A guided tour in targeted learning territory

David Benkeser, Antoine Chambaz, Nima Hejazi

08/23/2018

Contents

1 Introduction

```
redo_fixed <- c(TRUE, FALSE)[2]
redo_varying <- c(TRUE, FALSE)[2]
## if  'redo_$'  then  recompute  'learned_features_$_sample_size',  otherwise
## upload it if it is not already in the environment.
if (!redo_fixed) {
  if (!exists("learned_features_fixed_sample_size")) {
    learned_features_fixed_sample_size <-
```

```r
      loadObject("data/learned_features_fixed_sample_size.xdr")
  }
}
if (!redo_varying) {
  if (!exists("learned_features_varying_sample_size")) {
    learned_features_varying_sample_size <-
      loadObject("data/learned_features_varying_sample_size.xdr")
  }
}
```

```r
set.seed(54321) ## because reproducibility matters...
suppressMessages(library(R.utils)) ## make sure it is installed
suppressMessages(library(tidyverse)) ## make sure it is installed
suppressMessages(library(caret)) ## make sure it is installed
expit <- plogis
logit <- qlogis
```

Function `expit` implements the link function $\text{expit} : \mathbb{R} \to]0,1[$ given by $\text{expit}(x) \equiv (1 + e^{-x})^{-1}$. Function `logit` implements its inverse function $\text{logit} :]0,1[\to \mathbb{R}$ given by $\text{logit}(p) \equiv \log[p/(1-p)]$.

2 A simulation study

blabla

2.1 Reproducible experiment as a law. We are interested in a reproducible experiment. The generic summary of how one realization of the experiment unfolds, our observation, is called O. We view O as a random variable drawn from what we call the law P_0 of the experiment. The law P_0 is viewed as an element of what we call the model. Denoted by $\mathcal{M}$, the model is the collection of *all* laws from which O can be drawn and that meet some constraints. The constraints translate the knowledge we have about the experiment. The more we know about the experiment, the smaller is $\mathcal{M}$. In all our examples, model $\mathcal{M}$ will put very few restrictions on the candidate laws.

Consider the following chunk of code:

```r
draw_from_experiment <- function(n, ideal = FALSE) {
  ## preliminary
  n <- Arguments$getInteger(n, c(1, Inf))
  ideal <- Arguments$getLogical(ideal)
  ## ## 'Gbar' and 'Qbar' factors
  Gbar <- function(W) {
    expit(1 + 2 * W - 4 * sqrt(abs((W - 5/12))))
  }
  Qbar <- function(AW) {
    A <- AW[, 1]
    W <- AW[, 2]
    ## A * (cos((1 + W) * pi / 4) + (1/3 <= W & W <= 1/2) / 5) +
    ##  (1 - A) * (sin(4 * W^2 * pi) / 4 + 1/2)
    A * (cos((-1/2 + W) * pi) * 2/5 + 1/5 + (1/3 <= W & W <= 1/2) / 5 +
         (W >= 3/4) * (W - 3/4) * 2) +
      (1 - A) * (sin(4 * W^2 * pi) / 4 + 1/2)
```

```r
  }
  ## sampling
  ## ## context
  mixture_weights <- c(1/10, 9/10, 0)
  mins <- c(0, 11/30, 0)
  maxs <- c(1, 14/30, 1)
  latent <- findInterval(runif(n), cumsum(mixture_weights)) + 1
  W <- runif(n, min = mins[latent], max = maxs[latent])
  ## ## counterfactual rewards
  zeroW <- cbind(A = 0, W)
  oneW <- cbind(A = 1, W)
  Qbar.zeroW <- Qbar(zeroW)
  Qbar.oneW <- Qbar(oneW)
  Yzero <- rbeta(n, shape1 = 2, shape2 = 2 * (1 - Qbar.zeroW) / Qbar.zeroW)
  Yone <- rbeta(n, shape1 = 3, shape2 = 3 * (1 - Qbar.oneW) / Qbar.oneW)
  ## ## action undertaken
  A <- rbinom(n, size = 1, prob = Gbar(W))
  ## ## actual reward
  Y <- A * Yone + (1 - A) * Yzero
  ## ## observation
  if (ideal) {
    obs <- cbind(W = W, Yzero = Yzero, Yone = Yone, A = A, Y = Y)
  } else {
    obs <- cbind(W = W, A = A, Y = Y)
  }
  attr(obs, "Gbar") <- Gbar
  attr(obs, "Qbar") <- Qbar
  attr(obs, "QW") <- function(W) {
    out <- sapply(1:length(mixture_weights),
                  function(ii){
                    mixture_weights[ii] *
                      dunif(W, min = mins[ii], max = maxs[ii])
                  })
    return(rowSums(out))
  }
  attr(obs, "qY") <- function(AW, Y, Qbar){
    A <- AW[, 1]
    W <- AW[, 2]
    Qbar.AW <- do.call(Qbar, list(AW)) # is call to 'do.call' necessary?
    shape1 <- ifelse(A == 0, 2, 3)
    dbeta(Y, shape1 = shape1, shape2 = shape1 * (1 - Qbar.AW) / Qbar.AW)
  }
  ##
  return(obs)
}
```

We can interpret `draw_from_experiment` as a law P_0 since we can use the function to sample observations from a common law. It is even a little more than that, because we can tweak the experiment, by setting its `ideal` argument to `TRUE`, in order to get what appear as intermediary (counterfactual) variables in the regular experiment. The next chunk of code runs the (regular) experiment five times independently and outputs the resulting observations:

```r
(five_obs <- draw_from_experiment(5))
```

```
##              W A         Y
## [1,] 0.4533028 0 0.8979460
## [2,] 0.3716077 0 0.9905312
## [3,] 0.3875802 0 0.8080567
## [4,] 0.4008279 1 0.9954100
## [5,] 0.4038325 0 0.9772926
## attr(,"Gbar")
## function (W)
## {
##     expit(1 + 2 * W - 4 * sqrt(abs((W - 5/12))))
## }
## <bytecode: 0xda47a798>
## <environment: 0xd9dcc050>
## attr(,"Qbar")
## function (AW)
## {
##     A <- AW[, 1]
##     W <- AW[, 2]
##     A * (cos((-1/2 + W) * pi) * 2/5 + 1/5 + (1/3 <= W & W <=
##         1/2)/5 + (W >= 3/4) * (W - 3/4) * 2) + (1 - A) * (sin(4 *
##         W^2 * pi)/4 + 1/2)
## }
## <bytecode: 0xdb10db48>
## <environment: 0xd9dcc050>
## attr(,"QW")
## function (W)
## {
##     out <- sapply(1:length(mixture_weights), function(ii) {
##         mixture_weights[ii] * dunif(W, min = mins[ii], max = maxs[ii])
##     })
##     return(rowSums(out))
## }
## <bytecode: 0xdbd93c60>
## <environment: 0xd9dcc050>
## attr(,"qY")
## function (AW, Y, Qbar)
## {
##     A <- AW[, 1]
##     W <- AW[, 2]
##     Qbar.AW <- do.call(Qbar, list(AW))
##     shape1 <- ifelse(A == 0, 2, 3)
##     dbeta(Y, shape1 = shape1, shape2 = shape1 * (1 - Qbar.AW)/Qbar.AW)
## }
## <bytecode: 0xdc12e468>
## <environment: 0xd9dcc050>
```

We can view the `attributes` of object `five_obs` because, in this section, we act as oracles, *i.e.*, we know completely the nature of the experiment. In particular, we have included several features of P_0 that play an important role in our developments. The attribute `QW` describes the density of W, of which the law $Q_{0,W}$ is a mixture of the uniform laws over $[0,1]$ (weight $1/10$) and $[11/30, 14/30]$ (weight $9/10$).* The

*We fine-tuned (or tweaked, or something else?) the marginal law of W to make it easier later on to drive home important attribute Gbar describes the conditional probability of action $A = 1$ given W. For each $a \in \{0, 1\}$, we denote $\bar{G}_0(W) \equiv \Pr_{P_0}(A = 1|W)$ and $\ell\bar{G}_0(a, W) \equiv \Pr_{P_0}(A = a|W)$. The attribute qY describes the conditional density of Y given A and W. For each $y \in]0, 1[$, we denote by $q_{0,Y}(y, A, W)$ the conditional density evaluated at y of Y given A and W. Similarly, the attribute Qbar describes the conditional mean of Y given A and W, and we denote $\bar{Q}_0(A, W) = \mathrm{E}_{P_0}(Y|A, W)$ the conditional mean of Y given A and W.

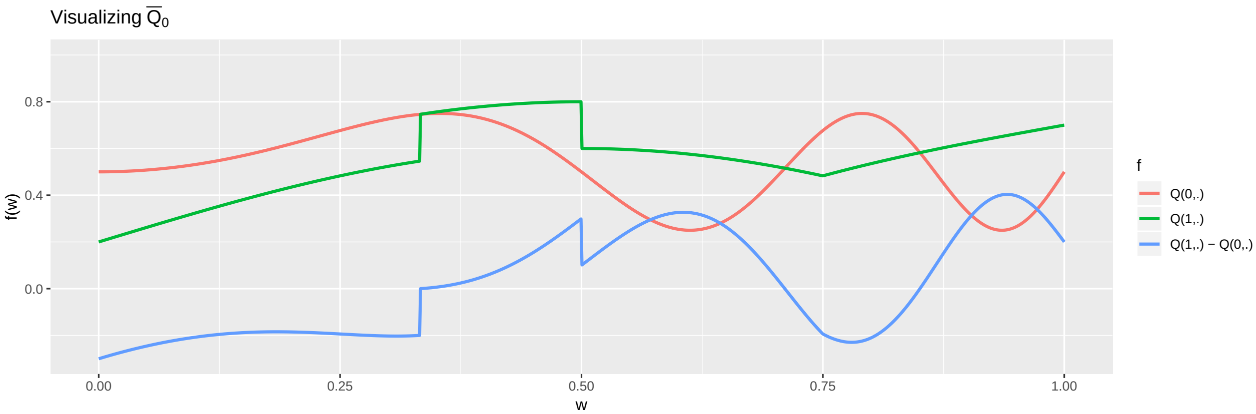
2.2 Visualizing infinite-dimensional features of the experiment

1. Run the following chunk of code. It visualizes the conditional mean $\bar{Q}_0$.

```
Gbar <- attr(five_obs, "Gbar")
Qbar <- attr(five_obs, "Qbar")
QW <- attr(five_obs, "QW")

features <- tibble(w = seq(0, 1, length.out = 1e3)) %>%
  mutate(Qw = QW(w),
         Gw = Gbar(w),
         Q1w = Qbar(cbind(A = 1, W = w)),
         Q0w = Qbar(cbind(A = 0, W = w)),
         blip_Qw = Q1w - Q0w)

features %>% select(-Qw, -Gw) %>%
  rename("Q(1,.)" = Q1w,
         "Q(0,.)" = Q0w,
         "Q(1,.) - Q(0,.)" = blip_Qw) %>%
  gather("f", "value", -w) %>%
  ggplot() +
  geom_line(aes(x = w, y = value, color = f), size = 1) +
  labs(y = "f(w)", title = bquote("Visualizing" ~ bar(Q)[0])) +
  ylim(NA, 1)
```



2. Adapt the above chunk of code to visualize the marginal density $Q_{0,W}$ and conditional probability $\bar{G}_0$.

2.3 The parameter of interest, first pass. It happens that we especially care for a finite-dimensional feature of P_0 that we denote by ψ_0. Its definition involves two of the aforementioned infinite-dimensional features:

messages. Specifically, ... (do we explain what happens?)

$$
\begin{aligned}
\psi_0 &\equiv \int \left(\bar{Q}_0(1,w) - \bar{Q}_0(0,w)\right) dQ_{0,W}(w) \qquad (1)\\
&= \mathrm{E}_{P_0}\left(\mathrm{E}_{P_0}(Y \mid A=1,W) - \mathrm{E}_{P_0}(Y \mid A=0,W)\right).
\end{aligned}
$$

Acting as oracles, we can compute explicitly the numerical value of ψ_0.

```
integrand <- function(w) {
  Qbar <- attr(five_obs, "Qbar")
  QW <- attr(five_obs, "QW")
  ( Qbar(cbind(1, w)) - Qbar(cbind(0, w)) ) * QW(w)
}
(psi_zero <- integrate(integrand, lower = 0, upper = 1)$val)
```

```
## [1] 0.08317711
```

Our interest in ψ_0 is of causal nature. Taking a closer look at `drawFromExperiment` reveals indeed that the random making of an observation O drawn from P_0 can be summarized by the following causal graph and nonparametric system of structural equations:

```
## plot the causal diagram
```

and, for some deterministic functions f_w, f_a, f_y and independent sources of randomness U_w, U_a, U_y,

1. sample the context where the counterfactual rewards will be generated, the action will be undertaken and the actual reward will be obtained, $W = f_w(U_w)$;

2. sample the two counterfactual rewards of the two actions that can be undertaken, $Y_0 = f_y(0, W, U_y)$ and $Y_1 = f_y(1, W, U_y)$;

3. sample which action is carried out in the given context, $A = f_a(W, U_a)$;

4. define the corresponding reward, $Y = AY_1 + (1 - A)Y_0$;

5. summarize the course of the experiment with the observation $O = (W, A, Y)$, thus concealing Y_0 and Y_1.

The above description of the experiment `draw_from_experiment` is useful to reinforce what it means to run the "ideal" experiment by setting argument `ideal` to `TRUE` in a call to `draw_from_experiment`. Doing so triggers a modification of the nature of the experiment, enforcing that the counterfactual rewards Y_0 and Y_1 be part of the summary of the experiment eventually. In light of the above enumeration, $\mathbb{O} \equiv (W, Y_0, Y_1, A, Y)$ is output, as opposed to its summary measure O. This defines another experiment and its law, that we denote $\mathbb{P}_0$.

It is straightforward to show[†] that

[†]For $a = 0, 1$,

$$
\begin{aligned}
\mathrm{E}_{\mathbb{P}_0}(Y_a) &= \int \mathrm{E}_{\mathbb{P}_0}(Y_a \mid W = w) dQ_{0,W}(w) = \int \mathrm{E}_{\mathbb{P}_0}(Y_a \mid A = a, W = w) dQ_{0,W}(w)\\
&= \int \mathrm{E}_{P_0}(Y \mid A = a, W = w) dQ_{0,W}(w) = \int \bar{Q}_0(a, W) dQ_{0,W}(w).
\end{aligned}
$$

The second equality follows from the conditional independence of the counterfactual rewards (Y_0, Y_1) and action A given W (in words, the *randomization assumption* "$(Y_0, Y_1) \perp A \mid W$" is met under $\mathbb{P}_0$). The third equality results from the facts that the observed reward Y equals the counterfactual reward Y_a when $A = a$ (in words, the *consistency assumption* "$Y_a = Y \mid A = a$" is

$$\psi_0 = \mathrm{E}_{\mathbb{P}_0}(Y_1 - Y_0) = \mathrm{E}_{\mathbb{P}_0}(Y_1) - \mathrm{E}_{\mathbb{P}_0}(Y_0). \tag{2}$$

Thus, ψ_0 describes the average difference in of the two counterfactual rewards. In other words, ψ_0 quantifies the difference in average of the reward one would get in a world where one would always enforce action $a = 1$ with the reward one would get in a world where one would always enforce action $a = 0$. This said, it is worth emphasizing that ψ_0 is a well-defined parameter beyond its causal interpretation and describes a standardized association between the action A and the reward Y.

To conclude this subsection, we use our position as oracles to sample observations from the ideal experiment. We call `draw_from_experiment` with its argument `ideal` set to `TRUE` in order to numerically approximate ψ_0. By the law of large numbers, the following code approximates ψ_0 and shows it approximate value.

```
B <- 1e6
ideal_obs <- draw_from_experiment(B, ideal = TRUE)
(psi_approx <- mean(ideal_obs[, "Yone"] - ideal_obs[, "Yzero"]))
```

```
## [1] 0.08293116
```

The object `psi_approx` contains an approximation to ψ_0 based on `B` observations from the ideal experiment. The random sampling of observations results in uncertainty in the numerical approximation of ψ_0. This uncertainty can be quantified by constructing a 95% confidence interval for ψ_0. The central limit theorem and Slutsky's lemma[‡] allow us to build such an interval as follows.

```
sd_approx <- sd(ideal_obs[, "Yone"] - ideal_obs[, "Yzero"])
alpha <- 0.05
(psi_approx_CI <- psi_approx + c(-1, 1) * qnorm(1 - alpha / 2) * sd_approx / sqrt(B))
```

```
## [1] 0.08232477 0.08353754
```

We note that the interpretation of this confidence interval is that in 95% of draws of size `B` from the ideal data generating experiment, the true value of ψ_0 will be contained in the generated confidence interval.

2.4 Difference in covariate-adjusted quantile rewards, first pass.

The problems come within the scope of Sections 2.3.

As discussed above, parameter ψ_0 (2) is the difference in average rewards if we enforce action $a = 1$ rather than $a = 0$. An alternative way to describe the rewards under different actions involves quantiles as opposed to averages.

Let $Q_{0,Y}(y, A, W) = \int_0^y q_{0,Y}(u, A, W)du$ be the conditional cumulative distribution of reward Y given A and W, evaluated at $y \in]0, 1[$, that is implied by P_0. For each action $a \in \{0, 1\}$ and $c \in]0, 1[$, introduce

$$\gamma_{0,a,c} \equiv \inf \left\{ y \in]0, 1[: \int Q_{0,Y}(y, a, w) dQ_{0,W}(w) \geq c \right\}. \tag{3}$$

met under $\mathbb{P}_0$) and that $\Pr_{P_0}(\ell\bar{G}_0(a, W) > 0) = 1$ (in words, the *positivity assumption* is met under P_0 — this is needed for $\mathrm{E}_{P_0}(Y \mid A = a, W)$ to be well-defined).

[‡]Let $X_1, \ldots, X_n$ be independently drawn from a law such that $\sigma^2 \equiv \mathrm{Var}(X_1)$ is finite. Let $m \equiv \mathrm{E}(X_1)$ and $\bar{X}_n \equiv n^{-1} \sum_{i=1}^n X_i$ be the empirical mean. It holds that $\sqrt{n}(\bar{X}_n - m)$ converges in law as n grows to the centered Gaussian law with variance σ^2. Moreover, if σ_n^2 is a (positive) consistent estimator of σ^2, then $\sqrt{n}/\sigma_n(\bar{X}_n - m)$ converges in law to the standard normal law. The empirical variance $n^{-1} \sum_{i=1}^n (X_i - \bar{X}_n)^2$ is such an estimator. In conclusion, denoting by Φ the standard normal distribution function, $[\bar{X}_n \pm \Phi^{-1}(1 - \alpha)\sigma_n/\sqrt{n}]$ is a confidence interval for m with asymptotic level $(1 - 2\alpha)$.

It is not difficult to check (see Problem 1 below) that

$$\gamma_{0,a,c} = \inf \left\{ y \in]0,1[: \Pr_{\mathbb{P}_0}(Y_a \leq y) \geq c \right\}. \tag{4}$$

Thus, $\gamma_{0,a,c}$ can be interpreted as a covariate-adjusted c-th quantile reward when action a is enforced. The difference

$$\delta_{0,c} \equiv \gamma_{0,1,c} - \gamma_{0,0,c}$$

is the c-th quantile counterpart to parameter ψ_0 (2).

1. Prove (4).

2. Compute the numerical value of $\gamma_{0,a,c}$ for each $(a,c) \in \{0,1\} \times \{1/4, 1/2, 3/4\}$ using the appropriate attributes of `five_obs`. Based on these results, report the numerical value of $\delta_{0,c}$ for each $c \in \{1/4, 1/2, 3/4\}$.

3. Approximate the numerical values of $\gamma_{0,a,c}$ for each $(a,c) \in \{0,1\} \times \{1/4, 1/2, 3/4\}$ by drawing a large sample from the "ideal" data experiment and using empirical quantile estimates. Deduce from these results a numerical approximation to $\delta_{0,c}$ for $c \in \{1/4, 1/2, 3/4\}$. Confirm that your results closely match those obtained in the previous problem.

2.5 The parameter of interest, second pass. Suppose we know beforehand that O drawn from P_0 takes its values in $\mathcal{O} \equiv [0,1] \times \{0,1\} \times [0,1]$ and that $\bar{G}(W) = P_0(A = 1|W)$ is bounded away from zero and one $Q_{0,W}$-almost surely (this is the case indeed). Then we can define model $\mathcal{M}$ as the set of all laws P on $\mathcal{O}$ such that $\bar{G}(W) \equiv P(A = 1|W)$ is bounded away from zero and one Q_W-almost surely, where Q_W is the marginal law of W under P.

Let us also define generically $\bar{Q}$ as

$$\bar{Q}(A, W) \equiv \mathrm{E}_P(Y|A, W),$$

where, for simplicity, we have suppressed the dependence of $\bar{Q}$ on P. Central to our approach is viewing ψ_0 as the value at P_0 of the statistical mapping Ψ from $\mathcal{M}$ to $[0,1]$ characterized by

$$\begin{aligned}\Psi(P) &\equiv \int \left(\bar{Q}(1,w) - \bar{Q}(0,w)\right) dQ_W(w) \\ &= \mathrm{E}_P \left(\bar{Q}(1,W) - \bar{Q}(0,W)\right),\end{aligned}$$

a clear extension of (1). For instance, although the law $\Pi_0 \in \mathcal{M}$ encoded by default (*i.e.*, with `h=0`) in `drawFromAnotherExperiment` defined below differs starkly from P_0,

```r
draw_from_another_experiment <- function(n, h = 0) {
  ## preliminary
  n <- Arguments$getInteger(n, c(1, Inf))
  h <- Arguments$getNumeric(h)
  ## ## 'Gbar' and 'Qbar' factors
  Gbar <- function(W) {
    sin((1 + W) * pi / 6)
  }
  Qbar <- function(AW, hh = h) {
    A <- AW[, 1]
    W <- AW[, 2]
    expit( logit( A *  W + (1 - A) * W^2 ) +
           hh * 10 * sqrt(W) * A )
  }
  ## sampling
  ## ## context
  W <- runif(n, min = 1/10, max = 9/10)
  ## ## action undertaken
  A <- rbinom(n, size = 1, prob = Gbar(W))
  ## ## reward
  shape1 <- 4
  QAW <- Qbar(cbind(A, W))
  Y <- rbeta(n, shape1 = shape1, shape2 = shape1 * (1 - QAW) / QAW)
  ## ## observation
  obs <- cbind(W = W, A = A, Y = Y)
  attr(obs, "Gbar") <- Gbar
  attr(obs, "Qbar") <- Qbar
  attr(obs, "QW") <- function(x){dunif(x, min = 1/10, max = 9/10)}
  attr(obs, "shape1") <- shape1
  attr(obs, "qY") <- function(AW, Y, Qbar, shape1){
    A <- AW[,1]; W <- AW[,2]
    Qbar.AW <- do.call(Qbar, list(AW))
    dbeta(Y, shape1 = shape1, shape2 = shape1 * (1 - Qbar.AW) / Qbar.AW)
  }
  ##
  return(obs)
}
```

the parameter $\Psi(\Pi_0)$ is well defined, and numerically approximated by `psi_Pi_zero` as follows.

```r
five_obs_from_another_experiment <- draw_from_another_experiment(5)
another_integrand <- function(w) {
  Qbar <- attr(five_obs_from_another_experiment, "Qbar")
  QW <- attr(five_obs_from_another_experiment, "QW")
  ( Qbar(cbind(1, w)) - Qbar(cbind(0, w)) ) * QW(w)
}
(psi_Pi_zero <- integrate(another_integrand, lower = 0, upper = 1)$val)
```

```
## [1] 0.1966687
```

Straightforward algebra confirms that indeed $\Psi(\Pi_0) = 59/300$.

2.6 Difference in covariate-adjusted quantile rewards, second pass. We continue with the exercise from Section 2.4. The problems come within the scope of Section 2.3.

As above, we define $q_Y(y, A, W)$ to be the conditional density of Y given A and W, evaluated at y, that is implied by a generic $P \in \mathcal{M}$. Similarly, we use Q_Y to denote the corresponding cumulative distribution function. The covariate-adjusted c-th quantile reward for action $a \in \{0, 1\}$ may be viewed as a mapping $\Gamma_{a,c}$ from $\mathcal{M}$ to $[0, 1]$ characterized by

$$\Gamma_{a,c}(P) = \inf\left\{y \in]0, 1[: \int Q_Y(y, a, w) dQ_W(w) \geq c\right\}.$$

The difference in c-th quantile rewards may similarly be viewed as a mapping Δ_c from $\mathcal{M}$ to $[0, 1]$, characterized by $\Delta_c(P) \equiv \Gamma_{1,c}(P) - \Gamma_{0,c}(P)$.

1. Compute the numerical value of $\Gamma_{a,c}(\Pi_0)$ for $(a, c) \in \{0, 1\} \times \{1/4, 1/2, 3/4\}$ using the appropriate attributes of `five_obs_from_another_experiment`. Based on these results, report the numerical value of $\Delta_c(\Pi_0)$ for each $c \in \{1/4, 1/2, 3/4\}$.

2. Approximate the value of $\Gamma_{0,a,c}(\Pi_0)$ for $(a, c) \in \{0, 1\} \times \{1/4, 1/2, 3/4\}$ by drawing a large sample from the "ideal" data experiment and using empirical quantile estimates. Deduce from these results a numerical approximation to $\Delta_{0,c}(\Pi_0)$ for each $c \in \{1/4, 1/2, 3/4\}$. Confirm that your results closely match those obtained in the previous problem.

3. Building upon the code you wrote to solve the previous problem, construct a confidence interval with asymptotic level 95% for $\Delta_{0,c}(\Pi_0)$, with $c \in \{1/4, 1/2, 3/4\}$.[§]

2.7 The parameter of interest, third pass. In the previous subsection, we reoriented our view of the target parameter to be a statistical functional of the law of the observed data. Specifically, we viewed the parameter as a function of specific features of the observed data law, namely Q_W and $\bar{Q}$. It is straightforward[¶] to show an equivalent representation of the parameter as

$$\begin{aligned}\psi_0 &= \int \frac{2a - 1}{\ell\bar{G}_0(a, w)} y dP_0(w, a, y) \\ &= \mathrm{E}_{P_0}\left(\frac{2A - 1}{\ell\bar{G}_0(A, W)} Y\right). \end{aligned} \tag{5}$$

Viewing again the parameter as a statistical mapping from $\mathcal{M}$ to $[0, 1]$, it also holds that

[§]Let $X_1, \ldots, X_n$ be independently drawn from a continuous distribution function F. Set $p \in]0, 1[$ and, assuming that n is large, find $k \geq 1$ and $l \geq 1$ such that $k/n \approx p - \Phi^{-1}(1 - \alpha)\sqrt{p(1 - p)/n}$ and $l/n \approx p + \Phi^{-1}(1 - \alpha)\sqrt{p(1 - p)/n}$, where Φ is the standard normal distribution function. Then $[X_{(k)}, X_{(l)}]$ is a confidence interval for $F^{-1}(p)$ with asymptotic level $1 - 2\alpha$.

[¶]We temporarily drop the subscript P_0 to save space and note, for the same reason, that $(2a - 1)$ equals 1 if $a = 1$ and -1 if $a = 0$. Now, for each $a = 0, 1$,

$$\begin{aligned}\mathrm{E}\left(\frac{\mathbf{1}\{A = a\}Y}{\ell\bar{G}(a, W)}\right) &= \mathrm{E}\left(\mathrm{E}\left(\frac{\mathbf{1}\{A = a\}Y}{\ell\bar{G}(a, W)}\middle| A, W\right)\right) = \mathrm{E}\left(\frac{\mathbf{1}\{A = a\}}{\ell\bar{G}(a, W)}\bar{Q}(A, W)\right) = \mathrm{E}\left(\frac{\mathbf{1}\{A = a\}}{\ell\bar{G}(a, W)}\bar{Q}(a, W)\right) \\ &= \mathrm{E}\left(\mathrm{E}\left(\frac{\mathbf{1}\{A = a\}}{\ell\bar{G}(a, W)}\bar{Q}(a, W)\middle| W\right)\right) = \mathrm{E}\left(\frac{\ell\bar{G}(a, W)}{\ell\bar{G}(a, W)}\bar{Q}(a, W)\middle| W\right) = \mathrm{E}\left(\bar{Q}(a, W)\right),\end{aligned}$$

where the first, fourth and sixth equalities follow from the tower rule, and the second and fifth hold by definition of the conditional expectation, This completes the proof.

$$
\begin{aligned}
\Psi(P) &= \int \frac{2a-1}{\ell\bar{G}(a,w)} y dP(w,a,y) \\
&= \mathrm{E}_P\left(\frac{2A-1}{\ell\bar{G}_0(A,W)} Y\right). \qquad (6)
\end{aligned}
$$

Our reason for introducing this alternative view of the target parameter will become clear when we discuss estimation of the target parameter. Specifically, the representations (1) and (5) naturally suggest different estimation strategies for ψ_0. The former suggests building an estimator of ψ_0 using estimators of $\bar{Q}_0$ and of $Q_{W,0}$. The latter suggests building an estimator of ψ_0 using estimators of $\ell\bar{G}_0$ and of P_0. We return to these ideas in later sections.

2.8 Difference in covariate-adjusted quantile rewards, third pass. We continue with the exercise from Section 2.4.

1. Show that for $a' = 0, 1$, $\gamma_{0,a',c}$ as defined in (3) can be equivalently expressed as

$$
\inf\left\{z \in]0,1[: \int \frac{\mathbf{1}\{a=a'\}}{\ell\bar{G}(a',W)} \mathbf{1}\{y \leq z\} dP_0(w,a,y) \geq c\right\}.
$$

2.9 Smooth parameters, first pass. Within our view of the target parameter as a statistical mapping, it is natural to inquire of properties this functional enjoys. For example, we may be interested in asking how the value of $\Psi(P)$ changes for laws that are, in a loose sense, near to P in $\mathcal{M}$. If small changes in the law result in large changes in the parameter value, then we might hypothesize that it will be difficult to produce "good estimators" of ψ_0. Fortunately, this turns out not to be the case for Ψ, and so we say that Ψ is a smooth parameter mapping. This colloquial expression refers to the fact that, for each $P \in \mathcal{M}$, if $P_h \to_h P$ in $\mathcal{M}$ from a direction s when the real parameter $h \to 0$ (the notion of convergence from a direction is formalized in Section 2.10), then not only $\Psi(P_h) \to_h \Psi(P)$ (continuity), but also $h^{-1}[\Psi(P_h) - \Psi(P)] \to_h c$, where the real number c depends on P and s (differentiability).

For instance, let $\Pi_h \in \mathcal{M}$ be the law encoded in `draw_from_another_experiment` with `h` ranging over $[-1,1]$. We will argue shortly that $\Pi_h \to_h \Pi_0$ in $\mathcal{M}$ from a direction s when $h \to 0$. The following chunk of code evaluates and represents $\Psi(\Pi_h)$ for h ranging in a discrete approximation of $[-1,1]$:

```
approx <- seq(-1, 1, length.out = 1e2)
psi_Pi_h <- sapply(approx, function(t) {
  obs_from_another_experiment <- draw_from_another_experiment(1, h = t)
  integrand <- function(w) {
    Qbar <- attr(obs_from_another_experiment, "Qbar")
    QW <- attr(obs_from_another_experiment, "QW")
    ( Qbar(cbind(1, w)) - Qbar(cbind(0, w)) ) * QW(w)
  }
  integrate(integrand, lower = 0, upper = 1)$val
})
slope_approx <- (psi_Pi_h - psi_Pi_zero) / approx
slope_approx <- slope_approx[min(which(approx > 0))]
ggplot() +
  geom_point(data = data.frame(x = approx, y = psi_Pi_h), aes(x, y),
             color = "#CC6666") +
```

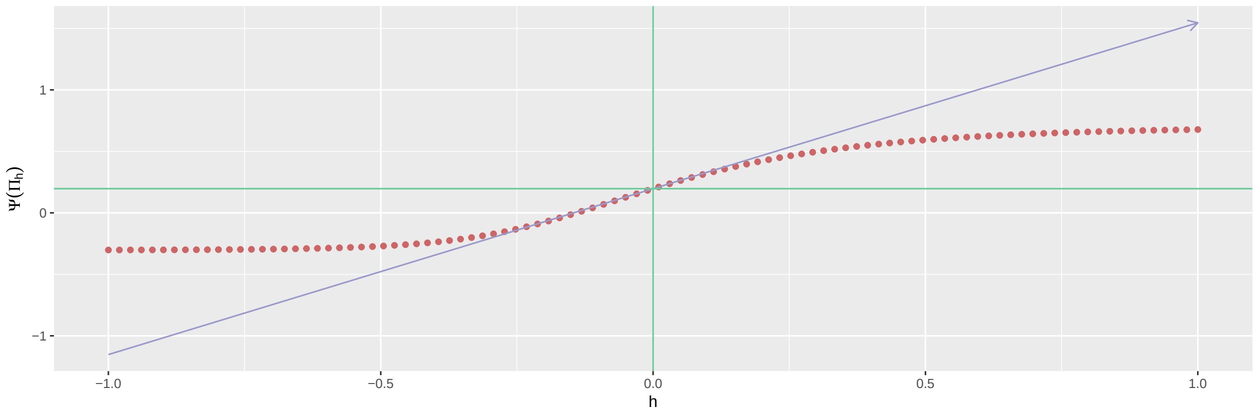


Figure 1: Evolution of statistical parameter Ψ along fluctuation $\{\Pi_h : h \in H\}$.

```
geom_segment(aes(x = -1, y = psi_Pi_zero - slope_approx,
                 xend = 1, yend = psi_Pi_zero + slope_approx),
             arrow = arrow(length = unit(0.03, "npc")),
             color = "#9999CC") +
geom_vline(xintercept = 0, color = "#66CC99") +
geom_hline(yintercept = psi_Pi_zero, color = "#66CC99") +
labs(x = "h", y = expression(Psi(Pi[h])))
```

The dotted curve represents the function $h \mapsto \Psi(\Pi_h)$. The blue line represents the tangent to the previous curve at $h = 0$, which is indeed differentiable around $h = 0$. It is derived by simple geometric arguments. In the next subsection, we formalize what it means to be smooth for the statistical mapping Ψ. This allows us to derive a closed-form expression for the slope of the blue curve from the chunk of code where `draw_from_another_experiment` is defined.

2.10 Being smooth, second pass. Let us now describe what it means for statistical mapping Ψ to be smooth at every $P \in \mathcal{M}$. To describe the rate of change of a statistical parameter at P as P is approached in model $\mathcal{M}$, we must formalize what is meant by "approach P". Toward that end, we introduce the idea of fluctuations of P. For every $h \in H \equiv \,]-M^{-1}, M^{-1}[$, we can define a law $P_h \in \mathcal{M}$ by setting $P_h \ll P$[‖] and

$$\frac{dP_h}{dP} \equiv 1 + hs, \tag{7}$$

where $s : \mathcal{O} \to \mathbb{R}$ is a P-measurable function of O such that $s(O)$ is not equal to zero P-almost surely, $\mathrm{E}_P(s(O)) = 0$, and s bounded by M. We make the observation that

$$(i)\ P_h|_{h=0} = P, \quad (ii)\ \frac{d}{dh} \log \frac{dP_h}{dP}(O)\Big|_{h=0} = s(O). \tag{8}$$

Because of *(i)*, we say that $\{P_h : h \in H\}$ is a *fluctuation* of P. The fluctuation is a one-dimensional submodel of $\mathcal{M}$ with univariate parameter $h \in H$. We note that *(ii)* indicates that the score of this submodel at $h = 0$ is s. Thus, we say that the fluctuation is *in the direction* of s. Fluctuations of P do not necessarily take the

[‖] That is, P_h is dominated by P: if an event A satisfies $P(A) = 0$, then necessarily $P_h(A) = 0$ too.

same form as in (7). No matter how the fluctuation is built, what really matters is its local shape in the neighborhood of P.

We are now prepared to provide a formal definition of smoothness of statistical mappings. Please check these modifications. We say that a statistical mapping Ψ is smooth at every $P \in \mathcal{M}$ if for each $P \in \mathcal{M}$, there exists a P-measurable function $D^*(P) : \mathcal{O} \to \mathbb{R}$ such that $\mathrm{E}_P(D^*(P)(O)) = 0$, $\mathrm{Var}_P(D^*(P)(O))$ finite and, for every fluctuation $\{P_h : h \in H\}$ with score s at $h = 0$, the real-valued mapping $h \mapsto \Psi(P_h)$ is differentiable at $h = 0$, with a derivative equal to

$$\mathrm{E}_P\left(D^*(P)(O)s(O)\right). \tag{9}$$

Interestingly, if a fluctuation $\{P_h : h \in H\}$ satisfies (8) for a direction s such that $s \neq 0$, $\mathrm{E}_P(s(O)) = 0$ and $\mathrm{Var}_P(s(O)) < \infty$, then $h \mapsto \Psi(P_h)$ is still differentiable at $h = 0$ with a derivative equal to (9) (beyond fluctuations of the form (7)). Maybe this needs reconciliation with my above modifications, as I'm defining things for a general fluctuation. Hmm...

The object $D^*(P)$ in (9) is called a gradient of Ψ at P. This terminology has a direct parallel to directional derivatives in the calculus of Euclidean geometry. That is, if $f : \mathbb{R}^p \to \mathbb{R}$, u is a unit vector in $\mathbb{R}^p$, and x is a point in $\mathbb{R}^p$, the directional derivative at x in the direction u is the dot product of the gradient of f and u. That is, the derivative can be represented as an inner product of the direction that we approach x and the change of the function's value at x. In the present problem, the law P is the point at which we evaluate the function, the score s of the fluctuation is the "direction" in which we approach the point, and the gradient describes the change in the function's value at the point.

Need a transition to calling this an influence function or influence curve. Right now, it seems a little out of place to be talking about estimation at this point since the section is all about being smooth, which is a property of parameters, not estimators. Maybe we move the next paragraphs to the following section and re-title it something like "Influence functions and efficient influence functions" or something? Not sure. Need to think about it more.

The influence curves $D^*(P)$ convey valuable information about Ψ. For instance, an important result from the theory of inference based on semiparametric models guarantees that if ψ_n is a regular** estimator of $\Psi(P)$ built from n independent observations drawn from P, then the asymptotic variance of the centered and rescaled $\sqrt{n}(\psi_n - \Psi(P))$ cannot be smaller than the variance of the P-specific efficient influence curve, that is,

$$\mathrm{Var}_P(D^*(P)(O)). \tag{10}$$

In this light, an estimator ψ_n of $\Psi(P)$ is said *asymptotically efficient* at P if it is regular at P and such that $\sqrt{n}(\psi_n - \Psi(P))$ converges in law to the centered Gaussian law with variance (10), which is called the Cramér-Rao bound.

2.11 The efficient influence curve. It is not difficult to check (do we give the proof?) that the efficient influence curve $D^*(P)$ of Ψ at $P \in \mathcal{M}$ writes as $D^*(P) \equiv D_1^*(P) + D_2^*(P)$ where $D_1^*(P)$ and $D_2^*(P)$ are given by

**We can view ψ_n as the by product of an algorithm $\widehat{\Psi}$ trained on independent observations $O_1, \ldots, O_n$ drawn from P. The estimator is regular at P (w.r.t. the maximal tangent space) if, for any direction $s \neq 0$ such that $\mathrm{E}_P(s(O)) = 0$ and $\mathrm{Var}_P(s(O)) < \infty$ and fluctuation $\{P_h : h \in H\}$ satisfying (8), the estimator $\psi_{n,1/\sqrt{n}}$ of $\Psi(P_{1/\sqrt{n}})$ obtained by training $\widehat{\Psi}$ on independent observations $O_1, \ldots, O_n$ drawn from $P_{1/\sqrt{n}}$ is such that $\sqrt{n}(\psi_{n,1/\sqrt{n}} - \Psi(P_{1/\sqrt{n}}))$ converges in law to a limit that does not depend on s.

$$D_1^*(P)(O) \equiv \bar{Q}(1,W) - \bar{Q}(0,W) - \Psi(P),$$
$$D_2^*(P)(O) \equiv \frac{2A-1}{\ell\bar{G}(A,W)}(Y - \bar{Q}(A,W)),$$

with shorthand notation $\ell\bar{G}(A,W) \equiv A\bar{G}(W) + (1-A)(1-\bar{G}(W))$. The following chunk of code enables the computation of the values of the efficient influence curve $D^*(P)$ at observations drawn from P (note that it is necessary to provide the value of $\Psi(P)$, or a numerical approximation thereof, through argument `psi`).

```
eic <- function(obs, psi) {
  Qbar <- attr(obs, "Qbar")
  Gbar <- attr(obs, "Gbar")
  QAW <- Qbar(obs[, c("A", "W")])
  QoneW <- Qbar(cbind(A = 1, W = obs[, "W"]))
  QzeroW <- Qbar(cbind(A = 0, W = obs[, "W"]))
  GW <- Gbar(obs[, "W", drop = FALSE])
  lGAW <- obs[, "A"] * GW + (1 - obs[, "A"]) * (1 - GW)
  (QoneW - QzeroW - psi) + (2 * obs[, "A"] - 1) / lGAW * (obs[, "Y"] - QAW)
}

(eic(five_obs, psi = psi_approx))
```

```
##               W
## [1,] -1.0154401
## [2,] -0.9019670
## [3,] -0.3260029
## [4,]  0.2457730
## [5,] -1.2678877
```

```
(eic(five_obs_from_another_experiment, psi = psi_Pi_zero))
```

```
##                W
## [1,] -1.14434889
## [2,] -0.04617269
## [3,]  0.15717587
## [4,]  0.38152278
## [5,]  0.09369052
```

2.12 Computing and comparing Cramér-Rao bounds. We can use `eic` to numerically approximate the Cramér-Rao bound at P_0:

```
obs <- draw_from_experiment(B)
(cramer_rao_hat <- var(eic(obs, psi = psi_approx)))
```

```
##           W
## W 0.2874799
```

and the Cramér-Rao bound at Π_0:

```
obs_from_another_experiment <- draw_from_another_experiment(B)
(cramer_rao_Pi_zero_hat <- var(eic(obs_from_another_experiment, psi = 59/300)))
```

```
##            W
## W 0.09476356
```

```
(ratio <- sqrt(cramer_rao_Pi_zero_hat/cramer_rao_hat))
```

```
##           W
## W 0.5741389
```

We thus discover that of the statistical parameters $\Psi(P_0)$ and $\Psi(\Pi_0)$, the latter is easier to target than the former. Heuristically, for large sample sizes, the narrowest (efficient) confidence intervals for $\Psi(\Pi_0)$ are approximately 0.57 (rounded to two decimal places) smaller than their counterparts for $\Psi(P_0)$.

2.13 Revisiting Section 2.9. It is not difficult either (though a little cumbersome) (do we give the proof? I'd rather not) to verify that $\{\Pi_h : h \in [-1,1]\}$ is a fluctuation of Π_0 in the direction of σ_0 (in the sense of (7)) given, up to a constant, by

$$\sigma_0(O) \equiv -10\sqrt{W}A \times \beta_0(A,W)\left(\log(1-Y) + \sum_{k=0}^{3}(k+\beta_0(A,W))^{-1}\right) + \text{constant},$$

$$\text{where } \beta_0(A,W) \equiv \frac{1-\bar{Q}_{\Pi_0}(A,W)}{\bar{Q}_{\Pi_0}(A,W)}.$$

Consequently, the slope of the dotted curve in Figure 1 is equal to

$$\mathrm{E}_{\Pi_0}(D^*(\Pi_0)(O)\sigma_0(O)) \tag{11}$$

(since $D^*(\Pi_0)$ is centered under Π_0, knowing σ_0 up to a constant is not problematic).

Let us check this numerically. In the next chunk of code, we implement direction σ_0 with `sigma0_draw_from_another_experiment`, then we numerically approximate (11) (pointwise and with a confidence interval of asymptotic level 95%):

```
sigma0_draw_from_another_experiment <- function(obs) {
  ## preliminary
  Qbar <- attr(obs, "Qbar")
  QAW <- Qbar(obs[, c("A", "W")])
  shape1 <- Arguments$getInteger(attr(obs, "shape1"), c(1, Inf))
  ## computations
  betaAW <- shape1 * (1 - QAW) / QAW
  out <- log(1 - obs[, "Y"])
  for (int in 1:shape1) {
    out <- out + 1/(int - 1 + betaAW)
  }
  out <- - out * shape1 * (1 - QAW) / QAW * 10 * sqrt(obs[, "W"]) * obs[, "A"]
  ## no need to center given how we will use it
  return(out)
}

vars <- eic(obs_from_another_experiment, psi = 59/300) *
  sigma0_draw_from_another_experiment(obs_from_another_experiment)
```

```
sd_hat <- sd(vars)
(slope_hat <- mean(vars))
```

```
## [1] 1.362463
```

```
(slope_CI <- slope_hat + c(-1, 1) * qnorm(1 - alpha / 2) * sd_hat / sqrt(B))
```

```
## [1] 1.357179 1.367746
```

Equal to 1.349 (rounded to three decimal places), the first numerical approximation `slope_approx` is not too off.

2.14 Double-robustness The efficient influence curve $D^*(P)$ at $P \in \mathcal{M}$ enjoys another remarkable property: it is double-robust. Specifically, if we define for all $P' \in \mathcal{M}$

$$\mathrm{Rem}_P(\bar{Q}', \bar{G}') \equiv \Psi(P') - \Psi(P) + \mathrm{E}_P(D^*(P')(O)), \tag{12}$$

then the so called remainder term $\mathrm{Rem}_P(\bar{Q}', \bar{G}')$ satisfies[††]

$$\mathrm{Rem}_P(\bar{Q}', \bar{G}')^2 \leq \|\bar{Q}' - \bar{Q}\|_P^2 \times \|(\bar{G}' - \bar{G})/\ell\bar{G}'\|_P^2. \tag{13}$$

In particular, if

$$\mathrm{E}_P(D^*(P')(O)) = 0, \tag{14}$$

and *either* $\bar{Q}' = \bar{Q}$ *or* $\bar{G}' = \bar{G}$, then $\mathrm{Rem}_P(\bar{Q}', \bar{G}') = 0$ hence $\Psi(P') = \Psi(P)$. In words, if P' solves the so called P-specific efficient influence curve equation (14) and if, in addition, P' has the same $\bar{Q}$-component or $\bar{G}$-component as P, then $\Psi(P') = \Psi(P)$ no matter how P' may differ from P otherwise. This property is useful to build consistent estimators of $\Psi(P)$.

However, there is much more to double-robustness than the above straightforward implication. Indeed, 12 is useful to build a consistent etimator of $\Psi(P)$ that, in addition, satisfies a central limit theorem and thus lends itsef to the construction of confidence intervals.

Let $P_n^0 \in \mathcal{M}$ be an element of model $\mathcal{M}$ of which the choice is data-driven, based on observing n independent draws from P. Equality 12 reveals that the statistical behavior of the corresponding *substitution* estimator $\psi_n^0 \equiv \Psi(P_n^0)$ is easier to analyze when the remainder term $\mathrm{Rem}_P(\bar{Q}_n^0, \bar{G}_n^0)$ goes to zero at a fast (relative to n) enough rate. In light of 13, this happens if the features $\bar{Q}_n^0$ and $\bar{G}_n^0$ of P_n^0 converge to their counterparts under P at rates of which *the product* is fast enough.

2.15 Inference assuming $\bar{G}_0$ known, or not, first pass. Let $O_1, \ldots, O_n$ be a sample of independent observations drawn from P_0. Let P_n be the corresponding empirical measure, *i.e.*, the law consisting in drawing one among $O_1, \ldots, O_n$ with equal probabilities n^{-1}.

Let us assume for a moment that we know $\bar{G}_0$. This may be the case indeed if P_0 was a controlled experiment. Note that, on the contrary, assuming $\bar{Q}_0$ known would be difficult to justify.

[††] For any (measurable) $f : \mathcal{O} \to \mathbb{R}$, we denote $\|f\|_P = \mathrm{E}_P(f(O)^2)^{1/2}$.

```r
Gbar <- attr(obs, "Gbar")

iter <- 1e3
```

Then, the alternative expression 5 suggests to estimate ψ_0 with

$$\psi_n^b \equiv \mathrm{E}_{P_n}\left(\frac{2A-1}{\ell\bar{G}_0(A,W)}Y\right) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{2A_i-1}{\ell\bar{G}_0(A_i,W_i)}Y_i\right). \tag{15}$$

Note how P_n is substituted for P_0 in (15) relative to (5).

It is easy to check that ψ_n^b estimates ψ_0 consistently, but this is too little to request from an estimator of ψ_0. Better, ψ_n^b also satisfies a central limit theorem: $\sqrt{n}(\psi_n^b - \psi_0)$ converges in law to a centered Gaussian law with asymptotic variance

$$v^b \equiv \mathrm{Var}_{P_0}\left(\frac{2A-1}{\ell\bar{G}_0(A,W)}Y\right),$$

where v^b can be consistently estimated by its empirical counterpart

$$v_n^b \equiv \mathrm{Var}_{P_n}\left(\frac{2A-1}{\ell\bar{G}_0(A,W)}Y\right) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{2A_i-1}{\ell\bar{G}_0(A_i,W_i)}Y_i - \psi_n^b\right)^2. \tag{16}$$

Let us investigate how ψ_n^b behaves based on `obs`. Because we are interested in the *law* of ψ_n^b, the next chunk of code constitutes `iter =` 1000 independent samples of independent observations drawn from P_0, each consisting of n equal to `nrow(obs)/iter =` 1000 data points, and computes the realization of ψ_n^b on all samples.

Before proceeding, let us introduce

$$\begin{aligned}
\psi_n^a &\equiv \mathrm{E}_{P_n}(Y|A=1) - \mathrm{E}_{P_n}(Y|A=0) \\
&= \frac{1}{n_1}\sum_{i=1}^{n}\mathbf{1}\{A_i=1\}Y_i - \frac{1}{n_0}\sum_{i=1}^{n}\mathbf{1}\{A_i=0\}Y_i \\
&= \frac{1}{n_1}\sum_{i=1}^{n}A_iY_i - \frac{1}{n_0}\sum_{i=1}^{n}(1-A_i)Y_i,
\end{aligned}$$

where $n_1 = \sum_{i=1}^{n} A_i = n - n_0$ is the number of observations O_i such that $A_i = 1$. It is an estimator of

$$\mathrm{E}_{P_0}(Y|A=1) - \mathrm{E}_{P_0}(Y|A=0).$$

We seize this opportunity to demonstrate numerically the obvious fact that ψ_n^a does not estimate ψ_0.

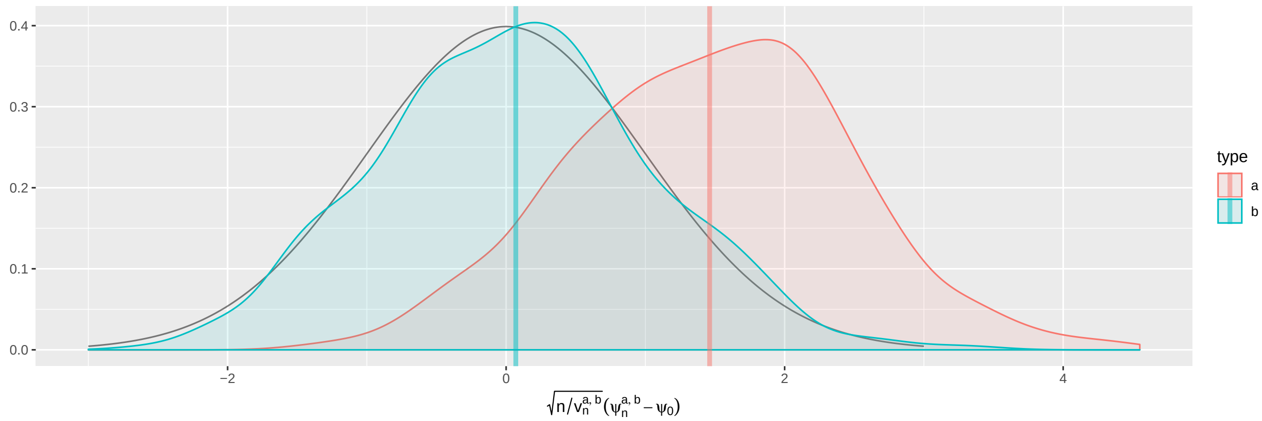


Figure 2: Kernel density estimators of the law of two estimators of ψ_0 (recentered with respect to ψ_0, and renormalized), one of them misconceived (a), the other assuming that $\bar{G}_0$ is known (b). Built based on `iter` independent realizations of each estimator.

```
psi_hat_ab <- obs %>% as_tibble() %>% mutate(id = 1:n() %% iter) %>%
  mutate(lGAW = A * Gbar(W) + (1 - A) * (1 - Gbar(W))) %>% group_by(id) %>%
  summarize(est_a = mean(Y[A==1]) - mean(Y[A==0]),
            est_b = mean(Y * (2 * A - 1) / lGAW),
            std_b = sd(Y * (2 * A - 1) / lGAW) / sqrt(n()),
            clt_b = (est_b - psi_approx) / std_b) %>%
  mutate(std_a = sd(est_a),
         clt_a = (est_a - psi_approx) / std_a) %>%
  gather("key", "value", -id) %>%
  extract(key, c("what", "type"), "([^_]+)_([ab])") %>%
  spread(what, value)

(bias_ab <- psi_hat_ab %>% group_by(type) %>% summarise(bias = mean(clt)))
```

```
## # A tibble: 2 x 2
##   type    bias
##   <chr>  <dbl>
## 1 a     1.46
## 2 b     0.0696
```

```
fig <- ggplot() +
  geom_line(aes(x = x, y = y),
            data = tibble(x = seq(-3, 3, length.out = 1e3),
                          y = dnorm(x)),
            linetype = 1, alpha = 0.5) +
  geom_density(aes(clt, fill = type, colour = type),
               psi_hat_ab, alpha = 0.1) +
  geom_vline(aes(xintercept = bias, colour = type),
             bias_ab, size = 1.5, alpha = 0.5)

fig +
  labs(y = "",
       x = expression(paste(sqrt(n/v[n]^{list(a, b)})*(psi[n]^{list(a, b)} - psi[0]))))
```

Let v_n^a be n times the empirical variance of the `iter` realizations of ψ_n^a. By the above chunk of code, the averages of $\sqrt{n/v_n^a}(\psi_n^a - \psi_0)$ and $\sqrt{n/v_n^b}(\psi_n^b - \psi_0)$ computed across the realizations of the two estimators are respectively equal to 1.461 and 0.07 (both rounded to three decimal places — see `bias_ab`). Interpreted as amounts of bias, those two quantities are represented by vertical lines in Figure 2. The red and blue bell-shaped curves represent the empirical laws of ψ_n^a and ψ_n^b (recentered with respect to psi_0, and renormalized) as estimated by kernel density estimation. The latter is close to the black curve, which represents the standard normal density.

2.16 Inference assuming $\bar{G}_0$ known, or not, second pass. At the beginning of Section 2.15, we assumed that $\bar{G}_0$ was known. Let us suppose now that it is not. The definition of ψ_n^b can be adapted to overcome this difficulty, by substituting an estimator of $\ell\bar{G}_0$ for $\ell\bar{G}_0$ in (15).

For simplicity, we consider the case that $\bar{G}_0$ is estimated by minimizing a loss function on a single working model, both fine-tune-parameter-free. By adopting this stance, we exclude estimating procedures that involve penalization (*e.g.* the LASSO) or aggregation of competing estimators (*via* stacking/super learning) – see Section 2.17. Defined in the next chunk of code, the generic function `estimate_G` fits a user-specified working model by minimizing the empirical risk associated to the user-specified loss function and provided data.

Comment on new structure of `estimate_G` and say a few words about `compute_lGhatAW`.

```
estimate_G <- function(dat, algorithm, ...) {
  if (!attr(algorithm, "ML")) {
    dat <- as.data.frame(dat)
    fit <- algorithm[[1]](formula = algorithm[[2]], data = dat)
    Ghat <- function(newdata) {
      newdata <- as.data.frame(newdata)
      predict(fit, newdata, type = "response")
    }
  } else {
    fit <- algorithm(dat, ...)
    Qhat <- function(newdata) {
      caret::predict.train(fit, newdata)
    }
  }
  return(Ghat)
}

compute_lGhatAW <- function(A, W, Ghat, threshold = 0.05) {
  dat <- data.frame(A = A, W = W)
  Ghat_W <- Ghat(dat)
  lGAW <- A * Ghat_W + (1 - A) * (1 - Ghat_W)
  pred <- pmin(1 - threshold, pmax(lGAW, threshold))
  return(pred)
}
```

Note how the prediction of any $\ell\bar{G}_0(A, W)$ is manually bounded away from 0 and 1 at the last but one line of `compute_lGhatAW`. This is desirable because the *inverse* of each $\ell\bar{G}_0(A_i, W_i)$ appears in the definition of ψ_n^b (15).

For sake of illustration, we choose argument `working_model_G_one` of function `estimate_G` as follows:

```
working_model_G_one <- list(
  model = function(...) {glm(family = binomial(), ...)},
  formula = as.formula(
    paste("A ~",
```

```
          paste(c("I(W^", "I(abs(W - 5/12)^"),
                rep(seq(1/2, 3/2, by = 1/2), each = 2),
                sep = "", collapse = ") + "),
          ")")
  ))
attr(working_model_G_one, "ML") <- FALSE
working_model_G_one$formula
```

```
## A ~ I(W^0.5) + I(abs(W - 5/12)^0.5) + I(W^1) + I(abs(W - 5/12)^1) +
##     I(W^1.5) + I(abs(W - 5/12)^1.5)
```

In words, we choose the so called logistic (or negative binomial) loss function L_a given by

$$-L_a(f)(A, W) \equiv A \log f(W) + (1 - A) \log(1 - f(W)) \tag{17}$$

for any function $f : [0, 1] \to [0, 1]$ paired with the working model $\mathcal{F} \equiv \left\{f_\theta : \theta \in \mathbb{R}^5\right\}$ where, for any $\theta \in \mathbb{R}^5$, $\text{logit}\, f_\theta(W) \equiv \theta_0 + \sum_{j=1}^4 \theta_j W^{j/2}$. The working model is well specified: it happens that $\bar{G}_0$ is the unique minimizer of the risk entailed by L_a over $\mathcal{F}$:

$$\bar{G}_0 = \mathop{\arg\min}_{f_\theta \in \mathcal{F}} \mathrm{E}_{P_0} \left(L_a(f_\theta)(A, W)\right).$$

Therefore, the estimator $\bar{G}_n$ output by `estimate_G` and obtained by minimizing the empirical risk

$$\mathrm{E}_{P_n} \left(L_a(f_\theta)(A, W)\right) = \frac{1}{n} \sum_{i=1}^n L_a(f_\theta)(A_i, W_i)$$

over $\mathcal{F}$ consistently estimates $\bar{G}_0$.

In light of (15), introduce

$$\psi_n^c \equiv \frac{1}{n} \sum_{i=1}^n \left(\frac{2A_i - 1}{\ell\bar{G}_n(A_i, W_i)} Y_i\right). \tag{18}$$

Because $\bar{G}_n$ minimizes the empirical risk over a finite-dimensional and well-specified working model, $\sqrt{n}(\psi_n^c - \psi_0)$ converges in law to a centered Gaussian law. Let us compute ψ_n^c on the same `iter` = 1000 independent samples of independent observations drawn from P_0 as in Section 2.15:

```
if (redo_fixed) {
  learned_features_fixed_sample_size <-
    obs %>% as_tibble() %>%
    mutate(id = 1:n() %% iter) %>%
    nest(-id, .key = "obs") %>%
    mutate(Ghat = map(obs, ~ estimate_G(., algorithm = working_model_G_one))) %>%
    mutate(lGAW = map2(Ghat, obs, ~ compute_lGhatAW(.y$A, .y$W, .x)))
}
```

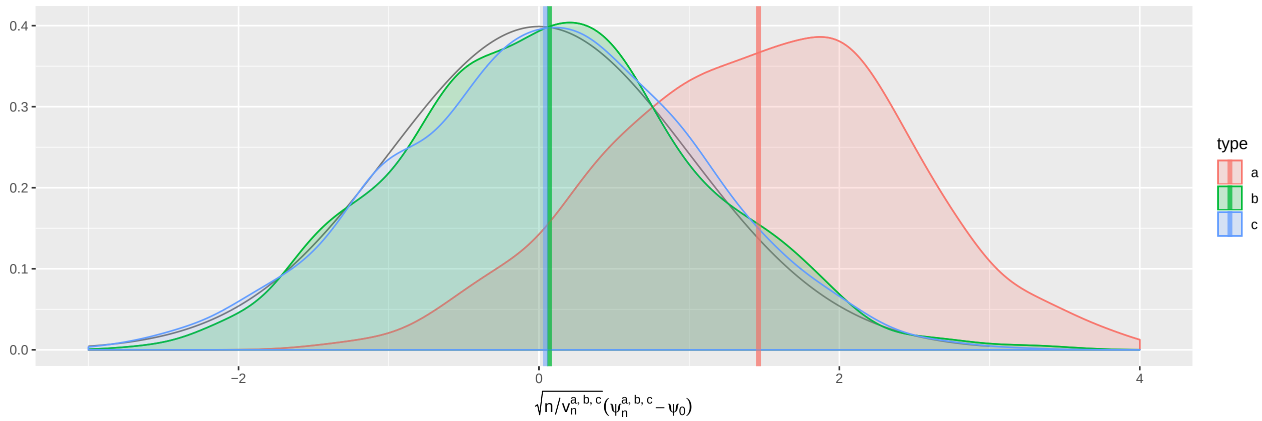


Figure 3: Kernel density estimators of the law of three estimators of ψ_0 (recentered with respect to ψ_0, and renormalized), one of them misconceived (a), one assuming that $\bar{G}_0$ is known (b) and one that hinges on the estimation of $\bar{G}_0$ (c). The present figure includes Figure 2 (but the colors differ). Built based on `iter` independent realizations of each estimator.

```
psi_hat_abc <-
  learned_features_fixed_sample_size %>%
  unnest(obs, lGAW) %>%
  group_by(id) %>%
  summarize(est = mean(Y * (2 * A - 1) / lGAW)) %>%
  mutate(std = sd(est),
         clt = (est - psi_approx) / std,
         type = "c") %>%
  full_join(psi_hat_ab)

(bias_abc <- psi_hat_abc %>% group_by(type) %>% summarise(bias = mean(clt)))
```

```
## # A tibble: 3 x 2
##   type    bias
##   <chr>  <dbl>
## 1 a     1.46
## 2 b     0.0696
## 3 c     0.0435
```

Note how we exploit the independent realizations of ψ_n^c to estimate the asymptotic variance of the estimator with v_n^c/n. By the above chunk of code, the average of $\sqrt{n/v_n^c}(\psi_n^c - \psi_0)$ computed across the realizations is equal to 0.043 (rounded to three decimal places — see `bias_abc`). We represent the empirical laws of the recentered (with respect to
psi_0) and renormalized ψ_n^a, ψ_n^b and ψ_n^c in Figures 3 (kernel density estimators) and 4 (quantile-quantile plots).

```
fig +
  geom_density(aes(clt, fill = type, colour = type), psi_hat_abc, alpha = 0.1) +
  geom_vline(aes(xintercept = bias, colour = type),
             bias_abc, size = 1.5, alpha = 0.5) +
  xlim(-3, 4) +
  labs(y = "",
       x = expression(paste(sqrt(n/v[n]^{list(a, b, c)})*
                            (psi[n]^{list(a, b, c)} - psi[0]))))
```

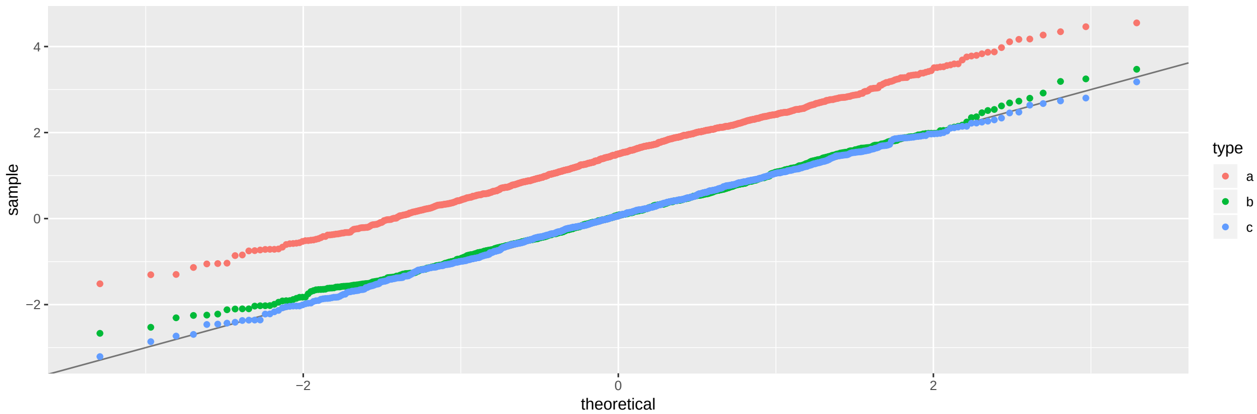


Figure 4: Quantile-quantile plot of the standard normal law against the empirical laws of three estimators of ψ_0, one of them misconceived (a), one assuming that $\bar{G}_0$ is known (b) and one that hinges on the estimation of $\bar{G}_0$ (c). Built based on `iter` independent realizations of each estimator.

```
ggplot(psi_hat_abc, aes(sample = clt, fill = type, colour = type)) +
  geom_abline(intercept = 0, slope = 1, alpha = 0.5) +
  geom_qq(alpha = 1)
```

Figures 3 and 4 reveal that ψ_n^c behaves as well as ψ_n^b — but remember that we did not discuss how to estimate its asymptotic variance.

2.17 **Exercises.** The problems come within the context of Sections 2.15 and 2.16.

1. Compute a numerical approximation of $\mathrm{E}_{P_0}(Y|A=1) - \mathrm{E}_{P_0}(Y|A=0)$. How accurate is it?

2. Building upon the piece of code devoted to the repeated computation of ψ_n^b and its companion quantities, construct confidence intervals for ψ_0 of (asymptotic) level 95%, and check if the empirical coverage is satisfactory. Note that if the coverage was exactly 95%, then the number of confidence intervals that would contain ψ_0 would follow a binomial law with parameters `iter` and `0.95`, and recall that function `binom.test` performs an exact test of a simple null hypothesis about the probability of success in a Bernoulli experiment against its three one-sided and two-sided alternatives.

3. The call to `compute_lGhatAW` makes predictions on the same data points as those exploited to learn $\bar{G}_0$ by fitting the user-supplied working model. Why could that be problematic? Can you think of a simple workaround, implement and test it?

4. Discuss what happens when the dimension of the (still well-specified) working model grows. You could use the following chunk of code

```
powers <- ## make sure '1/2' and '1' belong to 'powers', eg
  seq(1/4, 3, by = 1/4)
working_model_G_two <- list(
  model = function(...) {glm(family = binomial(), ...)},
  formula = as.formula(
    paste("A ~",
          paste(c("I(W^", "I(abs(W - 5/12)^"),
                rep(powers, each = 2),
```

```
                sep = "", collapse = ") + "),
          ")")
  ))
attr(working_model_G_two, "ML") <- FALSE
```

play around with argument `powers` (making sure that `1/2` and `1` belong to it), and plot graphics similar to those presented in Figures 3 and 4.

5. Discuss what happens when the working model is mis-specified. You could use the following chunk of code:

```
transform <- c("cos", "sin", "sqrt", "log", "exp")
working_model_G_three <- list(
  model = function(...) {glm(family = binomial(), ...)},
  formula = as.formula(
    paste("A ~",
          paste("I(", transform, sep = "", collapse = "(W)) + "),
          "(W))")
  ))
attr(working_model_G_three, "ML") <- FALSE
(working_model_G_three$formula)
```

```
## A ~ I(cos(W)) + I(sin(W)) + I(sqrt(W)) + I(log(W)) + I(exp(W))
```

6. Drawing inspiration from (16), one may consider estimating the asymptotic variance of ψ_n^c with the counterpart of v_n^b obtained by substituting $\ell\bar{G}_n$ for $\ell\bar{G}_0$ in (16). By adapting the piece of code devoted to the repeated computation of ψ_n^b and its companion quantities, discuss if that would be legitimate.

2.18 Inference based on the estimation of $\bar{Q}_0$. Comment on structure of `estimate_Q`, similar to that of `estimate_G`.

Demonstrating the inference of ψ_0 based on the estimation of $\bar{Q}_0$ (and of the marginal law of W). Once based on a (mis-specified) working model, and once based on a non-parametric algorithm.

```
estimate_Q <- function(dat, algorithm, ...) {
  if (!attr(algorithm, "ML")) {
    dat <- as.data.frame(dat)
    fit <- algorithm[[1]](formula = algorithm[[2]], data = dat)
    Qhat <- function(newdata) {
      newdata <- as.data.frame(newdata)
      predict(fit, newdata, type = "response")
    }
  } else {
    fit <- algorithm(dat, ...)
    Qhat <- function(newdata) {
      caret::predict.train(fit, newdata)
    }
  }
  return(Qhat)
}

compute_QhatAW <- function(Y, A, W, Qhat, blip = FALSE) {
```

```r
  if (!blip) {
    dat <- data.frame(Y = Y, A = A, W = W)
    pred <- Qhat(dat)
  } else {
    pred <- Qhat(data.frame(A = 1, W = W)) - Qhat(data.frame(A = 0, W = W))
  }
  return(pred)
}

working_model_Q_one <- list(
  model = function(...) {glm(family = binomial(), ...)},
  formula = as.formula(
    paste("Y ~ A * (",
          paste("I(W^", seq(1/2, 3/2, by = 1/2), sep = "", collapse = ") + "),
          "))")
  ))
attr(working_model_Q_one, "ML") <- FALSE
working_model_Q_one$formula
```

```
## Y ~ A * (I(W^0.5) + I(W^1) + I(W^1.5))
```

```r
## k-NN
kknn_algo <- function(dat, ...) {
  args <- list(...)
  if ("Subsample" %in% names(args)) {
    keep <- sample.int(nrow(dat), args$Subsample)
    dat <- dat[keep, ]
  }
  caret::train(Y ~ I(10*A) + W, ## a tweak
               data = dat,
               method = "kknn",
               verbose = FALSE,
               ...)
}
attr(kknn_algo, "ML") <- TRUE
kknn_grid <- expand.grid(kmax = 5, distance = 2, kernel = "gaussian")
control <- trainControl(method = "cv", number = 2,
                        predictionBounds = c(0, 1),
                        allowParallel = TRUE)
```

```r
if(redo_fixed) {
  learned_features_fixed_sample_size <-
    learned_features_fixed_sample_size %>% # head(n = 100) %>%
    mutate(Qhat_d = map(obs, ~ estimate_Q(., algorithm = working_model_Q_one)),
           Qhat_e = map(obs, ~ estimate_Q(., algorithm = kknn_algo,
                                          trControl = control,
                                          tuneGrid = kknn_grid))) %>%
    mutate(blip_QW_d = map2(Qhat_d, obs,
                            ~ compute_QhatAW(.y$Y, .y$A, .y$W, .x, blip = TRUE)),
           blip_QW_e = map2(Qhat_e, obs,
                            ~ compute_QhatAW(.y$Y, .y$A, .y$W, .x, blip = TRUE)))
}

psi_hat_de <- learned_features_fixed_sample_size %>%
```

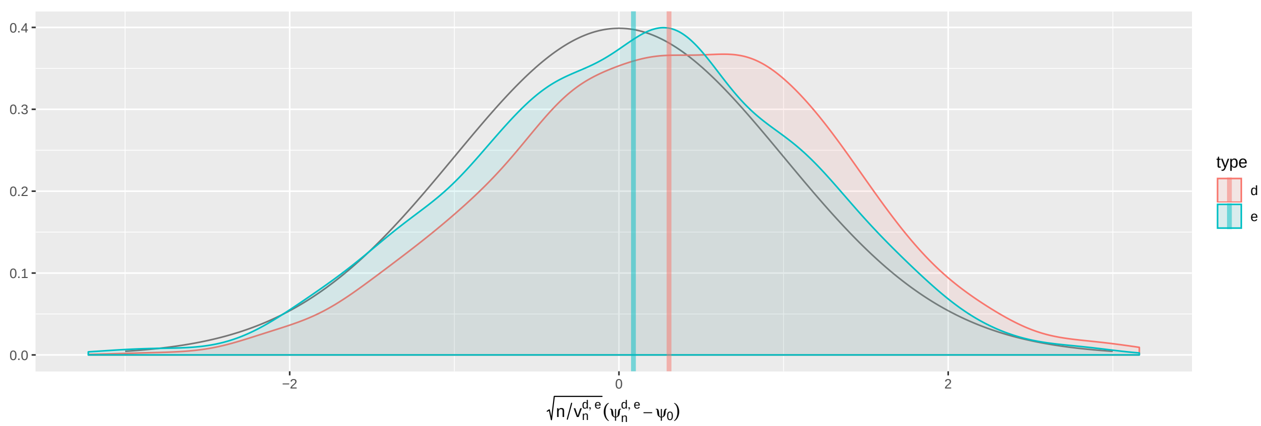
```
  unnest(blip_QW_d, blip_QW_e) %>%
  group_by(id) %>%
  summarize(est_d = mean(blip_QW_d),
            est_e = mean(blip_QW_e)) %>%
  mutate(std_d = sd(est_d),
         std_e = sd(est_e),
         clt_d = (est_d - psi_approx) / std_d,
         clt_e = (est_e - psi_approx) / std_e) %>%
  gather("key", "value", -id) %>%
  extract(key, c("what", "type"), "([^_]+)_([de])") %>%
  spread(what, value)

(bias_de <- psi_hat_de %>% group_by(type) %>% summarize(bias = mean(clt)))
```

```
## # A tibble: 2 x 2
##   type    bias
##   <chr>  <dbl>
## 1 d     0.304
## 2 e     0.0881
```

```
fig <- ggplot() +
  geom_line(aes(x = x, y = y),
            data = tibble(x = seq(-3, 3, length.out = 1e3),
                          y = dnorm(x)),
            linetype = 1, alpha = 0.5) +
  geom_density(aes(clt, fill = type, colour = type),
               psi_hat_de, alpha = 0.1) +
  geom_vline(aes(xintercept = bias, colour = type),
             bias_de, size = 1.5, alpha = 0.5)

fig +
  labs(y = "",
       x = expression(paste(sqrt(n/v[n]^{list(d, e)})*(psi[n]^{list(d, e)} - psi[0]))))
```



No that bad! Yet, we know that $\sqrt{n}$ times bias is bound to increase with sample size. To see this, check out the next chunks of code.

```
sample_size <- c(4e3, 9e3)
block_size <- sum(sample_size)
```

```
label <- function(xx, sample_size = c(1e3, 2e3)) {
  by <- sum(sample_size)
  xx <- xx[seq_len((length(xx) %/% by) * by)] - 1
  prefix <- xx %/% by
  suffix <- findInterval(xx %% by, cumsum(sample_size))
  paste(prefix + 1, suffix + 1, sep = "_")
}

if (redo_varying) {
  learned_features_varying_sample_size <- obs %>% as.tibble %>%
    head(n = (nrow(.) %/% block_size) * block_size) %>%
    mutate(block = label(1:nrow(.), sample_size)) %>%
    nest(-block, .key = "obs")
}
```

First, we cut the data set into independent sub-data sets of sample size n in { 4000, 9000 }. Second, we infer ψ_0 as shown two chunks earlier. We thus obtain 76 independent realizations of each estimator derived on data sets of 2, increasing sample sizes.

```
if(redo_varying) {
  learned_features_varying_sample_size <-
    learned_features_varying_sample_size %>%
    mutate(Qhat_d = map(obs, ~ estimate_Q(., algorithm = working_model_Q_one)),
           Qhat_e = map(obs, ~ estimate_Q(., algorithm = kknn_algo,
                                          trControl = control,
                                          tuneGrid = kknn_grid))) %>%
    mutate(blip_QW_d = map2(Qhat_d, obs,
                            ~ compute_QhatAW(.y$Y, .y$A, .y$W, .x, blip = TRUE)),
           blip_QW_e = map2(Qhat_e, obs,
                            ~ compute_QhatAW(.y$Y, .y$A, .y$W, .x, blip = TRUE)))
}

root_n_bias <- learned_features_varying_sample_size %>%
  unnest(blip_QW_d, blip_QW_e) %>%
  group_by(block) %>%
  summarize(clt_d = sqrt(n())*(mean(blip_QW_d) - psi_approx),
            clt_e = sqrt(n())*(mean(blip_QW_e) - psi_approx)) %>%
  gather("key", "value", -block) %>%
  extract(key, c("what", "type"), "([^_]+)_([de])") %>%
  spread(what, value) %>%
  mutate(block = unlist(map(strsplit(block, "_"), ~.x[2])),
         sample_size = sample_size[as.integer(block)])
```

The `tibble` called `root_n_bias` reports root-n times bias for all combinations of estimator and sample size. The next chunk of code presents visually our findings, see Figure 5. Note how we include the realizations of the estimators derived earlier and contained in `psi_hat_de` (thus breaking the independence between components of `root_n_bias`, a small price to pay in this context).

```
root_n_bias <- learned_features_fixed_sample_size %>%
  mutate(sample_size = B/iter) %>%  # because *fixed* sample size
  unnest(blip_QW_d, blip_QW_e) %>%
  group_by(id) %>%
  summarize(clt_d = sqrt(n()) * (mean(blip_QW_d) - psi_approx),
            clt_e = sqrt(n()) * (mean(blip_QW_e) - psi_approx),
```

```
            sample_size = sample_size[1]) %>%
  gather("key", "clt", -id, -sample_size) %>%
  extract(key, c("what", "type"), "([^_]+)_([de])") %>%
  mutate(block = "0") %>% select(-id, -what) %>%
  full_join(root_n_bias)

root_n_bias %>%
  ggplot() +
  stat_summary(aes(x = sample_size, y = clt,
                   group = interaction(sample_size, type),
                   color = type),
               fun.data = mean_se, fun.args = list(mult = 2),
               position = position_dodge(width = 250), cex = 1) +
  stat_summary(aes(x = sample_size, y = clt,
                   group = interaction(sample_size, type),
                   color = type),
               fun.data = mean_se, fun.args = list(mult = 2),
               position = position_dodge(width = 250), cex = 1,
               geom = "errorbar", width = 750) +
  stat_summary(aes(x = sample_size, y = clt,
                   color = type),
               fun.y = mean,
               position = position_dodge(width = 250),
               geom = "polygon", fill = NA) +
  geom_point(aes(x = sample_size, y = clt,
                 group = interaction(sample_size, type),
                 color = type),
             position = position_dodge(width = 250),
             alpha = 0.1) +
  scale_x_continuous(breaks = unique(c(B / iter, sample_size))) +
  labs(x = "sample size n",
       y = expression(paste(sqrt(n)*(psi[n]^{list(d, e)} - psi[0]))))
```

```
## execute
## rm(learned_features_fixed_sample_size)
## as soon as possible!
```

2.19 One-step estimation. Function `set_Qbar_Gbar` implements the change of the `Qbar` and `Gbar` attributes of `obs` (which are accessible only by oracles).

```
set_Qbar_Gbar <- function(obs, Qbar, Gbar) {
  attr(obs, "Qbar") <- Qbar
  attr(obs, "Gbar") <- Gbar
  return(obs)
}
```

First, we call function `set_Qbar_Gbar` to replace the true `Qbar` and `Gbar` attributes of `obs` by their estimated counterparts as stored in `learned_features_fixed_sample_size`. Second, we call function `eic` to compute the values of the estimated efficient influence at the observations in `obs`. Constructing the one-step estimators is then straightforward.

```
psi_hat_de_one_step <- learned_features_fixed_sample_size %>%
  mutate(obs_d = pmap(list(obs = obs, Qbar = Qhat_d, Gbar = Ghat),
```

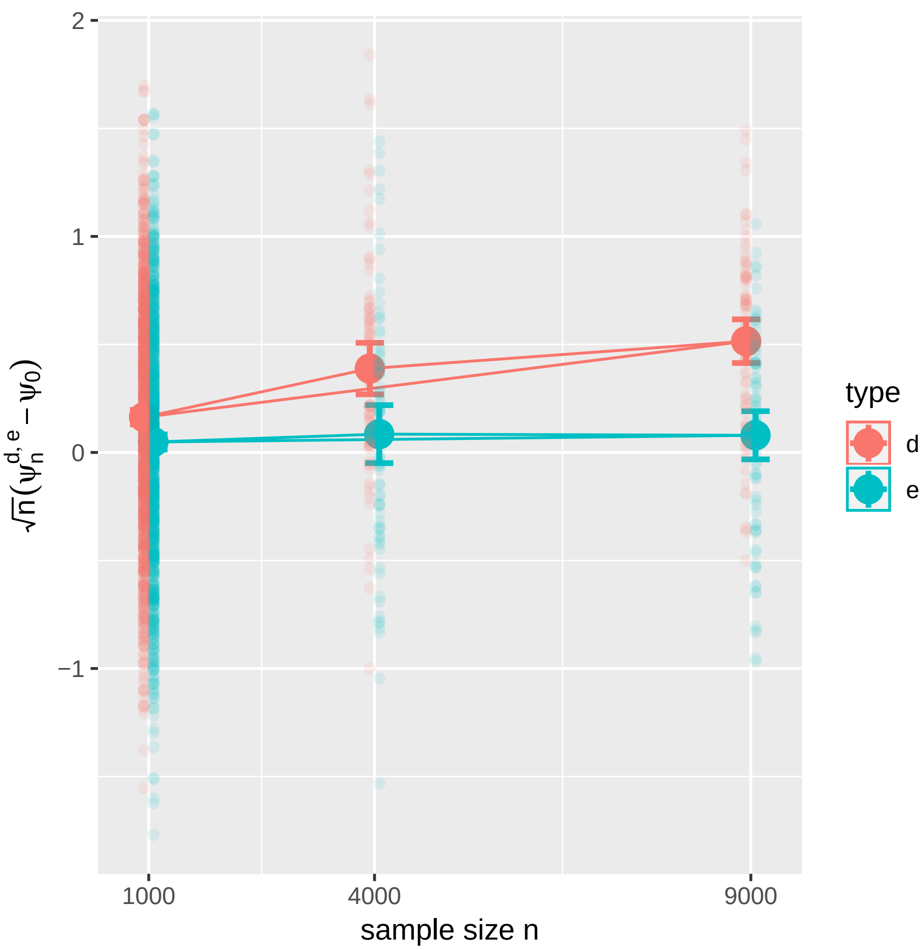


Figure 5: Evolution of root-n times bias versus sample size for two inference methodology of ψ_0 based on the estimation of $\bar{Q}_0$. Big dots represent the average biases and vertical lines represent twice the standard error.

```r
                      set_Qbar_Gbar),
         obs_e = pmap(list(obs = obs, Qbar = Qhat_e, Gbar = Ghat),
                      set_Qbar_Gbar)) %>%
  mutate(eic_obs_d = map2(obs_d, blip_QW_d, ~ eic(as.data.frame(.x), mean(.y))),
         eic_obs_e = map2(obs_e, blip_QW_e, ~ eic(as.data.frame(.x), mean(.y)))) %>%
  unnest(blip_QW_d, eic_obs_d,
         blip_QW_e, eic_obs_e) %>%
  group_by(id) %>%
  summarize(est_d = mean(blip_QW_d) + mean(eic_obs_d),
            std_d = sd(eic_obs_d),
            clt_d = sqrt(n()) * (est_d - psi_approx) / std_d,
            est_e = mean(blip_QW_e) + mean(eic_obs_e),
            std_e = sd(eic_obs_e),
            clt_e = sqrt(n()) * (est_e - psi_approx) / std_e) %>%
  gather("key", "value", -id) %>%
  extract(key, c("what", "type"), "([^_]+)_([de])") %>%
  spread(what, value) %>%
  mutate(type = paste0(type, "_one_step"))

(bias_de_one_step <- psi_hat_de_one_step %>%
   group_by(type) %>% summarize(bias = mean(clt)))
```

```
## # A tibble: 2 x 2
##   type          bias
##   <chr>        <dbl>
## 1 d_one_step 0.0414
## 2 e_one_step 0.0634
```

```r
ggplot() +
  geom_line(aes(x = x, y = y),
            data = tibble(x = seq(-3, 3, length.out = 1e3),
                          y = dnorm(x)),
            linetype = 1, alpha = 0.5) +
  geom_density(aes(clt, fill = type, colour = type),
               psi_hat_de_one_step, alpha = 0.1) +
  geom_vline(aes(xintercept = bias, colour = type),
             bias_de_one_step, size = 1.5, alpha = 0.5) +
  labs(y = "",
       x = expression(paste(sqrt(n/v[n]^{list(dos, eos)})*(psi[n]^{list(dos, eos)} - psi[0]))))
```

It seems that the one-step correction is quite good (in particular, compare `bias_de` with `bias_de_one_step`):

```r
bind_rows(bias_de, bias_de_one_step)
```

```
## # A tibble: 4 x 2
##   type          bias
##   <chr>        <dbl>
## 1 d          0.304
## 2 e          0.0881
## 3 d_one_step 0.0414
## 4 e_one_step 0.0634
```

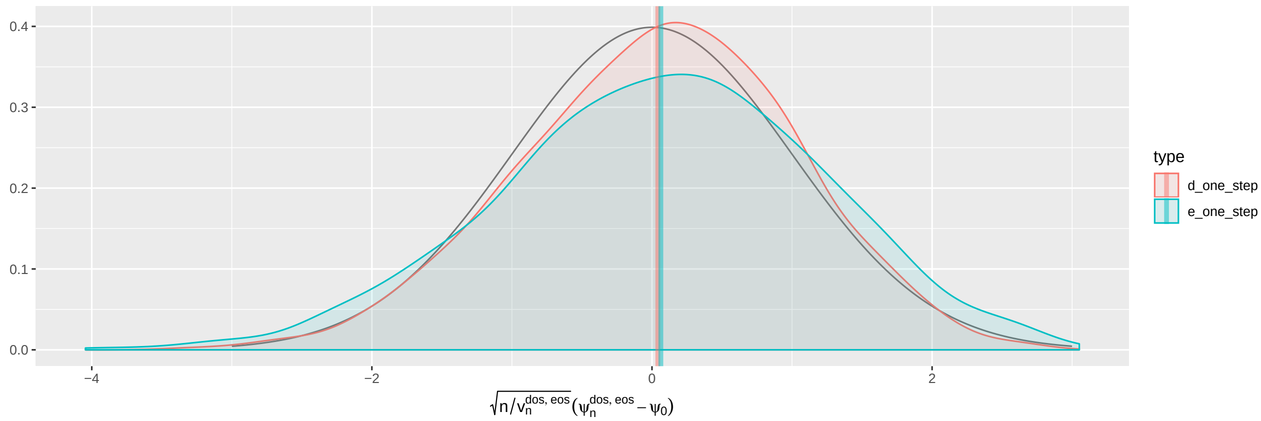


Figure 6: Write caption.

2.20 Targeted inference.

2.21 Appendix.

For later…

```
working_model_Q_two <- list(
  model = function(...) {glm(family = binomial(), ...)},
  formula = as.formula(
    paste("Y ~ A * (",
          paste("I(W^", seq(1/2, 3, by = 1/2), sep = "", collapse = ") + "),
          "))")
  ))
attr(working_model_Q_two, "ML") <- FALSE

## xgboost based on trees
xgb_tree_algo <- function(dat, ...) {
  caret::train(Y ~ I(10*A) + W,
               data = dat,
               method = "xgbTree",
               trControl = control,
               tuneGrid = grid,
               verbose = FALSE)
}
attr(xgb_tree_algo, "ML") <- TRUE
xgb_tree_grid <- expand.grid(nrounds = 350,
                             max_depth = c(4, 6),
                             eta = c(0.05, 0.1),
                             gamma = 0.01,
                             colsample_bytree = 0.75,
                             subsample = 0.5,
                             min_child_weight = 0)

## nonparametric kernel smoothing regression
npreg <- list(
  label = "Kernel regression",
  type = "Regression",
  library = "np",
  parameters = data.frame(parameter =
```

```r
                            c("subsample", "regtype",
                              "ckertype", "ckerorder"),
                          class = c("integer", "character",
                                    "character", "integer"),
                          label = c("#subsample", "regtype",
                                    "ckertype", "ckerorder")),
  grid = function(x, y, len = NULL, search = "grid") {
    if (!identical(search, "grid")) {
      stop("No random search implemented.\n")
    } else {
      out <- expand.grid(subsample = c(50, 100),
                         regtype = c("lc", "ll"),
                         ckertype =
                           c("gaussian",
                             "epanechnikov",
                             "uniform"),
                         ckerorder = seq(2, 8, 2))
    }
    return(out)
  },
  fit = function(x, y, wts, param, lev, last, classProbs, ...) {
    ny <- length(y)
    if (ny > param$subsample) {
      ## otherwise far too slow for what we intend to do here...
      keep <- sample.int(ny, param$subsample)
      x <- x[keep, ]
      y <- y[keep]
    }
    bw <- np::npregbw(xdat = as.data.frame(x), ydat = y,
                      regtype = param$regtype,
                      ckertype = param$ckertype,
                      ckerorder = param$ckerorder,
                      remin = FALSE, ftol = 0.01, tol = 0.01,
                      ...)
    np::npreg(bw)
  },
  predict = function (modelFit, newdata, preProc = NULL, submodels = NULL) {
    if (!is.data.frame(newdata)) {
      newdata <- as.data.frame(newdata)
    }
    np:::predict.npregression(modelFit, se.fit = FALSE, newdata)
  },
  sort = function(x) {
    x[order(x$regtype, x$ckerorder), ]
  },
  loop = NULL, prob = NULL, levels = NULL
)

npreg_algo <- function(dat, ...) {
  caret::train(working_model_Q_one$formula,
               data = dat,
               method = npreg, # no quotes!
               verbose = FALSE,
```

```
               ...)
}
attr(npreg_algo, "ML") <- TRUE
npreg_grid <- data.frame(subsample = 100,
                         regtype = "lc",
                         ckertype = "gaussian",
                         ckerorder = 4,
                         stringsAsFactors = FALSE)
```