WTSS - R interface to Web Time Series Service

Gilberto Queiroz *National Institute for Space Research (INPE), Brazil*
Gilberto Camara *National Institute for Space Research (INPE), Brazil*
Luiz Fernando Assis *National Institute for Space Research (INPE), Brazil*
Pedro Andrade *National Institute for Space Research (INPE), Brazil*
Karine Ferreira *National Institute for Space Research (INPE), Brazil*
Victor Maus *University of Vienna, Austria*

The WTSS-R package is a front-end to the Web Time Series Service \ (WTSS) that offers time series of remote sensing data using a simple API. A WTSS server takes as input a n Earth observation data cube, that has a spatial and a temporal dimension and can have multiple bands attributes. The WTSS API has three commands, which are are *list_coverages*, that returns a list of coverages available in the server; (b) *describe_coverage*, that returns the metadata for a given coverage; (c) *time_series*, that returns a time series for a spatio-temporal location.

Introduction

Recently, the concept of *data cubes* has emerged as a relevant paradigm for organising Earth observation data for performing analysis. In an intuitive sense, data cubes are temporal sequences of images from the same geographical area. This sequence is organised consistently to allow algorithms to explore the data in both the spatial and temporal directions[Appel2019]. Data cubes are thus an efficient way to explore satellite image archives spanning years or even decades.

Data cubes rely on the fact that Earth observation satellites revisit the same place at regular intervals. Thus measures can be calibrated so that observations of the same place in different times are comparable. These calibrated observations can be organised in regular intervals, so that each measure from sensor is mapped into a three dimensional multivariate array in space-time. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of remote sensing images which shows the same region at n consecutive times $T = \{t_1, t_2, \ldots, t_n\}$. Each location *[x, y, t]* of a pixel in an image (latitude, longitude, time) maps to a *[i, j, k]* position in a 3D array. Each array position *[i, j, k]* will be associated to a set of attributes values $A = \{a_1, a_2, \ldots, a_m\}$ which are the sensor measurements at each location in space-time (see figure below}). For optical sensors, these observations are proportional to Earth's reflexion of the incoming solar radiation at different wavelengths of the electromagnetic spectrum.

In what follows, we use the term "coverage" in the same sense as the definition of a "data cube", proposed from Appel and Pebesma [Appel and Pebesma, 2019]: *A regular, dense Earth observation (EO) data cube is a four-dimensional array with dimensions x (longitude or easting), y (latitude or northing), and time, with the following properties:*

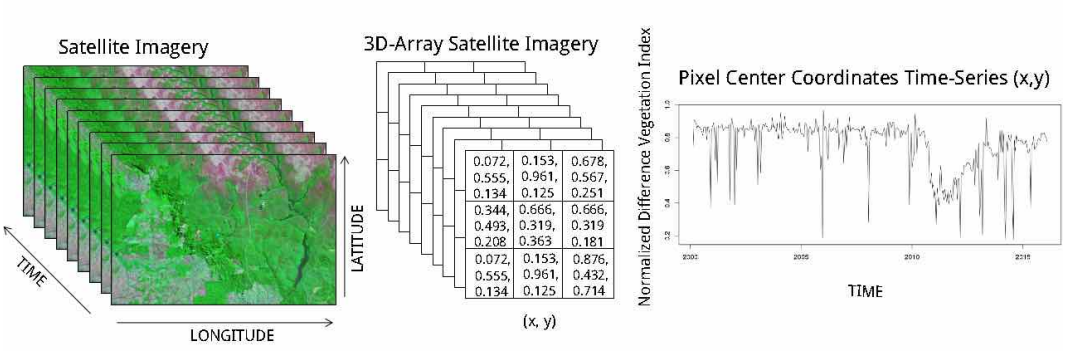

Figure 1: A Normalized Difference Vegetation Index (NDVI) time series

1. Spatial dimensions refer to a single spatial reference system (SRS)};

2. Cells of a data cube have a constant spatial size (with regard to the cube's SRS)};

3. The spatial reference is defined by a simple offset and the cell size per axis, i.e., the cube axes are aligned with the SRS axes;}

4. The temporal reference is a known set of non-overlapping temporal intervals; each temporal interval is defined by a simple start date/time and the temporal duration of cells;}

5. All cells have the same set of attributes; each attribute can be a scalar or a vector;}

6. For every combination of dimensions, a cell has a single set of attribute values.}

This definition of EO data cubes reflects their construction. Data cubes are built of 2D images collected on a specific date constrained by satellite orbits. These images are then grouped together in arbitrary intervals. In principle, a data cube can have an image on time 1, another image at time 5, another at time 8, and so on. Since interpolation in space is different from interpolation in time, a data set composed by these images would make a valid data cube. Space needs to be compact (dense) in a data cube, but time does not need to be. Thus, a sequence of 2D + time images need not be dense. 2D images in a data cube can correspond to different instants of acquisition.

Web Services for Earth Observation Data Cubes

Web services interfaces isolate users from concrete implementations, thus increasing portability There are many ways to implement Earth observation data cubes. One of the earlier approaches is to break large images in chunks and store them in an object-relational DBMS, as done in RASDAMAN [Baumann et al., 1998]. Google Earth Engine[Gorelick et al., 2017] uses a massive parallel design; image data is partitioned in hundreds or thousands of CPUs, processed separately and later aggregated using map-reduce techniques. The Open Data Cube software, after making the required geometric

and radiometric transformations in the images, stores them in files, which are indexed in a PostgreSQL database [Lewis et al., 2017]. Similar file-based approaches are used in the R package "gdalcubes" [Appel and Pebesma, 2019]. Each of these designs is associated to a different API, thus making compatibility and interoperability between them hard to achieve.

Given the different designs of Earth observation data cubes, and their inherent limitations for achieving interoperability, one natural approach is to propose Web services that would present a unified interface for accessing them. Web services interfaces isolate users from concrete implementations, thus increasing portability[Ferris and Farrell, 2003]. In the geospatial domain, web service standards developed by the Open Geospatial Consortium have achieved considerable impact [Percivall, 2010]. Since services such as WMS (Web Map Service) and WCS (Web Coverage Service) are used worldwide with success, it is natural to ask: *how to design web services for big Earth observation (EO) data?* The current work addresses this issue and proposes WTSS (Web Time Series Service), a new services for big EO data.

Satellite Image Time Series

Given a series of remote sensing snapshots, we can reorganise them into a set of time series. A satellite image time series is obtained by taking measurements in the same pixel location (x, y) in consecutive times $t_1, ..., t_m$, as shown in the figure below.

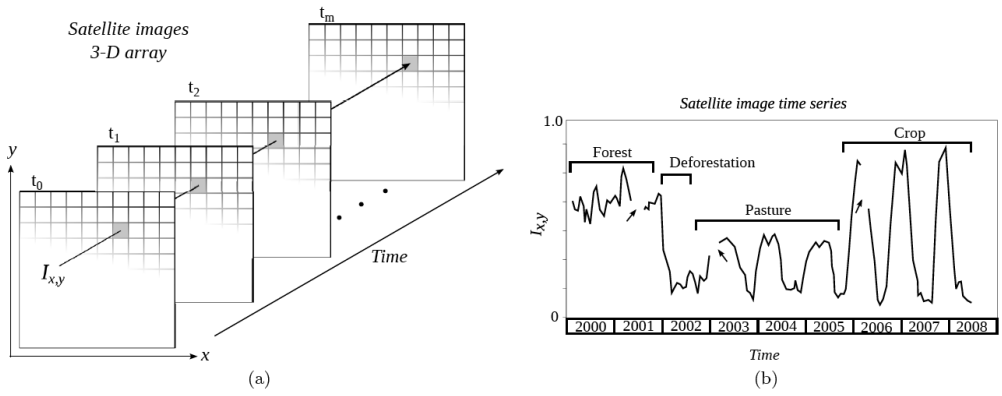

Figure 2: Events associated to a time series

Recent results in the literature show that analysis of satellite image times series enables extracting long-term trends of land change \citep[Pasquarella et al., 2016]. Such information which would be harder to obtain by processing 2D images separately. These analysis are supported by algorithms such as TWDTW [Maus et al., 2016], TIMESTAT [Jonsson and Eklundh, 2004] and BFAST [Verbesselt et al., 2010]. These algorithms process individual time-series and combine the results for selected periods to generate classified maps.

For example, classification methods such as TWDTW [Maus et al., 2019] can then break an image time series into a set of intervals. As an example, the figure below shows four events extracted from a remote sensing time series, expressed in terms of the its

intervals. From 2000 to 2001 the area was a forest that was deforested in 2002. From 2003 to 2005 the area it was used for pasture and from 2005 to 2008, it was transformed into a cropland. This kind of classification is done by algorithms such that split a time series into a set of events. Combining snapshots with time series, scientists can explore the full depth of big remote sensing data archives.

Web Time Series Service (WTSS)

Motivated by the need to retrieve satellite image time series from large 3D arrays, we have designed and implemented the Web Time Series Service (WTSS). A WTSS server takes as input a 3D array. Each array has a spatial and a temporal dimension and can be multidimensional in terms of its attributes. The WTSS service is independent of the actual data architecture used for 3D array store. It can work with solutions such as flat files, MapReduce distributed datasets, array databases or object-relational databases.

The WTSS API has three commands: (a) *list_coverages* that returns a list of coverages available in the server; (b) *describe_coverage* that returns the metadata for a given coverage; (c) *time_series* that returns a time series for a spatio-temporal location.

Connecting to a WTSS server

The first step towards using the service is connecting to a server that supports the WTSS protocol. Currenlty, Brazil's National Insitute for Space Research (INPE) runs such a service. In the package, the connection is enabled by using the URL of the service. The package informs if the connection has been correctly made.

```
# Connect to the WTSS server at INPE Brazil
wtss_inpe <-  wtss::WTSS("http://www.esensing.dpi.inpe.br/wtss/")
```

```
## Connected to WTSS server at http://www.esensing.dpi.inpe.br/wtss/
```

Listing coverages available at the WTSS server

This operation allows clients to retrieve the capabilities provided by any server that implements WTSS. It returns a list of coverage names available in a server instance.

```
# Connect to the WTSS server at INPE Brazil
wtss::list_coverages(wtss_inpe)
```

```
## Object of Class WTSS
## server-url:  http://www.esensing.dpi.inpe.br/wtss/
## Coverages: MOD13Q1 MOD13Q1_M
```

Describing a coverage from the WTSS server

This operation returns the metadata for a given coverage identified by its name. It includes its range in the spatial and temporal dimensions.

```
# Connect to the WTSS server at INPE Brazil
wtss::describe_coverage(wtss_inpe, name = "MOD13Q1")
```

```
## ------------------------------------------------------------------
## WTSS server URL = http://www.esensing.dpi.inpe.br/wtss/
## Coverage = MOD13Q1
##
## satellite  sensor  bands
## ---------  ------  ----------------------------------------
## TERRA      MODIS   c("mir", "blue", "nir", "red", "evi", "ndvi")
##
##
## |scale_factors                                                                  |
## |:------------------------------------------------------------------------------|
## |c(mir = 1e-04, blue = 1e-04, nir = 1e-04, red = 1e-04, evi = 1e-04, ndvi = 1e-04)|
##
##
## |minimum_values                                                  |
## |:---------------------------------------------------------------|
## |c(mir = 0, blue = 0, nir = 0, red = 0, evi = -2000, ndvi = -2000)|
##
##
## |maximum_values                                                                 |
## |:------------------------------------------------------------------------------|
## |c(mir = 10000, blue = 10000, nir = 10000, red = 10000, evi = 10000, ndvi = 10000)|
##
##
## nrows  ncols       xmin  xmax  ymin  ymax      xres      yres  crs
## ----- ----- --------- ---- ---- ---- -------- -------- ------------------------------
## 24000  24000 -81.23413  -30  -40   10 0.002087 0.002087 +proj=longlat +datum=WGS84 +no_de
##
## Timeline - 452 time steps
## start_date: 2000-02-18 end_date: 2019-09-30
## ------------------------------------------------------------------

## Coverage description saved in WTSS object
```

The coverage description is saved as a tibble in the wtss object, to be used whenever required.

```
# Coverage description available in the wtss object
wtss_inpe$description
```

```
## # A tibble: 1 x 19
##   URL   satellite sensor name  bands scale_factors missing_values
##   <chr> <chr>     <chr>  <chr> <lis> <list>        <list>
## 1 http~ TERRA     MODIS  MOD1~ <chr~ <dbl [6]>     <dbl [6]>
## # ... with 12 more variables: minimum_values <list>,
## #   maximum_values <list>, timeline <list>, nrows <dbl>, ncols <dbl>,
## #   xmin <dbl>, xmax <dbl>, ymin <dbl>, ymax <dbl>, xres <dbl>,
## #   yres <dbl>, crs <chr>
```

Obtaining a time series

This operation requests the time series of values of a coverage attribute at a given location. Its parameters are: (a) *wtss.obj*: either a WTSS object (created by the operation wtss::WTSS as shown above) or a valid WTSS server URL; (b) *name*: Coverage name; (c) *attributes*: vector of band names (optional). If omitted, all bands are retrieved; (d) *longitude*: longitude in WGS84 coordinate system; (e) *latitude*: Latitude in WGS84 coordinate system; (f) *start_date* (optional): Start date in the format yyyy-mm-dd or yyyy-mm depending on the coverage. If omitted, the first date on the timeline is used; (g) *end_date*(optional): End date in the format yyyy-mm-dd or yyyy-mm depending on the coverage. If omitted, the last date of the timeline is used.

```
# Request a time series from the "MOD13Q1" coverage
ts   <- wtss::time_series(wtss_inpe, name = "MOD13Q1",
        attributes = c("ndvi","evi"), longitude = -45.00, latitude  = -12.00,
        start_date = "2000-02-18", end_date = "2016-12-18")
ts
```

```
## # A tibble: 1 x 6
##   longitude latitude start_date end_date   coverage time_series
##       <dbl>    <dbl> <date>     <date>     <chr>    <list>
## 1       -45      -12 2000-02-18 2016-12-18 MOD13Q1  <tibble [388 x 3]>
```

The result of the operation is a `tibble`, which is a generalization of a `data.frame`, the usual way in *R* to organise data in tables. Tibbles are part of the `tidyverse`, a collection of R packages designed to work together in data manipulation [Wickham and Grolemund, 2017]. The tibble contains data and metadata. The first six columns contain the metadata: satellite, sensor, spatial and temporal information, and the coverage from where the data has been extracted. The spatial location is given in longitude and latitude coordinates for the "WGS84" ellipsoid. The `time_series` column contains the time series data for each spatiotemporal location. This data is also organized as a tibble, with a column with the dates and the other columns with the values for each spectral band.

```r
# Showing the contents of a time series
ts$time_series[[1]]
```

```
## # A tibble: 388 x 3
##    Index       ndvi   evi
##    <date>     <dbl> <dbl>
##  1 2000-02-18 0.374 0.302
##  2 2000-03-05 0.820 0.497
##  3 2000-03-21 0.802 0.481
##  4 2000-04-06 0.809 0.432
##  5 2000-04-22 0.749 0.409
##  6 2000-05-08 0.727 0.394
##  7 2000-05-24 0.698 0.374
##  8 2000-06-09 0.654 0.325
##  9 2000-06-25 0.608 0.310
## 10 2000-07-11 0.583 0.291
## # ... with 378 more rows
```

Plotting the time series

For convenience, the **WTSS** package provides a convenience funtion for plotting the time series.

```r
# Plotting the contents of a time series
plot(ts)
```

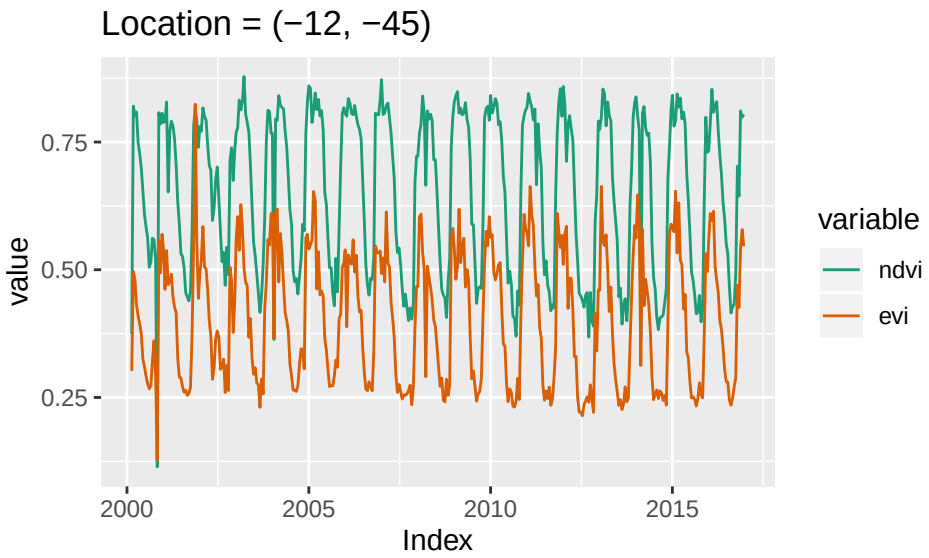

Figure 3: A Normalized Difference Vegetation Index (NDVI) time series

Conversion to "zoo" and "ts" formats

Since many time series analysis functions in R require data to be made available in the "zoo" and "ts" formats, the *wtss* package provides two convenience functions: *wtss_to_zoo* and "*wtss_to_ts*. The example below shows the detection of trends in a series converted to the"ts" format using the BFAST package [Verbesselt et al., 2010].

```
library(bfast)
```

```
## Registered S3 method overwritten by 'xts':
##   method     from
##   as.zoo.xts zoo
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
## Registered S3 methods overwritten by 'forecast':
##   method             from
##   fitted.fracdiff    fracdiff
##   residuals.fracdiff fracdiff
```

```
# create a connection using a serverUrl
server <-  wtss::WTSS("http://www.esensing.dpi.inpe.br/wtss/")
```

```
## Connected to WTSS server at http://www.esensing.dpi.inpe.br/wtss/
```

```
# get a time series for the "ndvi" attribute
ndvi_wtss <- wtss::time_series(server, "MOD13Q1", attributes = c("ndvi"),
                          latitude = -10.408, longitude = -53.495,
                          start = "2000-02-18", end = "2016-01-01")

# convert to ts
ndvi_ts <- wtss::wtss_to_ts(ndvi_wtss, band = "ndvi")

# detect trends
bf <- bfast::bfast01(ndvi_ts)
# plot the result
plot(bf)
```

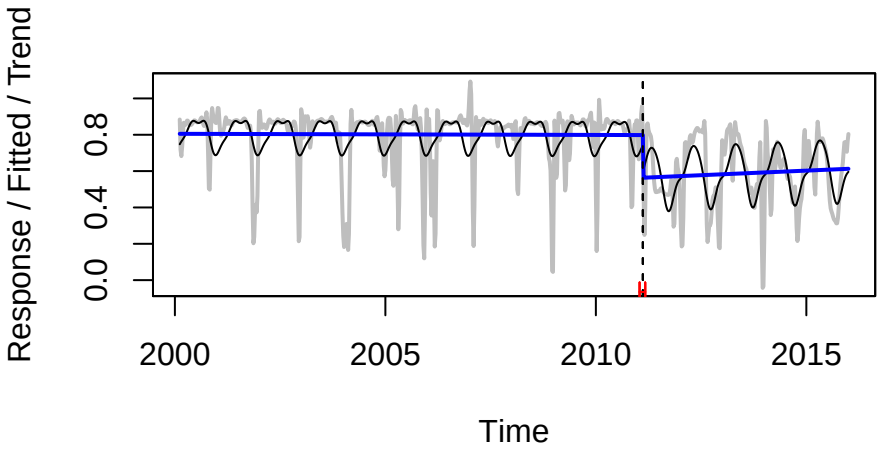

Figure 4: Breaks on an NDVI time series detected by BFAST

References

Marius Appel and Edzer Pebesma. On-demand processing of data cubes from satellite image collections with the gdalcubes library. *Data*, 4(3), 2019.

P. Baumann, A. Dehmel, P. Furtado, R. Ritsch, and N. Widmann. The multidimensional database system RasDaMan. *ACM SIGMOD Record*, 27(2):575–577, 1998.

Christopher Ferris and Joel Farrell. What are web services? *Communications of the ACM*, 46(6):31, 2003.

Noel Gorelick, Matt Hancher, Mike Dixon, Simon Ilyushchenko, David Thau, and Rebecca Moore. Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, 202:18–27, 2017.

Per Jonsson and Lars Eklundh. TIMESAT - a program for analyzing time-series of satellite sensor data. *Computers & Geosciences*, 30(8):833 – 845, 2004. ISSN 0098-3004.

Adam Lewis, Simon Oliver, Leo Lymburner, et al. The australian Geoscience Data Cube: Foundations and Lessons Learned. *Remote Sensing of Environment*, 202:276–292, 2017.

Victor Maus, Gilberto Camara, Ricardo Cartaxo, Alber Sanchez, Fernando M Ramos, and Gilberto R de Queiroz. A Time-Weighted Dynamic Time Warping method for Land-Use and Land-Cover mapping. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(8):3729 – 3739, 2016.

Victor Maus, Gilberto Câmara, Marius Appel, and Edzer Pebesma. dtwsat: Time-Weighted Dynamic Time Warping for Satellite Image Time Series analysis in R. *Journal of Statistical Software*, 88(5):1–31, 2019.

Valerie J. Pasquarella, Christopher E. Holden, Les Kaufman, and Curtis E. Woodcock. From imagery to ecology: leveraging time series of all available landsat observations to map and monitor ecosystem state and dynamics. *Remote Sensing in Ecology and Conservation*, 2(3):152–170, 2016. ISSN 2056-3485. doi: 10.1002/rse2.24.

George Percivall. Progress in ogc web services interoperability development. In *Standard-based data and information systems for earth observation*, pages 37–61. Springer, 2010.

Jan Verbesselt, Rob Hyndman, Glenn Newnham, and Darius Culvenor. Detecting trend and seasonal changes in satellite image time series. *Remote sensing of Environment*, 114(1):106–115, 2010.

Hadley Wickham and Garrett Grolemund. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, Inc., 2017.