

C>ONSTRUCTOR UNIVERSITY

Geospatial Big Data Processing with R

by Alina Amanbayeva

This notebook gives a short introduction to the Open Geospatial Consortium (OGC) standard WCS (Web Coverage Service) and WCPS (Web Coverage Processing Service). It provides small examples help you to get started retrieving data from

<https://inspire.rasdaman.org/rasdaman/ows> which hosts Rasdaman (Multidimensional Raster Database Manager) - <http://rasdaman.org> to process OGC standardized requests from data cubes.

Table of Contents

1. [OGC WCS Requests](#)
 - A. [Petascope endpoint](#)
 - B. [GetCapabilities](#)
 - C. [Describe Coverage](#)
 - D. [GetCoverage](#)
2. [integrating WCS Requests in R](#)
 - A. [Creating an interface in R](#)
 - B. [Checking the connection](#)
 - C. [Disclosing available coverages](#)
 - D. [Requesting a summary of a selected coverage ID](#)
 - E. [Coverage's axis label](#)
 - F. [Describing the coverage](#)
 - G. [List of coefficients](#)
 - H. [Time axis domain](#)
 - I. [Get a subset coverage](#)
3. [OGC Web Coverage Processing Services \(**WCPS**\) Requests](#)
4. [Integrating a WCPS query and displaying the results](#)
 - A. [Slicing, trimming, and downloading a TIFF file](#)
 - B. [Sending a query, to receive data encoded in CSV format](#)

5. Explanation of pixel computations in R

OGC WCS Requests

OGC Web Coverage Service (WCS, <http://www.opengeospatial.org/standards/wcs>) is a standardized data access protocol used for accessing data. It has three core requests: **GetCapabilities**, **DescribeCoverage**, and **GetCoverage**.

- **Petascop endpoint**

It should be used for any OGC requests and the URL for this endpoint is <https://inspire.rasdaman.org/rasdaman/ows>.

To access the Rasdaman demo server, you can use the following URL:

<https://inspire.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&request=...> This URL allows you to specify one of the core requests or the WCS extension.

- **GetCapabilities**

The GetCapabilities request returns an XML-encoded description of service properties and all data (coverages) available on the server. It returns the "Capabilities document," which contains information such as service provider, format encodings, supported interpolation methods, coverages on the server, and more

Example:

<https://inspire.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&request=GetCapabilities>

- **Describe Coverage**

The DescribeCoverage request returns an XML-encoded description of a specific coverage to check axes resolutions, dimensions, and other information. It returns the "Coverage description document" that contains information about spatial and temporal dimensions and general information about the type of coverage.

Example:

<https://inspire.rasdaman.org/rasdaman/ows?>

```
service=WCS&version=2.0.1&request=DescribeCoverage&coverageId=INSPIRE_EL
```

- **GetCoverage**

The **GetCoverage** request returns a full coverage or a subset by spatial/temporal dimensions of the coverage. It allows you to slice and trim the coverage's axes to get a subset coverage.

Example:

```
https://inspire.rasdaman.org/rasdaman/ows?
&SERVICE=WCS&VERSION=2.0.1&REQUEST=GetCoverage&COVERAGEID=INSPIRE_OI_
```



How to integrate WCS requests into an R terminal

You can install required R libraries using `install.packages("name_of_package_listed_in_CRAN")` function. List of all packages can be found here: https://cran.r-project.org/web/packages/available_packages_by_name.html

To save time I created a package already containing all the packages needed to download. In order to use it run:

```
In [3]: #Load pre-installed packages needed
install.packages("devtools")
install.packages("remotes")
library(devtools)
remotes::install_github("aamanbayev/CoverageProcessingWithR", dependencies = TRUE)
library(CoverageProcessingR)
```

Creation of an interface in R

In order to operate on a Web Coverage Service, you need to create an interface in R to this WCS. This is done with the class `WCSCClient`, as follows:

```
In [4]: wcs_client <- ows4R::WCSCClient$new("https://ows.rasdaman.org/rasdaman/ows", "2.1.0", 1
```

```
[ows4R][INFO] OWSGetCapabilities - Fetching https://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.1.0&request=GetCapabilities
```

```
|=====| 100%
```

Checking the connection

```
In [5]: caps <- tryCatch(
  wcs_client$getCapabilities(),
  error = function(e) {
    message("Error: Could not connect to WCS server.")
    return(NULL)
  }
) #Parse a message in case if it's successful
if (!is.null(caps)) {
  message("Connection to WCS server successful.")
  # Do something with the capabilities
} else {
  message("Connection to WCS server failed.")
} #Let the user know if connection failed
```

```
Connection to WCS server successful.
```

Disclosing all available coverages

Print available coverage IDs taken from the result of `GetCoverageSummaries()` function

```
In [7]: coverage_summaries <- caps$getCoverageSummaries()
coverage_ids <- sapply(coverage_summaries, function(summary) summary$CoverageId)
print(coverage_ids)
```

```
[1] "AverageChloroColor"
[2] "AverageChloroColorScaled"
[3] "AvgLandTemp"
[4] "AvgTemperatureColor"
[5] "AvgTemperatureColorScaled"
[6] "Bavaria_50_DSM"
[7] "BlueMarbleCov"
[8] "Germany_DTM"
[9] "NIR"
[10] "NN3_1"
[11] "NN3_2"
[12] "NN3_3"
[13] "NN3_4"
[14] "RadianceColorScaled"
[15] "S2_L2A_32631_B01_60m"
[16] "S2_L2A_32631_B03_10m"
[17] "S2_L2A_32631_B04_10m"
[18] "S2_L2A_32631_B08_10m"
[19] "S2_L2A_32631_B12_20m"
[20] "S2_L2A_32631_TCI_60m"
[21] "S2_federation_demo"
[22] "Temperature4D"
[23] "climate_cloud"
[24] "climate_earth"
[25] "land_cover_class_esa_test_6"
[26] "mean_summer_airtemp"
[27] "multiband"
[28] "nchc.earthserver.xyz:7000:AER_DLA_DEM_4326_2000_DMC_20010814_20061226"
[29] "nchc.earthserver.xyz:7000:AER_DLA_DEM_4326_2000_DMC_20010814_20061226_128"
[30] "nchc.earthserver.xyz:7000:AER_DLA_DEM_4326_2000_DMC_20010814_20061226_16"
[31] "nchc.earthserver.xyz:7000:AER_DLA_DEM_4326_2000_DMC_20010814_20061226_2"
[32] "nchc.earthserver.xyz:7000:AER_DLA_DEM_4326_2000_DMC_20010814_20061226_256"
[33] "nchc.earthserver.xyz:7000:AER_DLA_DEM_4326_2000_DMC_20010814_20061226_32"
[34] "nchc.earthserver.xyz:7000:AER_DLA_DEM_4326_2000_DMC_20010814_20061226_4"
[35] "nchc.earthserver.xyz:7000:AER_DLA_DEM_4326_2000_DMC_20010814_20061226_64"
[36] "nchc.earthserver.xyz:7000:AER_DLA_DEM_4326_2000_DMC_20010814_20061226_8"
[37] "nchc.earthserver.xyz:7000:AverageChlorophyll_new"
[38] "nchc.earthserver.xyz:7000:AverageTemperature"
[39] "nchc.earthserver.xyz:7000:LANDUSE_50Q_20200101_20210101"
[40] "nchc.earthserver.xyz:7000:LANDUSE_50Q_20200101_20210101_2"
[41] "nchc.earthserver.xyz:7000:LANDUSE_50Q_20200101_20210101_4"
[42] "nchc.earthserver.xyz:7000:LANDUSE_50Q_20200101_20210101_8"
[43] "nchc.earthserver.xyz:7000:LANDUSE_51Q_20200101_20210101"
[44] "nchc.earthserver.xyz:7000:LANDUSE_51Q_20200101_20210101_2"
[45] "nchc.earthserver.xyz:7000:LANDUSE_51Q_20200101_20210101_4"
[46] "nchc.earthserver.xyz:7000:LANDUSE_51Q_20200101_20210101_8"
[47] "nchc.earthserver.xyz:7000:LANDUSE_51R_20200101_20210101"
[48] "nchc.earthserver.xyz:7000:LANDUSE_51R_20200101_20210101_2"
[49] "nchc.earthserver.xyz:7000:LANDUSE_51R_20200101_20210101_4"
[50] "nchc.earthserver.xyz:7000:LANDUSE_51R_20200101_20210101_8"
[51] "nchc.earthserver.xyz:7000:LIDAR_DLA_DEM_4326_2000_Leica_20110101_20161101"
[52] "nchc.earthserver.xyz:7000:LIDAR_DLA_DEM_4326_2000_Leica_20110101_20161101_128"
[53] "nchc.earthserver.xyz:7000:LIDAR_DLA_DEM_4326_2000_Leica_20110101_20161101_16"
[54] "nchc.earthserver.xyz:7000:LIDAR_DLA_DEM_4326_2000_Leica_20110101_20161101_2"
[55] "nchc.earthserver.xyz:7000:LIDAR_DLA_DEM_4326_2000_Leica_20110101_20161101_256"
[56] "nchc.earthserver.xyz:7000:LIDAR_DLA_DEM_4326_2000_Leica_20110101_20161101_32"
[57] "nchc.earthserver.xyz:7000:LIDAR_DLA_DEM_4326_2000_Leica_20110101_20161101_4"
[58] "nchc.earthserver.xyz:7000:LIDAR_DLA_DEM_4326_2000_Leica_20110101_20161101_64"
[59] "nchc.earthserver.xyz:7000:LIDAR_DLA_DEM_4326_2000_Leica_20110101_20161101_8"
[60] "nchc.earthserver.xyz:7000:MSM1vltest"
```

```

[61] "nchc.earthserver.xyz:7000:MSMlvltest_2"
[62] "nchc.earthserver.xyz:7000:MSMlvltest_4"
[63] "nchc.earthserver.xyz:7000:TW_DLA_20010814_20061226_20M_3826_DEM"
[64] "nchc.earthserver.xyz:7000:TW_DLA_20100101_20191101_20M_3826_DEM"
[65] "nchc.earthserver.xyz:7000:TW_DLA_20110101_20161101_20M_3826_DEM"
[66] "nchc.earthserver.xyz:7000:m_msm_lvl_2021"
[67] "nchc.earthserver.xyz:7000:m_msm_lvl_2021_2"
[68] "nchc.earthserver.xyz:7000:m_msm_lvl_2021_4"
[69] "nchc.earthserver.xyz:7000:m_msm_sfc_2021"
[70] "nchc.earthserver.xyz:7000:m_msm_sfc_2021_2"
[71] "nchc.earthserver.xyz:7000:m_msm_sfc_2021_4"
[72] "nchc.earthserver.xyz:7000:mean_summer_airtemp"
[73] "nchc.earthserver.xyz:7000:sample"
[74] "nchc.earthserver.xyz:7000:sfc_allp"
[75] "nchc.earthserver.xyz:7000:sfc_allp_2"
[76] "nchc.earthserver.xyz:7000:sfc_allp_4"
[77] "nchc.earthserver.xyz:7000:sfc_test1"
[78] "nchc.earthserver.xyz:7000:sfc_test1_2"
[79] "nchc.earthserver.xyz:7000:sfc_test1_4"
[80] "nchc.earthserver.xyz:7000:sfc_test_4"
[81] "nchc.earthserver.xyz:7000:taiwan_air_quality_monitoring_data"
[82] "nchc.earthserver.xyz:7000:taiwan_air_quality_monitoring_data_benzene"
[83] "nchc.earthserver.xyz:7000:taiwan_air_quality_monitoring_data_co"
[84] "nchc.earthserver.xyz:7000:taiwan_air_quality_monitoring_data_no2"
[85] "nchc.earthserver.xyz:7000:taiwan_air_quality_monitoring_data_o3"
[86] "nchc.earthserver.xyz:7000:taiwan_air_quality_monitoring_data_pm2dot5"
[87] "nchc.earthserver.xyz:7000:taiwan_air_quality_monitoring_data_so2"

```

Requesting a summary of a chosen by ID coverage

the result is a printed summary

```

In [8]: chla <- caps$findCoverageSummaryById("AverageChloroColorScaled", exact = T)
        print(chla)

```

```

<WCSCoverageSummary>
  Inherits from: <OGCAbstractObject>
  Public:
    attrs: list
    BoundingBox: list
    clone: function (deep = FALSE)
    CoverageId: AverageChloroColorScaled
    CoverageSubtype: ReferenceableGridCoverage
    CoverageSubtypeParent: NULL
    defaults: list
    element: AbstractObject
    encode: function (addNS = TRUE, geometa_validate = TRUE, geometa_inspire = FALSE,
    ERROR: function (text)
    getBoundingBox: function ()
    getClass: function ()
    getClassName: function ()
    getCoverage: function (bbox = NULL, crs = NULL, time = NULL, elevation = NULL,
    getCoverageStack: function (time = NULL, elevation = NULL, bbox = NULL, filename_
handler = NULL,
    getDescription: function ()
    getDimensions: function ()
    getId: function ()
    getNamespaceDefinition: function (recursive = FALSE)
    getSubtype: function ()
    getSubtypeParent: function ()
    getWGS84BoundingBox: function ()
    INFO: function (text)
    initialize: function (xmlObj, capabilities, serviceVersion, owsVersion, logger =
NULL)
    isFieldInheritedFrom: function (field)
    logger: function (type, text)
    loggerType: INFO
    namespace: OWSNamespace, R6
    verbose.debug: FALSE
    verbose.info: TRUE
    WARN: function (text)
    WGS84BoundingBox: list
    wrap: FALSE
  Private:
    capabilities: WCSCapabilities, OWSCapabilities, OGCAbstractObject, R6
    description: NULL
    dimensions: NULL
    fetchCoverageSummary: function (xmlObj, serviceVersion, owsVersion)
    fromComplexTypes: function (value)
    owsVersion: 2.0
    system_fields: verbose.info verbose.debug loggerType wrap element names ...
    url: https://ows.rasdaman.org/rasdaman/ows
    version: 2.1.0
    xmlElement: AbstractObject
    xmlExtraNamespaces: NULL
    xmlNamespacePrefix: OWS
    xmlNodeToCharacter: function (x, ..., indent = "", tagSeparator = "\n")

```

Get coverage axis labels with order according to coverage's Coordinate Reference System (CRS)

```
In [10]: chla_dims <- chla$getDimensions()
for (dim in chla_dims) {
  print(dim$label)
```

```
[ows4R][INFO] WCSDescribeCoverage - Fetching https://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.1.0&coverageId=AverageChloroColorScaled&request=DescribeCoverage
|=====| 100%
[ows4R][INFO] WSCCoverageSummary - Fetching Coverage envelope dimensions by CRS interpretation
[ows4R][INFO] WSCCoverageSummary - Try to parse CRS from 'http://www.opengis.net/def/crs/OGC/0/AnsiDate'
[ows4R][INFO] WSCCoverageSummary - Try to parse CRS from 'http://www.opengis.net/def/crs/EPSSG/0/4326'
[1] "ansi"
[1] "Lat"
[1] "Lon"
```

Displaying the result of the describeCoverage function

which includes grid's upper and lower limits, and offset vectors in case you would like the result to be more user friendly `getDescription()` function can be used

```
In [14]: chla_des <- wcs_client$describeCoverage("AverageChloroColorScaled")
print(chla_des)
```


[ows4R][INFO] WCSClient - Fetching coverageSummary description for 'AverageChloroColorScaled' ...

<WCSCoverageDescription>

....|-- boundedBy [srsName=http://crs.rasdaman.com/def/crs-compound?1=http://crs.rasdaman.com/def/crs/OGC/0/AnsiDate&2=http://crs.rasdaman.com/def/crs/EPSSG/0/4326,axisLabels=ansi Lat Lon,uomLabels=d degree degree,srsDimension=3] <GMLEnvelope>

.....|-- lowerCorner: "2002-07-01T00:00:00.000Z" -90 -180

.....|-- upperCorner: "2015-05-01T00:00:00.000Z" 90 180

....|-- domainSet [dimension=3,gmlrgrid:id=AverageChloroColorScaled-grid] <GMLReferenceableGridByVectors>

.....|-- limits <GMLGridEnvelope>

.....|-- low

.....|-- value: 0 0 0

.....|-- high

.....|-- value: 154 449 899

.....|-- axisLabels

.....|-- value: ansi Lat Lon

.....|-- origin [gml:id=AverageChloroColorScaled-point,srsName=http://crs.rasdaman.com/def/crs-compound?1=http://crs.rasdaman.com/def/crs/OGC/0/AnsiDate&2=http://crs.rasdaman.com/def/crs/EPSSG/0/4326] <GMLPoint>

.....|-- pos: "2002-07-01T00:00:00.000Z" 89.8 -179.8

.....|-- generalGridAxis <GMLGeneralGridAxis>

.....|-- offsetVector: 1 0 0

.....|-- coefficients: 2002-07-01T00:00:00.000Z 2002-08-01T00:00:00.000Z 2002-09-01T00:00:00.000Z 2002-10-01T00:00:00.000Z 2002-11-01T00:00:00.000Z 2002-12-01T00:00:00.000Z 2003-01-01T00:00:00.000Z 2003-02-01T00:00:00.000Z 2003-03-01T00:00:00.000Z 2003-04-01T00:00:00.000Z 2003-05-01T00:00:00.000Z 2003-06-01T00:00:00.000Z 2003-07-01T00:00:00.000Z 2003-08-01T00:00:00.000Z 2003-09-01T00:00:00.000Z 2003-10-01T00:00:00.000Z 2003-11-01T00:00:00.000Z 2003-12-01T00:00:00.000Z 2004-01-01T00:00:00.000Z 2004-02-01T00:00:00.000Z 2004-03-01T00:00:00.000Z 2004-04-01T00:00:00.000Z 2004-05-01T00:00:00.000Z 2004-06-01T00:00:00.000Z 2004-07-01T00:00:00.000Z 2004-08-01T00:00:00.000Z 2004-09-01T00:00:00.000Z 2004-10-01T00:00:00.000Z 2004-11-01T00:00:00.000Z 2004-12-01T00:00:00.000Z 2005-01-01T00:00:00.000Z 2005-02-01T00:00:00.000Z 2005-03-01T00:00:00.000Z 2005-04-01T00:00:00.000Z 2005-05-01T00:00:00.000Z 2005-06-01T00:00:00.000Z 2005-07-01T00:00:00.000Z 2005-08-01T00:00:00.000Z 2005-09-01T00:00:00.000Z 2005-10-01T00:00:00.000Z 2005-11-01T00:00:00.000Z 2005-12-01T00:00:00.000Z 2006-01-01T00:00:00.000Z 2006-02-01T00:00:00.000Z 2006-03-01T00:00:00.000Z 2006-04-01T00:00:00.000Z 2006-05-01T00:00:00.000Z 2006-06-01T00:00:00.000Z 2006-07-01T00:00:00.000Z 2006-08-01T00:00:00.000Z 2006-09-01T00:00:00.000Z 2006-10-01T00:00:00.000Z 2006-11-01T00:00:00.000Z 2006-12-01T00:00:00.000Z 2007-01-01T00:00:00.000Z 2007-02-01T00:00:00.000Z 2007-03-01T00:00:00.000Z 2007-04-01T00:00:00.000Z 2007-05-01T00:00:00.000Z 2007-06-01T00:00:00.000Z 2007-07-01T00:00:00.000Z 2007-08-01T00:00:00.000Z 2007-09-01T00:00:00.000Z 2007-10-01T00:00:00.000Z 2007-11-01T00:00:00.000Z 2007-12-01T00:00:00.000Z 2008-01-01T00:00:00.000Z 2008-02-01T00:00:00.000Z 2008-03-01T00:00:00.000Z 2008-04-01T00:00:00.000Z 2008-05-01T00:00:00.000Z 2008-06-01T00:00:00.000Z 2008-07-01T00:00:00.000Z 2008-08-01T00:00:00.000Z 2008-09-01T00:00:00.000Z 2008-10-01T00:00:00.000Z 2008-11-01T00:00:00.000Z 2008-12-01T00:00:00.000Z 2009-01-01T00:00:00.000Z 2009-02-01T00:00:00.000Z 2009-03-01T00:00:00.000Z 2009-04-01T00:00:00.000Z 2009-05-01T00:00:00.000Z 2009-06-01T00:00:00.000Z 2009-07-01T00:00:00.000Z 2009-08-01T00:00:00.000Z 2009-09-01T00:00:00.000Z 2009-10-01T00:00:00.000Z 2009-11-01T00:00:00.000Z 2009-12-01T00:00:00.000Z 2010-01-01T00:00:00.000Z 2010-02-01T00:00:00.000Z 2010-03-01T00:00:00.000Z 2010-04-01T00:00:00.000Z 2010-05-01T00:00:00.000Z 2010-06-01T00:00:00.000Z 2010-07-01T00:00:00.000Z 2010-08-01T00:00:00.000Z 2010-09-01T00:00:00.000Z 2010-10-01T00:00:00.000Z 2010-11-01T00:00:00.000Z 2010-12-01T00:00:00.000Z 2011-01-01T00:00:00.000Z 2011-02-01T00:00:00.000Z 2011-03-01T00:00:00.000Z 2011-04-01T00:00:00.000Z 2011-05-01T00:00:00.000Z 2011-06-01T00:00:00.000Z 2011-07-01T00:00:00.000Z 2011-08-01T00:00:00.000Z 2011-09-01T00:00:00.000Z 2011-10-01T00:00:00.000Z 2011-11-01T00:00:00.000Z 2011-12-01T00:00:00.000Z 2012-01-01T00:00:00.000Z 2012-02-01T00:00:00.000Z 2012-03-01T00:00:00.000Z 2012-04-01T00:00:00.000Z 2012-05-01T00:00:00.000Z 2012-06-01T00:00:00.000Z 2012-07

```

-01T00:00:00.000Z 2012-08-01T00:00:00.000Z 2012-09-01T00:00:00.000Z 2012-10-01T00:
00:00.000Z 2012-11-01T00:00:00.000Z 2012-12-01T00:00:00.000Z 2013-01-01T00:00:00.0
00Z 2013-02-01T00:00:00.000Z 2013-03-01T00:00:00.000Z 2013-04-01T00:00:00.000Z 201
3-05-01T00:00:00.000Z 2013-06-01T00:00:00.000Z 2013-07-01T00:00:00.000Z 2013-08-01
T00:00:00.000Z 2013-09-01T00:00:00.000Z 2013-10-01T00:00:00.000Z 2013-11-01T00:00:
00.000Z 2013-12-01T00:00:00.000Z 2014-01-01T00:00:00.000Z 2014-02-01T00:00:00.000Z
2014-03-01T00:00:00.000Z 2014-04-01T00:00:00.000Z 2014-05-01T00:00:00.000Z 2014-06
-01T00:00:00.000Z 2014-07-01T00:00:00.000Z 2014-08-01T00:00:00.000Z 2014-09-01T00:
00:00.000Z 2014-10-01T00:00:00.000Z 2014-11-01T00:00:00.000Z 2014-12-01T00:00:00.0
00Z 2015-01-01T00:00:00.000Z 2015-02-01T00:00:00.000Z 2015-03-01T00:00:00.000Z 201
5-04-01T00:00:00.000Z 2015-05-01T00:00:00.000Z
.....|-- gridAxesSpanned
.....|-- value: ansi
.....|-- sequenceRule [axisOrder=+1]
.....|-- value: Linear
.....|-- generalGridAxis <GMLGeneralGridAxis>
.....|-- offsetVector: 0 -0.4 0
.....|-- coefficients
.....|-- gridAxesSpanned
.....|-- value: Lat
.....|-- sequenceRule [axisOrder=+1]
.....|-- value: Linear
.....|-- generalGridAxis <GMLGeneralGridAxis>
.....|-- offsetVector: 0 0 0.4
.....|-- coefficients
.....|-- gridAxesSpanned
.....|-- value: Lon
.....|-- sequenceRule [axisOrder=+1]
.....|-- value: Linear
....|-- coverageFunction <GMLGridFunction>
.....|-- sequenceRule [axisOrder=+2 +3 +1]
.....|-- value: Linear
.....|-- startPoint
.....|-- value: 0 0 0
....|-- rangeType <SWEDataRecord>
.....|-- field [definition=http://www.opengis.net/def/dataType/OGC/0/unsignedBy
te]<SWEQuantity>
.....|-- label
.....|-- value: Red
.....|-- description
.....|-- uom [code=10^0]
.....|-- constraint
.....|-- field [definition=http://www.opengis.net/def/dataType/OGC/0/unsignedBy
te]<SWEQuantity>
.....|-- label
.....|-- value: Green
.....|-- description
.....|-- uom [code=10^0]
.....|-- constraint
.....|-- field [definition=http://www.opengis.net/def/dataType/OGC/0/unsignedBy
te]<SWEQuantity>
.....|-- label
.....|-- value: Blue
.....|-- description
.....|-- uom [code=10^0]
.....|-- constraint
....|-- metadata
....|-- CoverageId
.....|-- value: AverageChloroColorScaled
....|-- ServiceParameters <ISOElementSequence>

```

```
.....|-- CoverageSubtype: ReferenceableGridCoverage  
.....|-- nativeFormat: application/octet-stream
```

Get the list of coefficients for irregular axis (time is an irregular axis of this coverage)

```
In [8]: for (dim in chla_dims) {  
        if (dim$type == "temporal") {  
          print(dim$coefficients)  
        }  
      }
```

```
[,1]
[1,] "2002-07-01T00:00:00.000Z"
[2,] "2002-08-01T00:00:00.000Z"
[3,] "2002-09-01T00:00:00.000Z"
[4,] "2002-10-01T00:00:00.000Z"
[5,] "2002-11-01T00:00:00.000Z"
[6,] "2002-12-01T00:00:00.000Z"
[7,] "2003-01-01T00:00:00.000Z"
[8,] "2003-02-01T00:00:00.000Z"
[9,] "2003-03-01T00:00:00.000Z"
[10,] "2003-04-01T00:00:00.000Z"
[11,] "2003-05-01T00:00:00.000Z"
[12,] "2003-06-01T00:00:00.000Z"
[13,] "2003-07-01T00:00:00.000Z"
[14,] "2003-08-01T00:00:00.000Z"
[15,] "2003-09-01T00:00:00.000Z"
[16,] "2003-10-01T00:00:00.000Z"
[17,] "2003-11-01T00:00:00.000Z"
[18,] "2003-12-01T00:00:00.000Z"
[19,] "2004-01-01T00:00:00.000Z"
[20,] "2004-02-01T00:00:00.000Z"
[21,] "2004-03-01T00:00:00.000Z"
[22,] "2004-04-01T00:00:00.000Z"
[23,] "2004-05-01T00:00:00.000Z"
[24,] "2004-06-01T00:00:00.000Z"
[25,] "2004-07-01T00:00:00.000Z"
[26,] "2004-08-01T00:00:00.000Z"
[27,] "2004-09-01T00:00:00.000Z"
[28,] "2004-10-01T00:00:00.000Z"
[29,] "2004-11-01T00:00:00.000Z"
[30,] "2004-12-01T00:00:00.000Z"
[31,] "2005-01-01T00:00:00.000Z"
[32,] "2005-02-01T00:00:00.000Z"
[33,] "2005-03-01T00:00:00.000Z"
[34,] "2005-04-01T00:00:00.000Z"
[35,] "2005-05-01T00:00:00.000Z"
[36,] "2005-06-01T00:00:00.000Z"
[37,] "2005-07-01T00:00:00.000Z"
[38,] "2005-08-01T00:00:00.000Z"
[39,] "2005-09-01T00:00:00.000Z"
[40,] "2005-10-01T00:00:00.000Z"
[41,] "2005-11-01T00:00:00.000Z"
[42,] "2005-12-01T00:00:00.000Z"
[43,] "2006-01-01T00:00:00.000Z"
[44,] "2006-02-01T00:00:00.000Z"
[45,] "2006-03-01T00:00:00.000Z"
[46,] "2006-04-01T00:00:00.000Z"
[47,] "2006-05-01T00:00:00.000Z"
[48,] "2006-06-01T00:00:00.000Z"
[49,] "2006-07-01T00:00:00.000Z"
[50,] "2006-08-01T00:00:00.000Z"
[51,] "2006-09-01T00:00:00.000Z"
[52,] "2006-10-01T00:00:00.000Z"
[53,] "2006-11-01T00:00:00.000Z"
[54,] "2006-12-01T00:00:00.000Z"
[55,] "2007-01-01T00:00:00.000Z"
[56,] "2007-02-01T00:00:00.000Z"
[57,] "2007-03-01T00:00:00.000Z"
[58,] "2007-04-01T00:00:00.000Z"
[59,] "2007-05-01T00:00:00.000Z"
```

```
[60,] "2007-06-01T00:00:00.000Z"  
[61,] "2007-07-01T00:00:00.000Z"  
[62,] "2007-08-01T00:00:00.000Z"  
[63,] "2007-09-01T00:00:00.000Z"  
[64,] "2007-10-01T00:00:00.000Z"  
[65,] "2007-11-01T00:00:00.000Z"  
[66,] "2007-12-01T00:00:00.000Z"  
[67,] "2008-01-01T00:00:00.000Z"  
[68,] "2008-02-01T00:00:00.000Z"  
[69,] "2008-03-01T00:00:00.000Z"  
[70,] "2008-04-01T00:00:00.000Z"  
[71,] "2008-05-01T00:00:00.000Z"  
[72,] "2008-06-01T00:00:00.000Z"  
[73,] "2008-07-01T00:00:00.000Z"  
[74,] "2008-08-01T00:00:00.000Z"  
[75,] "2008-09-01T00:00:00.000Z"  
[76,] "2008-10-01T00:00:00.000Z"  
[77,] "2008-11-01T00:00:00.000Z"  
[78,] "2008-12-01T00:00:00.000Z"  
[79,] "2009-01-01T00:00:00.000Z"  
[80,] "2009-02-01T00:00:00.000Z"  
[81,] "2009-03-01T00:00:00.000Z"  
[82,] "2009-04-01T00:00:00.000Z"  
[83,] "2009-05-01T00:00:00.000Z"  
[84,] "2009-06-01T00:00:00.000Z"  
[85,] "2009-07-01T00:00:00.000Z"  
[86,] "2009-08-01T00:00:00.000Z"  
[87,] "2009-09-01T00:00:00.000Z"  
[88,] "2009-10-01T00:00:00.000Z"  
[89,] "2009-11-01T00:00:00.000Z"  
[90,] "2009-12-01T00:00:00.000Z"  
[91,] "2010-01-01T00:00:00.000Z"  
[92,] "2010-02-01T00:00:00.000Z"  
[93,] "2010-03-01T00:00:00.000Z"  
[94,] "2010-04-01T00:00:00.000Z"  
[95,] "2010-05-01T00:00:00.000Z"  
[96,] "2010-06-01T00:00:00.000Z"  
[97,] "2010-07-01T00:00:00.000Z"  
[98,] "2010-08-01T00:00:00.000Z"  
[99,] "2010-09-01T00:00:00.000Z"  
[100,] "2010-10-01T00:00:00.000Z"  
[101,] "2010-11-01T00:00:00.000Z"  
[102,] "2010-12-01T00:00:00.000Z"  
[103,] "2011-01-01T00:00:00.000Z"  
[104,] "2011-02-01T00:00:00.000Z"  
[105,] "2011-03-01T00:00:00.000Z"  
[106,] "2011-04-01T00:00:00.000Z"  
[107,] "2011-05-01T00:00:00.000Z"  
[108,] "2011-06-01T00:00:00.000Z"  
[109,] "2011-07-01T00:00:00.000Z"  
[110,] "2011-08-01T00:00:00.000Z"  
[111,] "2011-09-01T00:00:00.000Z"  
[112,] "2011-10-01T00:00:00.000Z"  
[113,] "2011-11-01T00:00:00.000Z"  
[114,] "2011-12-01T00:00:00.000Z"  
[115,] "2012-01-01T00:00:00.000Z"  
[116,] "2012-02-01T00:00:00.000Z"  
[117,] "2012-03-01T00:00:00.000Z"  
[118,] "2012-04-01T00:00:00.000Z"  
[119,] "2012-05-01T00:00:00.000Z"
```

```
[120,] "2012-06-01T00:00:00.000Z"
[121,] "2012-07-01T00:00:00.000Z"
[122,] "2012-08-01T00:00:00.000Z"
[123,] "2012-09-01T00:00:00.000Z"
[124,] "2012-10-01T00:00:00.000Z"
[125,] "2012-11-01T00:00:00.000Z"
[126,] "2012-12-01T00:00:00.000Z"
[127,] "2013-01-01T00:00:00.000Z"
[128,] "2013-02-01T00:00:00.000Z"
[129,] "2013-03-01T00:00:00.000Z"
[130,] "2013-04-01T00:00:00.000Z"
[131,] "2013-05-01T00:00:00.000Z"
[132,] "2013-06-01T00:00:00.000Z"
[133,] "2013-07-01T00:00:00.000Z"
[134,] "2013-08-01T00:00:00.000Z"
[135,] "2013-09-01T00:00:00.000Z"
[136,] "2013-10-01T00:00:00.000Z"
[137,] "2013-11-01T00:00:00.000Z"
[138,] "2013-12-01T00:00:00.000Z"
[139,] "2014-01-01T00:00:00.000Z"
[140,] "2014-02-01T00:00:00.000Z"
[141,] "2014-03-01T00:00:00.000Z"
[142,] "2014-04-01T00:00:00.000Z"
[143,] "2014-05-01T00:00:00.000Z"
[144,] "2014-06-01T00:00:00.000Z"
[145,] "2014-07-01T00:00:00.000Z"
[146,] "2014-08-01T00:00:00.000Z"
[147,] "2014-09-01T00:00:00.000Z"
[148,] "2014-10-01T00:00:00.000Z"
[149,] "2014-11-01T00:00:00.000Z"
[150,] "2014-12-01T00:00:00.000Z"
[151,] "2015-01-01T00:00:00.000Z"
[152,] "2015-02-01T00:00:00.000Z"
[153,] "2015-03-01T00:00:00.000Z"
[154,] "2015-04-01T00:00:00.000Z"
[155,] "2015-05-01T00:00:00.000Z"
```

Get coverage's time axis's domain (as this coverage is 3D with time axis)

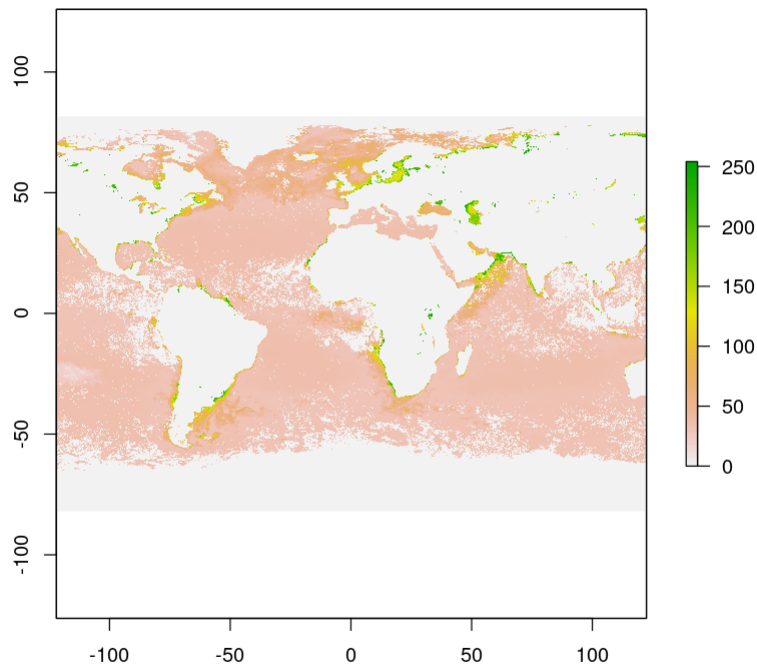
```
In [9]: temporal_dim <- NULL ## Get coverage's time axis's domain (as this coverage is 3D with
      for (dim in chla_dims) {
        if (dim$type == "temporal") {
          temporal_dim <- dim
        }
      }
      if (!is.null(temporal_dim)) {
        # Get the first and last time coefficients
        time_coefficients <- temporal_dim$coefficients
        first_time <- time_coefficients[1]
        last_time <- time_coefficients[length(time_coefficients)]
        print(paste("First time:", first_time))
        print(paste("Last time:", last_time))
      } else {
        print("No temporal dimension found.")
      }
```

```
[1] "First time: 2002-07-01T00:00:00.000Z"
[1] "Last time: 2015-05-01T00:00:00.000Z"
```

Get a subset coverage

By slicing on time axis, trimming on Lat and Long axes, downloading as TIFF file, and displaying the result directly

```
In [28]: data <- chla$getCoverage(
  bbox = ows4R::OWSUtils$toBBOX(-122.1420, 122.2185 , -81.7242, 81.7825),
  time = "2002-09-01T00:00:00.000Z",
  filename = "myfile.tif"
) # Get a subset coverage by slicing on time axis, trimming on Lat and Long axes, and
r <- raster::raster(file.path("~", "Downloads", "myfile.tif"))
#Note: It was assumed that the file gets downloaded in the Downloads directory
raster::plot(r) # Display result directly
```



Tip: There is also an option to download multiple coverages in a stack, by slightly changing a previous function, the way as in the cell below:

```
In [24]: cov_stack <- chla$getCoverageStack(
  bbox = ows4R::OWSUtils$toBBOX(-122.1420,122.2185 , -81.7242, 81.7825),
  time = tail(chla_dims[[1]]$coefficients, 5),
  filename_handler = WSCoverageFilenameHandler
) #Load stack of subset coverages
```

```

[ows4R][INFO] WCSCoverageSummary - Fetching coverage stack with 'temporal' dimension
[ows4R][INFO] WCSCoverageSummary - 2015-01-01T00:00:00.000Z
[ows4R][WARN] WCSCoverageSummary - Coverage without vertical dimension: 'elevation' a
rgument is ignored
<GMLEnvelope>
....|-- lowerCorner: "2002-07-01T00:00:00.000Z" -81.7242 -122.142
....|-- upperCorner: "2015-05-01T00:00:00.000Z" 81.7825 122.2185[ows4R][INFO] WCSGetC
overage - Fetching https://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.1.0&co
verageId=AverageChloroColorScaled&subset=ansi(%222015-01-01T00:00:00.000Z%22)&subset=
Lat(-81.7242,81.7825)&subset=Lon(-122.142,122.2185)&format=image/tiff&request=GetCove
rage
Downloading: 760 kB [ows4R][INFO] WCSCoverageSummary - 2015-02-01T00:00:00.000Z
[ows4R][WARN] WCSCoverageSummary - Coverage without vertical dimension: 'elevation' a
rgument is ignored
<GMLEnvelope>
....|-- lowerCorner: "2002-07-01T00:00:00.000Z" -81.7242 -122.142
....|-- upperCorner: "2015-05-01T00:00:00.000Z" 81.7825 122.2185[ows4R][INFO] WCSGetC
overage - Fetching https://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.1.0&co
verageId=AverageChloroColorScaled&subset=ansi(%222015-02-01T00:00:00.000Z%22)&subset=
Lat(-81.7242,81.7825)&subset=Lon(-122.142,122.2185)&format=image/tiff&request=GetCove
rage
Downloading: 760 kB [ows4R][INFO] WCSCoverageSummary - 2015-03-01T00:00:00.000Z
[ows4R][WARN] WCSCoverageSummary - Coverage without vertical dimension: 'elevation' a
rgument is ignored
<GMLEnvelope>
....|-- lowerCorner: "2002-07-01T00:00:00.000Z" -81.7242 -122.142
....|-- upperCorner: "2015-05-01T00:00:00.000Z" 81.7825 122.2185[ows4R][INFO] WCSGetC
overage - Fetching https://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.1.0&co
verageId=AverageChloroColorScaled&subset=ansi(%222015-03-01T00:00:00.000Z%22)&subset=
Lat(-81.7242,81.7825)&subset=Lon(-122.142,122.2185)&format=image/tiff&request=GetCove
rage
Downloading: 760 kB [ows4R][INFO] WCSCoverageSummary - 2015-04-01T00:00:00.000Z
[ows4R][WARN] WCSCoverageSummary - Coverage without vertical dimension: 'elevation' a
rgument is ignored
<GMLEnvelope>
....|-- lowerCorner: "2002-07-01T00:00:00.000Z" -81.7242 -122.142
....|-- upperCorner: "2015-05-01T00:00:00.000Z" 81.7825 122.2185[ows4R][INFO] WCSGetC
overage - Fetching https://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.1.0&co
verageId=AverageChloroColorScaled&subset=ansi(%222015-04-01T00:00:00.000Z%22)&subset=
Lat(-81.7242,81.7825)&subset=Lon(-122.142,122.2185)&format=image/tiff&request=GetCove
rage
Downloading: 760 kB [ows4R][INFO] WCSCoverageSummary - 2015-05-01T00:00:00.000Z
[ows4R][WARN] WCSCoverageSummary - Coverage without vertical dimension: 'elevation' a
rgument is ignored
<GMLEnvelope>
....|-- lowerCorner: "2002-07-01T00:00:00.000Z" -81.7242 -122.142
....|-- upperCorner: "2015-05-01T00:00:00.000Z" 81.7825 122.2185[ows4R][INFO] WCSGetC
overage - Fetching https://ows.rasdaman.org/rasdaman/ows?service=WCS&version=2.1.0&co
verageId=AverageChloroColorScaled&subset=ansi(%222015-05-01T00:00:00.000Z%22)&subset=
Lat(-81.7242,81.7825)&subset=Lon(-122.142,122.2185)&format=image/tiff&request=GetCove
rage
Downloading: 760 kB

```

OGC Web Coverage Processing Services (WCPS) Requests

The OGC Web Coverage Processing Service (WCPS) is an extension of the OGC WCS standard that allows for the extraction, analysis, and processing of multi-dimensional coverages. WCPS is designed with syntax similar to the XQuery language, and it establishes a protocol for sending a query string to a server and receiving a set of coverages as the result of the server's processing.

The processing expression is applied to each coverage specified in the given list (coverageList) that satisfies the optional boolean expression. The query references each coverage using the corresponding identifier variableName in the processing expression. The processing expression can include sub-expressions that return scalars or encoded marrays, which operate on both the data and metadata of a coverage. More information can be found at <http://tutorial.rasdaman.org/rasdaman-and-ogc-ws-tutorial/#ogc-web-services-web-coverage-processing-service>.

To execute a query, use the following request (ProcessCoverages is an extension of the WCS request):

[https://inspire.rasdaman.org/rasdaman/ows?
service=WCS&version=2.0.1&request=ProcessCoverages&query=](https://inspire.rasdaman.org/rasdaman/ows?service=WCS&version=2.0.1&request=ProcessCoverages&query=)

The query parameter should contain a WCPS query with valid syntax for the server to process. An example of subsetting a 3D coverage to a 2D coverage and encoding the result in image/png is provided in the previous demo for the WCS GetCoverage request.

```
In [ ]: # NOTE: for GET requests, WCPS query cannot contain new lines character (below query i
for $c in (AvgTemperatureColorScaled) return encode(
    $c[Lat(-81.7242:81.7825),
        Lon(-122.1420:122.2185),
        ansi("2002-09-01T00:00:00.000Z")],
    "png")
```

WCPS provides the capability for significantly more intricate processing requests than what can be achieved with WCS. For instance, it allows for time-series processing to determine the difference between two specific date-time slices ("2002-09-01T00:00:00.000Z" (reference) and "2009-05-01T00:00:00.000Z") of the coverage AvgTemperatureColorScaled.

****NOTE:**** It should be noted that when using WCPS queries that contain special characters such as '[', ']', '{', '}', it is necessary to use POST requests.

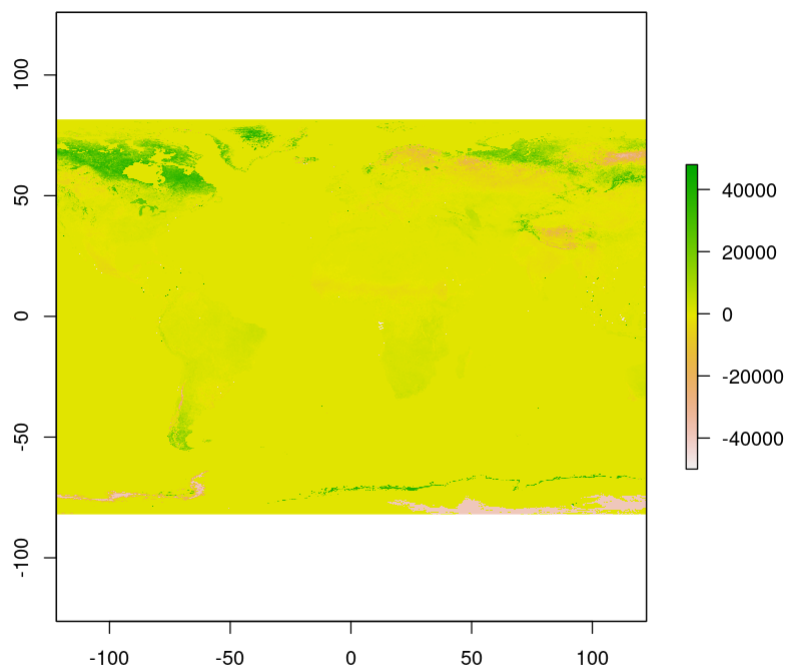
Some of the libraries used are not available in CRAN R packages, have been found on GitHub, such as myCubeR (<https://github.com/ARPASMR/myCubeR>), and have been installed using devtools package's "install_github()" function.

Integrating a WPCS query and displaying the results

Slicing, trimming, and downloading a TIFF file

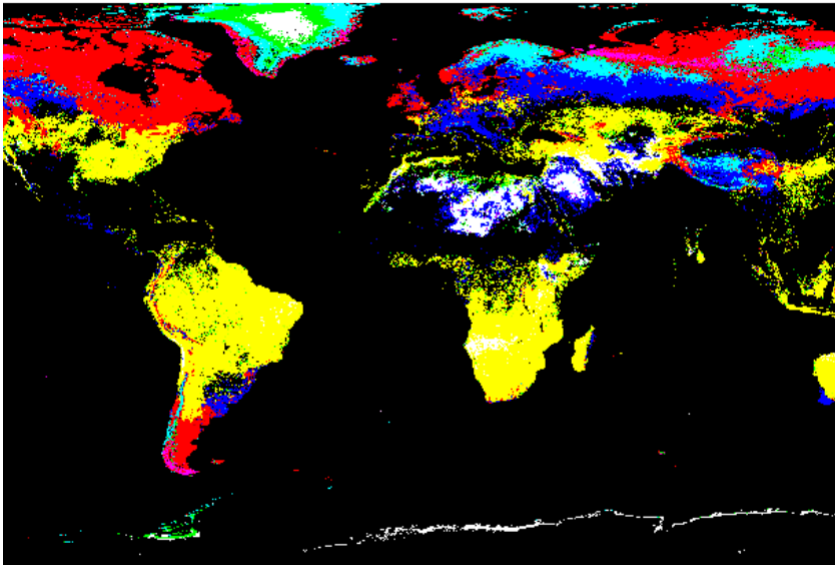
`myCubeR::WPCS_query()` function send a query to a server, and receives the results. The variables needed are the query itself, format, name of the file where we put the retrieved data, and URL to which the query is sent. In fact you can send any type of an executable query.

```
In [22]: query='for $c in (AvgTemperatureColorScaled) return encode(
  ($c[Lat(-81.7242:81.7825), Lon(-122.1420:122.2185), ansi("2002-09-01T00:00:00.000Z")]
  - $c[Lat(-81.7242:81.7825), Lon(-122.1420:122.2185), ansi("2009-05-01T00:00:00.000Z")
  * 200, "tiff")' # Query for a TIFF format
raster_tot=myCubeR::WPCS_query(proper_query=query, FORMAT="image/tiff", filename="AvgT
# Send a WPCS query with special characters to server and download the result
file_path <- file.path("~", "Downloads", "AvgTemperatureColorScaled.tiff")
ra <- raster::raster(file_path)
#Note: It was assumed that the file gets downloaded in the Downloads directory
raster::plot(ra)
# Display a TIFF file directly
```



Tip: The data can also be downloaded and displayed in a PNG or JPEG formats. Here we display the same query with a PNG encoding

```
In [18]: query='for $c in (AvgTemperatureColorScaled) return encode(
  ($c[Lat(-81.7242:81.7825), Lon(-122.1420:122.2185), ansi("2002-09-01T00:00:00.000Z")])
  - $c[Lat(-81.7242:81.7825), Lon(-122.1420:122.2185), ansi("2009-05-01T00:00:00.000Z")])
  * 200, "png")'
raster_top=myCubeR::WPCS_query(proper_query=query, FORMAT="image/png", filename="AvgTe
# Send a WPCS query with special characters to server and download the result
file_path = file.path("~", "Downloads", "AvgTemperatureColorScaled.png")
img <- png::readPNG(file_path)
#Note: It was assumed that the file gets downloaded in the Downloads directory
grid::grid.raster(img)
```



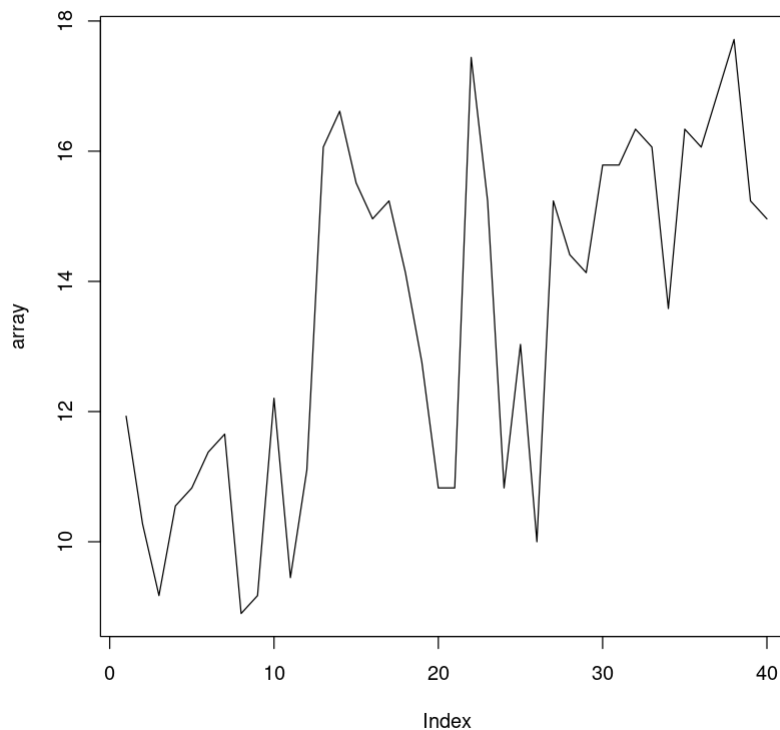
Sending a query, to receive data encoded in CSV format

The last query sent by this vignette to show, that the data can be further displayed after converting it to a JSON format. Any one in interest may change the queries and its parameters to receive different results.

```
In [27]: queryplot='for $c in (AvgLandTemp) return encode(
  $c[Lat(41.7242:45.7352),
  Long(12.1420),
  ansi("2000-02-01T00:00:00.000Z")])
  , "text/csv")'
# Select a 1D subset coverage result encoded in JSON
raster_toj=myCubeR::WPCS_query(proper_query=queryplot, FORMAT="text/csv", filename="Av
df <- read.csv(file.path("~", "Downloads", "AvgLandTemp.csv"))
json_string <- jsonlite::toJSON(df)
write(json_string, file.path("~", "Downloads", "AvgLandTemp.json"))
```

```
# Send the query and download the result
file_path = file.path("~", "Downloads", "AvgLandTemp.json")
json_data <- jsonlite::read_json(file_path)
#Note: It was assumed that the file gets downloaded in the Downloads directory
array <- as.numeric(json_data)

# Display the raw values of 1D array in a plot
raster::plot(array, type = "l")
```



Explanation of pixel computations in R

The query language enables us to process data inside it, as we can trim and slice subsets, by choosing the timestamps, coordinates, and bands. Although the functionality of the R language also gives us an opportunity for further processing, as for example displayed in a previous cell. Now we will explain the processes taking part in these computations.

Lets start from basics, e.g., how we send the queries to a server:

```
query_encode <- urltools::url_encode(proper_query)
request<- paste(query_url, query_encode, collapse = NULL, sep="")
res <- GET(request)
```

url_encode() function works by replacing each character in the input string with its corresponding hexadecimal representation, preceded by a percent sign (%). The hexadecimal representation of a character is a two-digit code that represents the ASCII code for that character.

paste(query_url, query_encode, collapse = NULL, sep="") takes one or more vectors of strings and combines them into a single string, optionally separated by a specified separator.

GET(request) function sends an HTTP GET request to the specified URL, and the server responds with the requested data. The function waits for the response and returns the result as an R object. The response body is the data that was requested from the server, and it can be in various formats such as JSON, XML, or text.

Now let's observe how we save the response:

```
out<- content(res, response_format)
if (is.null(filename)) {
  return(out)
} else {
  # Save to local disk <br>
  savefile(response=res, filename=filename)<br>
} </div>
```

The **content()** function extracts the content of the HTTP response object **res** and returns it in the specified format. If **filename** is not specified, the function simply returns the content in the desired format.

If **filename** is specified, the content is saved to a local file with the specified name using the **savefile()** function. The **savefile()** function takes two arguments: **response**, which is the content to be saved, and **filename**, which is the name of the file to which the content should be saved.

Now let's dive in to details of the processing of the retrieved data.

```
df <- read.csv(file.path("~", "Downloads", "AvgLandTemp.csv"))
json_string <- jsonlite::toJSON(df)
write(json_string, file.path("~", "Downloads", "AvgLandTemp.json"))
json_data <- jsonlite::read_json(file.path("~", "Downloads", "AvgLandTemp.json"))
```

```
array <- as.numeric(json_data)
raster::plot(array, type = "l")
```

The `read.csv()` function is called with a file path or connection argument. The function then creates a connection to the file. The first line of the file is read to determine the column names and data types. The function then reads the remaining lines of the file, storing them in memory as a list. The list is then converted into a data frame. If the `header` argument is set to `TRUE`, the first row of the file is treated as the column names for the data frame. If the `row.names` argument is set to `TRUE`, the first column of the file is treated as the row names for the data frame. If the file contains missing values, the `na.strings` argument can be used to specify how missing values are represented in the file. Finally, the data frame is returned.

The computational steps that `jsonlite::toJSON()` performs: The function first checks the class of the input object to determine how to serialize it. If the object is a data frame or a list, it is serialized as a JSON object. If the object is a vector or an array, it is serialized as a JSON array. The function then iterates over the input object's elements, converting each element to its JSON representation. For example, a numeric value is converted to a JSON number, a character string is converted to a JSON string, and so on. If the input object contains nested elements (such as a list of lists), the function recursively calls itself on each nested element to serialize it. The function then assembles the JSON string by concatenating the JSON representations of the input object's elements together with appropriate separators (e.g., commas between elements).

Finally, the function returns the JSON string.

`write()` is a function that writes data to a file. It takes two arguments: the first argument is the data to be written, and the second argument is the file name or connection where the data will be written. If the file already exists, it will be overwritten. For example, `write("hello", "myfile.txt")` will create a new file called "myfile.txt" in the current working directory, and write the string "hello" to that file.

Here is how `read_json()` function works computationally: The input to `read_json()` can either be a file path or a JSON string. If a file path is provided, the function reads the JSON data from the file using the `readLines()` function and stores it as a character vector. If a JSON string is provided, it is directly stored as a character vector. The character vector obtained from step 1 is parsed by the `jsonlite` package using the `fromJSON()` function. This function converts the JSON string to an R object. The R object is returned as the output of `read_json()`. The type of the R object depends on the structure of the JSON data. For example, if the JSON data represents an array, the R object will be a list. If the JSON data represents a JSON object, the R object will be a

named list.

In our example the result will be:

```
In [ ]: <b> X..DOCTYPE.HTML.PUBLIC...IETF..DTD.HTML.2.0..EN. <br>
1                                     <html><head>
2                                     <title>404 Not Found</title>
3                                     </head><body>
4                                     <h1>Not Found</h1>
5                                     <p>The requested URL was not found on this server.</p>
6                                     <hr>
7 <address>Apache/2.4.52 (Ubuntu) Server at inspire.rasdaman.org Port 443</address>
8                                     </body></html>

[1] NA NA NA NA NA NA NA NA,
```

Then when we convert the result into JSON, it gives us:

```
In [ ]: [{"X..DOCTYPE.HTML.PUBLIC...IETF..DTD.HTML.2.0..EN.": "<html><head>"},
        {"X..DOCTYPE.HTML.PUBLIC...IETF..DTD.HTML.2.0..EN.": "<title>404 Not Found\\",
        {"X..DOCTYPE.HTML.PUBLIC...IETF..DTD.HTML.2.0..EN.": "<\\head><body>"}, \
        {"X..DOCTYPE.HTML.PUBLIC...IETF..DTD.HTML.2.0..EN.": "<h1>Not Found\\</h1>"},
        {"X..DOCTYPE.HTML.PUBLIC...IETF..DTD.HTML.2.0..EN.": "<p>The requested URL w",
        {"X..DOCTYPE.HTML.PUBLIC...IETF..DTD.HTML.2.0..EN.": "<hr>"},
        {"X..DOCTYPE.HTML.PUBLIC...IETF..DTD.HTML.2.0..EN.": "<address>Apache/2.4.52",
        {"X..DOCTYPE.HTML.PUBLIC...IETF..DTD.HTML.2.0..EN.": "<\\body><br><\\html>"}]
```

When we apply `as.numeric()` function on our data, we receive:

[1] NA NA NA NA NA NA NA NA

Which is basically a converted input object to a numeric data type, using the following rules:

If the input object is a character string that contains only digits and optional signs (+/-) and decimal points, then the function converts it to a numeric value.

If the input object is a logical value (TRUE or FALSE), then the function converts it to 1 (TRUE) or 0 (FALSE).

If the input object is a factor or a character string that does not contain only digits and optional signs (+/-) and decimal points, then the function throws an error.

If the input object is already numeric, then the function returns the input object as is.

If the input object is missing (NA), then the function returns NA.

Finally, we apply `plot()` function on the array we have and receive the final graph. The function is a part of the R package "raster". It creates a line plot with the values of the numeric array on the y-axis and the index of the values on the x-axis. The `type = "l"` argument is used to specify that a line plot should be created.

Overall R gives a vast range of possibilities for Geospatial Data Processing, as shown above in the previous example. Using the functionality of R, given existing functions, including mentioned functions, any user can advance in the further processing.