# A guided tour in targeted learning territory

*David Benkeser, Antoine Chambaz, Nima Hejazi*

*08/13/2018*

## Contents



## 1 Introduction

This is a very first draft of our article. The current *tentative* title is "A guided tour in targeted learning territory".

Explain our objectives and how we will meet them. Explain that the symbol indicates more delicate material.

Use sectioning a lot to ease cross-referencing.

Do we include exercises? I propose we do, and to flag the corresponding subsections with symbol .

```
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(ggplot2)) ## make sure it is installed
suppressMessages(library(caret)) ## make sure it is installed
expit <- plogis
logit <- qlogis
```

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

```
draw_from_experiment <- function(n, full = FALSE) {
  ## preliminary
  n <- Arguments$getInteger(n, c(1, Inf))
  full <- Arguments$getLogical(full)
  ## ## 'Gbar' and 'Qbar' factors
  Gbar <- function(W) {
    expit(-0.2 + 3 * sqrt(W) - 1.5 * W)
  }
  Qbar <- function(AW) {
    A <- AW[, 1]
    W <- AW[, 2]
    ## A * cos((1 + W) * pi / 5) + (1 - A) * sin((1 + W^2) * pi / 4)
    A * (cos((1 + W) * pi / 5) + (1/3 <= W & W <= 1/2) / 10) +
      (1 - A) * (sin(4 * W^2 * pi) / 4 + 1/2)
  }
  ## sampling
  ## ## context
  W <- runif(n)
  ## ## 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 (full) {
    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") <- dunif
  ##
  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 `full` 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:

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

```
##              W A         Y
## [1,] 0.4290078 0 0.9426242
## [2,] 0.4984304 1 0.7202482
## [3,] 0.1766923 1 0.8768885
## [4,] 0.2743935 0 0.8494665
## [5,] 0.2165102 1 0.3849406
## attr(,"Gbar")
## function (W)
## {
##     expit(-0.2 + 3 * sqrt(W) - 1.5 * W)
## }
## <bytecode: 0x221947d0>
## <environment: 0x3440f358>
## attr(,"Qbar")
## function (AW)
## {
##     A <- AW[, 1]
##     W <- AW[, 2]
##     A * (cos((1 + W) * pi/5) + (1/3 <= W & W <= 1/2)/10) + (1 -
##         A) * (sin(4 * W^2 * pi)/4 + 1/2)
## }
## <bytecode: 0xb3def40>
## <environment: 0x3440f358>
## attr(,"QW")
## function (x, min = 0, max = 1, log = FALSE)
## .Call(C_dunif, x, min, max, log)
## <bytecode: 0xb301cb0>
## <environment: namespace:stats>
```

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. From a probabilistic point of view, the attributes `Gbar`, `Qbar` and `QW` are infinite-dimensional features of $P_0$. There is more to $P_0$ than $\bar{G}_0$ (`Gbar`), $\bar{Q}_0$ (`Qbar`), formally defined by

$$\bar{G}_0(W) \equiv P_0(A = 1|W), \quad \bar{Q}_0(A, W) \equiv E_{P_0}(Y|A, W), \tag{1}$$

and the marginal law $Q_{0,W}$ of $W$ under $P_0$ (`QW`) — for instance the conditional law (not expectation) of $Y$ given $(A, W)$, but $\bar{G}_0$, $\bar{Q}_0$ and $Q_{0,W}$ will play a prominent role in our story.

**2.2 The parameter of interest, first pass.** It happens that we especially care for a finite-dimensional feature of $P_0$ that we denote by $\psi_0$. Its definition involves two of the aforementioned infinite-dimensional features:

$$\begin{aligned}\psi_0 &\equiv E_{P_0}\left(\bar{Q}_0(1, W) - \bar{Q}_0(0, W)\right) \\ &= \int \left(\bar{Q}_0(1, w) - \bar{Q}_0(0, w)\right) dQ_{0,W}(w).\end{aligned} \tag{2}$$

Acting as oracles, we can compute explicitely the numerical value of $\psi_0$.

Our interest in $\psi_0$ 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 rest of the experiment will take place, $W = f_w(U_w)$;
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 ram home what it means to run the "full" experiment by setting argument `full` 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 well known (do we give the proof or refer to other articles?) that

$$\psi_0 = E_{\mathbb{P}_0}\left(Y_1 - Y_0\right).$$

Thus, $\psi_0$ compares (additively) the averages of the two counterfactual rewards. In other words, $\psi_0$ 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 $\psi_0$ is a well defined parameter beyond its causal interpretation.

To conclude this subsection, we draw advantage from the possibility to sample full observations from `draw_from_experiment` by setting its argument `full` to `TRUE` in order to numerically approximate $\psi_0$. By the law of large numbers, the following chunk of code approximates $\psi_0$ and shows it approximate value:

```
B <- 1e6 ## Antoine: 1e6 eventually
full_obs <- draw_from_experiment(B, full = TRUE)
(psi_hat <- mean(full_obs[, "Yone"] - full_obs[, "Yzero"]))
```

```
## [1] 0.06062293
```

In fact, the central limit theorem and Slutsky's lemma allow us to build a confidence interval with asymptotic level 95% for $\psi_0$:

```
sd_hat <- sd(full_obs[, "Yone"] - full_obs[, "Yzero"])
alpha <- 0.05
(psi_CI <- psi_hat + c(-1, 1) * qnorm(1 - alpha / 2) * sd_hat / sqrt(B))
```

```
## [1] 0.05997656 0.06126929
```

**2.3 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

$$\bar{Q}(A, W) \equiv E_P(Y|A, W).$$

Central to our approach is viewing $\psi_0$ as the value at $P_0$ of the statistical mapping $\Psi$ from $\mathcal{M}$ to $[0,1]$ characterized by

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

a clear extension of (2). 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 $P_0$,

```
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
  ##
  return(obs)
}
```

parameter $\Psi(\Pi_0)$ is well defined, and numerically approximated by `psi.Pi.zero` in the following chunk of code:

```
five_obs_from_another_experiment <- draw_from_another_experiment(5)
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(integrand, lower = 0, upper = 1)$val)
```

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

(easy algebra reveals that $\Psi(\Pi_0) = 59/300$ indeed).

**2.4 Being smooth, first pass.** Luckily, the statistical mapping $\Psi$ is well behaved, or smooth. Here, 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 $h \to 0$, 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)
```

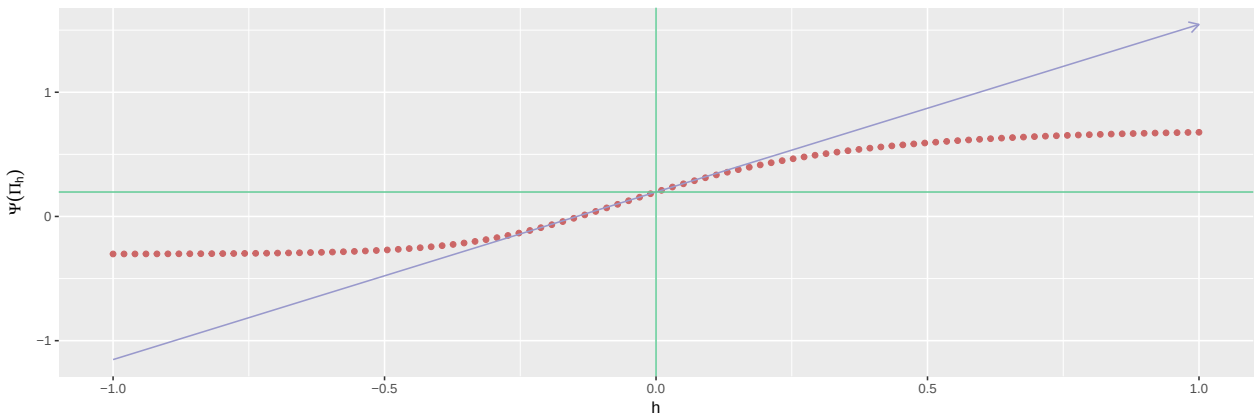



Figure 1: Evolution of statistical parameter $\Psi$ along fluctuation $\{\Pi_h : h \in H\}$.

```
  }
  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") +
  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 $\Psi$. Once the presentation is complete, we will be able 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.

**2.5 Being smooth, second pass.** Let us now describe what it means for statistical mapping $\Psi$ to be smooth at every $P \in \mathcal{M}$. The description necessitates the introduction of fluctuations.

For every direction* $s : \mathcal{O} \to \mathbb{R}$ such that $s \neq 0^\dagger$, $E_P(s(O)) = 0$ and $s$ bounded by, say, $M$, for every $h \in H \equiv ]-M^{-1}, M^{-1}[$, we can define a law $P_h \in \mathcal{M}$ by setting $P_h \ll P^\ddagger$ and

$$\frac{dP_h}{dP}(O) \equiv 1 + hs(O), \tag{3}$$

*A direction is a measurable function.
†That is, $s(O)$ is not equal to zero $P$-almost surely.
‡That is, $P_h$ is dominated by $P$: if an event $A$ satisfies $P(A) = 0$, then necessarily $P_h(A) = 0$ too.

that is, $P_h$ has density $(1 + hs)$ with respect to (w.r.t.) $P$. We call $\{P_h : h \in H\}$ a fluctuation of $P$ in direction $s$ because

$$(i)\ P_h|_{h=0} = P, \quad (ii)\ \frac{d}{dh} \log \frac{dP_h}{dP}(O)\bigg|_{h=0} = s(O). \tag{4}$$

The fluctuation is a one-dimensional parametric submodel of $\mathcal{M}$.

Statistical mapping $\Psi$ is smooth at every $P \in \mathcal{M}$ because, for each $P \in \mathcal{M}$, there exists a so called efficient influence curve[§] $D^*(P) : \mathcal{O} \to \mathbb{R}$ such that $E_P(D^*(P)(O)) = 0$ and, for any direction $s$ as above, if $\{P_h : h \in H\}$ is defined as in (3), then the real-valued mapping $h \mapsto \Psi(P_h)$ is differentiable at $h = 0$, with a derivative equal to

$$E_P\left(D^*(P)(O)s(O)\right). \tag{5}$$

Interestingly, if a fluctuation $\{P_h : h \in H\}$ satisfies (4) for a direction $s$ such that $s \neq 0$, $E_P(s(O)) = 0$ and $\mathrm{Var}_P(s(O)) < \infty$, then $h \mapsto \Psi(P_h)$ is still differentiable at $h = 0$ with a derivative equal to (5) (beyond fluctuations of the form (3)).

The influence curves $D^*(P)$ convey valuable information about $\Psi$. For instance, an important result from the theory of inference based on semiparametric models guarantees that if $\psi_n$ 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,

$$\mathrm{Var}_P(D^*(P)(O)). \tag{6}$$

In this light, an estimator $\psi_n$ 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 (6), which is called the Cramér-Rao bound.

**2.6 The efficient influence curve.** It is not difficult to check (do we give the proof?) that the efficient influence curve $D^*(P)$ of $\Psi$ 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

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

[§]It is a measurable function.

[¶]We can view $\psi_n$ 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 $E_P(s(O)) = 0$ and $\mathrm{Var}_P(s(O)) < \infty$ and fluctuation $\{P_h : h \in H\}$ satisfying (4), 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 $s$.

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")])
  gW <- Gbar(obs[, "W"])
  lgAW <- obs[, "A"] * gW + (1 - obs[, "A"]) * (1 - gW)
  ( Qbar(cbind(1, obs[, "W"])) - Qbar(cbind(0, obs[, "W"])) - psi ) +
    (2 * obs[, "A"] - 1) / lgAW * (obs[, "Y"] - QAW)
}

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

```
## [1] -1.0729162  0.1645268  0.2829249 -0.5969299 -0.4555560
```

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

```
## [1] -0.15461023 -0.11720740  0.05266769 -0.09541458 -0.19365206
```

**2.7 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_hat)))
```

```
## [1] 0.2546553
```

and the Cramér-Rao bound at $\Pi_0$:

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

```
## [1] 0.09512008
```

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

```
## [1] 0.6111668
```

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 0.61 (rounded to two decimal places) smaller than their counterparts for $\Psi(P_0)$.

**2.8 Revisiting Section 2.4.** 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 $\Pi_0$ in the direction of $\sigma_0$ (in the sense of (3)) given, up to a constant, by

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

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

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

$$E_{\Pi_0}(D^*(\Pi_0)(O)\sigma_0(O)) \tag{7}$$

(since $D^*(\Pi_0)$ is centered under $\Pi_0$, knowing $\sigma_0$ up to a constant is not problematic).

Let us check this numerically. In the next chunk of code, we implement direction $\sigma_0$ with `sigma0_draw_from_another_experiment`, then we numerically approximate (7) (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] 1.35969
```

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

```
## [1] 1.354437 1.364944
```

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

**2.9 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 $P' \in \mathcal{M}$

$$\mathrm{Rem}_P(\bar{Q}', \bar{G}') \equiv \Psi(P') - \Psi(P) + E_P(D^*(P')(O)), \tag{8}$$

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

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

$$\mathrm{Rem}_P(\bar{Q}', \bar{G}')^2 \leq \|\bar{Q}' - \bar{Q}\|_P^2 \times \|(\bar{G}' - \bar{G})/\ell\bar{G}'\|_P^2. \tag{9}$$

In particular, if

$$E_P(D^*(P')(O)) = 0, \tag{10}$$

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 (10) 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, 8 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 8 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 9, 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.

**2.10 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.

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

iter <- 1e3
```

Then, the alternative expression

$$\psi_0 = E_{P_0}\left(\frac{2A-1}{\ell\bar{G}_0(A,W)}Y\right) \tag{11}$$

suggests to estimate $\psi_0$ with

$$\psi_n^b \equiv E_{P_n}\left(\frac{2A-1}{\ell\bar{G}_0(A,W)}Y\right) = \frac{1}{n}\sum_{i=1}^n\left(\frac{2A_i-1}{\ell\bar{G}_0(A_i,W_i)}Y_i\right). \tag{12}$$

Note how $P_n$ is substituted for $P_0$ in (12) relative to (11).

It is easy to check that $\psi_n^b$ estimates $\psi_0$ consistently, but this is too little to request from an estimator of $\psi_0$. Better, $\psi_n^b$ 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

$$v_n^b \equiv \mathrm{Var}_{P_n}\left(\frac{2A-1}{\ell\bar{G}_0(A,W)}Y\right) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{2A_i-1}{\ell\bar{G}_0(A_i,W_i)}Y_i - \psi_n^b\right)^2. \tag{13}$$

Let us investigate how $\psi_n^b$ behaves based on `obs`. Because we are interested in the *law* of $\psi_n^b$, 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 $\psi_n^b$ on all samples.

Before proceeding, let us introduce

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

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

$$E_{P_0}(Y|A=1) - E_{P_0}(Y|A=0).$$

We seize this opportunity to demonstrate numerically the obvious fact that $\psi_n^a$ does not estimate $\psi_0$.

```
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_hat) / std_b)
std_a <- sd(psi_hat_ab$est_a)
psi_hat_ab <- psi_hat_ab %>%
  mutate(std_a = std_a,
         clt_a = (est_a - psi_hat) / 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
```

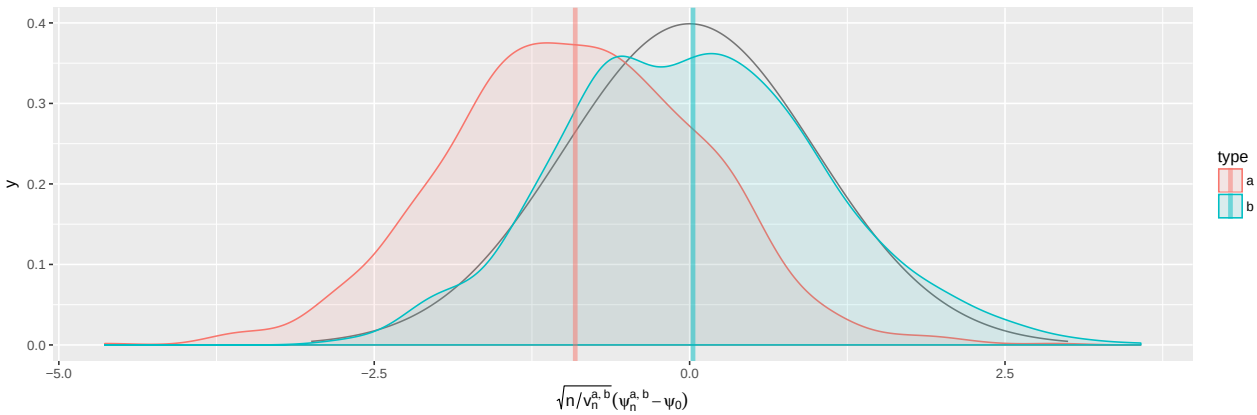



Figure 2: Kernel density estimators of the law of two estimators of $\psi_0$ (recentered and renormalized), one of them misconceived (a), the other assuming that $\bar{G}_0$ is known (b). Built based on `iter` independent realizations of each estimator.

```
##   <chr>   <dbl>
## 1 a     -0.907
## 2 b      0.0269
```

```
debug(ggplot2::stat_density)
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(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 $\psi_n^a$. 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 -0.907 and 0.027 (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 $\psi_n^a$ and $\psi_n^b$ (recentered and renormalized) as estimated by kernel density estimation. The latter is close to the black curve, which represents the standard normal density.

**2.11 Inference assuming $\bar{G}_0$ known, or not, second pass.** At the beginning of Section 2.10, we assumed that $\bar{G}_0$ was known. Let us suppose now that it is not. The definition of $\psi_n^b$ can be adapted to overcome this difficulty, by substituting an estimator of $\ell\bar{G}_0$ for $\ell\bar{G}_0$ in (12).

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 2.12. 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, and

the generic function `predict_lGAW` (merely a convenient wrapper) estimates $\ell\bar{G}_0(A, W)$ for any $(A, W)$ based on the output of `estimate_G`.

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

predict_lGAW <- function(A, W, algorithm, threshold = 0.05, ...) {
  ## a wrapper to use in a call to 'mutate'
  ## (a) fit the working model
  dat <- data.frame(A = A, W = W)
  Ghat <- estimate_G(dat, algorithm, ...)
  ## (b) make predictions based on the fit
  Ghat_W <- Ghat(dat)
  lGAW <- A * Ghat_W + (1 - A) * (1 - Ghat_W)
  pmin(1 - threshold, pmax(lGAW, threshold))
}
```

Comment on new structure of `estimate_G`.

Note how the prediction of any $\ell\bar{G}_0(A, W)$ is manually bounded away from 0 and 1 at the last line of `predict_lGAW`. This is desirable because the *inverse* of each $\ell\bar{G}_0(A_i, W_i)$ appears in the definition of $\psi_n^b$ (12).

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("I(W^", seq(1/2, 2, by = 1/2), sep = "", collapse = ") + "),
          ")")
  ))
attr(working_model_G_one, "ML") <- FALSE
working_model_G_one$formula
```

```
## A ~ I(W^0.5) + I(W^1) + I(W^1.5) + I(W^2)
```

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

$$-L_a(f)(A, W) \equiv A \log f(W) + (1 - A) \log(1 - f(W)) \tag{14}$$

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 $\theta \in \mathbb{R}^5$,

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 $\mathcal{F}$:

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

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

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

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

In light of (12), introduce

$$\psi_n^c \equiv \frac{1}{n}\sum_{i=1}^n \left(\frac{2A_i - 1}{\ell\bar{G}_n(A_i, W_i)} Y_i\right). \tag{15}$$

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 $\psi_n^c$ on the same `iter = 1000` independent samples of independent observations drawn from $P_0$ as in Section 2.10:

```
psi_hat_c <- obs %>% as_tibble() %>% mutate(id = 1:n() %% iter) %>%
  group_by(id) %>%
  mutate(lgAW = predict_lGAW(A, W, working_model_G_one)) %>%
  summarize(est = mean(Y * (2 * A - 1) / lgAW),
            try = sd(Y * (2 * A - 1) / lgAW) / sqrt(n()))
std_c <- sd(psi_hat_c$est)
psi_hat_abc <- psi_hat_c %>%
  mutate(std = std_c,
         try = try,
         clt = (est - psi_hat) / 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     -0.907
## 2 b      0.0269
## 3 c     -0.0870
```

Note how we exploit the independent realizations of $\psi_n^c$ 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 -0.087 (rounded to three decimal places — see `bias_abc`). We represent the empirical laws of the recentered and renormalized $\psi_n^a$, $\psi_n^b$ and $\psi_n^c$ in Figures 3 (kernel density estimators) and 4 (quantile-quantile plots).

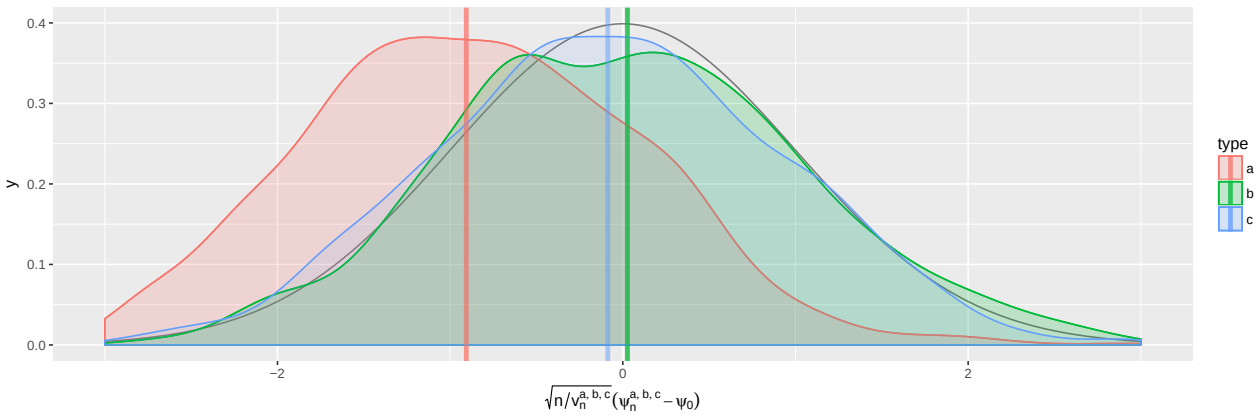



Figure 3: Kernel density estimators of the law of three estimators of $\psi_0$ (recentered and renormalized), one of them misconceived (a), one assuming that $\bar{G}_0$ is known (b) and one that hinges on the estimation of $\bar{G}_0$ (c). The present figure includes Figure 2 (but the colors differ). Built based on `iter` independent realizations of each estimator.

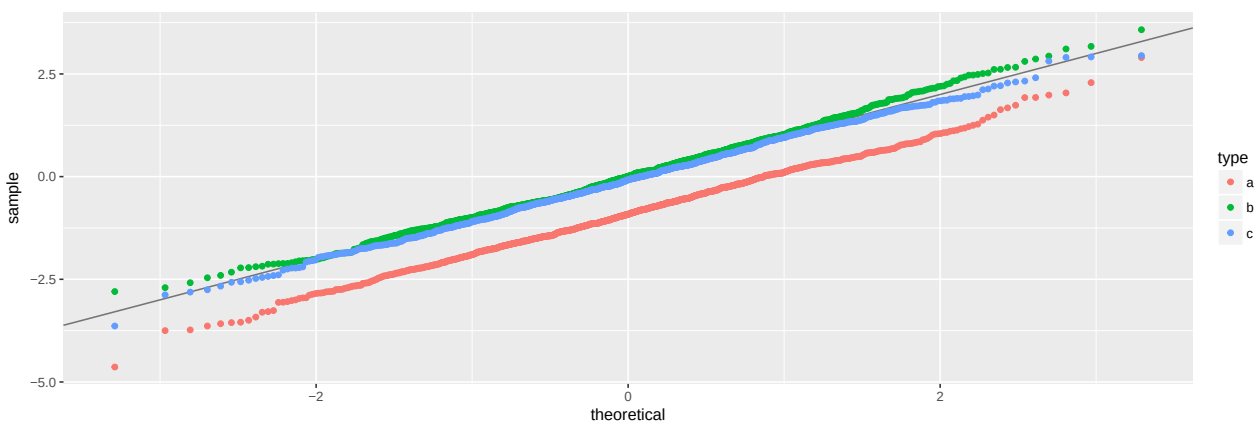



Figure 4: Quantile-quantile plot of the standard normal law against the empirical laws of three estimators of $\psi_0$, one of them misconceived (a), one assuming that $\bar{G}_0$ is known (b) and one that hinges on the estimation of $\bar{G}_0$ (c). Built based on `iter` independent realizations of each estimator.

```
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, 3) +
  labs(x = expression(paste(sqrt(n/v[n]^{list(a, b, c)})*
                            (psi[n]^{list(a, b, c)} - psi[0]))))
```

```
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 $\psi_n^c$ behaves as well as $\psi_n^b$ — but remember that we did not discuss how to estimate its asymptotic variance.

**2.12** **Exercises.** The questions are asked in the context of Sections 2.10 and 2.11.

1. Building upon the piece of code devoted to the repeated computation of $\psi_n^b$ and its companion quantities, construct confidence intervals for $\psi_0$ 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 $\psi_0$ 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.

2. The wrapper `predict_lGAW` makes predictions by fitting a working model on the same data points as those for which predictions are sought. Why could that be problematic? Can you think of a simple workaround, implement and test it?

3. 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("I(W^", powers, 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.

4. 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))
```

5. Drawing inspiration from (13), one may consider estimating the asymptotic variance of $\psi_n^c$ with the counterpart of $v_n^b$ obtained by substituting $\ell\bar{G}_n$ for $\ell\bar{G}_0$ in (13). By adapting the piece of code devoted to the repeated computation of $\psi_n^b$ and its companion quantities, discuss if that would be legitimate.

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

predict_QAW <- function(Y, A, W, algorithm, blip = FALSE, ...) {
  ## a wrapper to use in a call to 'mutate'
  ## (a) carry out the estimation based on 'algorithm'
  dat <- data.frame(Y = Y, A = A, W = W)
  Qhat <- estimate_Q(dat, algorithm, ...)
  ## (b) make predictions based on the fit
  if (!blip) {
    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, 2, by = 1/2), sep = "", collapse = ") + "),
          "))")
  ))
attr(working_model_Q_one, "ML") <- FALSE
working_model_Q_one$formula
```

### 2.13 Targeted inference.

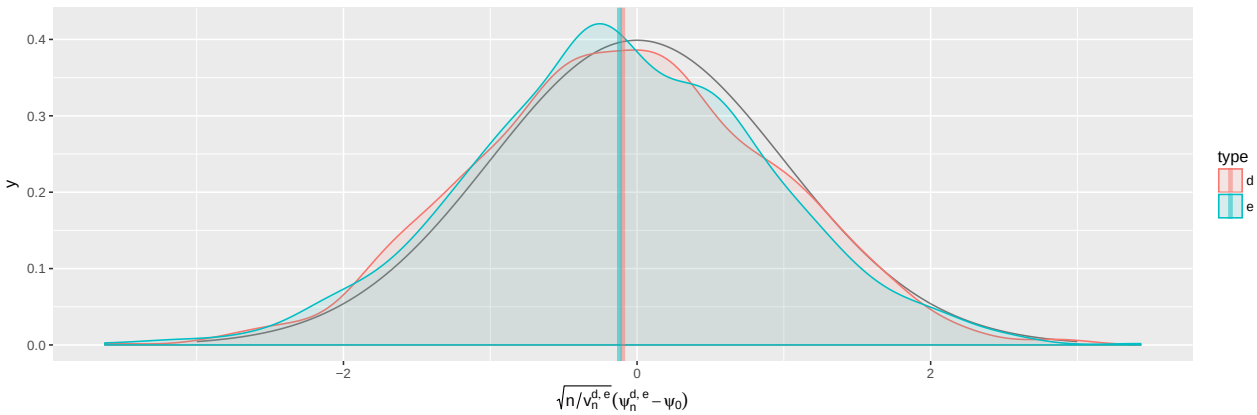
```
## Y ~ A * (I(W^0.5) + I(W^1) + I(W^1.5) + I(W^2))
```

```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,
```

```r
                ...)
}
attr(kknn_algo, "ML") <- TRUE
kknn_grid <- expand.grid(kmax = c(3, 5), distance = 2, kernel = "gaussian")
control <- trainControl(method = "cv", number = 2,
                        predictionBounds = c(0, 1),
                        allowParallel = TRUE)

psi_hat_de <- obs %>% as_tibble() %>% mutate(id = 1:n() %% iter) %>%
  group_by(id) %>%
  mutate(blipQW_d = predict_QAW(Y, A, W, working_model_Q_one, blip = TRUE),
         blipQW_e = predict_QAW(Y, A, W, kknn_algo, blip = TRUE,
                                trControl = control,
                                tuneGrid = kknn_grid,
                                Subsample = 100)) %>%
  summarize(est_d = mean(blipQW_d),
            est_e = mean(blipQW_e))

std_d <- sd(psi_hat_de$est_d)
std_e <- sd(psi_hat_de$est_e)
psi_hat_de <- psi_hat_de %>%
  mutate(std_d = std_d,
         clt_d = (est_d - psi_hat) / std_d,
         std_e = std_e,
         clt_e = (est_e - psi_hat) / 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     -0.0970
## 2 e     -0.121
```

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




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 =
                            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)
```