + (.90/4)*(replicate-1)
estimate restore pxc
predict pxc

* Set x0 to the 5 quantiles
gen x0 = e(rmse)*invnormal(D) + pxc
* Set x1 to x_cont for all replications
gen x1 = x_cont

* Imputation of the outcome
estimate restore py
replace x_cont = x0
predict y_hat


*-----Natural effect models

***Conditional
glm y_hat c.x0##c.x1 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8, fam(bin) link(logit)
matrix coef[1,1] = 0.7*_b[x0]
matrix coef[1,2] = (_b[x1] + 0.7*_b[x0#x1])*0.7
matrix coef[1,3] = coef[1,1] + coef[1,2]

restore

*-----Results

***Colname
matrix colnames coef = "NDEc" "NIEc" "TEc"

*****Return
ereturn post  coef, esample(`touse')

end
```

```
bootstrap, rep(1000) : medflex_cont
estat bootstrap, eform norm perc
```

Extended-imputation approach: weights based on M_2

```
cap program drop extImputaton2_cont
program define extImputaton2_cont, eclass
version 18

* Sample to use for bootstrap
tempvar touse
gen `touse' = condition

* Exposure model for data replications
regress x_cont i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store pxc

* Mediator model (M2) for calculating the weights
logit m2 c.x_cont##c.m1 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store m2

* Outcome model for imputation
logit y c.x_cont##c.m1##i.m2 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store py

* Set the value of J
local J = 3

preserve
gen tempID = _n
* Matrix to store estimates
matrix coef = J(1, 5, .)
* Expand the data set by a factor of J^2=9 and
*generate a duplication indicator
expand `J'^2
```

```
bys tempID : gen replicate = _n

estimate restore pxc
predict pxc

* x0 replicates the 0.1 quantile, the 0.9 quantile, and
*the original value sequence J times
gen x0 = cond(inlist(replicate, 1, 4, 7), e(rmse)*invnormal(0.1) + ///
pxc, cond(inlist(replicate, 2, 5, 8), e(rmse)*invnormal(0.9) + ///
pxc, x_cont))
* Set x1 to x_cont for all replications
gen x1 = x_cont
* x2 replicates the 0.1 quantile J times, the 0.9 quantile J times, and
*the original value J times
gen x2 = cond(inlist(replicate, 1, 2, 3), e(rmse)*invnormal(0.1) + ///
pxc, cond(inlist(replicate, 7, 8, 9), e(rmse)*invnormal(0.9) + ///
pxc, x_cont))

* Compute weights for conditional estimands (based on M2)
estimate restore m2
replace x_cont = x2
predict pm2_2
replace pm2_2 = 1-pm2_2 if m2 == 0
replace x_cont = x1
predict pm2_1
replace pm2_1 = 1-pm2_1 if m2 == 0
gen w_c = pm2_2/pm2_1

* Imputation of the outcome
estimate restore py
replace x_cont = x0
predict y_hat

*-----Natural effect models

***Conditional
```

```
glm y_hat c.x0##c.x1##c.x2 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8 ///
[pweight=w_c], fam(bin) link(logit)
matrix coef[1,1] = 0.7*_b[x0]
matrix coef[1,2] = 0.7*(_b[x1] + 0.7*(_b[x0#x1] + _b[x1#x2] + ///
0.7*_b[x0#x1#x2]))
matrix coef[1,3] = 0.7*(_b[x2] + 0.7*_b[x0#x2])
matrix coef[1,4] = coef[1,2] + coef[1,3]
matrix coef[1,5] = coef[1,1] + coef[1,4]

restore

***Colname
matrix colnames coef = "NDE_m_c" "NIE_m1y_c" "NIE_m2_y_c" "NIE_m_c" "TE_c"

*****Return
ereturn post  coef, esample(`touse')

end

bootstrap, rep(1000) : extImputaton2_cont
estat bootstrap, eform norm perc
```

Inverse probability weighted approach

```
cap program drop ipw_cont
program define ipw_cont, eclass
version 18

* Sample to use for bootstrap
tempvar touse
gen `touse' = condition

* Exposure model for data replications
regress x_cont i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store pxc
```

```
* Set the value of J
local J = 5

* Mediation model (M1) for calculating weights
regress m1 x_cont i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store m1

* Mediation model (M2) for calculating weights
regress m2 x_cont i.c1 c2 c3 c4 c5 c6 i.c7 i.c8
estimate store m2

preserve
* Matrix to save estimates
matrix coef = J(1, 6, .)
gen tempID = _n

* Expand the data set by a factor of J
expand `J'
sort tempID

estimate restore pxc
predict meanA

* Set x0 to x_cont for all replications
gen x0 = x_cont
* Randomly draw J values for x1, and then expand the data set by a
*factor of J
* x1 replicates the J sampled values sequence J times
gen x1 = rnormal(meanA, e(rmse))
expand `J'
bys tempID x1 : gen counter_id = _n
* Arrange the values of x2
* x2 replicates each of the J sampled values J times
by tempID : gen x2 = x1[1+ `J'*(counter_id-1)]

* Computing weights based on M1
```

```
estimate restore m1
replace x_cont = x1
predict meanM1_1
gen num1 = normalden(m1, meanM1_1, e(rmse))
replace x_cont = x0
predict meanM1_0
gen den1 = normalden(m1, meanM1_0, e(rmse))
gen w1 = num1/den1

* Computing weights based on M2
estimate restore m2
replace x_cont = x2
predict meanM2_1
gen num2 = normalden(m2, meanM2_1, e(rmse))
replace x_cont = x0
predict meanM2_0
gen den2 = normalden(m2, meanM2_0, e(rmse))
gen w2 = num2/den2

* Computing conditional weights based on w1 and w2
gen w_c = w1*w2


*-----Natural effect models

***Conditional
glm y c.x0##c.x1##c.x2 i.c1 c2 c3 c4 c5 c6 i.c7 i.c8 ///
[pweight=w_c], fam(bin) link(logit)
matrix coef[1,1] = 0.7*_b[x0]
matrix coef[1,2] = 0.7*(_b[x1] + 0.7*_b[x0#x1])
matrix coef[1,3] = 0.7*(_b[x2] + 0.7*_b[x0#x2])
matrix coef[1,4] = 0.7*(0.7*_b[x1#x2] + 0.7^2*_b[x0#x1#x2])
matrix coef[1,5] = coef[1,2] + coef[1,3] + coef[1,4]
matrix coef[1,6] = coef[1,1] + coef[1,5]
```

```
restore

***Colname
matrix colnames coef = "NDE_m_c" "NIE_m1_y_c" "NIE_m2_y_c" "MI_m_c" ///
"NIE_m_c" "TE_c"
*****Return
ereturn post  coef, esample(`touse')

end

bootstrap, rep(1000) : ipw_cont
estat bootstrap, eform norm perc
```

Extended quasi-Bayesian Monte Carlo approach

The code for the extended quasi-Bayesian Monte Carlo approach when the exposure is continuous is identical to the one provided for a binary exposure. The argument of the `multimediate` function must however be changed to compute the total and natural effects when the exposure is set to 0 versus 0.7.

```
putmata C=(c12 c2 c3 c4 c5 c6 c72 c82), replace

qui regress m1 x_cont c12 c2 c3 c4 c5 c6 c72 c82
estimate store m1
cap drop r1
qui predict r1, resid
* Simulate the parameters
mata:
paramM1 = st_matrix("e(b)")
vcovM1 = st_matrix("e(V)")
simParM1 = (cholesky(vcovM1)*rnormal(1000,10,0,1)' :+ paramM1')'
end

qui regress m2 x_cont c12 c2 c3 c4 c5 c6 c72 c82
estimate store m2
```

```
cap drop r2
qui predict r2, resid
* Simulate the parameters
mata:
paramM2 = st_matrix("e(b)")
vcovM2 = st_matrix("e(V)")
simParM2 = (cholesky(vcovM2)*rnormal(1000,10,0,1)' :+ paramM2')'
end

qui logit y x_cont m1 xm1 m2 xm2 m1m2 xm1m2 c12 c2 c3 c4 c5 c6 c72 c82
estimate store y
* Simulate the parameters
mata:
paramY = st_matrix("e(b)")
vcovY = st_matrix("e(V)")
simParY = (cholesky(vcovY)*rnormal(1000,16,0,1)' :+ paramY')'
end

qui corr r1 r2, cov
mata: cov = st_matrix("r(C)")

mata: sim = multimediate(C, simParM1, simParM2, simParY, cov, 0.7, 0)
matrix colnames result = "OR" "2.5% UL" "97.5% LL"
matrix rownames result = "ACME_joint_treat" "ACME_joint_control" ///
"ACME_treat_m1" "ACME_control_m1" "ACME_treat_m2" "ACME_control_m2" ///
"ADE_treat" "ADE_control" "Total Effect"
matlist result
```

References

Jun, S. J., & Lee, S. (2023, April). Average adjusted association: Efficient estimation with high dimensional confounders. *In International Conference on Artificial Intelligence and Statistics* (pp. 5980-5996). PMLR.

Karlson, K. B., Popham, F., & Holm, A. (2023). Marginal and conditional confounding using logits. *Sociological Methods and Research*, *52*(4), 1765–1784. https://doi.org/10.1177/0049124121995548

Lange, T., Rasmussen, M., & Thygesen, L. C. (2014). Assessing natural direct and indirect effects through multiple pathways. *American Journal of Epidemiology*, *179*(4), 513–518. https://doi.org/10.1093/aje/kwt270

Lange, T., Vansteelandt, S., & Bekaert, M. (2012). A simple unified approach for estimating natural direct and indirect effects. *American Journal of Epidemiology*, *176*(3), 190–195. https://doi.org/10.1093/aje/kwr525

Samoilenko, M., & Lefebvre, G. (2021). Parametric-regression-based causal mediation analysis of binary outcomes and binary mediators: Moving beyond the rareness or commonness of the outcome. *American Journal of Epidemiology*, *190*(9), 1846–1858. https://doi.org/10.1093/aje/kwab055

StataCorp. (2023). *Stata statistical software: Release 18.* College Station, TX: StataCorp LLC.

Steen, J., Loeys, T., Moerkerke, B., & Vansteelandt, S. (2017). Medflex: An R package for flexible mediation analysis using natural effect models. *Journal of Statistical Software*, *76*(11), 1–45. https://doi.org/10.18637/jss.v076.i11

Strauss, M. (2011). International dating violence study (2001–2006). Interuniversity consortium for political and social research. https://doi.org/10.3886/ICPSR29583.v1