An R package for calculating repeat-sale price indices[1]

Megan Kirby-McGregor[2] (megan.kirby-mcgregor@canada.ca)
Producer Prices Division, Statistics Canada

Steve Martin (steve.martin5@canada.ca)
Producer Prices Division, Statistics Canada

ABSTRACT

Residential property price indices are valuable indicators of economic activity and financial stability. Calculating pure, quality adjusted price changes using real estate data presents a number of obstacles, however, as properties each have a unique set of price determining characteristics. The repeat-sales method offers a means to construct a constant-quality price index by exploiting multiple sales for the same property over time, although calculating reproducible results on an ongoing basis following this methodology is not trivial. This paper outlines an R package developed at Statistics Canada for calculating a variety of repeat-sale price indices found in the literature and used in practice, as well as supporting tools to get price data into a form that is suitable for calculating an index. The package is designed to be simple, easy to use, and suitable for operational use, without requiring much prior knowledge of the repeat-sales approach or R.

Keywords: *Price index, repeat-sales, Case-Shiller, R.*
JEL codes: *C43, C87, R31.*

1. INTRODUCTION

Residential property price indices relay important information about economic activity and financial stability to analysts, policymakers, investors, financial institutions, and households. Following the global financial crisis of 2008, the G-20 identified real estate prices indices as important indicators of financial soundness. Linked to these efforts, residential property price indices form a core set of data necessary for financial stability analysis under a new tier of the IMF's Special Data Dissemination Standard, known as SDDS plus.

1. This paper has benefited from helpful comments by Kate Burnett-Isaacs and Liam Moore, seminar participants at Statistics Canada and uRos 2019 (Bucharest), and two anonymous referees. The views of the authors do not represent those of Statistics Canada. All errors are ours.

2. Corresponding author.

Calculating price changes using data on residential properties presents a number of obstacles, as properties each have a unique set of price determining characteristics and can vary drastically in quality. Accounting for quality differences between properties is a perennial concern when constructing a property price index, as the goal of a price index is to capture a pure price movement over time. If there are systematic differences in the properties being compared over time that also affect price—so called differences in quality—then a pure price movement cannot be captured using transaction prices alone. Without additional information, there is no way to disentangle a pure price movement from a movement in price due to changing quality over time. The ideal is a constant-quality price index that holds the determinants of price fixed across periods. This allows for an apples-to-apples comparison using transaction prices, and any difference in prices between periods must reflect a pure price movement.

The repeat-sales method offers a means to construct a constant-quality property price index by exploiting multiple sales for the same property over time to control for time-invariant differences in quality between properties. A repeat-sales index can be used to construct a price index for goods and services outside of real estate where changing quality is an issue, although its main application is for calculating property price indices.[1] In practice, a repeat-sales index comes in a number of different flavours. There are two broad classes of repeat-sales price indices—the geometric repeat-sales index (GRS index) proposed by Bailey et al. (1963) and the arithmetic repeat-sales index (ARS index) proposed by Shiller (1991)—as well as various weighting schemes that can be used to weight the prices in the index calculation (e.g., Case and Shiller, 1987, 1989; Abraham and Schauman, 1991; Calhoun, 1996).[2]

Calculating a repeat-sales index it is not trivial without a matrix-based programming language. Although the GRS index can (sometimes) be calculated with a panel-data econometrics library (e.g., the plm package in R (Croissant and Millo, 2008), or the PROC PANEL function in SAS), calculating the ARS index requires creating two non-trivial matrices. Incorporating any of the weighting schemes found in the literature adds further complications, as canned panel-data routines do not usually compute these weights. Calculating

1. See Wang and Zorn (1997) and de Haan and Diewert (2013, chapter 6) for an overview of the repeat-sales method, and Jansen et al. (2008) for an application; see Hansen (2009) for a comparison between the repeat-sales approach and other approaches for constructing a property price index.

2. There are also hybrid repeat-sales indices that can make hedonic quality adjustments using a random-effects model (see de Haan and Diewert, 2013, section 6.25-6.27), although these methods require data on property characteristics which are not usually available if one is contemplating a repeat-sales index.

a repeat-sales index thus requires an analyst to be familiar with panel-data econometrics and either navigate an econometrics library, or explicitly program the index in a matrix language. Neither of these solutions are ideal in a production setting, where an index needs to be produced in a consistent and timely manner, and with minimal use of econometric resources.

This paper outlines an R (R Core Team, 2017) package created at Statistics Canada for calculating repeat-sales prices indices that is accessible to non-econometricians and is suitable for a production environment where the index is calculated frequently, usually monthly or quarterly. The package is designed to be simple, easy to use, and appropriate for operational use, without requiring much prior knowledge of the repeat-sales approach or R. To the best of the authors' knowledge, at the time of writing, the only R package on CRAN that can calculate a repeat-sales price index is the McSpatial package by McMillen (2013). However, this package can only calculate the GRS index, only implements one weighting scheme, and is not generally suitable for production work. The Residential Property Price Index at Statistics Canada uses an ARS index, and this provided the impetus to create a package in-house that can be used in production.

The remainder of the paper proceeds as follows. Section 2 gives the theoretical background for the repeat-sales method, with the aim of outlining the computational steps to make a repeat-sales index; section 3 details the structure of the package, provides an example of how to use the package to compute a repeat-sales index, and outlines the design choices that were made when creating the package; section 4 concludes. The package is available upon request from the authors, and may be made available on CRAN in the future.

2. THE REPEAT-SALES METHOD

2.1 The GRS and ARS indices

Historically the GRS index came before the ARS index, starting with the seminal paper by Bailey et al. (1963), and it is easier to understand the ARS index by first developing the GRS index. Letting time periods be indexed by $t \in \{0,1,\dots,T\}$ and properties be indexed by $i \in \{1,2,\dots,N\}$, the starting point for the GRS index is a structural (hedonic) model of property prices

$$\log(p_{it}) = \log(P_t) + x_{it}\beta + \log(\epsilon_{it}),$$

where p_{it} is the transaction price of property i at time t, P_t is a common city-level price reflecting aggregate price movements, x_{it} is a (row) vector of property characteristics (e.g., number of bedrooms for property i at time t), β is a vector of implicit (hedonic) prices for these characteristics, and ϵ_{it} is an

error term.[1] This is simply a time-dummy hedonic model in which properties can sell more than once (e.g., de Haan and Diewert, 2013, chapter 5). In the context of this model, the constant-quality (geometric) price index in period τ with base period 0 is given by $I_\tau^G \equiv P_\tau / P_0 \cdot 100$. Importantly, P_t is not random - it is a parameter that governs the joint distribution of property prices.

Under the assumption that property characteristics do not change over time (i.e., $x_{it} = x_i$, for all t) and that each property sells twice, the first-difference transformation can be used to deliver

$$\log\left(\frac{p_{is(i)}}{p_{if(i)}}\right) = \log\left(\frac{P_{s(i)}}{P_{f(i)}}\right) + \log\left(\frac{\epsilon_{is(i)}}{\epsilon_{if(i)}}\right) = \sum_{t=1}^{T} D_{it} \log\left(\frac{P_t}{P_0}\right) + \log\left(\frac{\epsilon_{is(i)}}{\epsilon_{if(i)}}\right),$$

where *s(i)* gives the time of the second sale for property *i*, *f(i)* gives the time of the first sale for property *i*, and D_{it} is a variable that takes the value 1 if a property sells for the second time in period *t* (i.e., *s(i) = t*), -1 if the property sells for the first time in period *t* (i.e., *f(i) = t*), and 0 otherwise. The assumption that property characteristics do not change over time means that the percent change in a property's price follows the aggregate percent change in property prices, up to an additive error. Properties that sell three or more times can be incorporated in the first difference transformation by treating consecutive pairs of sales as distinct properties.

Under the assumption that the error terms are strictly exogenous - or, equivalently, that the structural model for price is correctly specified - $E[\log(\epsilon_{it}) | D_{i1}, D_{i2}, \ldots, D_{iT}] = 0$, and so the assumption that property characteristics do not change over time allows for the price index to be identified from the linear regression

$$\log\left(\frac{p_{is(i)}}{p_{if(i)}}\right) = \sum_{t=1}^{T} D_{it}\gamma_t + \log\left(\frac{\epsilon_{is(i)}}{\epsilon_{if(i)}}\right),$$

so that $I_\tau^G \equiv P_\tau / P_0 \cdot 100 = \exp(\gamma_\tau) \cdot 100$. The first difference transformation turns a structural model that depends on property characteristics into an estimating equation that depends on only the time when a property sells.[2] Letting $y_i = \log(p_{is(i)}/p_{if(i)})$ and $D_i = (D_{i1}, D_{i2}, \ldots, D_{iT})$, the entire series of GRS indices from period to period is computed as

1. The structural models in Case and Shiller (1987) and Shiller (1991, section V) are special cases of this model.

2. If property characteristics systematically change over time, then Dit is correlated with the error $\log(\epsilon_{is(i)}/\epsilon_{if(i)})$, although the direction of this correlation depends on whether there is net depreciation or net appreciation of properties over time.

$$(I_1^G, I_2^G, \ldots, I_T^G)' = \exp\left(\left[\sum_{i=1}^{N} D_i' D_i\right]^{-1} \sum_{i=1}^{N} D_i' y_i\right)_{T\times 1} \cdot 100.$$

It is instructive to derive the form of the GRS index as an index number to make the link with the ARS index. Letting $N_f(\tau)$ be the set of properties that sell for the first time in period τ, $N_s(\tau)$ be the set of properties that sell for the second time in period τ, and $N(\tau) = |N_f(\tau)| + |N_s(\tau)|$ (the number of properties that sell in period τ), it can be shown that

$$I_\tau^G = \prod_{i\in N_f(\tau)} \left(\frac{p_{i\tau}}{\frac{p_{is(i)}}{I_{s(i)}^G}}\right)^{\frac{1}{N(\tau)}} \prod_{i\in N_s(\tau)} \left(\frac{p_{i\tau}}{\frac{p_{if(i)}}{I_{f(i)}^G}}\right)^{\frac{1}{N(\tau)}}.$$

The GRS index is simply a matched-model (geometric) index with a twist. Rather than use only property transactions that occur in period 0 and period τ, the index itself is used to extrapolate prices across time for all properties that sell in period τ by deflating prices for sales that do not occur in the base period with that period's index. This allows all properties that sell in period τ to be used in the index calculation, whether that property sells in period 0 or not.

As an alternative to a geometric index, Shiller (1991) proposes the ARS index, denoted by I_τ^A, that simply replaces the geometric averages in the GRS index with arithmetic averages:

$$I_\tau^A = \frac{\sum_{i\in N_f(\tau)} p_{i\tau} + \sum_{i\in N_s(\tau)} p_{i\tau}}{\sum_{i\in N_f(\tau)} \frac{p_{is(i)}}{I_{s(i)}^A} + \sum_{i\in N_s(\tau)} \frac{p_{if(i)}}{I_{f(i)}^A}}.$$

Price relatives are formed in the same way as the GRS index, except now an arithmetic (Laspeyres) index is used to combine price relatives, rather than a geometric index.

Computing the ARS index requires solving a system of equations to calculate the index in each period. As with the GRS index, the ARS index can be computed as a linear regression, although now with a set of instrumental variables - this provides a convenient way to calculate the index and determine its statistical properties. Letting

$$Y_i = \begin{cases} p_{if(i)} \text{ if } f(i) = 0 \\ 0 \text{ if } f(i) > 0 \end{cases},$$

and

$$X_{it} = \begin{cases} -p_{if(i)} \text{ if } f(i) = t \\ p_{is(i)} \text{ if } s(i) = t \\ 0 \text{ otherwise} \end{cases},$$

the ARS index is the reciprocal of the instrumental variables (IV) estimator for the regression

$$Y_i = \sum_{t=1}^{T} X_{it}\, \theta_t + v_i,$$

with D_{it} as an instrument for X_{it}. Letting $X_{it} = (X_{i1}, X_{i2}, ..., X_{iT})$, the entire series of ARS indices from period 1 to period T is computed as

$$(I_1^A, I_2^A, \ldots, I_T^A)' = \text{diag}\left(\left[\sum_{i=1}^{N} D_i'\, X_i\right]^{-1} \sum_{i=1}^{N} D_i' Y_i\right)_{T\times T}^{-1} \begin{bmatrix} 100 \\ 100 \\ \vdots \\ 100 \end{bmatrix}_{T\times 1}.$$

Given a sample of repeat-sales transactions for properties over T periods, both the GRS and ARS indices are consistent under fairly weak conditions on the sampling process (e.g., White, 2001, theorem 3.15; Wooldridge, 2010, theorems 5.1 and 8.1), so that the estimators for these indices converge in probability to their population counterparts.

2.2 Inverse-variance weights

Case and Shiller (1987, 1989) argue that the variance of transaction prices for pairs of sales increases with the holding period for a property, in which case the error term in the regression for the GRS index can be heteroskedastic.[1] This means that the usual OLS standard errors for the GRS index are inconsistent, and the OLS estimator is no longer minimum variance; the same applies to the IV estimator for the ARS index. If the relationship between holding period and variance in transaction prices is known, the generalized least squares (GLS) and generalized instrumental variables (GIV) estimators, using inverse-variance weights, are more efficient alternatives than their unweighted counterparts, and provide a consistent estimator for their standard errors (White, 2001, theorem 4.62; Wooldridge, 2010, theorem 8.5). This has led to a variety of weighting schemes for the GRS and ARS

1. Under the assumption of strict exogeneity, $E(\log(p_{is(i)}/p_{if(i)})|D_{i1}, D_{i2}, \ldots, D_{iT})$ is linear, so that $var(\log(p_{is(i)}/p_{if(i)})\,|D_{i1}, D_{i2}, \ldots, D_{iT}) = var(\log(\epsilon_{is(i)}/\epsilon_{if(i)})\,|D_{i1}, D_{i2}, \ldots, D_{iT})$.

indices, depending on the underlying model for the relationship between variance and holding period, with the original weights by Case and Shiller, and the modifications by Abraham and Schauman (1991) and Calhoun (1996), appearing in application.

Heteroskedasticity is not particularly problematic in application, as standard errors are not reported for the most indices, and there is often a sufficiently large sample of transactions that asymptotic efficiency is not a concern (Wang and Zorn, 1997, section 4.4). Using inverse-variance weights, however, modifies the index values. This is problematic as the GLS and GIV estimators require stronger assumptions than the usual OLS and IV estimators (e.g., the relationship between variance and holding period must be known), and failure of these assumptions can result in the GLS or GIV estimator becoming inconsistent or having poor small-sample properties, as well as failing to produce correct standard errors or give a more efficient estimator (e.g., Angrist and Pischke, 2009, section 3.4.1; Wooldridge, 2010, section 4.2.3). There is also no guarantee that inverse-variance weights can be calculated at any point in time (e.g., Calhoun, 1996), and since the weights affect the index values, the index cannot be calculated if the weights fail. Consequently, weights may serve as a good robustness check, but are unlikely to be useful for producing an index.[1] In practice these weights can have an at most marginal impact on the index (e.g., Goetzmann, 1992; Hansen, 2009), especially with large samples. Previous studies have also found that the unweighted indices are not inferior to the weighted indices (de Haan and Diewert, 2013, section 6.14).

Standard errors that are robust to unknown heteroskedasticity can easily be computed for the GRS and ARS indices without changing the index values, and these are proposed by Shiller (1991, section II). It is also straightforward to generalize these standard errors to account for serially-correlated errors over time for properties that sell three or more times, so called cluster-robust standard errors (e.g., White, 2001, chapter 6; Wooldridge, 2010, chapters 8 and 10). For the GRS index, these standard errors are given by

1. Inverse-variance weights do not offer a correction for quality changes within a property over time. The original weights by Case and Shiller (1987) put less weight on properties with longer holding periods, giving properties with a longer time between sales, and hence more opportunity for quality changes, less weight in the index calculation because price variance increased with holding time. Jansen et al. (2008), however, find a non-monotonic relationship between holding period and variance, so that these weights cannot adjust for quality differences within a property. If the assumptions required for inverse-variance weighting hold, then the weighted and unweighted indices converge in probability to the same value—if weighting offered a correction for quality changes over time, then the weighted index should converge to a different value than the unweighted index.

$$SE_G = \sqrt{\text{diag}\left(\left[\sum_{i=1}^{N} D_i' D_i\right]^{-1}\left(\sum_{i=1}^{N} D_i' e_i e_i' D_i\right)\left[\sum_{i=1}^{N} D_i' D_i\right]^{-1}\right)} \cdot (I_1^G, I_2^G, \ldots, I_T^G)',$$

where $e_i = \log(\epsilon_{is(i)}/\epsilon_{if(i)})$. For the ARS index these standard errors are given by

$$SE_A = \sqrt{\text{diag}\left(\left[\sum_{i=1}^{N} D_i' X_i\right]^{-1}\left(\sum_{i=1}^{N} D_i' \nu_i \nu_i' D_i\right)\left[\sum_{i=1}^{N} X_i' D_i\right]^{-1}\right)} \cdot \left(\frac{I_1^{A^2}}{100}, \frac{I_2^{A^2}}{100}, \ldots, \frac{I_T^{A^2}}{100}\right)'.$$

3. CREATING THE REPEAT-SALES PACKAGE

3.1 Structure of the package

The **rsi** package contains a collection of tools for making the repeat-sale price indices presented in section 2. It provides specialized functions to wrangle transaction data into sales pairs and to calculate an ARS or a GRS index, with or without weights, as well as cluster-robust standard errors. Specialized functions simplify calculating an index in a production setting, with errors and warnings to guide users who may not be familiar with R programming, while ensuring that the indices are calculated in a consistent manner. This section outlines the general structure of the **rsi** package. An example of calculating an index using simulated repeat-sales data follows, detailing additional features of the package.

The **rsi** package is quite small, and has four main components that are visible in the package namespace:

- `rs_pair`: a function that takes a long dataset with repeat-sale information—the usual form in which sales data are captured—and turns it into a dataset of sales pairs.
- `rs_unpair`: a function that does the opposite of rs_pair.
- `rs`: a function that calculates a variety of repeat-sale prices indices using sales-pair data, and is the main function in the package.
- `rs_data`: a data frame that gives some sample repeat-sales data.

Repeat-sales information is likely stored as a long dataset, where each row represents a unique sale of a property. Transforming this dataset into a dataset with sales pairs is equivalent to taking the first difference of the data, and can easily and quickly be done using the `rs_pair` function. The `rs_unpair` function can be used to transform the data back to a long format. Once sales data are in the form of sales pairs, the `rs` function can be used to calculate the index. This function returns the index series, the number of sales

pairs, as well as cluster-robust standard errors, if requested (only available without inverse-variance weights). The user also has the option to print the underlying regression coefficients, the data matrices used to calculate the index, and, if requested, the weights.

In addition to the objects in the visible namespace, `rs` automatically calls the following functions that are not visible:

- `cse`: calculates cluster-robust standard errors using a matrix of residuals, covariates, and instruments.
- `period`: calculates the distance between consecutive elements in a vector and returns a numeric vector.
- `is.consec`: calls `period` to determine if the distance between consecutive elements is constant throughout the vector. A logical value is returned. If this is determined to be false, a warning will be issued to the user.

The output from `rs` is returned as an object of class ***rs***, an S3 class to hold the calculated index and supporting data. Methods for `print` and `as.data.frame` are implemented in **rsi**; other generics such as `coef` and weights also work with the ***rs*** class. Few generics are implements in the package as the normal process flow for calculating the index is to export it as a .csv file to input into a downstream production environment.

The **rsi** package imports the following dependencies:

- **stats**: for R statistical functions, such as `complete.cases`.
- **Matrix**: to generate sparse matrices and perform matrix operations (Bates and Maechler, 2018).
- **zoo**: to work with months and quarters using the `yearmon` and `yearqtr` functions (Zeileis and Grothendieck, 2005).

The number of dependencies was deliberately kept to a minimum when developing the **rsi** package, with the functions relying mostly on base-R features. This is important in a production setting, where dependencies need to be managed to ensure reproducibility.

3.2 Example

The **rsi** package was designed to be easy to use with minimal knowledge of R or the repeat-sales method. As an example, we use the packaged dataset `rs_data`, a randomly generated long dataset with 820 observations of repeat-sales data, to calculate a repeat-sales price index. The 3 column dataset has variables “price”, an integer between 1 and 1,000; “id”, a unique identifier; and “date”, a date between January 2016 and December

2018. This is the canonical form in which sales data are received to calculate a repeat-sales index.

```
> head(rs_data)
   id       date price
1   1 2017-02-01   417
2   1 2018-02-01   853
3 100 2016-07-01   508
4 100 2017-11-01   655
5 100 2018-04-01   211
6 101 2016-05-01   415
```

In practice, the repeat-sales data should always have these three variables. For example, the administrative data used to calculate Statistics Canada's Residential Property Price Index has a unique property identifier (address), the closing date for the sale of the property (i.e., the date at which the title of the land transfers from the seller to the buyer), and the final transaction price at the closing date for the sale of the property. As no other data are required to calculate a repeat-sales index, all the functions in the package are specifically designed to work with these kind of data.

The first task to calculate a repeat-sales index is to turn the data in `rs_data` into pairs of consecutive sales. Once the data are in this form, the index-number formula in section 2.1 can be directly used to calculate the index. The `rs_pair` function can be used to turn a long dataset with repeat-sales information into a dataset of sales pairs that is suitable for calculating a repeat-sales price index: `rs_pair(x, cols)`. This function takes two inputs, x, a data frame (or something that can be coerced into one), and `cols`, a named vector of column names giving a unique identifier to make sales pairs (id), a date variable (date), and a price variable (price). It returns a data frame. The date variable must be stored as a date, or in a form that can be coerced into one (e.g., in the form YYYY-MM-DD). By default, `rs_pair` looks for columns named "id", "date", and "price". The function filters out missing values and duplicate rows, as these observations cannot be used to create sales pairs, as well as observations for which there is only one sale for that id, and returns a warning if observations have been omitted. A warning is also included to signal that no sales pairs exist in the dataset.

```
> head(rs_pair(rs_data))
   id       date price  date_prev price_prev
1   1 2018-02-01   853 2017-02-01        417
2 100 2017-11-01   655 2016-07-01        508
3 100 2018-04-01   211 2017-11-01        655
4 101 2016-12-01   876 2016-05-01        415
5 101 2017-02-01   192 2016-12-01        876
6 101 2018-03-01   900 2017-02-01        192
```

Additional filtering is usually necessary to remove aberrant observations that could distort the index calculation. For example, an extremely short holding period for a property might indicate a distressed sale that would fall outside the scope of the index. Unusually large price increases between sales could suggest major quality changes, and these types of properties are not suitable for calculating a repeat-sales index. In general, this sort of filtering should be considered before passing the dataset to the `rs_pair` function, although in certain cases filters should be applied after the data have been turned into pairs. In the event that data are received as sales pairs, the `rs_unpair` function serves as the inverse of the `rs_pair` function.

```
> head(rs_unpair(rs_pair(rs_data)))
   id       date price
1   1 2017-02-01   417
2   1 2018-02-01   853
3 100 2016-07-01   508
4 100 2017-11-01   655
5 100 2018-04-01   211
6 101 2016-05-01   415
```

This can also help to simplify the filtering process if the data need to be paired and unpaired multiple times to apply the filters. Both of these functions preserve columns that contain data that does not vary over time for a given identifier.

Once the `rs_data` dataset has been turned into a dataset of sales pairs, the index can be calculated using the `rs` function: `rs(x, cols, type, weight, period, ref, se)`. The data frame x is ideally created using the `rs_pair` function, although any data frame of sales pairs can be used. Any rows with missing values are removed (with a warning). The `cols` argument is a named vector giving the column names for the most recent sales date and price (date and price), as well as the previous sales date

and price (date_prev and price_prev). Note that date and date_prev variables must either be stored as dates, or a in a form that can be directly coerced into a date. An optional column with a unique identifier (id) can be included if standard errors are required. By default `rs` looks for columns named "date", "date_prev", "price", "price_prev", and "id" to be consistent with the `rs_pair` and `rs_unpair` functions.

The `type` argument specifies if the GRS index of Bailey et al. (1963) (`type = "grs"`) or the ARS index of Shiller (1991) (`type = "ars"`) is to be computed. If no type is specified, a GRS index is calculated by default. Weights can be applied using the `weight` argument—these include "cs" for Case-Shiller (1987, 1989) weights, "as" for Abraham-Schauman (1991) weights, or "cal" for Calhoun (1996) weights. If no weights are specified, an unweighted index is calculated by default. The `period` argument can be set to "month" for a monthly index, "quarter" for a quarterly index, or "year" for an annual index. All of these arguments support partial matching. In total, the `rs` function can calculate 24 different repeat-sale indices, and is essentially a wrapper to calculate repeat-sale indices with a common structure.

The index reference period, that is, the period when the index is equal to 100, is determined by the `ref` argument. The reference period must be a date, or directly coercible into a date, and the default is the first period in the dataset `x`. Finally, cluster-robust standard errors can be computed for an unweighted ARS or GRS index by specifying `se = TRUE`. The built-in `cse` function calculates the delta-method standard errors in section 2.2 if standard errors are requested in the call to `rs`. Groups are formed using the identification variable in the dataset `x`; if no identification variable is specified, then each sales pair is treated as its own group. Standard errors are not reported for any of the weighted indices, as they are likely too small and can be very misleading.

The `rs` function includes several stops and warnings for missing data. For example, the `is.consec` function, called in `rs`, returns a warning if the index is not being calculated for consecutive periods. This is particularly useful when calculating the index on a dataset with a large number of periods. The program will also stop if there are not at least two periods of data.

Once the `rs_data` dataset has been formatted appropriately by the `rs_pair` function, calculating a repeat-sales index is straightforward and only requires a superficial understanding of R. By default, the function returns the index value for each month, along with the number of sales pairs used. In the interest of space, a quarterly index is calculated instead of a monthly index.

```
> rs(rs_pair(rs_data), type = "ars", period =
"quarter")
      date     index pairs
1  2016 Q1 100.00000    58
2  2016 Q2  92.05103    57
3  2016 Q3 114.50373    83
4  2016 Q4 121.45766    82
5  2017 Q1  98.85645    82
6  2017 Q2  93.58567    99
7  2017 Q3 131.74431    97
8  2017 Q4 122.63508   101
9  2018 Q1 124.34845    72
10 2018 Q2 106.31694    78
11 2018 Q3 121.99040    82
12 2018 Q4 104.21874    65
```

Requesting standard errors will appropriately add another column with these values.

```
> rs(rs_pair(rs_data), type = "ars", period =
"quarter", se = TRUE)
      date     index pairs       se
1  2016 Q1 100.00000    58  0.00000
2  2016 Q2  92.05103    57 13.71049
3  2016 Q3 114.50373    83 14.40602
4  2016 Q4 121.45766    82 16.36837
5  2017 Q1  98.85645    82 13.00594
6  2017 Q2  93.58567    99 12.44592
7  2017 Q3 131.74431    97 17.52286
8  2017 Q4 122.63508   101 17.54517
9  2018 Q1 124.34845    72 15.47995
10 2018 Q2 106.31694    78 13.51269
11 2018 Q3 121.99040    82 16.41320
12 2018 Q4 104.21874    65 15.23143
```

All of the underlying data for the index calculation, along with the weights or standard errors if requested, are stored in the output of the `rs` function.

```
> str(rs(rs_pair(rs_data)))
List of 6
 $ index       : Named num [1:36] 100 113 179 130 113 ...
```

```
  ..- attr(*, "names")= chr [1:36] "Jan 2016" "Feb 2016"
"Mar 2016" ...
 $ pairs       : Named num [1:36] 23 17 24 18 15 30 40 18
37 18 ...
  ..- attr(*, "names")= chr [1:36] "Jan 2016" "Feb 2016"
"Mar 2016" ...
 $ se          : NULL
 $ data        :List of 2
  ..$ Z:Formal class 'dgCMatrix' [package "Matrix"] with
6 slots
  .. .. ..@ i        : int [1:1037] 8 33 46 54 72 73 103
151 160 215 ...
  .. .. ..@ p       : int [1:36] 0 17 41 59 74 104 144 162
199 217 ...
  .. .. ..@ Dim     : int [1:2] 530 35
  .. .. ..@ Dimnames:List of 2
  .. .. .. ..$ : chr [1:530] "1" "100" "100" "101" ...
   .. .. .. ..$ : chr [1:35] "Feb 2016" "Mar 2016" "Apr
2016" ...
  .. .. ..@ x       : num [1:1037] -1 -1 -1 -1 -1 1 -1 -1
-1 -1 ...
  .. .. ..@ factors : list()
  ..$ y: Named num [1:530] 0.716 0.254 -1.133 0.747 -1.518
...
   .. ..- attr(*, "names")= chr [1:530] "1" "100" "100"
"101" ...
 $ coefficients: Named num [1:35] 0.124 0.579 0.263 0.123
0.252 ...
  ..- attr(*, "names")= chr [1:35] "Feb 2016" "Mar 2016"
"Apr 2016" ...
 $ weights     : NULL
 - attr(*, "class")= chr "rs"
```

3.3 Design choices

The **rsi** package was designed with production work in mind, and this necessitated certain design choices when creating the package. An initial design choice that affected subsequent development of the package was the use of as few external packages as possible. Having minimal dependencies makes it much easier to keep track of the versions of packages needed to use the code for production, a requirement for using R at Statistics Canada.

Although the ideal from a versioning point of view would be to use only base-R functionality, the `yearmon` and `yearqtr` functions in the **zoo** package are used in the `rs` function to turn dates into factors for calculating the index. The **zoo** package is well established, and it is easier to use it than program these features in base R. In addition to **zoo**, the **Matrix** package is

also used in `rs` to reduce the computational burden of calculating a repeat-sales index. By having a class and functions to work with sparse matrices, the **Matrix** package produced a significant reduction in both the memory usage and calculation time of an index, especially for large datasets, compared to base-R matrices or regression routines. This is particularly important in a production setting as it means that specific hardware is not needed to calculate an index—any analyst can use their computer to produce the index without needing to worry about running out of memory. As with **zoo**, **Matrix** is well established and has no dependencies outside of base R (or packages that are bundled with base R).

Other R packages (e.g., plm) can be used to compute robust standard errors, and another design choice was having a purpose-built function to compute standard errors, rather than using a more general function for this purpose. Given the expected size of the datasets that are used to calculate a repeat-sales index for residential properties, performing matrix operations to calculate the covariance matrix can be memory intensive. To minimize memory usage, the `rs` function calls the built-in `cse` function when `se = TRUE` and `weight = none`. This function groups observations into clusters by their identification variable, then loops through each cluster to calculate the covariance matrix. This approach to constructing the covariance matrix uses very little memory, but comes at the expense of taking longer to compute.

A final design choice that went into the **rsi** package was the use of simple functions with minimal features, particularly for the `rs` function. The immediate benefit of having few features is that the function can be used by non-econometricians who are not familiar with panel-data econometrics. This also helps to ensure that the output is easily understood, and that an index is always calculated the same way - simplicity forces consistency into the index calculation. Despite this, however, the underlying matrices used to calculate the index are stored as part of the output, and these can be used to do further computations with the index, such as constructing a Wald statistic for the underlying regression coefficients.

4. CONCLUSION

The **rsi** package, developed at Statistics Canada, provides a complete set of tools for calculating commonly-used repeat-sale price indices. Both geometric and arithmetic indices can be calculated, and various weighting schemes found in the literature are available. The package was designed to be accessible to non-econometricians and requires very limited knowledge of R. Although simple to use, the results are robust and reproducible, making it a suitable tool to implement in a production environment.

The **rsi** package is still in its infancy, and is only beginning to be used at Statistics Canada. Future developments for the package could work to incorporate standard errors that can account for both serial correlation over time and spatial dependence (Cameron, Gelbach, and Miller, 2011), as most price indices are stratified and calculated with a hierarchical structure. Diagnostic tools that can be used to perform quality assurance work on an index that is calculated with `rs` could also be a useful addition to the package. Finally, although the main repeat-sale indices can be calculated with `rs`, it may be worth including other variants that can serve as a further robustness check for the unweighted ARS and GRS indices.

References

1. **Abraham, J. M.** and **Schauman, W. S.**, 1991, *New evidence on home prices for Freddie Mac repeat sales*. Real Estate Economics, 19(3): 333-352.
2. **Angrist, J.** and **Pischke, J.-S**., 2009, *Mostly Harmless Econometrics*. Princeton University Press.
3. **Bailey, M., Muth, R., and Nourse, H.**, 1963, *A regression method for real estate price index construction*. Journal of the American Statistical Association, 58(304): 933-942.
4. **Bates, D.** and **Maechler, M.**, 2018, *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 1.2-14. *https://CRAN.R-project.org/package=Matrix*.
5. **Calhoun, C**., 1996, *OFHEO House Price Indexes: HPI Technical Description*. Office of Federal Housing Enterprise Oversight. Retrieved from *http://www.ofheo.gov/Media/Archive/house/hpi_tech.pdf*.
6. **Cameron, A. C., Gelbach, J. B.,** and **Miller, D. L.**, 2011, *Robust inference with multiway clustering*. Journal of Business and Economics Statistics, 29(2): 238-249.
7. **Case, K.** and **Shiller, R**., 1987, *Prices of single-family homes since 1970: New indexes for four cities*. New England Economic Review: 45-56.
8. **Case. K.** and **Shiller, R.**, 1989, *The efficiency of the market for single-family homes*. American Economic Review, 79(1): 125-137.
9. **Croissant, Y.** and **Millo, G.**, 2008, *Panel Data Econometrics in R: The plm Package*. Journal of Statistical Software, 27(2): 1-43.
10. **de Haan, J.** and **Diewert, W. E.** (Eds.), 2013,. *Handbook on Residential Property Prices Indices (RPPIs)*. Eurostat.
11. *Goetzmann, W.*, 1992, *The accuracy of real estate indices: Repeat sale estimators*. Journal of Real Estate Finance and Economics, 5(1): 5-53.
12. **Hansen, J.**, 2009, *Australian house prices: A comparison of hedonic and repeat-sales measures*. Economic Record, 85(269): 132-145.
13. **Jansen, S., de Vries, P., Coolen, H., Lamain, C.,** and **Boelhouwer, P.**, 2008, *Developing a house price index for The Netherlands: A practical application of weighted repeat sales*. Journal of Real Estate Finance and Economics, 37(2): 163-186.
14. **McMillen, D.**, 2013, *McSpatial: Nonparametric spatial data analysis*. R package version 2.0. *https://CRAN.R-project.org/package=McSpatial*.
15. **R Core Team**, 2017, *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/
16. **Shiller, R.**, 1991, *Arithmetic repeat sales price estimators*. Journal of Housing Economics, 1(1): 110-126.

17. **Wang, F.** and **Zorn, P.,** 1997, *Estimating house price growth with repeat sales data: What's the aim of the game?* Journal of Housing Economics, 6: 93-118.
18. **White. H.**, 2001, *Asymptotic Theory for Econometricians* (revised edition). Emerald Group Publishing.
19. **Wooldridge, J.**, 2010, *Econometric Analysis of Cross Section and Panel Data* (2nd edition). MIT University Press.
20. **Zeileis, A.** and **Grothendieck, G**., 2005, *zoo: S3 Infrastructure for Regular and Irregular Time Series*. Journal of Statistical Software, 14(6): 1-27.