Introduction to the `bunching` Package

Panos Mavrokonstantis

2019-08-07

Introduction

This vignette is an introduction to the basic toolkit of the `bunching` package, and shows examples of how it can be applied to data that feature bunching. The package offers a flexible implementation of the bunching estimator. This was originally developed to tackle questions in labor and public economics, but can be applied to any setting where a constrained optimization problem involves a discontinuity in a constraint, which can be related to a discontinuity in the observed density of the decision variable. The main aim is to measure behavioral responses to such changes in incentives, by estimating how much excess mass, in an otherwise smooth distribution, can be attributed to the responses to a discrete change in constraints at that same level.

`bunching` allows the user to conduct such bunching analysis in a kink or notch setting and returns a rich set of results. Important features of the package include functionality to control for (different levels of) round-number bunching or other bunching masses within the estimation bandwidth, options to split bins by placing the bunching point as the minimum, median or maximum in its bin (for robustness analysis), and can return estimates of both parametric and reduced-form versions of elasticities associated with the bunching mass. It also provides an exploratory visualization function to speed up pre-analysis, and produces plots in the Chetty et al. (2011) style with lots of options on editing the plot appearance. Further, it returns bootstrapped estimates of all the main estimable parameters, which can be used for further analysis such as incorporation into structural models that rely on bunching moments.

This vignette proceeds by explaining how the `bunching` package estimates the bunching mass, and then provides several examples using simulated data with kinks and notches. It does not cover any theory behind the bunching estimator. For a review of bunching optimization theory, see the companion vignette `bunching_theory`.

Overview of the `bunching` package

Main functions and parameter input options

`bunchit()` is the main function of the package. This does all the analysis and returns a range of results, including a bunching plot. Another function, designed for pre-analysis, is `plot_hist()`. This can be used to inspect the binned data and plot the density of a given vector without having to run any estimations. A quick visualization can help pick appropriate parameters for the inputs required by `bunchit()`.

The main parameters the user must choose for `bunchit()` are:

- **`z_vector`**: the name of the (unbinned) vector to be analyzed
- **`zstar`**: the location of the bunching point
- **`binwidth`**: how wide the grouping bins should be
- **`bins_l`** and **`bins_r`**: how many bins to the left and right of **`zstar`** to consider in the bandwidth
- **`t0`** and **`t1`**: the marginal (average) tax rates below and above **`zstar`** in a kink (notch) setting

Note that without inputs for `t0` and `t1`, no elasticity can be calculated which will throw an error. To avoid this estimation process, use `plot_hist()` for exploratory analysis, which requires these same inputs except for `t0` and `t1`.

The rest of the inputs have set defaults. Among these, the ones that the user may want to experiment with (and which affect estimation) are:

- `binv`: This is the bin version, which controls how the bins are grouped around `zstar`. The default is `"median"`, which places `zstar` in the median position in its bin. The other options are `"min"` and `"max"`, which create bins with `zstar` being in the minimum or maximum position. Default is `"median"`
- `poly`: The order of the polynomial used to fit the bin counts. Default is `9`
- `bins_excl_l` and `bins_excl_r`: How many bins to the left and right of `zstar` to also include in the bunching (i.e. "excluded") region. This is particularly important when the density exhibits diffuse bunching around `zstar`. Defaults are `0`
- `extra_fe`: Other fixed points, featuring a bunching mass (or hole), that the counterfactual estimate should also control for. This is useful when there is another focal point in the bandwidth, which can affect the estimate of the bunching mass at `zstar`. Default is `NA`
- `rn`: Round numbers (up to two) to control for through fixed effects. Default is `NA`
- `n_boot`: How many bootstrapped samples to use to estimate (residual-based) standard errors. Default is `100`
- `correct`: Whether to correct for the integration constraint. Default is `TRUE`
- `correct_above_zu`: When applying the integration constraint correction, this controls where to start shifting the counterfactual distribution up from. If set to `TRUE`, it only shifts bins above z_U (upper bound of bunching region). Default is `FALSE`, which shifts all bins above z^*
- `notch`: Whether the analysis is for a notch or a kink. Default is `FALSE` (kink)
- `force_notch`: In the case of a notch, whether to force the user's choice of `bins_excl_r`. The default is `FALSE`, whereby the upper bound of the excluded region is estimated through an iterative process that equates the bunching and missing masses
- `e_parametric`: Whether to estimate elasticities using the parametric form or the reduced-form specifications. Default is `FALSE` (non-parametric)
- `e_parametric_lb` and `e_parametric_ub`: If both `notch = TRUE` and `e_parametric = TRUE` are chosen, the elasticity is found by a non-linear equation solver. `e_parametric_lb` and `e_parametric_ub` set the lower and upper bound of possible solution values for the elasticity. Defaults are `1e-04` and `3` respectively
- `seed`: A value used as a seed for reproducability of standard errors. Default is `NA`

The rest of the inputs control the plot's output, and are explained in the last section of the vignette.

How does `bunchit()` estimate the bunching mass?

Using the package's helper functions `bin_data()` and `prep_data_for_fit()`, the chosen vector is binned into groups of binwidth δ around the bunching point z^*. The following specification is then run:

$$c_j = \sum_{i=0}^{p} \beta_i (z_j)^i + \sum_{i=z_L}^{z_U} \gamma_i \mathbb{1}[z_j = i] + \sum_{r \in R} \rho_r \mathbb{1}\Big[\frac{z_j}{r} \in \mathbb{N}\Big] + \sum_{k \in K} \theta_k \mathbb{1}\Big[z_j \in K \wedge z_j \notin [z_L, z_U]\Big] + v_j$$

c_j is the observation count in bin j, p is the order of polynomial used to fit the counts, and z_L and z_U stand for the lower and upper region that define the bunching region. The specification also allows the user to control for bunching at round numbers in a set R (defined by `rn`), and for other fixed effects in a set K (`extra_fe`) that feature a bunching mass in the estimation bandwidth outside the bunching range $z \in [z_L, z_U]$ but that are not associated with z^* (through the ρ and θ coefficient vectors respectively).

The specification allows for (up to) two different levels of round number bunching. This is useful in cases that typically feature bunching at round numbers (such as earnings distributions), and (may also) exhibit uneven bunching because some numbers are "rounder" than others. For instance, there could be bunching at all multiples of 1000's and 500's, but bunching at the former being much stronger than at the latter. Not controlling for round number bunching can significantly bias the bunching estimate upwards if z^* is also a round number. This is because some of the observed bunching will be driven by factors unrelated to the

change in incentives driven by the discrete change in the constraint that we want to attribute the bunching to. Similarly, not controlling for other bunching masses can exert a downward bias in the bunching estimate at z^* by biasing the counterfactual estimate upwards.

Given this estimation strategy, the predicted counterfactual density in the absence of the kink is given by:

$$\hat{B}_0 = \sum_{j=z_L}^{z_U} (c_j - \hat{c}_j)$$

and in the case of (the absence of) a notch:

$$\hat{B}_0 = \sum_{j=z_L}^{z^*} (c_j - \hat{c}_j)$$

where $\hat{c}_j$ is the estimated count excluding the contribution of the dummies in the bunching region:

$$\hat{c}_j = \sum_{i=0}^{p} \hat{\beta}_i (z_j)^i + \sum_{r \in R} \hat{\rho}_r \mathbb{1}\Big[\frac{z_j}{r} \in \mathbb{N}\Big] + \sum_{k \in K} \hat{\theta}_k \mathbb{1}\Big[z_j \in K \wedge z_j \notin [z_L, z_U]\Big]$$

$\hat{B}_0$ estimates the excess number of observations locating at z^* due to the kink or notch. To be able to compare bunching masses across different kinks or notches that feature varying heights of counterfactuals, we use a normalization where we divide the total excess mass with the height of the counterfactual at z^*. This returns the *normalized excess mass*, which is a central parameter of interest besides elasticity estimates:

$$\hat{b}_0 = \frac{\hat{B}_0}{\hat{c}_0}$$

The integration constraint correction

The initial estimate $\hat{B}_0$ will be slightly biased, because it ignores the fact that those with counterfactual earnings above $z^* + \Delta z^*$ (those of the marginal buncher) are also exhibiting an interior response to the introduction of a kink or notch at z^*. Hence, what we observe in the actual distribution above the bunching region is not the true counterfactual, since this has been shifted. Technically, it is not straightforward to fully account for this because the response is coming from *all* levels above the bunching region, including those beyond our estimation bandwidth. A feasible solution is to approximate this total response to that among those we do observe in our estimation bandwidth. With a large enough bandwidth (and not extremely large bunching estimates), this typically yields good results. The solution, named the *integration constraint correction*, is to shift the counterfactual distribution lying to the right of z^* upwards until the count of observations under the empirical distribution equals that under the counterfactual distribution. This is done by running:

$$c_j \left(1 + \mathbb{1}[j > z_U] \frac{\hat{B}_0}{\sum_{j=z^*+1}^{\infty} c_j}\right) = \sum_{i=0}^{p} \beta_i (z_j)^i + \sum_{i=z_L}^{z_U} \gamma_i \mathbb{1}[z_j = i] + \sum_{r \in R} \rho_r \mathbb{1}\Big[\frac{z_j}{r} \in \mathbb{N}\Big] + \sum_{k \in K} \theta_k \mathbb{1}\Big[z_j \in K \wedge z_j \notin [z_L, z_U]\Big] + v_j$$

This is estimated by iteration until a fixed point is found. The final $\hat{B}$ is then based on this updated counterfactual. Note that an alternative formulation is to implement the upward shift starting from bin $j = z_U + 1$ instead of z^*. Which alternative used has typically very little effect on the actual bunching estimate (because the counterfactual should not shift much at z^*), but the latter may over-shift the counterfactual in

bins above the bunching region. The next sub-section explains how this may be an issue when setting z_U in the notch setting. The package allows the user to conduct robustness checks of the effect of each alternative, by choosing the preferred version through the input parameter `correct_above_zu` (the default is set to `FALSE`, i.e. shifting starts at the right of z^*).

How are z_L and z_U set?

With kinks, both can be set visually as these will be obvious from a simple inspection of the density. With notches, it is typically easy to visually determine the location of z_L but not of z_U, because it must span both the bunching mass and hole, and the latter could be very diffuse. In this case, it is possible to estimate z_U conjointly with the bunching mass through an iterative procedure relying on the intuition that the bunching mass must be equal to the missing mass. The approach starts at z^*, shifting z_U marginally rightwards until this equality is reached. `bunchit()` allows the user to choose between estimating z_U through this iterative procedure, or by forcing a particular level of z_U instead. The latter can be done by specifying a given value for `bins_excl_r` and setting `force_notch = TRUE` (the default is `FALSE`).

Note that in the case of notches, there is a further complication when choosing to find z_U iteratively, while also applying the integration constraint correction and setting the shifting option `correct_above_zu` to `TRUE`. This approach may shift the counterfactual estimate significantly upwards (in some cases, too much!), especially if the bunching mass is very large, leading to estimates of z_U quite far from z^*. While this will not usually affect the bunching estimate by much, it can distort the appearance of the counterfactual estimate to the right of z^*. It will also reduce the estimate of α, the proportion of observations "stuck" in the hole between z^* and z_D. It is advisable to try different options for these input parameters to find which one works best.

What does `bunchit()` return?

The function returns a list with the following:

- **`plot`**: A plot (a `ggplot2` object) with the observed and estimated counterfactual
- **`data`**: A dataframe with the binned data and other generated variables used for the bunching estimation
- **`cf`**: The estimated counterfactual density
- **`model_fit`**: The coefficients of the fitted model
- **`B`**: The estimated bunching mass
- **`B_vector`**: A vector of bootstrapped estimates of `B`
- **`B_sd`**: The standard deviation of `B_vector`
- **`b`**: The normalized estimated bunching mass
- **`b_vector`**: A vector of bootstrapped estimates of `b`
- **`b_sd`**: The standard deviation of `b_vector`
- **`e`**: The estimated elasticity
- **`e_vector`**: A vector of bootstrapped estimates of `e`
- **`e_sd`**: The standard deviation of `e_vector`
- **`alpha`**: The estimated proportion of observations in the hole (in a notch setting only)
- **`alpha_vector`**: A vector of bootstrapped estimates of `alpha` (in a notch setting only)
- **`alpha_sd`**: The standard deviation of `alpha_vector` (in a notch setting only)
- **`zD`**: The upper bound of the dominated region (in a notch setting only)
- **`zD_bin`**: The bin in which the upper bound of the dominated region is in (in a notch setting only)
- **`zU_bin`**: The bin in which the upper bound of the dominated region is in (in a notch setting only, relevant where `zU` is estimated internally, otherwise it will match the choice of `bins_exl_r`)
- **`marginal_buncher`**: The estimated counterfactual level of the marginal buncher, i.e. $z^* + \Delta z^*$
- **`marginal_buncher_vector`**: A vector of bootstrapped estimates of `marginal_buncher`
- **`marginal_buncher_sd`**: The standard deviation of `marginal_buncher_vector`

Examples

The package's example data

`bunching` comes with some simulated example data, which can be loaded using `data(bunching_data)`. Note that this will return a "lazy" load, so it will not appear as data unless you view or use it in a function. The data consists of a dataframe with two vectors of earnings, named `kink_vector` and `notch_vector`. Both feature bunching at an earnings level of 10000, and range between 8000 and 12000.

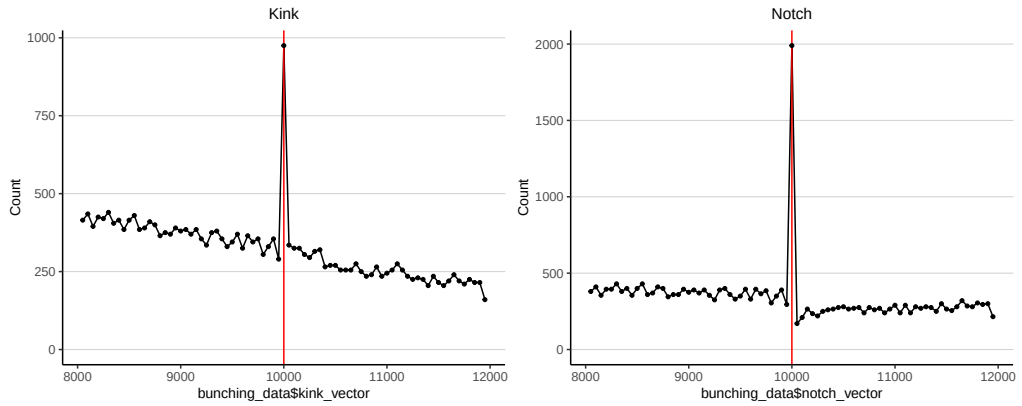
Exploring the data using `plot_hist()`

Let's load the data and visualize the two vectors using the package's `plot_hist()` function. We'll set `binwidth` to 50 and `bins_l` and `bins_r` to 40 to get a plot with a bandwidth of 2000 around `zstar`, which is at 10000.

```
data(bunching_data)

plot_hist(z_vector = bunching_data$kink_vector,
          zstar = 10000, binwidth = 50,
          bins_l = 40, bins_r = 40,
          p_title = "Kink", p_title_size = 11)$plot

plot_hist(z_vector = bunching_data$notch_vector,
          zstar = 10000, binwidth = 50,
          bins_l = 40, bins_r = 40,
          p_title = "Notch", p_title_size = 11)$plot
```



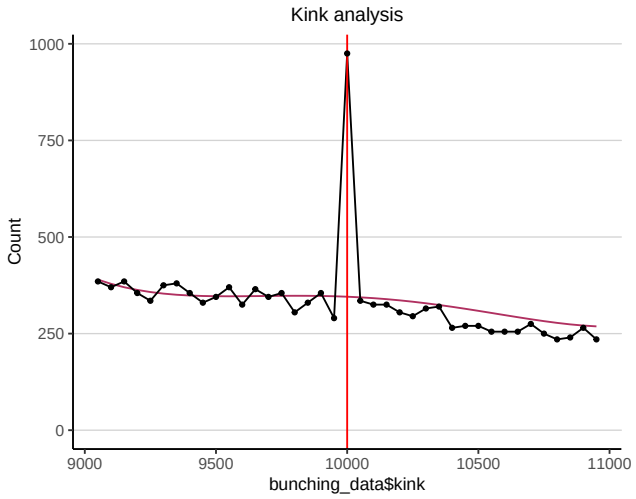
Both plots show sharp bunching at z^*, with the notch case also exhibiting a diffuse hole to the right, confirming they are appropriate for bunching analysis. The next section builds on these to show several examples of how the main function, `bunchit()`, can be used to apply the bunching estimator, including cases that feature diffuse bunching, round number bunching and other bunching points in the bandwidth. We first focus on the kink case.

Kinks

Kink example 1: Sharp bunching

The first example is based on the distribution of our data's `kink_vector`, as plotted above. To apply the estimator, we need to choose an appropriate polynomial, as well as lower and upper limits of the bunching region. Since there does not appear to be any diffuse bunching around 10000, we can simply set the bunching region to be the kink, i.e. `bins_excl_l` and `bins_excl_r` can be left to their default values of zero. The distribution also looks fairly smooth outside the bunching region, so we can use a moderate level of polynomial order for fitting purposes, say `poly` $= 4$. For expositional purposes, let's also restrict the bandwidth to 20 bins below and above z^*, by setting `bins_l` and `bin_r` to 20. We also need to choose values for the marginal tax rate below and above the kink, which we'll set to `t0` $= 0$ and `t1` $= 0.2$.

```
kink1 <- bunchit(z_vector = bunching_data$kink, zstar = 10000, binwidth = 50,
                 bins_l = 20, bins_r = 20, poly = 4, t0 = 0, t1 = .2,
                 p_title = "Kink analysis")
# return plot
kink1$plot
```



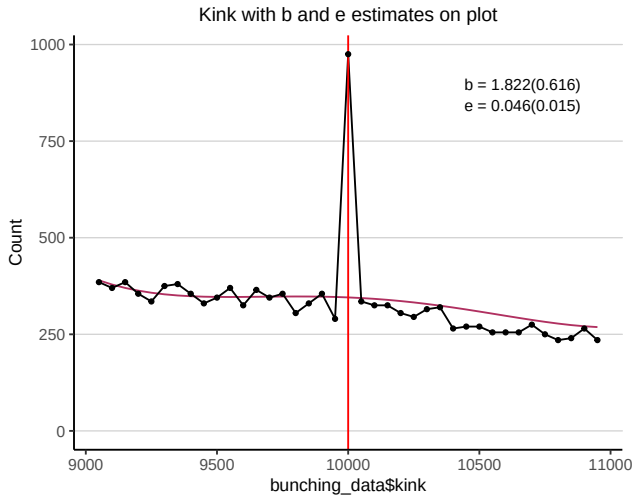
This shows the standard output of the function's `plot` object. The black line with circular markers represents the true density, and the maroon line the counterfactual estimate. The vertical red line marks `zstar`. Let's next return some of the estimated parameters:

```
# Bunching mass
kink1$B
#> [1] 629.511
# Normalized bunching mass
kink1$b
#> [1] 1.822
# Elasticity
kink1$e
#> [1] 0.046
```

We can also report the estimates of the normalized bunching mass and elasticity directly on the plot. This is done by simply setting `p_b = TRUE` and `p_e = TRUE`. This will also display the standard errors, so let's set a `seed` to make these reproducible. As an example, we'll also manually set their y-position through `p_b_e_ypos`.

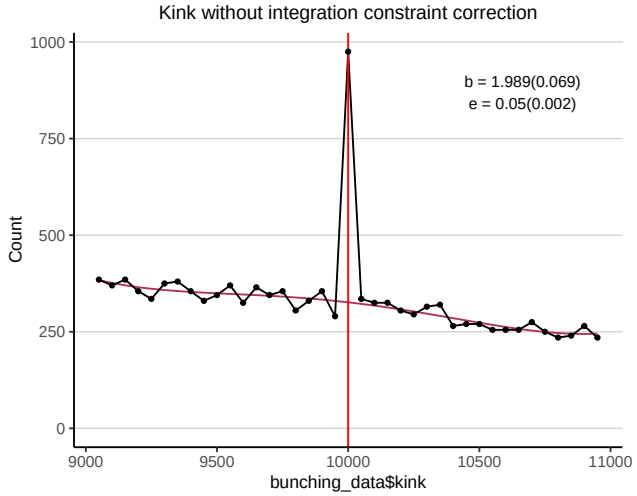
```
kink1_param <- bunchit(z_vector = bunching_data$kink, zstar = 10000, binwidth = 50,
                       bins_l = 20, bins_r = 20, poly = 4, t0 = 0, t1 = .2,
```

```
                        p_b = TRUE, p_e = TRUE, p_b_e_ypos = 870,  seed = 1,
                        p_title = "Kink with b and e estimates on plot")
kink1_param$plot
```



Note how the counterfactual lies somewhat above the actual density, for bins above z^*. This is because the estimator applied the integration constraint correction, which is the default setting. We can override this by setting `correct = FALSE`. The output in this case is:

```
kink1_no_corr <- bunchit(z_vector = bunching_data$kink, zstar = 10000, binwidth = 50,
                         bins_l = 20, bins_r = 20, poly = 4, t0 = 0, t1 = .2,
                         p_b = TRUE, p_e = TRUE, p_b_e_ypos = 870,  seed = 1,
                         correct = FALSE,
                         p_title = "Kink without integration constraint correction")
kink1_no_corr$plot
```

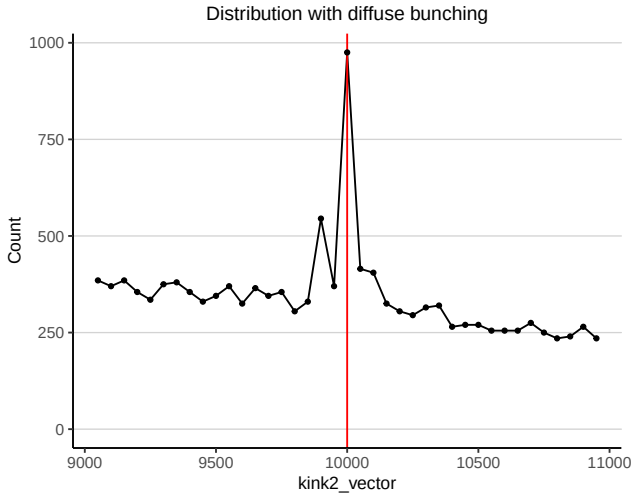


As expected, the counterfactual to the right of z^* is now lower. The effect of this is to pull the height of the counterfactual at z^* to a slightly lower level than before, leading to a larger estimate for the normalized excess mass of 1.989 compared to 1.822.

Kink example 2: Diffuse bunching

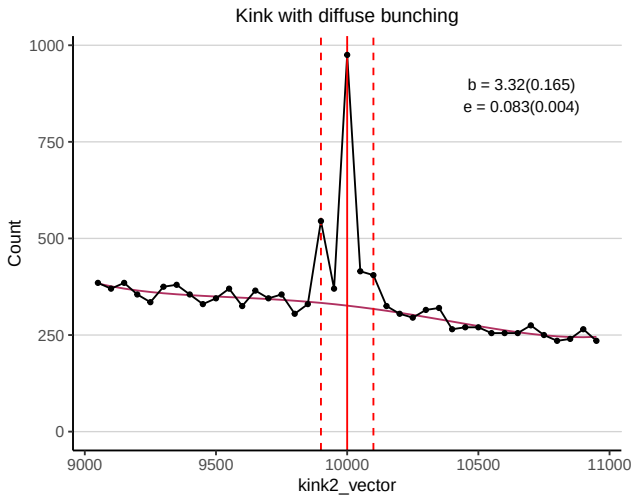
Next, we consider a case where bunching is diffuse around z^*:

```
# create diffuse bunching
bpoint <- 10000; binwidth <- 50
kink2_vector <- c(bunching_data$kink_vector,
                  rep(bpoint - binwidth,80), rep(bpoint - 2*binwidth,190),
                  rep(bpoint + binwidth,80), rep(bpoint + 2*binwidth,80))
# visualization
plot_hist(z_vector = kink2_vector, zstar = 10000, binwidth = 50,
          bins_l = 20, bins_r = 20,  p_title = "Distribution with diffuse bunching")$plot
```



This case exhibits diffuse bunching, spanning the region two bins below to two bins above z^*. This suggests optimization frictions, where agents are bunching but cannot precisely target the kink. To account for this, we can set `bins_excl_l = 2` and `bins_excl_r = 2`.

```
kink2 <- bunchit(z_vector = kink2_vector, zstar = 10000, binwidth = 50,
                 bins_l = 20, bins_r = 20, poly = 4,  t0 = 0, t1 = .2,
                 bins_excl_l = 2, bins_excl_r = 2, correct = FALSE,
                 p_b = TRUE, p_e = TRUE, p_b_e_ypos = 870,
                 p_title = "Kink with diffuse bunching")
kink2$plot
```

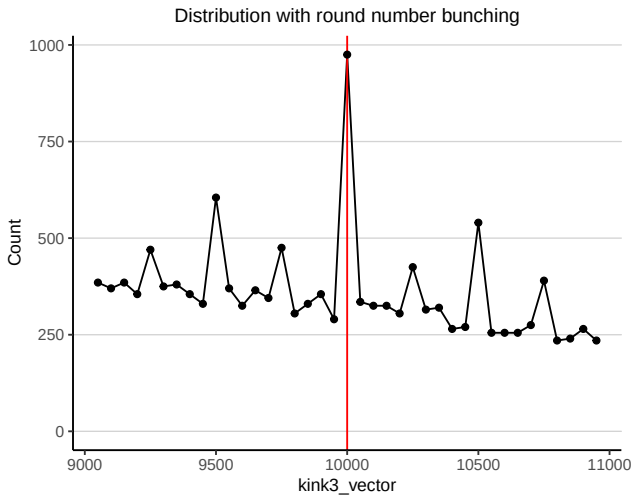


The plot now also marks the bounds of the bunching region by default, using vertical dashed lines. The resulting b estimate is 3.32.

Kink example 3: Round number bunching

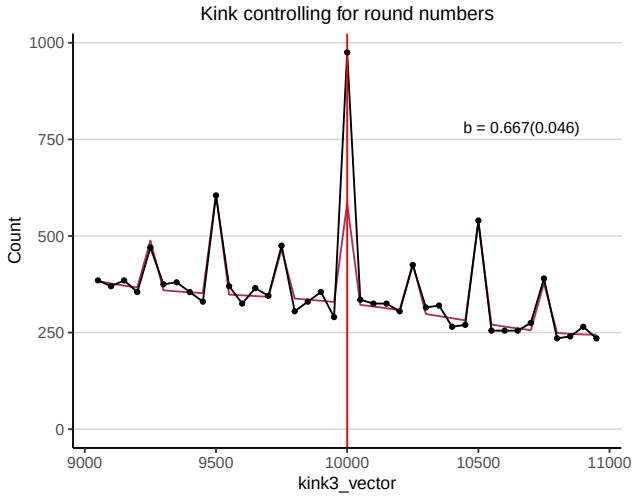
Distributions may also exhibit bunching for reasons unrelated to the examined discontinuity. This is common for instance when analyzing earnings or profits, which tend to be set at or reported in round numbers. If z^* is also at a round number, then some (or all) of the bunching may actually just be because of the round number bunching, and not of the discontinuity in question. In such cases, we want to residualize this effect by introducing controls. As an example, consider the following distribution:

```
# create round number bunching
rn1 <- 500; rn2 <- 250; bpoint <- 10000
kink3_vector <- c(bunching_data$kink_vector,
                  rep(bpoint + rn1, 270),rep(bpoint + 2*rn1,230),
                  rep(bpoint - rn1,260), rep(bpoint - 2*rn1,275),
                  rep(bpoint + rn2, 130), rep(bpoint + 3*rn2,140),
                  rep(bpoint - rn2,120), rep(bpoint - 3*rn2,135))
plot_hist(z_vector = kink3_vector, zstar = 10000, binwidth = 50,
          bins_l = 20, bins_r = 20, p_freq_msize = 1.5,
          p_title = "Distribution with round number bunching")$plot
```



This distribution features the usual bunching at z^*, but also clear round number bunching at multiples of 500 and 250. Moreover, it appears that there are different magnitudes of round number bunching, with that at multiples of 500 being larger than at 250 multiples. This can occur when some numbers are "rounder" than others and therefore get targeted more. In this case, we need to account for them separately. `bunchit()` allows the user to specify (up to) two different levels of round numbers to control for, by passing a vector to `rn`. This is the result:

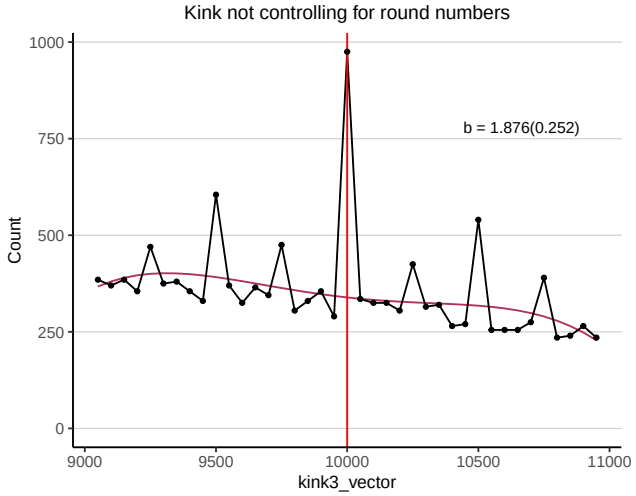
```
kink3_rn<- bunchit(z_vector = kink3_vector, zstar = 10000, binwidth = 50,
                   bins_l = 20, bins_r = 20, poly = 4, t0 = 0, t1 = .2,
                   correct = FALSE, p_b = TRUE, seed = 1, rn = c(250,500),
                   p_title = "Kink controlling for round numbers")
kink3_rn$plot
```



The counterfactual now features spikes at round number multiples of 250 and 500, accounting for the fact that we would have observed such a distribution with bunching masses even in the absence of a kink at z^*. Consequently, the counterfactual at z^* also features a spike, implying that much of the bunching is driven by the targeting of round numbers instead of the actual kink.

If we had not controlled for these, the result would have been:

```
kink3_no_rn <- bunchit(z_vector = kink3_vector, zstar = 10000, binwidth = 50,
                       bins_l = 20, bins_r = 20, poly = 4,  t0 = 0, t1 = .2,
                       correct = FALSE, p_b = TRUE, seed = 1,
                       p_title = "Kink not controlling for round numbers")
kink3_no_rn$plot
```



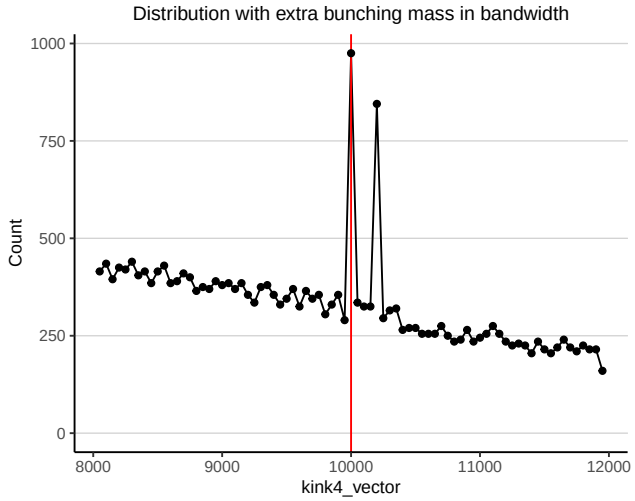
Notice how the counterfactual at z^* is much lower in this case, resulting in a much larger estimated b of 1.876, instead of the corrected estimate of 0.667.

Kink example 4: Other bunching mass in bandwidth

Another case that may be empirically relevant is that of another bunching mass, unrelated to round number bunching, but present in the estimation bandwidth. This can occur when the kink of interest has been recently created by shifting a former kink in the vicinity, creating new bunching mass but also leaving behind residual mass, presumably because optimization frictions precluded those bunching at the former kink to

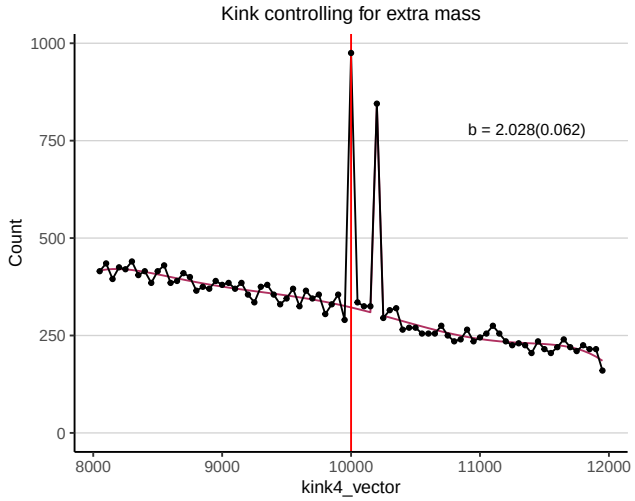
de-bunch.[1] Here is an example of such a case:

```
# create extra bunching mass
kink4_vector <- c(bunching_data$kink_vector, rep(10200,540))
plot_hist(z_vector = kink4_vector, zstar = 10000, binwidth = 50,
          bins_l = 40, bins_r = 40, p_freq_msize = 1.5,
          p_title = "Distribution with extra bunching mass in bandwidth")$plot
```



This distribution exhibits an extra bunching mass at a value of $z = 10200$, which cannot be related to round number bunching. Controlling for this through `extra_fe`, we get:

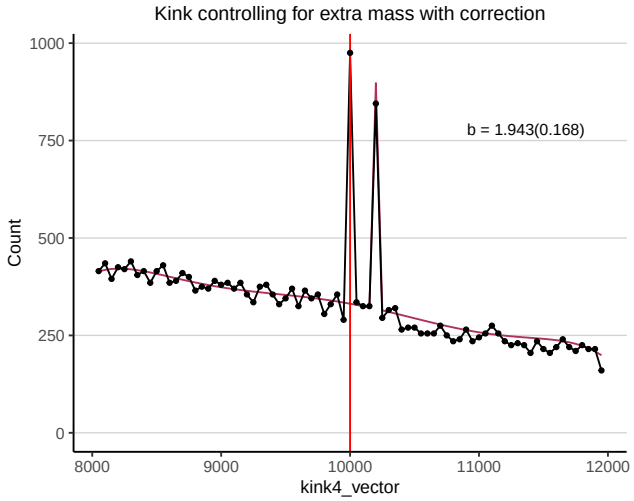
```
kink4_fe <- bunchit(z_vector = kink4_vector, zstar = 10000, binwidth = 50,
                    bins_l = 40, bins_r = 40, poly = 6, t0 = 0, t1 = .2,
                    bins_excl_l = 0, bins_excl_r = 0, correct = FALSE,
                    p_b = TRUE, extra_fe = 10200,
                    p_title = "Kink controlling for extra mass")
kink4_fe$plot
```



By adding this control, the counterfactual goes exactly through the bunching mass at $z = 10200$. If we had also applied the integration constraint correction, then it would lie slightly above it due to the correction's upward shift:

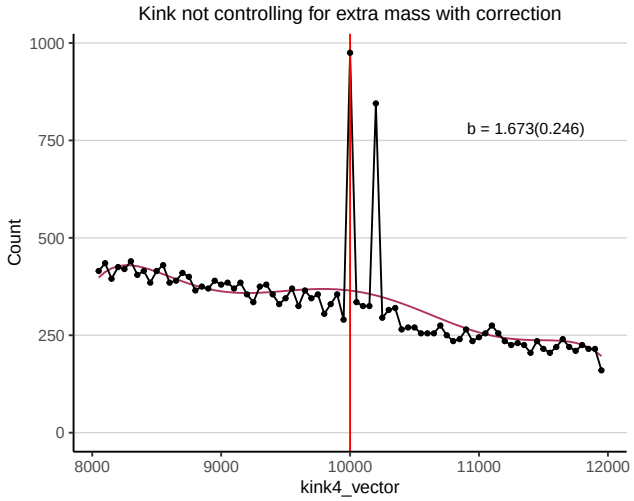
[1] For a study analyzing such a setting, see Mavrokonstantis (2019).

```
kink4_fe_corrected <- bunchit(z_vector = kink4_vector, zstar = 10000, binwidth = 50,
                        bins_l = 40, bins_r = 40, poly = 6,  t0= 0 , t1 = .2,
                        correct = TRUE, p_b=TRUE, extra_fe = 10200, seed = 1,
                        p_title = "Kink controlling for extra mass with correction")
kink4_fe_corrected$plot
```



Applying the integration constraint correction without controlling for the extra bunching mass would have instead returned:

```
kink4_no_fe <- bunchit(z_vector = kink4_vector, zstar = 10000, binwidth = 50,
                        bins_l = 40, bins_r = 40, poly = 6, t0= 0 , t1 = .2,
                        correct = TRUE, p_b=TRUE, seed = 1,
                        p_title = "Kink not controlling for extra mass with correction")
kink4_no_fe$plot
```

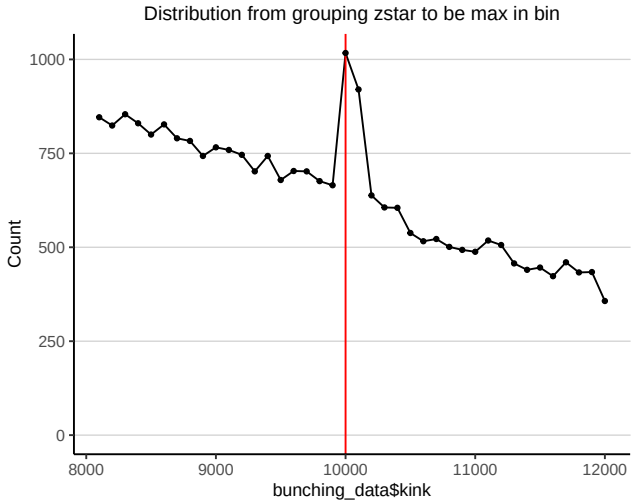


In this case, the counterfactual is biased upwards because the estimator is trying to fit it smoothly through the extra bunching mass, effectively pulling it up, which reduces the b estimate from 1.943 to 1.673.

Kink example 5: Changing some parameters

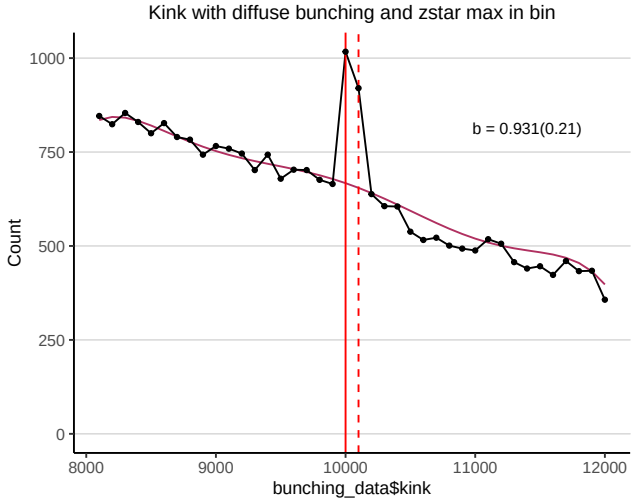
Finally, let's explore changing some parameters. Going back to the first example, let's change `binv` to group the data by forcing `zstar` to be the maximum value in its bin, and set `binwidth` to 100. Let's visualize this before setting further parameters:

```
plot_hist(z_vector = bunching_data$kink, zstar = 10000, binv = "max",
          binwidth = 100, bins_l = 20, bins_r = 20,
          p_title = "Distribution from grouping zstar to be max in bin")$plot
```



This version now creates some diffuse bunching by shifting some to the first bin to the right of z^*, so `bins_excl_r` should be set to 1. We can also increase the flexibility of the polynomial by setting `poly = 9`.

```
kink5 <- bunchit(z_vector = bunching_data$kink, zstar = 10000, binv = "max",
                 binwidth = 100, bins_l = 20, bins_r = 20, bins_excl_r = 1,
                 poly  = 6, t0 = 0, t1 = .2, p_b = TRUE, seed = 1,
                 p_title = "Kink with diffuse bunching and zstar max in bin")
kink5$plot
```



Note how b has now dropped from 1.822 to 0.931, showing that the estimate can be sensitive to these choices. Running the analysis for various values of the main input parameters should be done for robustness, which can help uncover any irregularities in the data that should be taken into account.

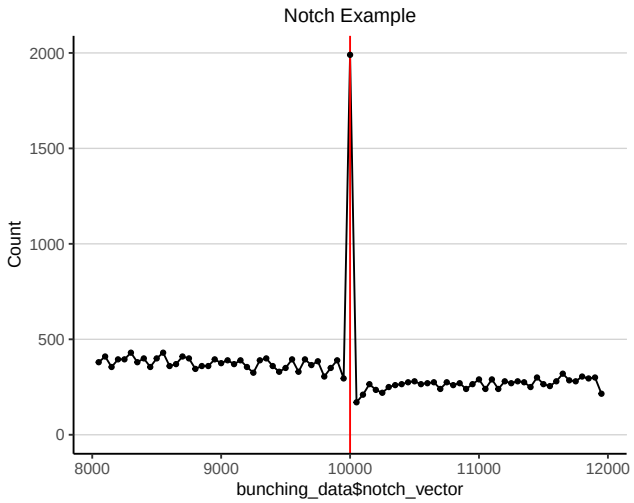
Notches

We now move on to bunching examples in a setting with notches. For this, we'll use `bunching_data$notch_vector` and consider a case of a tax notch where the average tax rate in the first bracket is $t_0 = 0.18$, and jumps to $t_1 = 0.25$ as earnings cross the $z^* = 10000$ threshold.

Notch example 1: Sharp bunching with hole

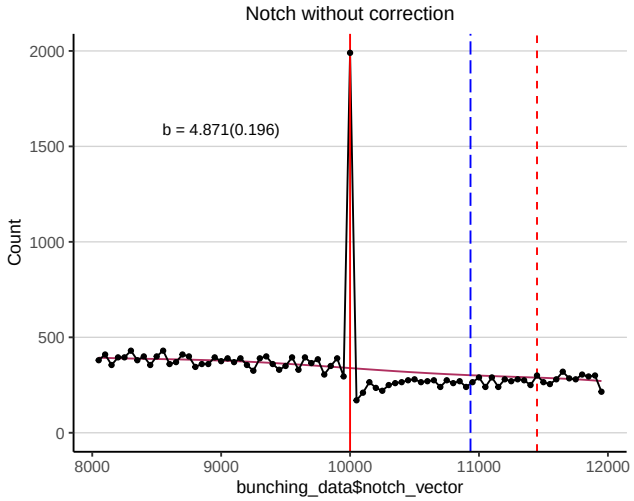
Let's visualize the simulated distribution:

```
plot_hist(z_vector = bunching_data$notch_vector, zstar = 10000, binwidth = 50,
          bins_l = 40, bins_r = 40, p_title = "Notch Example")$plot
```



First, we will apply the bunching estimator without enforcing the integration constraint correction, and then see how this affects our results. We will also not force z_U to a particular value (the default setting). Since the distribution shows no diffuse bunching below z^*, `bins_excl_l` can be kept to the default value of 0. Note that in settings with notches, we must explicitly set `notch = TRUE`. The outcome of these input choices is:

```
notch1 <- bunchit(z_vector = bunching_data$notch_vector, zstar = 10000, binwidth = 50,
                  bins_l = 40, bins_r = 40, poly = 5, t0=0.18, t1=.25, correct = FALSE,
                  notch = TRUE, p_b = TRUE, p_b_e_xpos = 9000, seed = 1,
                  p_title = "Notch without correction")
notch1$plot
```



The plot now includes two vertical lines: the red dashed vertical line is the usual marker for z_U, where this has been estimated internally. The new blue line marks z_D, the upper bound of the dominated region. The plot shows a clear sign of missing mass in the range $z \in (z^*, z_D]$, since the counterfactual lies strictly above the actual density. It also reveals optimization frictions, since optimization theory predicts an empty hole in a frictionless world. We can get (the exact value of) z_D and its bin, and an estimate of the fraction of observations "stuck" in the hole, α, by returning the following objects:

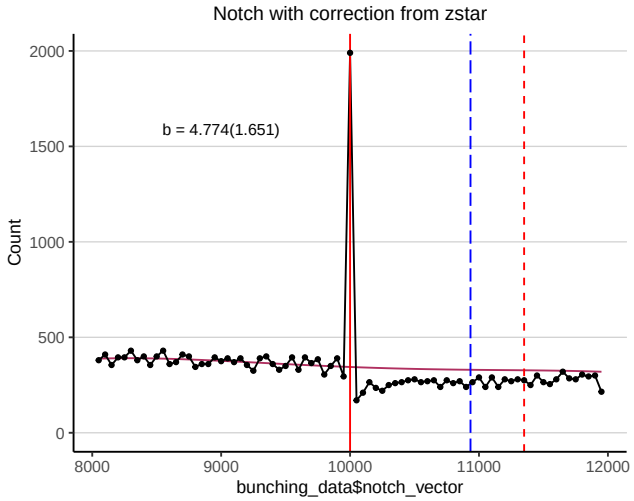
```
# zD
notch1$zD
#> [1] 10934
# zD_bin
notch1$zD_bin
#> [1] 19
# alpha
notch1$alpha
#> [1] 0.794
```

z_D is estimated to be 1.0934×10^4 and lies in bin 19, and the proportion of individuals who are unresponsive due to frictions is 0.794. Further, we can use `notch1$zU_bin` to get z_u, which was estimated at 29 bins above z^*.

Notch example 2: Effect of integration constraint - shifting from z^* Vs z_U

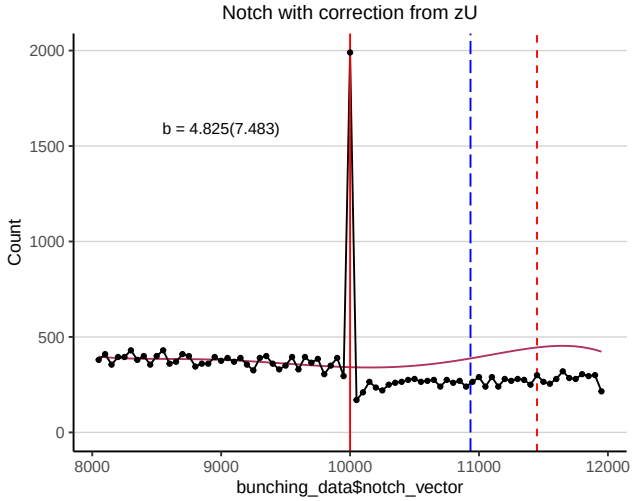
As has been already mentioned, the integration constraint correction can be applied in two different ways. The main difference is whether we start shifting the counterfactual upwards from the bin above z^*, or the bin above z_U. Let's see the results of the first case, where we set `correct = TRUE` and rely on the default `correct_above_zu = FALSE`:

```
notch2 <- bunchit(z_vector = bunching_data$notch_vector, zstar = 10000, binwidth = 50,
                  bins_l = 40, bins_r = 40, poly = 4, t0=0.18, t1=.25, correct = TRUE,
                  notch = TRUE, p_b = TRUE, p_b_e_xpos = 9000, seed = 1,
                  p_title = "Notch with correction from zstar")
notch2$plot
```



Now let's see what happens if we instead set `correct_above_zu = TRUE`:

```
notch3 <- bunchit(z_vector = bunching_data$notch_vector, zstar = 10000, binwidth = 50,
                  bins_l = 40, bins_r = 40, poly = 5, t0=0.18, t1=.25,  correct = TRUE,
                  notch = TRUE, correct_above_zu = TRUE, p_b = TRUE, p_b_e_xpos = 9000,
                  seed = 1, p_title = "Notch with correction from zU")
#> Warning in bunchit(z_vector = bunching_data$notch_vector, zstar = 10000, : estimated zD (upper bound
#>  Are you sure this is a notch?
#>  If yes, check your input choices for t0, t1, force_notch, correct and correct_above_zu.
notch3$plot
```



In this case, we find that the counterfactual distribution is shifted much more. This does not affect the estimate of b by much because the counterfactual at z^* remains stable. It can however make the bootstrapped estimates unstable, blowing up the standard errors. Furthermore, it can decrease α because it shifts the counterfactual significantly upwards and increases the missing mass. This impact is expected in cases where z_U is much higher than z^*, because it approaches the limit of the bandwidth and forces the same mass to be accounted for by fewer bins, shifting the counterfactual to the right of z_U to much higher levels than otherwise, which also pulls it up for bins between z^* and z_U. In this case, it may be better to consider shifting up from z^* (the default), or forcing z_U to some value based on visual inspection.

Finally, note that the last run returns a warning:

estimated zD (upper bound of dominated region) is larger than estimated marginal buncher's counterfactual z

level

Are you sure this is a notch?

If yes, check your input choices for t0, t1, and force_notch.

This is telling us that with this last type of correction, the results imply that $z_D > z^* + \Delta z^*$, which cannot be true. The user should take the warnings (and their suggestions) seriously.

Further optional parameter inputs in `bunchit()` related to plot's appearance

All parameters associated with the plot are prefixed with `p_`:

- **`p_title`**: Title displayed in the plot. Default is empty
- **`p_xtitle`**: x-axis title. Default is the name of the analyzed vector
- **`p_ytitle`**: y-axis title. Default is `"Count"`
- **`p_title_size`**: Size of plot title. Default is `9`
- **`p_axis_title_size`**: Size of x- and y-axis titles. Default is `9`
- **`p_axis_val_size`**: Size of x- and y-axis value labels. Default is `7.5`
- **`p_miny`**: Minimum value of y-axis. Default is `0`
- **`p_maxy`**: Maximum value of y-axis. Default is set internally
- **`p_ybreaks`**: y-axis value(s) at which to add horizontal line markers. Default is optimized internally
- **`p_freq_color`**: Color of the frequency line. Default is `"black"`
- **`p_cf_color`**: Color of the counterfactual line. Default is `"maroon"`
- **`p_zstar_color`**: Color of the vertical line marking z^*. Default is `"red"`
- **`p_grid_major_y_color`**: Color of the y-axis major-y line marker. Default is `"lightgrey"`
- **`p_freq_size`**: Thickness of frequency line. Default is `0.5`
- **`p_freq_msize`**: Size of frequency line markers. Default is `1`
- **`p_cf_size`**: Thickness of counterfactual line. Default is `0.5`
- **`p_zstar_size`**: Thickness of vertical line marking z^*. Default is `0.5`
- **`p_b`**: Whether to show the normalized bunching estimate on the plot. Default is `FALSE`
- **`p_e`**: Whether to show the elasticity estimate on the plot. Default is `FALSE`
- **`p_b_e_xpos`**: x-coordinate of bunching/elasticity estimate on plot. Default is optimized internally
- **`p_b_e_ypos`**: y-coordinate of bunching/elasticity estimate on plot. Default is optimized internally
- **`p_b_e_size`**: Text size of bunching/elasticity estimate on plot. Default is `3`
- **`p_domregion_color`**: Color of vertical line marking upper bound of dominated region (in notch case). Default is `"blue"`
- **`p_domregion_ltype`**: Line type/style of vertical line marking upper bound of dominated region (in notch case). Default is `"longdash"`. Any line type compatible with `geom_vline()` of `ggplot2` will work (e.g. "dotted").

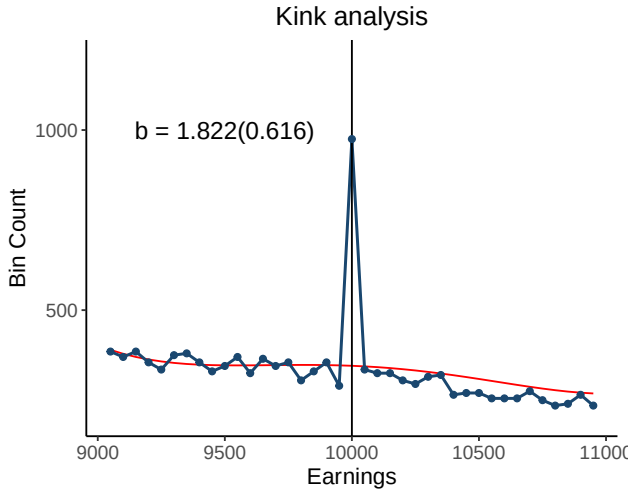
Let's see how to use these in the following examples.

Editing plot options

We will again analyze the original kink vector, but change the plot's appearance in the following ways. First, we'll edit the x- and y-axis titles to "Earnings" and "Bin Count" using `p_xtitle = "Earnings"` and `p_ytitle = "Bin Count"`. Second, we'll drop the horizontal line markers by setting their color to white using `p_grid_major_y_color = "white"`. Third, we'll increase the size of the plots and axis labels: we'll increase the title's size by setting `p_title_size = 15`, the axes' title size with `p_axis_title_size = 13` and the axes' values' size with `p_axis_val_size = 11`. Further, we'll change some colors: the counterfactual line to red using `p_cf_color = "red"`, the true density's color to a navy offshoot using a Hex value: `p_freq_color = "#1A476F"`, and set the z^* marker to black using `p_zstar_color = "black"`. Next, let's change the frequency line's thickness using `p_freq_size = .8`, and increase the size of its markers with `p_freq_msize = 1.5`. We'll also set the minimum y-axis value to 200 and the maximum to 1200 using `p_miny = 200` and

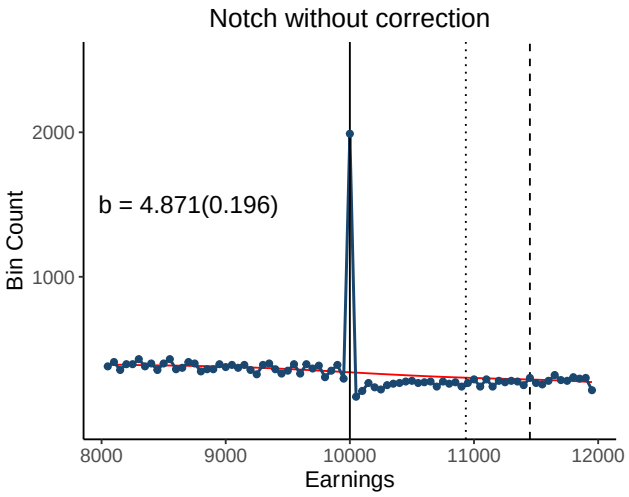
`p_maxy = 1200` but only label the values at 500 and 1000 using `p_ybreaks = c(500,1000)`. Finally, we'll increase the size of the text showing the estimates of b using `p_b_e_size = 5`, and change their x- and y-coordinates to `p_b_e_xpos = 9500` and `p_b_e_ypos = 1000`. This is the resulting plot:

```
kink_p <- bunchit(z_vector = bunching_data$kink, zstar = 10000, binwidth = 50,
                  bins_l = 20, bins_r = 20, poly = 4, t0 = 0, t1 = .2,
                  p_title = "Kink analysis", p_xtitle = "Earnings", p_ytitle = "Bin Count",
                  p_title_size = 15, p_axis_title_size = 13, p_axis_val_size = 11,
                  p_grid_major_y_color = "white",  p_cf_color =  "red",
                  p_freq_color = "#1A476F", p_freq_size = .8, p_zstar_color = "black",
                  p_freq_msize = 1.5, p_miny = 200, p_maxy = 1200, p_ybreaks = c(500,1000),
                  p_b = TRUE, p_b_e_size = 5, p_b_e_xpos = 9500, p_b_e_ypos = 1000,
                  seed = 1)
kink_p$plot
```



The plotting options are the same for kinks and notches. The only addition is that with notches, the user can also change the appearance (color and linetype) of the vertical line marking the upper range of the dominated region, z_D, through `p_domregion_color` and `p_domregion_ltype`. Let's set these to "black" and "dotted" respectively. Note that to remove this line completely, simply set `p_domregion_color = "white"`. We'll also change some of the other settings (`p_miny`, `p_maxy`, `p_ybreaks`, etc.) to better match the notch output. The resulting plot is:

```
notch_p <- bunchit(z_vector = bunching_data$notch_vector, zstar = 10000, binwidth = 50,
                   bins_l = 40, bins_r = 40, poly = 5, t0=0.18, t1=.25, correct = FALSE,
                   notch = TRUE, p_title = "Notch without correction",
                   p_xtitle = "Earnings", p_ytitle = "Bin Count",
                   p_title_size = 15, p_axis_title_size = 13, p_axis_val_size = 11,
                   p_grid_major_y_color = "white",  p_cf_color =  "red",
                   p_freq_color = "#1A476F", p_freq_size = .8, p_zstar_color = "black",
                   p_freq_msize = 1.5, p_maxy = 2500, p_ybreaks = c(1000,2000),
                   p_b = TRUE, p_b_e_size = 5, p_b_e_xpos = 8700, p_b_e_ypos = 1500,
                   seed = 1, p_domregion_color = "black", p_domregion_ltype = "dotted")
notch_p$plot
```



Some final notes on plotting

Please note that there can be a difference in the appearance of marker and font sizes between the plot, as viewed in RStudio, and its exported version, so you may need to experiment with a few settings to find the one that best matches your final document.

If you require further flexibility than what is provided, you can instead build the whole plot from scratch. From the list of results returned from `bunchit()`, simply use `cf` for the estimated counterfactual, and columns `bin` and `freq_orig` from the `data` dataframe for the bins and per-bin count.

References

Chetty, R., J.N. Friedman, Olsen, T. and Pistafferi, L. (2011) "Adjustment Costs, Firm Responses, and Micro vs. Macro Labor Supply Elasticities: Evidence from Danish Tax Records", *Quarterly Journal of Economics*, vol. 126(2), pp. 749-804.

Mavrokonstantis, P. (2019) "Bunching and Adjustment Costs: Evidence from Cypriot Tax Reforms", mimeo.