Semiparametric Thresholding Least Squares Inference for Causal Effects with R

Pierre Chausse[*], Mihai Giurcanu[†], George Luta[‡]

Abstract

The vignette explains how to use the causalTLSE package to estimate thet causal effects using a semiparametric thresholding least squares methods.

Introduction

This document presents the `causalTLSE` package explaining in details all functions implemented in the package. It is intended for users interested in all the details about the methods presented in the paper and how they are implemented.

The main model is

$$Y = \beta_0(1-Z) + \beta_1 Z + f_0(X) + f_1(X) + \varepsilon\,,$$

and it is approximated by the regression

$$Y = \beta_0(1-Z) + \beta_1 Z + \psi_0^T U_0(X) + \psi_1^T U_1(X) + u\,,$$

where $Y \in \mathbb{R}$ is the response variable, X is a $k \times 1$ vector of confounders, and $U_0(X) \in \mathbb{R}^{p_0}$ and $U_1(X) \in \mathbb{R}^{p_1}$ are spline vectors satisfying $U_0(X) = 0$ if $Z = 1$ and $U_1(X) = 0$ if $Z = 0$[1]. We can represent $U_j(X)$, for $j = 0, 1$, into a block vector $\{U_{j1}(X_1)^T, U_{j2}(X_2)^T, ..., U_{jk}(X_k)^T\}^T$, where $U_{jl}(X_l) \in \mathbb{R}^{p_{jl}}$ is the vector of basis functions for group j associated with X_l, with $\sum_{l=1}^k p_{jl} = p_j$, $j = 0, 1$. The paper proposes a data-driven method for selecting the vectors $U_0(X)$ and $U_1(X)$.

To understand the package, it is important to know how the $U_{jl}(X_l)$'s are defined. To simplify the notation, we remove the subscript l from X and the subscripts j and l from $U_{jl}(X_l)$ and p_{jl}. We just need to keep in mind that $U(X) = \{U_s(X)\} \in \mathbb{R}^p$ is different for the treated and control groups and also for different confounders. Let $\{\kappa_1, ..., \kappa_{p-1}\}$ be a set of $p-1$ knots strictly inside the support of X satisfying $\kappa_1 < \kappa_2 <, ..., < \kappa_{p-1}$. For a realization x and $p \geq 3$, we have the following bases:

$$\begin{aligned}
U_1(x) &= xI(x \leq \kappa_1) + \kappa_1 I(x > \kappa_1)\\
U_p(x) &= (x - \kappa_{p-1})I(x > \kappa_{p-1})\\
U_s(x) &= (x - \kappa_{s-1})I(\kappa_{s-1} \leq x \leq \kappa_s) + (\kappa_s - \kappa_{s-1})I(x > \kappa_s)\,,
\end{aligned}$$

[*] University of Waterloo, pchausse@uwaterloo.ca

[†] University of Chicago, giurcanu@uchicago.edu

[‡] Georgetown University, George.Luta@georgetown.edu

[1] This is a simplification for clarify. Alternatively, we could define the regression as

$$Y = \beta_0(1-Z) + \beta_1 Z + \psi_0^T U_0(X)(1-Z) + \psi_1^T U_1(X) Z + u\,.$$

where the last $U_s(x)$ is defined for $2 < s < p$. Therefore, if the number of knots is equal to 1, we only have the first two bases. Since knots must be strictly inside the support of X, any categorical variable with two levels, which includes as a special case binary variables, the number of knots must be equal to zero. In this case, $U(X) = X$. For general nominal variables, the number of knots cannot exceed the number of levels minus two.

Note that for the sample regression

$$Y_i = \beta_0(1 - Z_i) + \beta_1 Z_i + \psi_0^T U_0(X_i) + \psi_1^T U_1(X_i) + u_i\,,$$

for $i = 1, ..., n$, the knots of X_l, $l = 1, ..., k$, must be strictly inside the sample range of the vector $\{X_{li}\} \in \mathbb{R}^n$ instead of inside the support of X_l.

The causalTLSE package

Setting up the Model

The first step in using the package is to define a model. The model contains the information about the outcome, the treatment indicator, the covariates and their knots. This is the starting point before applying any basis selection method. To illustrate how to use the package, we are using the dataset from Lalonde (1986). The dataset, called `nsw`, contains some continuous and categorical variables, so we can illustrate how knots are selected initially. The dataset is included in the `causalTLSE` package.

```
library(causalTLSE)
data(nsw)
```

The outcome is the real income in 1978 (`re78`) and the purpose is to estimate the causal effect of a training program (`treat`) on the outcome. The dataset includes the continuous covariates age (`age`), education (`ed`) the 1975 real income (`re75`) and some binary variables (`black`, `hisp`, `married` and `nodeg`). We start by considering the covariates `age`, `re75`, `ed` and ‘`married`. We can setup the model simply by running the following command:

```
model1 <- setModel(re78 ~ treat | ~ age + re75 + ed + married, data=nsw)
```

The left of | is designated for the formula linking the outcome (re78) and the treatment indicator (treat). The covariates are entered after | as a formula without a dependent variable. This method works like for formulas in `lm`. For example, we can add interactions, transformations of the variables, etc. The following is an example:

```
model0 <-  setModel(re78 ~ treat | ~ age + I(age^2) + re75 + ed * married,
                data=nsw)
```

This will create the vector of covariates {age, age^2, re75, ed, married, ed×married}. The function `setModel` creates an object of class `tlseModel` with its own print method, which will be presented later.

The following sub-sections explain all arguments of the function.

The starting knots

By default, the function automatically generates knots for each variable based on the following procedure. This procedure is applied separately for the treated and control groups. The term `sample size` means the number of observations in the treated or control group.

1. The starting number of knots is a function of the sample size and is determined by the argument `nbases`, a function of one argument, the sample size. The floor of what the function returns is the number of bases. The starting number of knots is therefore equal to the `floor` of what the function returns minus 1 (or 0 if this operation results in a negative number). The default function is `function(n)`

`n^0.3`. For example, if the total sample size is 500, with 200 treated and 300 control, the starting number of knots in the treated and control groups are respectively equal to 3 (`floor(200^0.3)-1`) and 4 (`floor(300^0.3)-1`). It is possible to have a number of knots that does not depend on the sample size. All we need is to set the argument `nbases` to a function that returns an integer.

2. Let $(p-1)$ be the number of knots determined by the previous step. The knots are obtained by computing $p+1$ quantiles of X for equally spaced probabilities from 0 to 1, and by dropping the first and last ones. For example, if the number of knots is equal to 3, we compute the quantiles for the probabilities 0.25, 0.5 and 0.75.
3. We drop any duplicated knots and any knots equal to either the max or the min of X. If the resulting number of knots is equal to 0, the vector of knots is set to `NULL`. When the knots is `NULL` for a variable X, it means that $U(X) = X$.

The last step implies that the number of knots for all categorical variables with two levels, which includes as a special case binary variables, is equal to 0. For nominal variables with a small number of levels, the number of knots may be smaller than the ones defined by `nbases`. For example, when the number of levels for a nominal variable is 3, the number of knots cannot exceed 1.

We can inspect the knots of the current model as follows. Note that each object in the package is S3-class, so the elements can be accessed using the operator \$. The elements `knots0` and `knots1` are the list of knots for the control and treated groups. For example, the knots for the treated are:

```
model1$knots1
```

```
## $age
## 20% 40% 60% 80% 
##  19  22  25  28 
## 
## $re75
##       40%       60%       80% 
##  357.9499 1961.8640 5588.6640 
## 
## $ed
## 20% 40% 60% 80% 
##   9  10  11  12 
## 
## $married
## NULL
```

We see that it is set to `NULL` for `married`, because it is a binary variable. The sample size for the treated is 297. Given the default `nbases`, it implies a number of starting knots equal to 4 ($297^{0.3} - 1 = 4.519$). This is the number of knots we have for `ed` and `age`, but not for `re75`. The reason is that `re75` contains a large fraction of zeros. Since the 20% quantile is equal to 0 and 0 in also the minimum value of `ed75`, it is dropped (the `type` argument of the `quantile` function is the same as it is implemented in the package).

```
quantile(nsw[nsw$treat==1,'re75'], c(.2,.4,.6,.8), type=1)
```

```
##       20%       40%       60%       80% 
##    0.0000  357.9499 1961.8640 5588.6640
```

By printing the object, we see a summary of the model. It includes the list of variable with a positive number of knots and the ones with no knots.

```
model1
```

```
## Semiparametric TLSE Model
## *************************
## 
```

```
## Number of treated:  297
## Number of control:  425
## Number of missing values:  0
## Selection Method: SLSE
## Covariates approximated by semiparametric TLSE:
##  age, re75, ed
## Covariates not approximated by semiparametric TLSE:
##  married
```

> **SLSE**: We see that the selection method is set to SLSE, which stands for Semiparametric Least Squares Estimator. We refer to this when the knots are automatically selected by the method described above. Later in the document, we will present methods for selecting a subset of the SLSE using TLSE.

As another example, the simulated dataset `simDat4` contains special types of covariates. It helps to further illustrate how the knots are determined. The dataset contains a continuous variable `X1` with a large proportion of zeros, the nominal variables `X2` and `X3`, with respectively 2 and 3 levels (the levels are {3,4} for `X2` and {1,2,3} for `X3`), and a binary variable `X4`.

```
data(simDat4)
model2 <- setModel(Y~Z |~X1+X2+X3+X4, data=simDat4)
model2$knots0
```

```
## $X1
##        40%        60%        80%
##  0.2531388  2.9118507 12.1110772
##
## $X2
## NULL
##
## $X3
## 40%
##   2
##
## $X4
## NULL
```

We see that the number of knots for the nominal variable with 2 levels is set to 0 and it is equal to 1 for the one with 3 levels.

Setting the number of knots to 0 for specific variables

To avoid having a positive number of knots for a variable, we can enter its name in the argument `userRem`. For example, if we want the number of knots to be zero for `ed` and `age`, we can create the model as follows:

```
model3 <- setModel(re78~treat | ~age+re75+ed+married, data=nsw,
                   userRem=c("ed","age"))
model3
```

```
## Semiparametric TLSE Model
## **************************
##
## Number of treated:  297
## Number of control:  425
## Number of missing values:  0
## Selection Method: SLSE
## Covariates approximated by semiparametric TLSE:
```

```
##  re75
## Covariates not approximated by semiparametric TLSE:
##  age, ed, married
```

We see that only `re75` has a positive number of knots.

Setting the knots manually

We have the control over the knots through the arguments `knots0` and `knots1`. When the arguments are missing (the default), all knots are set automatically. One way to set the number of knots to 0 for all variables in a given group is to set the argument to `NULL`. For example, the number of knots is equal to 0 for all variables of the treated group in the following:

```
setModel(re78~treat | ~age+re75+ed+married, data=nsw, knots1=NULL)
```

```
## Semiparametric TLSE Model
## **************************
##
## Number of treated:  297
## Number of control:  425
## Number of missing values:  0
## Selection Method: User Based
## Covariates approximated by semiparametric TLSE:
##  Treated: None
##  Control: age, re75, ed
## Covariates not approximated by semiparametric TLSE:
##  Treated: age, re75, ed, married
##  Control: married
```

Notice that the selection method is defined as "User Based" whenever knots are provided manually by the user. The other option is to provide a list of knots. The list must have the same length as the number of covariates. For each element, we have three options:

- `NA`: The knots are set automatically for this variable only.
- `NULL`: The number of knots is set to 0 for this variable only.
- A numeric vector: The vector cannot contain missing or duplicated values and must be strictly inside the range of the variable for the group.

Suppose you want to set for the control group an automatic selection for `age`, no knots for `ed` and the knots $\{1000, 5000, 10000\}$ for `re75`, and the knots be automatically selected for the treated group. We proceed as follows. Note that setting the value to `NA` or `NULL` has the same effect for the binary variable `married`. In the following, the argument `knots=TRUE` is added to the `print` method to only print the knots.

```
model <- setModel(re78~treat | ~age+re75+ed+married, data=nsw,
                  knots0=list(NA, c(1000,5000,10000), NULL, NA))
print(model, knots=TRUE)
```

```
## Lists of knots for the treated group
## ************************************
## age:
## 20%  40%  60%  80%
##  19   22   25   28
## re75:
##       40%       60%       80%
##  357.9499 1961.8640 5588.6640
## ed:
## 20%  40%  60%  80%
##   9   10   11   12
## married:
## None
##
## Lists of knots for the Control group
## ************************************
## age:
## 16.66667%  33.33333%        50%  66.66667%  83.33333%
##        18         20         23         26         30
## re75:
##    k1     k2     k3
##  1000   5000  10000
## ed:
## None
## married:
## None
```

Estimating the model

Given the set of knots from the model object, the estimation is just a least squares method applied to the extended set of covariates. We want to estimate the model

$$Y = \beta_0(1-Z) + \beta_1 Z + \psi_0' U_0(X) + \psi_1' U_1(X) + u\,,$$

where $U_0(X)$ and $U_1(X)$ are defined above and depends on the model knots. The function that estimates the model is `estModel`. The function has three arguments, but two of them are mostly used internally by other functions. We present it in case it is needed. The arguments are:

- `model`: A model created by the function `setModel`.
- `w0`: A list of integers to select knots for the control group from the model. By default, all the knots are used.
- `w1`: A list of integers to select knots for the treated group from the model. By default, all the knots are used.

We illustrate with a simple model containing only two covariates and one knot per eligible variables.

```r
model <- setModel(re78~treat | ~age+married, data=nsw,
                  nbases=function(n) 2)
print(model, knots=TRUE)
```

```
## Lists of knots for the treated group
## ************************************
## age:
## 50%
##  23
## married:
## None
##
## Lists of knots for the Control group
## ************************************
## age:
## 50%
##  23
## married:
```

```
## None
```

```
fit <- estModel(model)
fit
```

```
## Semiparametric TLSE Estimate
## ****************************
## Selection Method: SLSE
##
## factor(treat)0  factor(treat)1        Xf0age_1        Xf0age_2      Xf0married
##     4558.28061      3754.98326        27.79868       -12.51415      -115.81593
##       Xf1age_1        Xf1age_2      Xf1married
##       89.25358        22.22331      1435.28205
```

The object has its own print method that returns the coefficient estimates. A more detrailed presentation of the results can be obtained using the `summary` method. The following is an example with just one knot per eligible variable.

```
summary(fit)
```

```
## Semiparametric TLSE Estimate
## ****************************
## Selection Method: SLSE
##
##                Estimate Std. Error t value Pr(>|t|)
## factor(treat)0  4558.28    3380.43   1.348    0.178
## factor(treat)1  3754.98    4043.48   0.929    0.353
## Xf0age_1          27.80     164.59   0.169    0.866
## Xf0age_2         -12.51      67.11  -0.186    0.852
## Xf0married      -115.82     859.66  -0.135    0.893
## Xf1age_1          89.25     194.19   0.460    0.646
## Xf1age_2          22.22      76.46   0.291    0.771
## Xf1married      1435.28    1014.69   1.415    0.157
##
## Multiple R-squared:  0.009618,   Adjusted R-squared:  -9.119e-05
```

For example, the coefficient of `Xf0age_1` is the effect of age for the control on `re78` when age is less than the first knot (23) and `Xf0age_2` is the effect when age is greater then the first knot. Note that the R^2 and adjusted R^2 are different from what we obtain using the summary of the `lm` object:

```
summary(fit$lm.out)[c("r.squared","adj.r.squared")]
```

```
## $r.squared
## [1] 0.4379272
##
## $adj.r.squared
## [1] 0.4316295
```

This is because our model does not contain an intercept and the R^2 is computed differently for models without an intercept. The definition of the R^2 used by R is the following (RSS means residual sum of squares):

$$R^2 = 1 - \frac{\text{RSS for the model with the regressors}}{\text{RSS for the model without the regressors}}$$

In a model with an intercept, the residual of the model without the regressors is $Y_i - \bar{Y}$, but it is equal to Y_i when the model does not have an intercept. As a result, the R^2 with and without an intercept are

$$R^2_{with} = 1 - \frac{\sum_{i=1}^n \hat{e}_i^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2}$$
$$R^2_{without} = 1 - \frac{\sum_{i=1}^n \hat{e}_i^2}{\sum_{i=1}^n Y_i^2}$$

However, our model does contain an intercept since we include a binary variable for both the control and treated groups.

The predict and plot method

The `predict` method is very similar to the `predict.lm` method. We use the same arguments: `object`, `interval`, `se.fit`, `newdata` and `level`. The difference is that it returns the predicted outcome for the treated and control groups separately and the argument `vcov.`, a function like `vcovHC` or `vcovCL` fro the `sandwich` package, can be used to compute robust standard errors. By default, the standard errors are computed using `vcov.lm` (valid under the homoskedasticity assumption). Since there are many options for computing robust standard errors, we let the user chooses what is the most appropriate one when it is needed. The function return a list of 2 elements, `treated` and `control`. By default (`se.fit=FALSE` and `interval="none"`), each element contains a vector of predictions. Here is an example with the previously fitted model `fit`:

```
predict(fit, newdata=data.frame(treat=c(1,1,0,0),age=20:23, married=1))
```

```
## $treated
## [1] 6975.337 7064.591
## 
## $control
## [1] 5054.036 5081.834
```

If `interval` is set to "confidence", but `$se.fit` remains equal to `FALSE`, each element contains a matrix containing the prediction, and the lower and upper bound of the confidence interval, with the confidence level determined by the argument `level`. Here is an example with the same fitted model:

```
predict(fit, newdata=data.frame(treat=c(1,1,0,0),age=20:23, married=1),
        interval="confidence")
```

```
## $treated
##        fit    lower    upper
## 1 6975.337 4960.082 8990.592
## 2 7064.591 5119.244 9009.937
## 
## $control
##        fit    lower    upper
## 3 5054.036 3455.978 6652.093
## 4 5081.834 3423.558 6740.110
```

If `se.fit` is set to `TRUE`, each element, treated or control, is a list with the elements `pr`, containing the predictions, and `se.fit`, containing the standard errors. In the following, we only show the result for the treated:

```
predict(fit, newdata=data.frame(treat=c(1,1,0,0),age=20:23, married=1),
        se.fit=TRUE)$treated
```

```
## $fit
## [1] 6975.337 7064.591
## 
## $se.fit
##         1         2
## 1028.2100  992.5422
```

The `predict` method is called by the `plot` method to visually assess the predicted outcome for the treated and control groups with respect to a given covariate, controlling for the other covariates in the model. The arguments of the `plot` method are:

- **x**: An object of class `tlseFit`.
- **y**: An alias for `which` for compatibility with the generic `plot` function.
- **which**: covariate to plot against the outcome variable. It could be an integer (the position of the covariate) or a character (the name of the covariate)
- **interval**: The type of confidence interval to include. The default is “none”. The other alternative is “confidence”.
- **level**: The confidence level when `interval="confidence"`. The default is 0.95.
- **newdata**: An optional named vector of fixed values for some or all other covariates.
- **legendPos**: The position of the legend. The default is “topright”.
- **vcov.**: An optional function to compute the estimated matrix of covariance of the least squares estimators. This argument only affects the confidence intervals.
- **col0, col1, lty0, lty1**: The line colors and shapes for the control and treated. The defaults are `col0=1` (black), `col1=2` (red), `lty0=1` (solid) and `lty1=2` (dashed).
- **add.**: Should the curves be added to an existing plot? The default is `FALSE`.
- **addToLegend**: An optional character string to add to the legend next to “treated” and “control”.
- **cex**: The font size for the legend. The default is 1.
- **ylim, xlim**: optional ranges for the y-axis and x-axis.
- **addPoints**: Should we include the scatterplot of the outcome and covariate to the graph? The default is `FALSE`.
- **FUN**: A function to determine how the other covariates are fixed. The default is `mean`. Note that the function is applied to each group separately.
- **main**: An optional title to replace the default one.
- **…*: Other arguments are passed to the `vcov.` function.

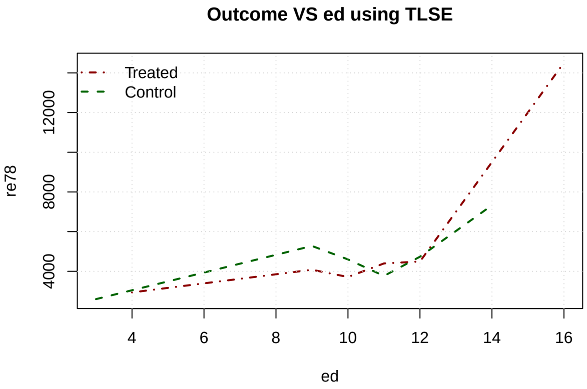
In the following, we illustrate some examples. Consider the model:

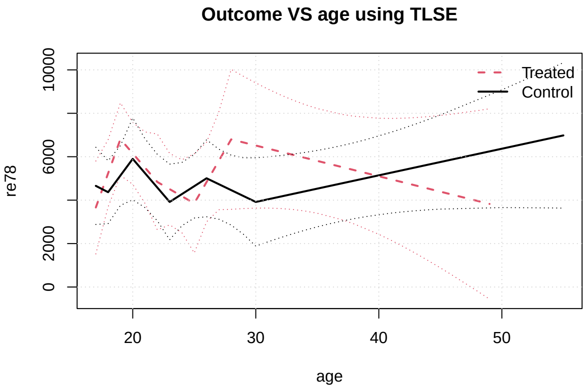
```
model1 <- setModel(re78~treat | ~age+re75+ed+married, data=nsw)
fit1 <- estModel(model1)
```

Suppose we want to compare the predicted income with respect to age or education, holding the other covariates fixed to their group means (the default).

The following are two examples with some of the default arguments modified:

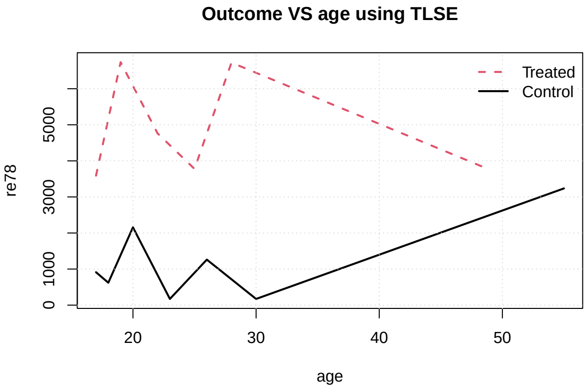
```
library(sandwich)
plot(fit1, "ed", col0="darkgreen", col1="darkred", lty0=2, lty1=4,
     legendPos="topleft")
plot(fit1, "age", interval='confidence', level=0.9, vcov.=vcovHC)
```

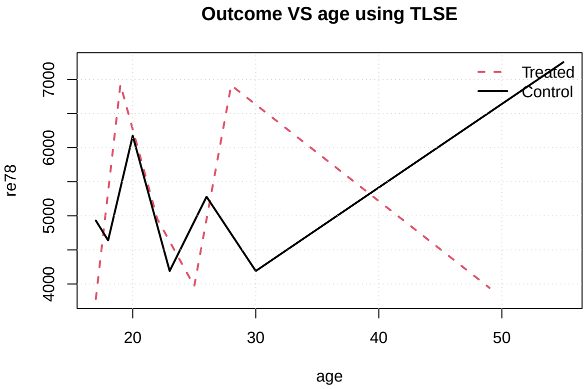




If we want to fix the other covariates using another function, we can change the argument `FUN`. The new function must be a function of one argument. For example, if we want to fix the other covariates to their group medians, we set `FUN` to `median` (no quotes). We proceed the same way for any function that requires only one argument (e.g. `mode`). If the function requires more than one argument, we have to create a new function. For example, if we want to fix them to their 20% group quantiles, we can set the argument to `function(x) quantile(x, .20)`. The following illustrates the two cases:

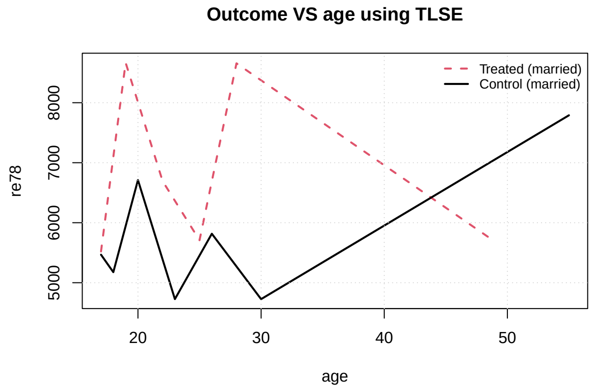
```
plot(fit1, "age", FUN=mode)
plot(fit1, "age", FUN=function(x) quantile(x, .20))
```

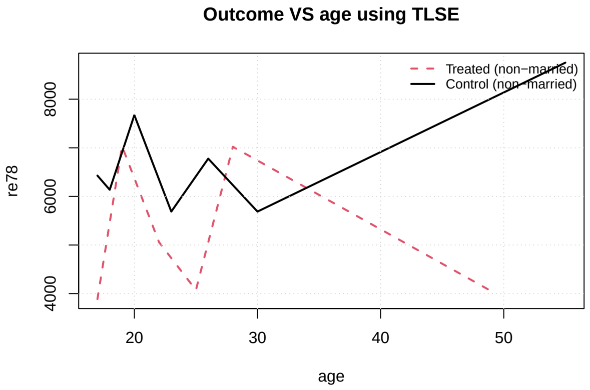




It is also possible to set some of the other covariates to a specific value by changing the argument `newdata`. This argument must be a named vector with the names corresponding to the variables you want to fix. You can also add a description to the legend with the argument `addToLegend`.

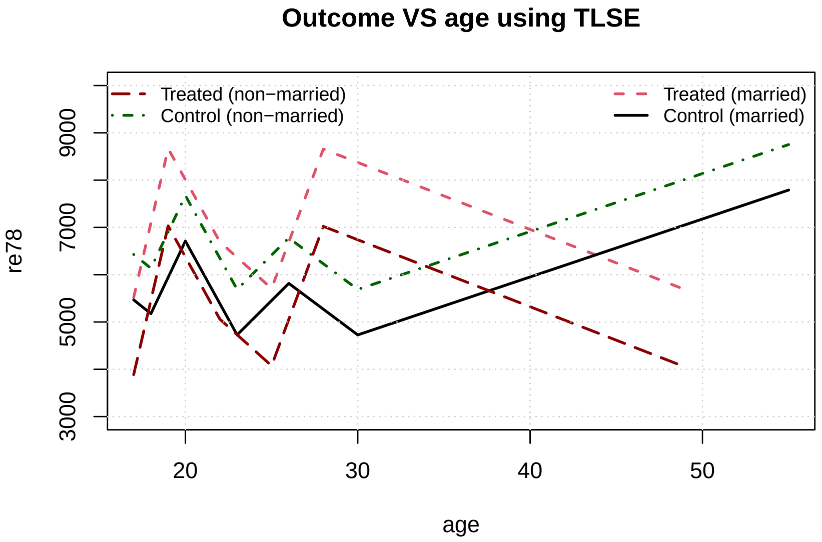
```
plot(fit1, "age", newdata=c(married=1, re75=10000), addToLegend="married", cex=0.8)
plot(fit1, "age", newdata=c(married=0, re75=10000), addToLegend="non-married", cex=0.8)
```





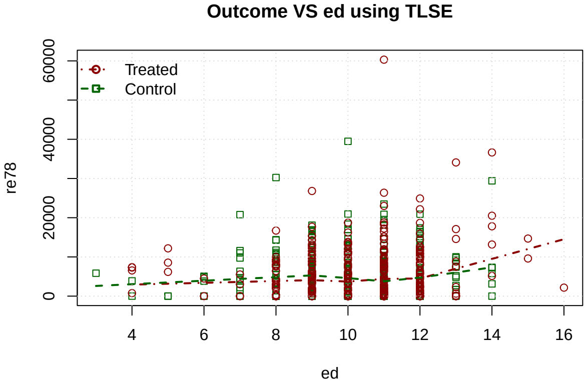
To be better compare the two, it is also possible to have them plotted on the same graph by setting the argument `add.` to `TRUE`. We just need to adjust some of the arguments to better distinguish the different curves. In the following example, we set the colors and line shapes to different values and change the position of the legend in the second `plot` function.

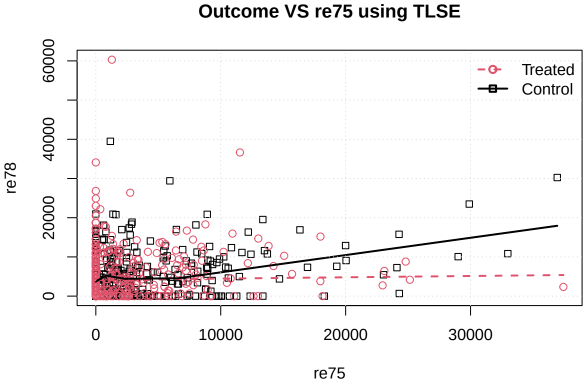
```
plot(fit1, "age", newdata=c(married=1, re75=10000), addToLegend="married", cex=0.8,
     ylim.=c(3000,10000))
plot(fit1, "age", newdata=c(married=0, re75=10000), addToLegend="non-married", cex=0.8,
     legendPos='topleft', col0="darkgreen", col1="darkred", lty0=4, lty1=5,
     add.=TRUE,)
```



Finally, it is also possible to add the observed points to the graph.

```
plot(fit1, "ed", col0="darkgreen", col1="darkred", lty0=2, lty1=4,
     legendPos="topleft", addPoints=TRUE)
plot(fit1, "re75", addPoints=TRUE)
```





The `causal` function

Once we have a model with knots, we can estimate the different causal effects. This is done by the `causal` function. The function assumes we are satisfied with the knots and estimate the causal effects and their standard errors. To define the different causal effect measures, let's redefine $U_0(X)$ and $U_1(X)$ as the spline bases using the knots of the control and treated group respectively, but with all data points. This differs from how it is defined in the introduction, because this $U_0(X_i)$ is not equal to 0 when $Z_i = 1$ and $U_1(X_i)$ is not equal to 0 when $Z_i = 0$. The regression estimated by `estModel`, or the one defined in the introduction, can be written as

$$Y = \beta_0(1-Z) + \beta_1 Z + \psi_0'[U_0(X)(1-Z)] + \psi_1'[U_1(X)Z] + u\,.$$

Let $\hat{\beta}_0$, $\hat{\beta}_1$, $\hat{\psi}_0$ and $\hat{\psi}_1$ be the least squares estimates. Then, the estimated causal effects are defined as:

$$\begin{aligned}
\text{ACE} &= \hat{\beta}_1 - \hat{\beta}_0 + \hat{\phi}_1'\overline{U_1(X)} - \hat{\phi}_0'\overline{U_0(X)} \\
\text{ACT} &= \hat{\beta}_1 - \hat{\beta}_0 + \hat{\phi}_1'\overline{U_1(X)Z} - \hat{\phi}_0'\overline{U_0(X)Z} \\
\text{ACN} &= \hat{\beta}_1 - \hat{\beta}_0 + \hat{\phi}_1'\overline{U_1(X)(1-Z)} - \hat{\phi}_0'\overline{U_0(X)(1-Z)}\,,
\end{aligned}$$

where

$$\begin{aligned}
\overline{U_j(X)} &= \frac{1}{n}\sum_{i=1}^{n} U_j(X_i), \text{ for j=0,1} \\
\overline{U_j(X)Z} &= \frac{1}{n_1}\sum_{i=1}^{n} U_j(X_i)Z_i, \text{ for j=0,1} \\
\overline{U_j(X)(1-Z)} &= \frac{1}{n_0}\sum_{i=1}^{n} U_j(X_i)(1-Z_i), \text{ for j=0,1}
\end{aligned}$$

and n_0 and n_1 are the number of individuals in the control and treated groups. The function `causal` is a method registered for `tlseFit` and `tlseModel` objects. In other words, we can compute the causal effects directly from the model:

```
causal(model1)
```

```
## Causal Effect using Semiparametric TLSE
## ***************************************
## Selection Method: SLSE
##
## ACE = 814.3083
## ACT = 831.8856
## ACN = 802.0249
```

or from the estimated model:

```
causal(fit1)
```

```
## Causal Effect using Semiparametric TLSE
## ***************************************
## Selection Method: SLSE
##
## ACE = 814.3083
## ACT = 831.8856
## ACN = 802.0249
```

We see that the selection method (for the knots) and the criteria used to select the knots are set to unknown. This is because it is not specified in the model object how the knots were selected. We will clarify this below. The method return an object of class `causaltlse`. We see above what its `print` method returns and the following show its `summary` method:

```
ce <- causal(model1)
summary(ce)
```

```
## Causal Effect using Thresholding Least Squares
## **********************************************
## Selection Method: SLSE
##     Estimate Std. Error t value Pr(>|t|)
## ACE    814.3      482.1   1.689   0.0912 .
## ACT    831.9      499.5   1.665   0.0958 .
## ACN    802.0      498.9   1.608   0.1079
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

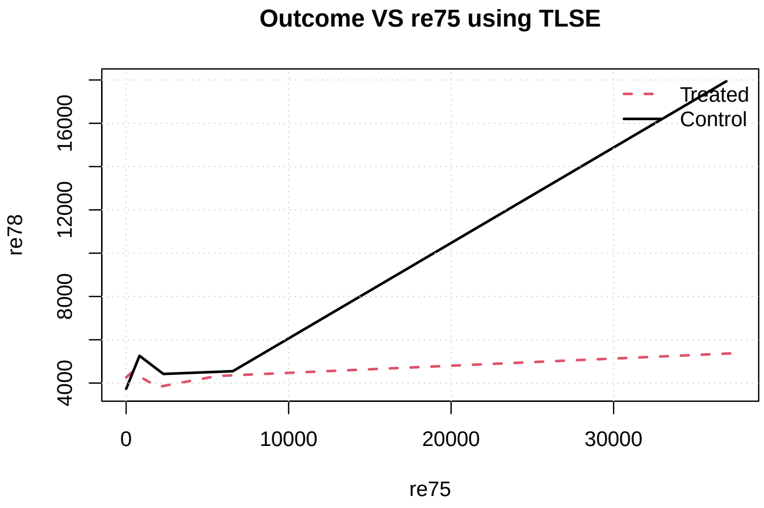
The standard errors are computed using an analytical expression derived in the paper (need to add a citation to our paper), which takes into account the variance of the sample means of the covariates. Asymptotically, these variances converge to 0, so it only makes a difference in small samples. Alternatively, we can set the argument `seType` to "lm" and use the least squares standard errors based on the asymptotic properties. By default, `vcov.lm` is used, but it is possible to modify it by changing the argument `vcov.`. In the following, we estimate the standard errors using the HC3 type of heteroskedasticity robust standard errors.

```
ce2 <- causal(model1, seType="lm", vcov.=vcovHC, type="HC3")
summary(ce2)
```

```
## Causal Effect using Thresholding Least Squares
## **********************************************
## Selection Method: SLSE
##     Estimate Std. Error t value Pr(>|t|)
## ACE    814.3      506.1   1.609    0.108
## ACT    831.9      527.4   1.577    0.115
## ACN    802.0      514.2   1.560    0.119
```

The object `causaltlse` inherits from the class `tlseFit`, so we can apply the `plot` (or the `predict`) method directly on this object.

```r
plot(ce2, "re75")
```



Optimal selection of the knots

We propose two methods for selecting the knots: a backward (BTLSE) and a forward (FTLSE) methods. For each method, we propose three criteria: the asymptotic (ASY), the Akaike Information (AIC) and the Bayesian Information (BIC). The two selection methods can be summarized as follows:

BTLSE:

1. We estimate the model with all knots included in the model.
2. For each knot, we test if the slope of the piecewise linear polynomial is the same before and after, and return the p-value.
3. The knots are selected using one of the following criteria
    - **ASY**: We remove all knots with a p-value greater than a specified threshold.
    - **AIC** or **BIC**: We order the p-values in descending order. Then, going from the largest to the smallest, we remove the knot associated with the p-value one by one, estimate the model and return the information criterion. We keep the model with the smallest information citerion.

FTLSE:

1. We estimate the model by including a subset of the knots one variable at the time. When we test a knot for one variable, the number of knots is set to 0 for all the others.
2. For each knot, we test if the slope of the piecewise linear polynomial is the same before and after, and return the p-value. The set of knots used for each test depends on the following:
    - Variables with 1 knot: we return the p-value of the test of equality before and after the knot.
    - Variables with 2 knots: we include the two knots and return the p-values of the test of equality before and after for each knot.
    - Variables with p knots ($p > 2$): We test the equality before and after the knot i, for $i = 1, ..., p$, using the sets of knots $\{1, 2\}$, $\{1, 2, 3\}$, $\{2, 3, 4\}$, $\ldots$, $\{p-2, p-1, p\}$ and $\{p-1, p\}$ respectively.

3. The knots are selected using one of the following criteria
    - **ASY**: We remove all knots with a p-value greater than a specified threshold.
    - **AIC** or **BIC**: We order the p-values in ascending order. Then, starting with a model with no knots and going from the smallest to the highest highest p-value, we add the knot associated with the p-value one by one, estimate the model and return the information criterion. We keep the model with the smallest information citerion.

The selection is done using the function `selTLSE`. The arguments are:

- **model**: An object of class `tlseModel`.
- **method**: This is the selection method. We have the choice between "FTLSE" (the default) and "BTLSE".
- **crit**: This is the criterion used by the selection method. We have the choice between "AIC" (the default), "BIC" or "ASY".
- **minPV**: This is a function that returns the p-value threshold. It is a function of one argument, the average number of knots per covariate. The default is `function(p) 1/log(p)`. It is also possible to set it to a fix threshold. For example, `function(p)`   `0.20` set the threshold to 0.2. This argument affects the result only when `method` is set to "ASY".
- **vcov.**: By default, the p-values are computed with the `lm` covariance matrix method `vcov`. Alternatively, we can use sandwich estimators like `vcovHC`.
- `...`: This is used to pass arguments to the `vcov.` function.

The function returns a model of class `tlseModel` with the optimal selection of knots. For example, we can compare the starting knots of `model1`, with the model selected by the default arguments.

```
print(model1, knots=TRUE)
```

```
## Lists of knots for the treated group
## ************************************
## age:
## 20% 40% 60% 80%
##  19  22  25  28
## re75:
##       40%       60%       80%
##  357.9499 1961.8640 5588.6640
## ed:
## 20% 40% 60% 80%
##   9  10  11  12
## married:
## None
##
## Lists of knots for the Control group
## ************************************
## age:
## 16.66667% 33.33333%       50% 66.66667% 83.33333%
##        18        20        23        26        30
## re75:
##       50% 66.66667% 83.33333%
##  823.2544 2292.1710 6567.3290
## ed:
## 16.66667% 33.33333% 66.66667% 83.33333%
##         9        10        11        12
## married:
## None
```

```
model2 <- selTLSE(model1)
print(model2, knots=TRUE)
```

```
## Lists of knots for the treated group
## ************************************
## age:
```

```
## 20%  60%  80%
##  19   25   28
## re75:
## None
## ed:
## 80%
##  12
## married:
## None
##
## Lists of knots for the Control group
## ************************************
## age:
## None
## re75:
##       50%  83.33333%
##  823.2544  6567.3290
## ed:
## 16.66667%  66.66667%
##         9         11
## married:
## None
```

For example, the method has removed all knots from `re75` for the treated group and kept two knots for the control group. We can then compute the causality measures for the new model. Notice that the selection method and criterion reflects what was used to update the model. In this case, we see FTLSE as selection method and AIC as criterion.

```
causal(model2)
```

```
## Causal Effect using Semiparametric TLSE
## ***************************************
## Selection Method: FTLSE
## Criterion: AIC
##
## ACE = 817.4254
## ACT = 835.2916
## ACN = 804.9401
```

We can compare with other methods:

```
model3 <- selTLSE(model1, method="BTLSE", crit="BIC")
causal(model3)
```

```
## Causal Effect using Semiparametric TLSE
## ***************************************
## Selection Method: BTLSE
## Criterion: BIC
##
## ACE = 818.8162
## ACT = 889.3806
## ACN = 769.5041
```

The `extract` method

The package comes with an `extract` method for objects of class `causaltlse`, which is a required method for creating Latex tables using the `texreg` package. For example, we can compare different methods in a single table.

```
library(texreg)
c1 <- causal(model1)
c2 <- causal(selTLSE(model1, method="BTLSE"))
```

```
c3 <- causal(selTLSE(model1, method="FTLSE"))
texreg(list(SLSE=c1, BTLSE=c2, FTLSE=c3), table=FALSE, digits=4)
```

	SLSE	BTLSE	FTLSE
ACE	814.3083	824.4901	817.4254
	(482.1393)	(481.8267)	(483.0555)
ACT	831.8856	852.4659	835.2916
	(499.4948)	(496.6795)	(499.2405)
ACN	802.0249	804.9401	804.9401
	(498.8671)	(490.4101)	(491.4644)
Num. knots (Control)	12	6	4
Num. knots (Treated)	11	4	4
Num. covariates	4	4	4
Num. obs.	722	722	722
R^2	0.0869	0.0840	0.0812
R^2_{adj}	0.0445	0.0592	0.0590

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$

The `causalTLSE` function

We just saw how to estimate the causal effects step by step. The function `causalTLSE` estimate them in one step, once the model has been created. It returns an object of class `causaltlse` like the `causal` method does, so we can apply the same `print`, `summary` and `predict` and `plot` method to it. The last two can be applied to the object, because it inherits from the `tlseFit` class. The arguments are almost like the ones from the `selTLSE` and `causal` functions.

- **model**: An object of class `tlseModel`.
- **selType**: This is the selection method. We have the choice between “SLSE” (the default), “FTLSE” and “BTLSE”. The SLSE method implies no selection, so all knots from the model are kept. It is therefore identical to estimating the model using the `causal` method.
- **selCrit**: This is the criterion used by the selection method. We have the choice between “AIC” (the default), “BIC” or “ASY”.
- **causal**: What causality measure should the function compute? We have the choice between “All” (the default), “ACT”, “ACE” or “ACT”.
- **seType**: The method to compute the standard error of the causality measures. We have the choice between “analytical” (the default) or “lm”. We have explained the difference when we presented the `causal` method.
- **minPV**: This is a function that returns the p-value threshold. We explained this argument when we presented the `sellTLSE` function.
- **vcov.**: An alternative was to compute the covariance matrix of the least squares estimates.
- `...`: This is used to pass arguments to the `vcov.` function.

For example, we can generate the previous table as follows:

```
c1 <- causalTLSE(model1, selType="SLSE")
c2 <- causalTLSE(model1, selType="BTLSE")
c3 <- causalTLSE(model1, selType="FTLSE")
texreg(list(SLSE=c1, BTLSE=c2, FTLSE=c3), table=FALSE, digits=4)
```

	SLSE	BTLSE	FTLSE
ACE	814.3083	824.4901	817.4254
	(482.1393)	(481.8267)	(483.0555)
ACT	831.8856	852.4659	835.2916
	(499.4948)	(496.6795)	(499.2405)
ACN	802.0249	804.9401	804.9401
	(498.8671)	(490.4101)	(491.4644)
Num. knots (Control)	12	6	4
Num. knots (Treated)	11	4	4
Num. covariates	4	4	4
Num. obs.	722	722	722
R^2	0.0869	0.0840	0.0812
R^2_{adj}	0.0445	0.0592	0.0590

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$

An example with simulated data

In the package, the data set `datSim1` was generated using the following data generating process.

$$\begin{aligned}
Y(0) &= 1 + X + X^2 + e \\
Y(1) &= 1 - 2X + u \\
Z &= \text{Ber}[\Lambda(1+X)] \\
Y &= Y(1)Z + Y(0)(1-Z)
\end{aligned}$$

where X, e and u are standard normal, $\Lambda(x)$ is the CDF of the standard logistic distribution and Ber(p) is the Bernoulli distribution. The true causal effects ACE, ACT and ACN are approximately equal to -1, -1.6903 and 0.5867. We can start by building starting model:

```r
data(simDat1)
mod <- setModel(Y~Z | ~X, data=simDat1)
```

Then we can compare three different methods:

```r
c1 <- causalTLSE(mod, selType="SLSE")
c2 <- causalTLSE(mod, selType="BTLSE", selCrit="BIC")
c3 <- causalTLSE(mod, selType="FTLSE", selCrit="BIC")
texreg(list(SLSE=c1, BTLSE=c2, FTLSE=c3), table=FALSE, digits=4)
```

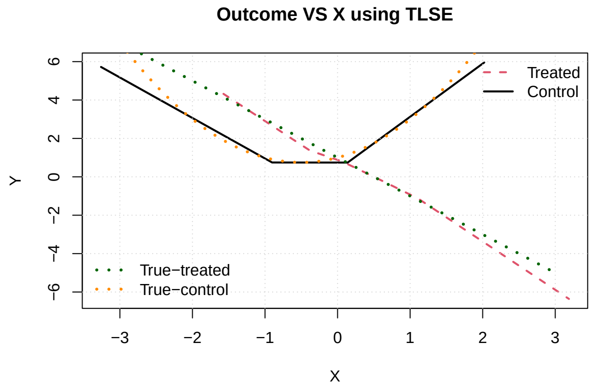
	SLSE	BTLSE	FTLSE
ACE	-1.4396^{***}	-1.4530^{***}	-1.4530^{***}
	(0.2614)	(0.2605)	(0.2605)
ACT	-1.9316^{***}	-1.9316^{***}	-1.9316^{***}
	(0.3030)	(0.3024)	(0.3024)
ACN	-0.0865	-0.1369	-0.1369
	(0.3263)	(0.3224)	(0.3224)
Num. knots (Control)	2	2	2
Num. knots (Treated)	4	0	0
Num. covariates	1	1	1
Num. obs.	300	300	300
R^2	0.7434	0.7386	0.7386
R^2_{adj}	0.7354	0.7342	0.7342

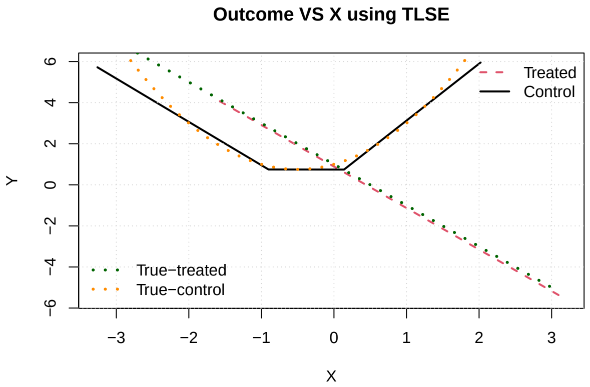
$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$

We see that both selection methods choose to assign 0 knots for the treated group, which is not surprising since the true $f_1(x)$ is linear. We can compare the different fits (we ignore the FTLSE because the selected knots are the same):

```r
plot(c1, "X")
curve(1-2*x, -3,3, col="darkgreen", lty=3, lwd=3, add=TRUE)
curve(1+x+x^2, -3,3, col="darkorange", lty=3, lwd=3, add=TRUE)
```

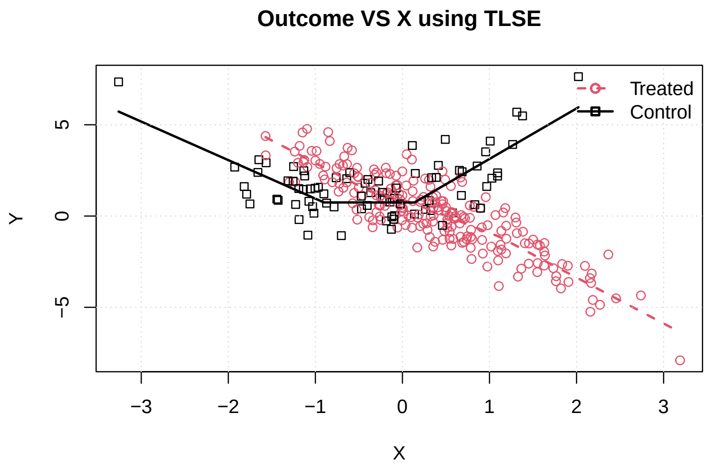
```
legend("bottomleft", c("True-treated","True-control"),
       col=c("darkgreen","darkorange"), lty=3, lwd=3, bty='n')
plot(c2, "X")
curve(1-2*x, -3,3, col="darkgreen", lty=3, lwd=3, add=TRUE)
curve(1+x+x^2, -3,3, col="darkorange", lty=3, lwd=3, add=TRUE)
legend("bottomleft", c("True-treated","True-control"),
       col=c("darkgreen","darkorange"), lty=3, lwd=3, bty='n')
```





We see that the piecewise polynomials are very close to the true $f_1(x)$ and $f_2(x)$. We can see from the folllowing graph how the lines are fit through the observations by group.

```
plot(c1, "X", addPoints=TRUE)
```



An example with another simulated data

The dataset `datSim2` was generated using the following data generating process.

$$
\begin{aligned}
Y(0) &= (1+X)I(X \leq -1) + (-1-X)I(X > -1) + e \\
Y(1) &= (1-2X)I(X \leq 0) + (1+2X)I(X > 0) + e \\
Z &= \text{Ber}[\Lambda(1+X)] \\
Y &= Y(1)Z + Y(0)(1-Z)
\end{aligned}
$$

where $I(A)$ is the indicator function equal to 1 if A is true, X, e and u are standard normal, $\Lambda(x)$ is the CDF of the standard logistic distribution and Ber(p) is the Bernoulli distribution. The true causal effects ACE, ACT and ACN are approximately equal to 3.763, 3.858 and 3.545. We can compare the SLSE, BTLSE with AIC and BTLSE with BIC.

```
data(simDat2)
mod <- setModel(Y~Z | ~X, data=simDat2)
```

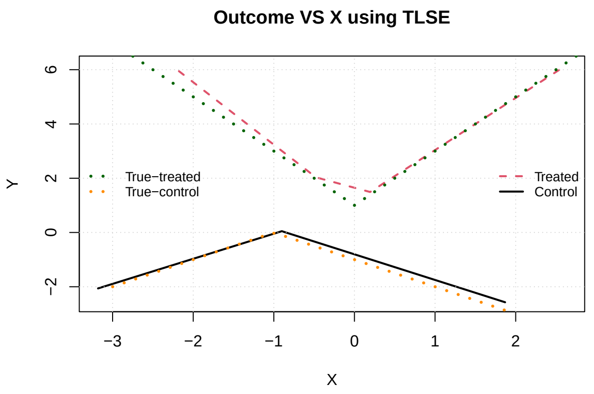
```
c1 <- causalTLSE(mod, selType="SLSE")
c2 <- causalTLSE(mod, selType="BTLSE", selCrit="BIC")
c3 <- causalTLSE(mod, selType="BTLSE", selCrit="AIC")
texreg(list(SLSE=c1, BTLSE.BIC=c2, BTLSE.AIC=c3), table=FALSE, digits=4)
```

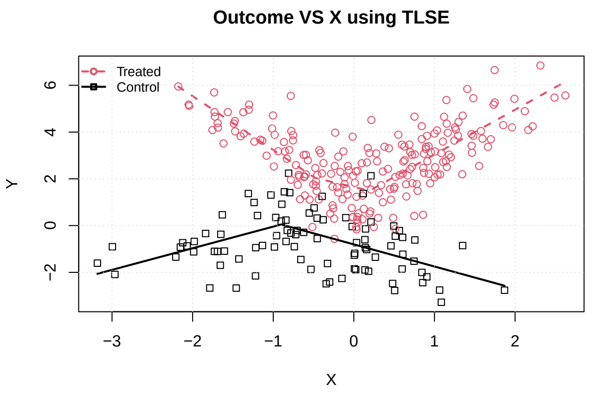
	SLSE	BTLSE.BIC	BTLSE.AIC
ACE	3.9290***	3.9201***	3.9201***
	(0.1703)	(0.1717)	(0.1717)
ACT	3.9552***	3.9404***	3.9404***
	(0.1891)	(0.1904)	(0.1904)
ACN	3.8670***	3.8721***	3.8721***
	(0.2371)	(0.2362)	(0.2362)
Num. knots (Control)	2	1	1
Num. knots (Treated)	3	2	2
Num. covariates	1	1	1
Num. obs.	300	300	300
R^2	0.7833	0.7829	0.7829
R^2_{adj}	0.7774	0.7784	0.7784

$^{***}p < 0.001$; $^{**}p < 0.01$; $^{*}p < 0.05$

The following illustrate the fit of BTLSE-AIC with the true $f_1(x)$ and $f_0(x), and the observations.

```
plot(c2, "X", legendPos="right", cex=.8)
curve((1-2*x)*(x<=0)+(1+2*x)*(x>0), -3,3,
      col="darkgreen", lty=3, lwd=3, add=TRUE)
curve((1+x)*(x<=-1)+(-1-x)*(x>-1),
      -3,3, col="darkorange", lty=3, lwd=3, add=TRUE)
legend("left", c("True-treated","True-control"),
       col=c("darkgreen","darkorange"), lty=3, lwd=3, bty='n', cex=.8)
plot(c2, "X", addPoints=TRUE, legendPos="topleft", cex=.8)
```


Outcome VS X using TLSE
Y
X
True−treated
True−control
Treated
Control


Outcome VS X using TLSE
Treated
Control
Y
X