Journal of
open research software

Korosov, A A, et al 2016 Nansat: a Scientist-Orientated Python Package for Geospatial Data Processing. *Journal of Open Research Software,* 4: e39, DOI: http://dx.doi.org/10.5334/jors.120

SOFTWARE METAPAPER

Nansat: a Scientist-Orientated Python Package for Geospatial Data Processing

Anton A. Korosov[1], Morten W. Hansen[1], Knut-Frode Dagestad[2], Asuka Yamakawa[1], Aleksander Vines[1], and Maik Riechert[3]

[1] Nansen Environmental and Remote Sensing Center, Bergen, Norway

[2] Norwegian Meteorological Institute, Bergen, Norway

[3] University of Reading, GB

Corresponding author: Anton A. Korosov
(anton.korosov@nersc.no)

Nansat is a Python toolbox for analysing and processing 2-dimensional geospatial data, such as satellite imagery, output from numerical models, and gridded in-situ data. It is created with strong focus on facilitating research, and development of algorithms and autonomous processing systems. Nansat extends the widely used Geospatial Abstraction Data Library (GDAL) by adding scientific meaning to the datasets through metadata, and by adding common functionality for data analysis and handling (e.g., exporting to various data formats). Nansat uses metadata vocabularies that follow international metadata standards, in particular the Climate and Forecast (CF) conventions, and the NASA Directory Interchange Format (DIF) and Global Change Master Directory (GCMD) keywords. Functionality that is commonly needed in scientific work, such as seamless access to local or remote geospatial data in various file formats, collocation of datasets from different sources and geometries, and visualization, is also built into Nansat. The paper presents Nansat workflows, its functional structure, and examples of typical applications.

Keywords: Python, Nansat, GDAL, geospatial data, satellite remote sensing, data synergy, data handling

Funding Statement: This work was supported via the NORMAP project under the Research Council of Norway (grant no. 195397/V30), the EU FP7 project SeaU under contract no. 263246, the European Space Agency (ESA) and the Norwegian Space Centre (NSC) through the Prodex Experiment Arrangement (Prodex ISAR, project no. 4000108820), and the NSC through researcher support (JOP.15.13.2).

(1) Overview

Introduction

Geospatial data is information that identifies the geographic location of features and boundaries on Earth, such as natural or constructed features, oceans, and more, usually stored as coordinates and topology, that can be mapped [1]. This paper describes a new tool to handle the interpretation and processing of geospatial data from satellite images and numerical models.Typical operations on such data include visualization for human perception of spatial patterns, extraction of geophysical values, pixel-per-pixel or contextual calculation of new geophysical variables based on one or several input datasets, re-gridding to another coordinate system, co-location of several datasets on one spatial grid, subsetting in space, and automatic recognition and description of objects.

With a growing amount of available geospatial data, the complexity of this work is, however, increasing. As an example, a collection of 2-dimensional (2D) datasets acquired over one geographic region at a given point of time will form a 3D cube of data, and time series of such cubes represent 4D datasets. The combination of multidimensional data analysis with significant input data heterogeneity and size (a single satellite may, e.g., provide 20TB of data per day) creates a challenge for researchers [2], and powerful and customized tools are thus required.

A number of analysis and processing tools for gridded data already exist, e.g., ERDAS IMAGINE [3], Bilko [4], and ArcGis [5]. Both the European Space Agency (ESA) and the National Aeronautics and Space Administration (NASA) have developed and provide software tools for processing their respective satellite data, including, e.g., the Sentinel Toolbox [6] and SeaDAS [7]. However, these tools are written by programmers, from a programmer perspective, in compiled low-level and object-oriented languages [8], such as Java or C/C++ to accomplish highly generic software and reduced execution time. However, scientists

often prefer higher-level programming languages in order to minimize the abstraction from their scientific ideas. The nature of scientific work also requires researchers to be able to quickly modify and extend their software tools to be able to analyse new sets of information. The possibility to work from the command line, and create quick and simple visualizations is therefore very important, whereas execution speed is usually less of a concern. Python [9] is a programming language that can meet these needs. It has a large user community, and it is free and open source. Existing Python packages, such as Numerical Python (Numpy) [10], Scientific Python (Scipy) [11] and Matplotlib [12], already provide much of the required flexibility, and code can be speeded up by clever programming practices. Compared to, e.g., Java, it is relatively easy to get started with Python, and the threshold for being able to produce scientific results is lower. As Python is object oriented, it also allows programmers to create reusable and complex software tools. As such, Python is highly suitable for scientists, while overlapping into what programmers want.

Many of the above mentioned tools are available only as Graphical User Interfaces (GUIs). A challenge with such tools is to automate operations that could be performed by computers alone [13, 14]. In an attempt to solve this problem, scientific users of geospatial data often develop their own software tools which can be more easily integrated into processing chains handling multiple datasets (e.g., an archive of satellite images) [15]. With the boom of open source software, the programming paradigm has gradually changed from development of large, strongly coupled packages to modular, unifunctional libraries that can be used together as building blocks for multifunctional applications. The Geospatial Data Abstraction Library (GDAL [16]) is an example of an open source software rarely used alone, but which is included in many applications (even proprietary) and online services such as, e.g., QGIS [17] and GRASS [18]. GDAL has a very usable and well documented application programming interface (API), with bindings for many programming languages, and a number of community developed drivers that are capable of reading geospatial data and metadata from many file formats.

The advantage of GDAL is that it provides access to various sources of information using the same universal data model. However, a limitation for scientific users is that GDAL only provides access to the data or metadata stored in the file and does not attach any geophysical meaning to the data. Thus, it may become difficult for a researcher to correctly interpret the content of the dataset with regard to the relevant band name, calibration, physical units, spatial scale, etc. In addition, GDAL provides low-level access to the data, and a scientific user is often required to perform several complicated operations in order to eventually fetch the values of the required geophysical variable at a given resolution and region of interest. Furthermore, the structure of files containing geospatial data varies significantly between its sources, and GDAL does not provide a unified interface. In many datasets, there are specific details about, e.g., subdatasets, ground control points, or the location of metadata, which should be taken into account.

The main goal of Nansat is thus to combine the versatility and power of GDAL with scientific knowledge about the content of geospatial data from various sources in a user-friendly Python package. In order to simplify the data use and interpretation, Nansat adds a *priori* known scientific metadata following the CF and GCMD conventions [19, 20] to the datasets, unifies the access to various data (either in local files or through the OpenDAP protocol) as much as possible, and provides a high level API to enable use in diverse, as well as automated operations. Via GDAL, spatial transformation of data and exporting the data into common formats (e.g., NetCDF-CF, PNG, GeoTIFF) is also facilitated.

Implementation and architecture

In this section, the instantiation of a **Nansat** object with geospatial satellite data is summarized and schematically presented on **Fig. 1** (subsection A), and the Nansat structure including details about classes, mappers and use of third party libraries are given (subsection B). Examples of more advanced processing are provided in section (3) 'Reuse Potential'.

A. Nansat workflow

(1) A user invokes the command `n=Nansat (input_file_name)`, where `input_file_name` is a string that includes the full path to the input file containing the geospatial data.
(2) The **Nansat** constructor uses GDAL to open the input file and extract provided metadata.
(3) The **Nansat** constructor generates a list of available mappers based on the content of the nansat.mappers subpackage. A mapper is a Python class named **Mapper**, where the structure of the opened file is defined a priori, or where qualified guesses based on, e.g., filenames and metadata following known conventions (e.g., the CF-conventions) are defined. A mapper maps the structure of the input file to the structure of the Nansat object and provides meaning to the dataset through added metadata. Since there are many file types which can be opened by Nansat, many mapper classes have been developed. In each mapper class, the structure of the input file (e.g., the number of bands and their content, type and name of the sensor and platform, relevant global metadata) is hardcoded or partly dynamically constructed. The Nansat constructor then loops through the available mappers, and.
(4) Each of the mapper constructors tests whether or not the opened file matches the given mapper.
(5) If the opened file matches the given mapper, a GDAL Virtual XML file [21] (hereinafter referred to as VRT file), which represents the description of the GDAL data model, is created. The mapper constructor then adds relevant bands from a *priori* specified bands of data provided in the input file. The Mapper constructor also adds conventional metadata

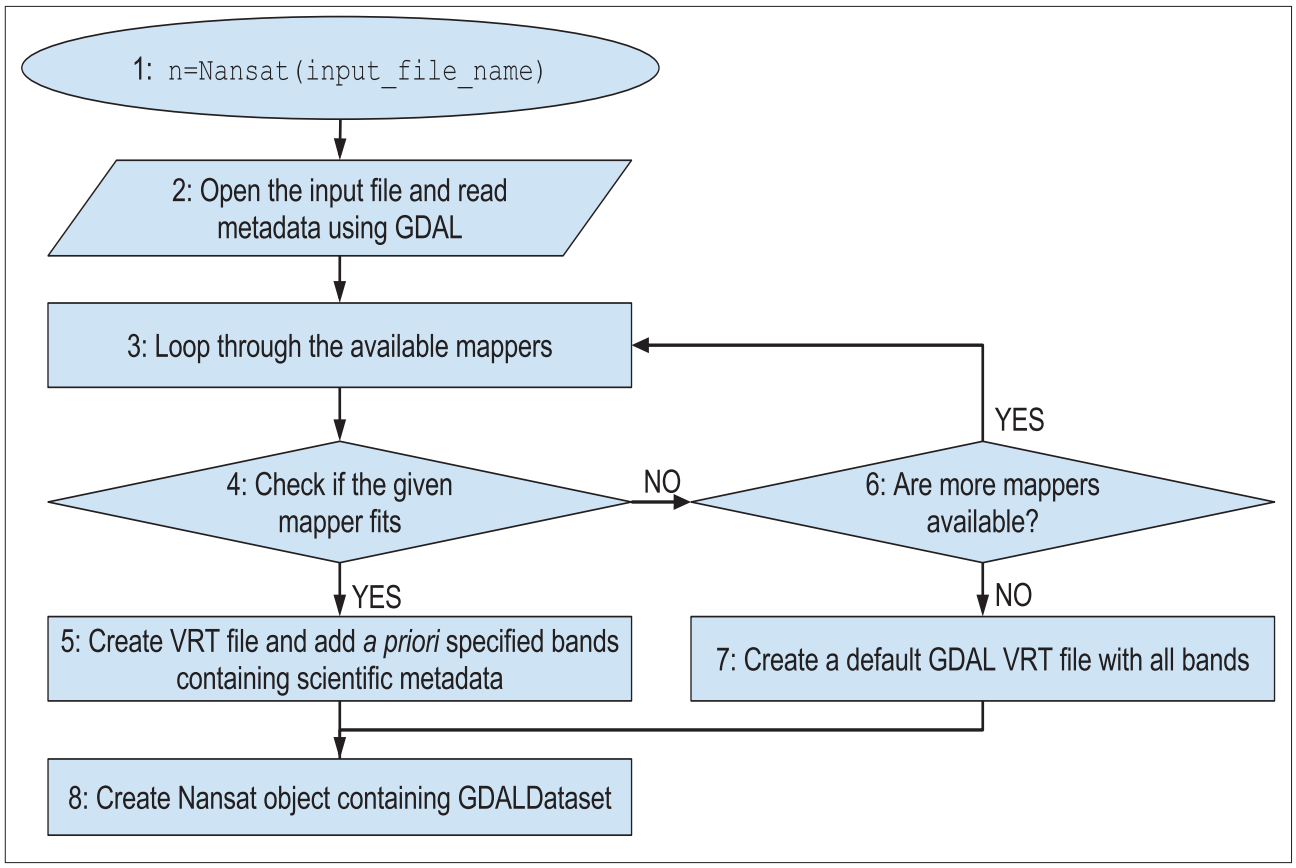

Figure 1: Schematic flowchart of **Nansat** instantiation. Details of the process are explained in the text.

with the standard and short name of geophysical variables, their units, valid minimum and maximum to the VRT file for each band. If scaling of values is needed a corresponding scales and offsets are also added to the VRT file.

(6) If the given mapper does not match the provided file, a **MapperError**, which is a dedicated Python Exception, is immediately raised, and the next mapper is tested.

(7) If none of the mappers matches the input file, the Nansat constructor creates the VRT file without scientific metadata.

(8) Finally, the Nansat constructor creates a GDAL **Dataset** object from the VRT file, and adds this to the attributes of the generated Nansat object. A GDAL Dataset is a class that provides various methods for low level access to a set of associated raster bands and metadata [22].

B. Nansat functional structure

Nansat is designed as a modular collection of classes (**Fig. 2**) to make the code more cohesive and less coupled [23], and thus to facilitate independent development of each block. The class **Domain** describes geographical projection, span and resolution of any kind of gridded data. It contains an instance of the class **VRT**, which operates on VRT files.The class **VRT** contains an instance of a GDAL **Dataset** for the actual reading of data and metadata. **VRT** is a base class for **Mapper** classes, which add scientific meaning to various data read by GDAL. A **Domain** instance also contains an **NSR** object, which subclasses OGR **SpatialReference** for work with spatial reference systems. **Nansat** is the central class in the package and a subclass of **Domain**. **Nansat** contains **Figure** (convenient generation of images) and **PointBrowser** (interactive digitizer). **Nansat** is a base class for **Mosaic**, which performs mosaicking of several files.

The **Nansat** class is the main container for the geospatial data, and performs all high-level operations. An instance of the **Nansat** class contains information about the geographical reference of the data (e.g., raster size, pixel resolution, type of projection, etc.) and about bands with geophysical variables (e.g., water leaving radiance, normalized radar cross section, chlorophyll concentration). A **Nansat** instance contains methods for high-level operations on the data, including mainly the following:

- **__get_item__,** or **[]** (square brackets): fetch data as a Numpy ndarray;
- **write_figure:** use the class **Figure** for writing full resolution data to an RGB or indexed image with, e.g., land mask, automatic correction of brightness/contrast, and logarithmic scaling;
- **reproject:** change the projection of the dataset, and consequently its span and resolution. This is a lazy operation, meaning that only the projection

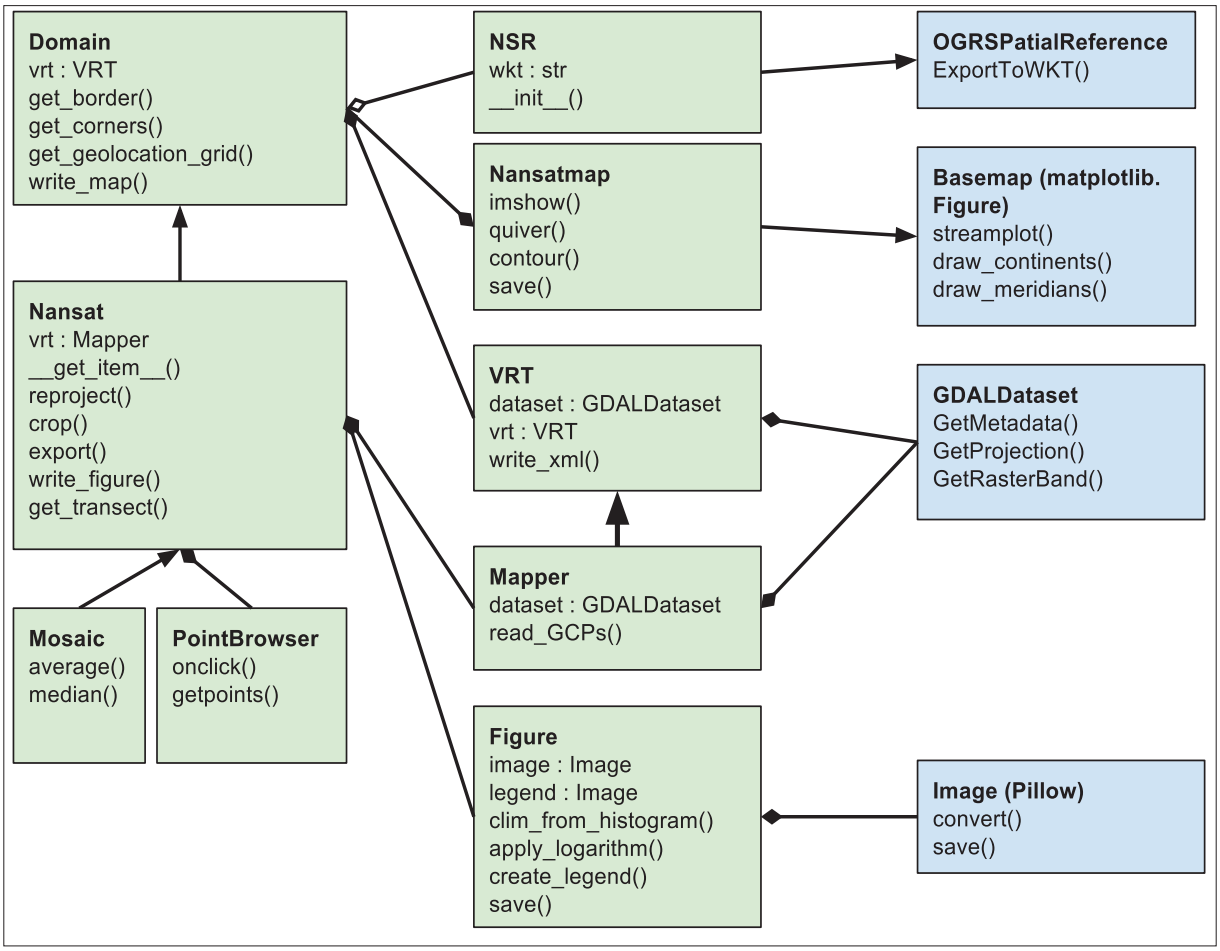

Figure 2: UML diagram of the Nansat package including the most important class methods. White boxes describe third party classes.

parameters in the core VRT file are modified. The actual resampling happens only when the data is requested (e.g., within the method write_**figure**); and
- **export:** write data to a GeoTIFF or NetCDF [24] file following OGC conventions and standards.

The **Domain** class is a container for the geographical reference of the raster data. A **Domain** instance is a GDAL Dataset with information about the geolocation information only (i.e., no bands), containing, e.g.,

- The type and parameters of a spatial reference system (SRS) (projection), e.g. cylindrical or stereographic, and central meridian and parallel;
- The grid width and height (number of pixels or grid cells);
- The pixel size in decimal degrees, metres or other units (depends on the selected projection); and
- The relation between pixel/line coordinates and geographical coordinates, e.g., as a linear relation or as ground control points (GCP).

There are three ways to store the geo-reference in a GDAL dataset:

1. As a GDAL GeoTransform, which defines a linear relationship between the raster pixels and lines, and the geographical X and Y coordinates;
2. As GCPs (Ground Control Points), which define the relationship between raster pixels and lines, and geographical X and Y coordinates; and
3. As a Geolocation Array, which is a grid of geographical X and Y coordinates for each pixel in a raster dataset.

The relation between row and column coordinates of the raster and geographic (e.g. latitude and longitude) coordinates is defined by the projection type and projection parameters.

The **Domain** class has methods for basic operations with georeference information:

- **__init__:** (the constructor) creates the georeference based on provided arguments;
- **get_corners, get_border, get_geolocation_grid:** fetch coordinates of the grid corners, the footprint/coverage of the grid, or the entire raster; and
- **write_map:** create map of the grid footprint, and add coastlines and graticule.

The **VRT** class is a wrapper around a GDAL vrt file. The vrt file is in XML format, and contains the geo-reference,

the global and band metadata, and reference to the data grids (or bands) provided in the source file. The class VRT performs all operations on the **VRT** files, i.e., copying, modifying, reading, and writing of files, adding bands, setting spatial reference attributes as such projection or coordinate transformation, and more. These operations are performed either directly by methods in the **VRT** class (e.g., remove_geotransform and get_warped_vrt), or using GDAL functions (e.g., Create, AddBand, SetMetadata, and AutoCreateWarpedVRT). The core of the **VRT** object is a GDAL Dataset instance generated by the GDAL VRT Driver. The respective vrt file is located in computer memory, and is accessible using the GDAL VSI functionality (the path to the vrt file is /vsimem). The **VRT** class can also be used to write binary files with actual data accompanied with a wrapping vrt file that describes its dimensions and data type. This is used when adding data to existing **Nansat** instances or caching CPU consuming operations.

When **Domain** is instantiated, the **VRT** object is created without any bands, only with geo-reference information using the GDAL **Dataset** methods. On the contrary, when a file is opened with **Nansat**, the **VRT** object with both the georeference and the bands is created using the class **Mapper**.

When **Nansat** performs operations that change the vrt file, e.g., by reprojecting, cropping, resizing or adding bands, an additional vrt file with a corresponding **VRT** object is generated. The new **VRT** object keeps a reference to the original **VRT** object. This allows to easily and safely undo operations.

The **Figure** class performs graphical operations. It is used to create figures, append a legend, add land mask, and adjust brightness and colours, as well as saving figures in PNG, JPG or TIFF formats. An instance of **Figure** can be created in the **Nansat write_figure** method, or from a Numpy 2D array (or three 2D layers in case of an RGB image). The **Figure** class uses two Pillow (Python Imaging Library) **Image** objects, one for the image canvas and one for the legend. **Figure** has several convenient methods for optimizing the data presentation:

- **clim_from_histogram:** estimates optimal minimum and maximum brightness limits based on the data distribution;
- **apply_logarithm:** applies logarithmic transformation of the original data;
- **create_legend:** applies one of the Matplotlib colormaps, and adds a legend with colorbar and caption; and
- **array_mod_function:** modifies the given band array by a user provided function.

The **Nansatmap** class generates nice looking maps where several datasets can be displayed together. It extends the Python **Basemap** [25] class and overrides its constructor for simpler instantiation based on a given Nansat object. **Nansatmap** also overrides several Basemap methods by taking advantage of a priori knowledge about the spatial extent:

- **imshow:** adds a raster layer to the map in a fast manner and colorizes with a given colormap and value limits;
- **smooth:** spatially smooths the input data;
- **contour(f):** creates (filled) contour lines;
- **quiver:** adds regularly spaced and correctly rotated arrows indicating direction and speed;
- **draw_grid:** adds labeled graticule; and
- **save:** saves to a graphical format at a given resolution.

The **Mosaic** class extends Nansat and provides capability to create mosaics of several input files using one of two methods:

- **average:** a memory-friendly and multi-threaded method for averaging data. It reads all input files using Nansat, re-projects the raster data of the required bands onto the original **Mosaic** object, calculates the mean and standard deviation, and adds these as new bands in the **Mosaic** object. The method tries to get a band with name 'mask' from the input files. The mask should have the following encoding:
    - 0: invalid 1 (e.g. nodata),
    - 1: invalid 2 (e.g. clouds),
    - 2: invalid 3 (e.g. land),
    - 64: valid pixel.

If the "mask" band is present, it is used to select the valid pixels for averaging (i.e. where mask equals 64). Otherwise it is assumed that all input pixels are valid.

- **median:** a more memory greedy method working in only one computational thread. Instead of returning the mean, this function calculates the median of specific bands in the input files, and adds the resulting array as a new band in the Mosaic object.

Quality control

Quality control of the code is implemented in two blocks: unit tests and integration tests. Currently, the unit tests cover 78% of the core code [26] and thus the most important functionality. Sample datasets are pre-generated for running the unit tests, and consists of synthetic satellite images with data either on a regular projection or with the projection defined by ground control points (GCPs). The code repository is integrated with the Travis Continuous Integration platform [27] and tests are automatically run on each commit.

The integration tests are developed for testing mappers and operation chains on actual data. The integration tests are designed as a separate package based on the Python unittest [28] package, and are run only locally (i.e.,not on TravisCI). This is to prevent downloading of large amounts of satellite data on the test server. Including the integration tests, the coverage of all code including all mappers is 59% (this is not reflected in the Coveralls badge in the readme file at GitHub).

(2) Availability

Operating system

Nansat has been tested on OS X and Linux and is also provided in a Vagrant/Ansible configuration for virtual machines (CentOS 7 or Ubuntu 14.04). Thus, it is easy to install the tool also on Windows, although the actual OS would be Ubuntu.

Programming language

Most of Nansat is written in Python 2.7. A small and optional part (GDAL pixel functions [16]) is written in C.

Additional system requirements

The code is 8 MB, and the system requirements largely depend on the application of Nansat. Small satellite images (less than 1000 x 1000 pixels) can be processed with 100 MB of free RAM and a 1.5 GHz processor. Large datasets (e.g. synthetic aperture radar (SAR) images exceeding 10000 x 10000 pixels) require > 4 GB RAM and > 2.5 GHz CPU.

Dependencies

Nansat has both required and optional dependencies. Installation of all the dependencies for full Nansat functionality, i.e., to access data in netCDF, HDF4 and HDF5 files via GDAL, may sometimes be cumbersome. Therefore, we have pre-built fully functional binaries of GDAL for Linux (32 and 64 bits), which are available in the Anaconda Cloud [29, 30]. To further simplify the installation, we have created a Vagrant/Ansible configuration to install Nansat and all required dependencies on a local virtual machine as referenced in the "Emulation Environment" section.

The basic requirements of Nansat are:

- GDAL >= 1.11.4 [16]
- Numpy >= 1.10.4 [10]
- Scipy >= 0.17.1 [11]
- Matplotlib >= 1.5.1 [12]
- Basemap >= 1.0.8dev [25]
- Pillow >= 3.2.0 [31]
- py-thesaurus-interface = 1.1.1 [32]

In addition, there are optional dependencies for specific Nansat functionality, in particular:

- netcdf4 >= 1.2.4 required to read data from OPeNDAP
- gcc >= 4.8.2 required to compile the pixel functions

List of contributors

Anton Korosov, Nansen Environmental and Remote Sensing Center, Bergen, Norway
Morten W. Hansen, Nansen Environmental and Remote Sensing Center, Bergen, Norway
Asuka Yamakawa, Nansen Environmental and Remote Sensing Center, Bergen, Norway
Knut-Frode Dagestad, Norwegian Meteorological Institute, Bergen, Norway
Aleksander Vines, Nansen Environmental and Remote Sensing Center, Bergen, Norway
Maik Riechert, University of Reading, Reading, United Kingdom
Alexander Myasoedov, SoLab, Russian State Hydrometeorological University, St. Petersburg, Russia
Evgeny Morozov, Nansen International Environmental and Remote Sensing Center, St. Petersburg, Russia
Natalia Zakhvatkina, Nansen International Environmental and Remote Sensing Center, St. Petersburg, Russia

Software location

Archive
Name: Zenodo
Persistent identifier: 10.5281/zenodo.59998
Licence: GNU GPLv3
Publisher: Anton Korosov
Date published: 11/08/2016 (last release, 0.6.14)

Code repository

Name: GitHub
Identifier: https://github.com/nansencenter/nansat.git
Licence: GNU GPLv3
Date published: 18/01/2013 (first release, 0.4)

Emulation environment

Name: Provisioning of virtual machines with Vagrant/Ansible
Identifier: https://github.com/nansencenter/nersc-vagrant.git
Licence: GNU GPLv3
Date published: 10/09/2015

Language

English

(3) Reuse potential

The potential to reuse Nansat for scientific applications is illustrated in three use cases. These examples, plus other tutorials are available in the nansat-lectures repository [33].

Nansat is at the time of writing capable of reading data from tens of platforms and instruments in 8 file formats, as specified in the online table at https://github.com/nansencenter/nansat/blob/master/nansat/mappers/README.yml. Nansat can be extended by inheritance or by adding new mappers, as described in the online manual [34]. User support and feedback is realized via GitHub issues and through a mailing list nansat-dev@google-groups.com.

Example 1: Open, collocate, and write figure

This example covers spatial collocation of two images acquired in May 2016 by the Synthetic Aperture Radar (SAR) instrument onboard the Sentinel-1A [35] satellite, and the Moderate Resolution Imaging Spectroradiometer (MODIS) [36] onboard the Aqua satellite. The two original level-1 datasets have different spatial reference, resolution and coverage, and are collocated and projected onto the same spatial grid, and then visualized using Nansat. The resulting images (**Fig. 3**) show MODIS

```
from nansat import Nansat, Domain

# Open file with MODIS image
n1 = Nansat('MYD02HKM.A2016148.1050.006.2016149162607.hdf')

# Open file with Sentinel-1A image
n2 = Nansat('S1A_EW_GRDM_1SDH_20160527T163927_20160527T164031_011447' \
               '_0116DE_3ED0.SAFE')

# Define region of interest (south of Lofoten islands)
d = Domain('+proj=stere +lon_0=12 +lat_0=67 +no_defs',
           '-te -50000 -50000 50000 50000 -tr 100 100')

# Collocate the images
n1.reproject(d)
n2.reproject(d)

# Write RGB graphic file from MODIS data
f = n1.write_figure('test_rgb_MODIS.png', ['L_858', 'L_555', 'L_469'],
                    clim='hist', ratio=0.95)

# Write grayscale file from Sentinel-1A data
f = n2.write_figure('test_SENTINEL1.png', 'sigma0_HH',
                    clim='hist', ratio=0.95, cmapName='gray')

# Write map of the region of interest

d.write_map('test_domain_map.png', resolution='h', dpi=300,
            meridians=4, parallels=4,
            latBorder=1, lonBorder=1,
            merLabels=[False, False, True, False],
            parLabels=[True, False, False, False])
```

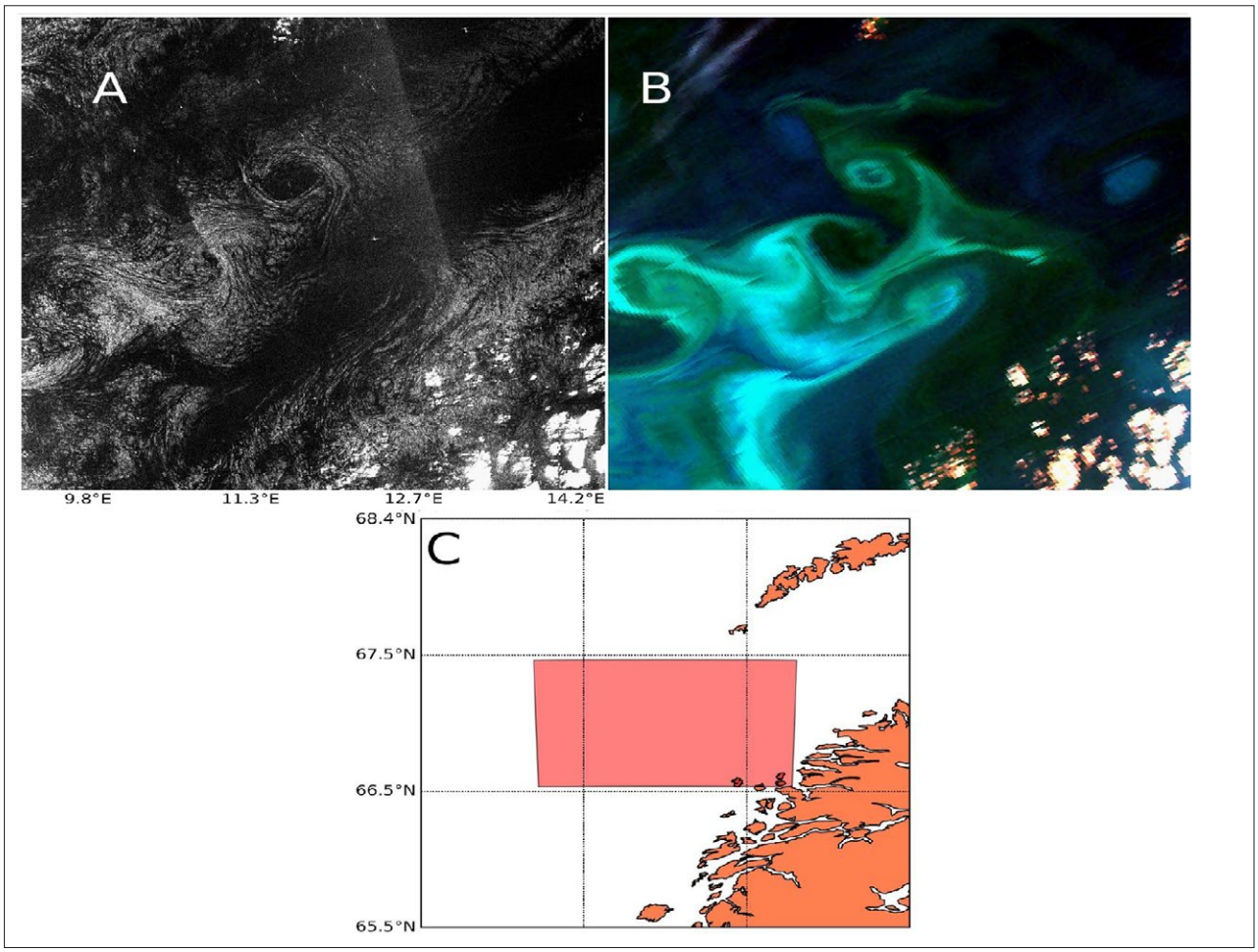

Figure 3: Example of a simple application of Nansat to collocate and visualize synchronous observations acquired by MODIS and Sentinel1A on 25 May 2011. (A) An RGB image from a MODIS/Aqua acquisition over the Norwegian Sea, (B) a grayscale SAR image from Sentinel-1A, and (C) the data coverage.

```
from nansat import Nansat, Domain

# Access online sea surface temperature data streamed from
# the Norwegian Meteorological Institute (met.no) via the
# OpenDAP protocol
n1 = Nansat('http://thredds.met.no/thredds/dodsC/sea_ice/' \
                    'SSTMETNOARCSST_L4OBSV2V1/sst_arctic_aggregated',
            date = '20120601',
            bands = ['analysed_sst'])

# Access online Chlorophyll-A data streamed from
# Plymouth Marine Laboratory, UK, via the OpenDAP protocol
n2 = Nansat('https://rsg.pml.ac.uk/thredds/dodsC/CCI_ALLv2.08DAY',
            date = '20120601',
            bands = ['chlor_a'])

# Get transects from points given by [longitude, latitude]
t1 = n1.get_transect([[24, 24], [71, 77]], ['analysed_sst'])
t2 = n2.get_transect([[24, 24], [71, 77]], ['chlor_a'])

# Fetch values from the transects
sst_lat = t1['lat']
sst_val = t1['analysed_sst']
chl_lat = t2['lat']
chl_val = t2['chlor_a']

# Mask invalid SST values
sst_val[sst_val < 0] = np.nan
# ...[Plotting is performed using standard Matplotlib and Basemap
# libraries (Fig. 4)]...
```

top-of-atmosphere radiances measured at 469, 555 and 858 nm displayed in RGB (**Fig. 3A**) and a grayscale image of the same area generated from SAR HH (horizontal transmit and horizontal receive) polarized Normalized Radar Cross Section (NRCS) (**Fig. 3B**) covering parts of the Norwegian Sea. In these images, we see ocean mesoscale eddies as manifested by both the ocean colour (**Fig. 3A**) and the sea surface roughness (**Fig. 3B**).

Example 2: Analyse transects using data streamed via the OPeNDAP protocol

In this example Nansat is used to compare two datasets of surface Chlorophyll-A (Chl-A) and Sea Surface Temperature (SST) in the Barents Sea. As shown in the code below, Nansat accesses remote Chl-A and SST datasets streamed from the Norwegian Meteorological Institute and Plymouth Marine Laboratory (PML) (the latter dataset is provided as part of

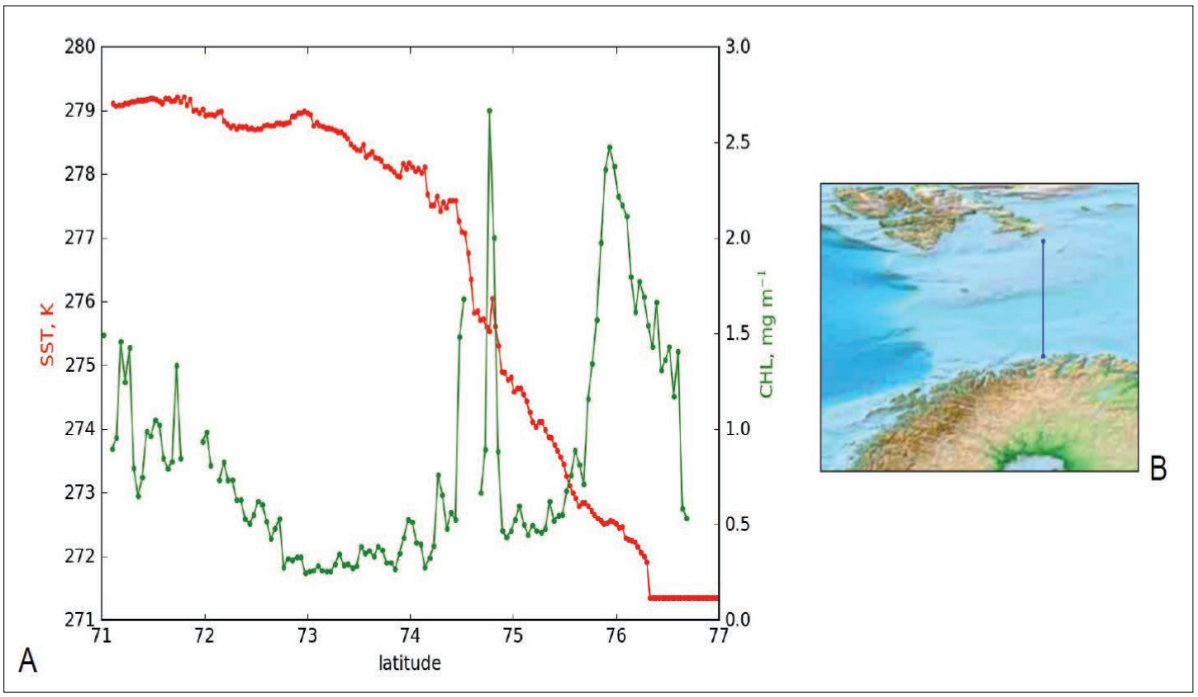

Figure 4: Comparison (A) of Sea Surface Temperature (SST) and Chlorophyll-A (Chl-A) from transect (B) across the Barents Sea (the data is retrieved using Nansat, and the figure is generated using Matplotlib and Basemap).

the ESA Ocean Colour Climate Change Initiative) via the OpenDAP protocol. When connection to the data stream is established, Nansat is used to define the transects and retrieve and plot the actual data (**Fig. 4**). The Matplotlib and Basemap code used for creating **Fig. 4** is out-of-scope of the paper and is thus not provided.

Example 3: Fusion and visualization of Sea Surface Temperature (SST), sea ice concentration and sea ice drift

The example below illustrates how to use Nansat to access online datasets of sea ice concentration, SST and sea ice drift streamed from the Norwegian Meteorological

```
from nansat import Nansat, Domain, Nansatmap

# Access online sea ice concentration data (IC) from met.no
n1 = Nansat('http://thredds.met.no/thredds/dodsC/osisaf/met.no/' \
                'ice/conc/2016/04/' \
                'ice_conc_nh_polstere100_multi_201604011200.nc',
             bands = ['ice_conc'])

# Access online SST data from met.no
n2 = Nansat('http://thredds.met.no/thredds/dodsC/myocean/' \
               'siwtac/sstmetnoarcsst03/'\
               '20160401000000METNOL4_GHRSST-SSTfnd' \
               'METNO_OIARCv02.0fv02.0.nc',
            bands = ['analysed_sst'])

# Access online sea ice drift data from met.no
n3 = Nansat('http://thredds.met.no/thredds/dodsC/osisaf/met.no/' \
                'ice/drift_lr/merged/2016/04/' \
                'ice_drift_nh_polstere625_multi' \
                'oi_2016040112002016040312000. nc',
            bands = ['dX', 'dY'])
# Define domain of the region of interest
d = Domain('+proj=stere +lon_0=45 +lat_0=90 +no_defs',
           '-te 300000 -1200000 1700000 300000 -tr 10000 10000')

# Project the collocated datasets onto the same domain
n1.reproject(d)
n2.reproject(d)
n3.reproject(d)

# Retrieve data covering the region of interest
ice_conc = n1['ice_conc']
analysed_sst = n2['analysed_sst']
dX = n3['dX']
dY = n3['dY']

# Mask invalid data with Numpy NotANumber values
ice_conc[ice_conc <=1] = np.nan
analysed_sst[analysed_sst < 0] = np.nan

# Create canvas for drawing a map
nmap = Nansatmap(n2, resolution ='l')

# Add SST data as a raster layer, then the corresponding colorbar
nmap.imshow(analysed_sst, vmin = 270, cmap = 'viridis')
nmap.add_colorbar(shrink = 0.5, pad = 0.05)

# Add the ice concentration as a raster layer
nmap.imshow(ice_conc, vmin = 0, vmax = 100, cmap = 'bone')

# Add the ice drift data as vectors
nmap.quiver(dX, dY, step = 5)

# Decorate the map with a grid
nmap.drawmeridians([-20, 0, 20, 40], labels = [False, True, False, True],
                   fontsize = 6)
nmap.drawparallels([75, 80, 85], labels = [True, False, True, False],
                   fontsize = 6)
# Save the image as a graphical file
nmap.save('nmap_example5.png', dpi = 300)
plt.close('all')
```

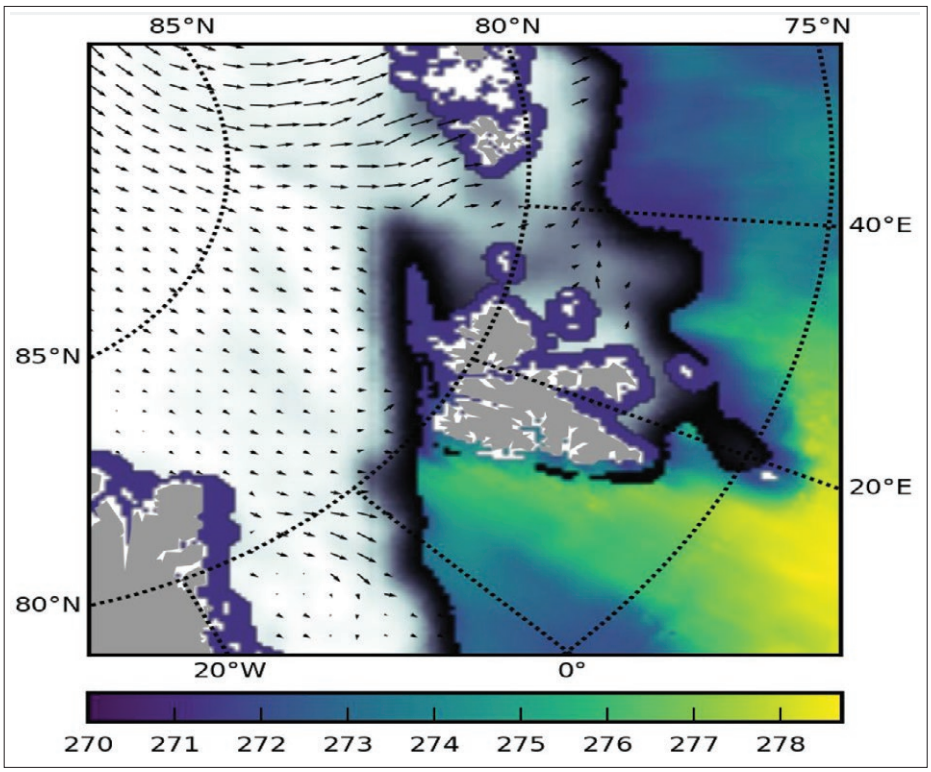

Figure 5: Map of Sea Surface Temperature (SST), sea ice concentration, and sea ice drift streamed from the Norwegian Meteorological Institute, collocated over the Greenland, Barents, and Arctic Seas. The land mass in the centre of the image is Svalbard.

Institute, collocate the datasets, and create a nice looking map (**Fig. 5**).

Competing Interests

The authors declare that they have no competing interests.

References

1. **Beal, F, Stroud, F, Shread, P.** 2016 WEBOPEDIA, URL: http://www.webopedia.com/TERM/S/spatial_data.html.
2. **Korosov, A, Counillon, F, Johannessen, J A.** 2014 Monitoring the spreading of the Amazon freshwater plume by MODIS, SMOS, Aquarius, and TOPAZ, *Journal of Geophysical Research: Oceans*, 120(1). DOI: http://dx.doi.org/10.1002/2014JC010155
3. **ERDAS, I** 2016 URL: http://www.hexagongeospatial.com/products/producer-suite/erdas-imagine
4. **Bilko software for ESA LearnEO!** 2012 URL: http://www.learn-eo.org/software.php.
5. **ArcGIS** 2010 URL: https://www.arcgis.com.
6. **Toolboxes Sentinel Online – ESA** 2014 URL: https://sentinel.esa.int/web/sentinel/toolboxes.
7. **Feldman, G C.** 2002 SeaDAS Home Page, URL: http://seadas.gsfc.nasa.gov.
8. **"Low-level programming language"** 2016 WIKIPEDIA, URL: https://en.wikipedia.org/wiki/Low-level_programming_language.
9. **Rossum, G.** 1997 Scripting the Web with Python. In "Scripting Languages: Automating the Web", World Wide Web Journal, 2(2), O'Reilly., URL: https://www.python.org/.
10. **Oliphant, T,** et al. 2016 Numpy, URL: http://www.numpy.org/.
11. **Oliphant, T, Peterson, P, Jones, E,** et al. 2016 Scipy, URL: https://www.scipy.org/.
12. **Hunter, J D,** et al. 2016 Matplotlib, URL: http://matplotlib.org/.
13. **Korosov, A A, Pozdnyakov, D V, Grassl, H.** 2012 Spaceborne quantitative assessment of dissolved organic carbon fluxes in the Kara Sea, *Advances in Space Research*, 50, 1173–1188, DOI: http://dx.doi.org/10.1016/j.asr.2011.10.008
14. **Muckenhuber, S, Nilsen, F, Korosov, A** and **Sandven, S.** 2016 Sea ice cover in Isfjorden and Hornsund, Svalbard (2000–2014) from remote sensing data, *The Cryosphere,* 10, 149–158, DOI: http://dx.doi.org/10.5194/tc-10-149-2016
15. **Hansen, M W, Collard, F, Dagestad, K F, Johannessen, J A, Fabry, P, Chapron, B.** 2011 Retrieval of Sea Surface Range Velocities From Envisat ASAR Doppler Centroid Measurements, *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING,* 49, (10)6: 3582–3592, DOI: http://dx.doi.org/10.1109/TGRS.2011.2153864
16. **Rouault, E, Jolma, A, Baryshnikov, D.** 2004 GDAL Geospatial Data Abstraction Library, URL: http://www.gdal.org/.
17. **Sutton, T, Dassau, O, Sutton, M.** 2015 QGIS A Free and Open Source Geographic Information System, URL: http://www.qgis.org/.
18. **GRASS Development Team.** 2015 Geographic Resources Analysis Support System (GRASS) Software,

Version 7.0. Open Source Geospatial Foundation. URL : http://grass.osgeo.org.
19. **Eaton, B, Gregory, J, Drach, B, Taylor, K, Hankin, S, Caron, J, Signell, R, Bentley, P, Rappa, G, Höck, H, Pamment, A, Juckes, M.** 2011 NetCDF Climate and Forecast (CF) Metadata Conventions, Version 1.6, URL: http://cfconventions.org/.
20. **Wharton, S.** 2016 National Aeronautics and Space Administration (NASA), Global Change Master Directory (GCMD), Version 10.0, URL: http://gcmd.nasa.gov/.
21. **Rouault, E.** 2004 GDAL Virtual Format Tutorial, 2004 URL: http://www.gdal.org/gdal_vrttut.html.
22. **Rouault, E, Jolma, A, Baryshnikov, D.** 2004 GDAL Dataset Class Reference, URL: http://www.gdal.org/classGDALDataset.html.
23. **Stevens, W P, Myers, G J, Constantine, L L.** 1974 "Structured design". *IBM Systems Journal,* 13 (2): 115–13 DOI: http://dx.doi.org/10.1147/sj.132.0115
24. **Russ, R** and **Davis, G.** 1990 NetCDF: an interface for scientific data access. Computer Graphics and Applications, IEEE 10(4), p 76–82.
25. **Basemap Toolkit documentation.** 2012 URL: http://matplotlib.org/basemap/.
26. **Merwin, N, Donahoe, L, McAngus, E.** 2016 Coverage of nansat at github, URL: https://coveralls.io/github/nansencenter/nansat.
27. **Travis CI.** 2011 URL: https://travis-ci.org.
28. **Unit testing framework.** 2014 URL: https://docs.python.org/2/library/unittest.html.
29. **Continuum Analytics.** 2015 Anaconda Cloud, URL: https://anaconda.org.
30. **Gillingham, S,** et al. 2016 A conda-smithy repository for gdal, URL: https://github.com/conda-forge/gdal-feedstock.
31. **Lundh, F, Clarck, A,** et al. 2016 Pillow, URL: https://python-pillow.org/.
32. **Korosov, A A, Hansen, M W, Vines, A.** 2016 Python Thesaurus Interface, URL: https://github.com/nansencenter/py-thesaurus-interface.
33. **Korosov, A A** and **Hansen, M W.** 2016 Nansat-lectures URL: https://github.com/nansencenter/nansat-lectures.
34. **Korosov, A A, Dagestad, K F, Hansen, M W.** 2014 How to create a mapper, URL: https://github.com/nansencenter/nansat/wiki/How-to-create-a-mapper
35. **Meadows, P.** 2015 Sentinel-1A Annual Performance Report for 2015, CLS Technical report #MPC0139, CLS, URL: https://earth.esa.int/documents/247904/1814124/Sentinel-1A-Annual-Performance-Report-2015.
36. **Frazier, S.** 2016 MODIS Moderate Resolution Imaging Spectroradiometer, URL: http://modis.gsfc.nasa.gov/.

How to cite this article: Korosov, A A, Hansen, W M, Dagestad, F K, Yamakawa, A, Vines, A, Riechert, A 2016 Nansat: a Scientist-Orientated Python Package for Geospatial Data Processing. *Journal of Open Research Software,* 4: e39, DOI: http://dx.doi.org/10.5334/jors.120

Submitted: 16 February 2016 **Accepted:** 27 September 2016 **Published:** 24 October 2016

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS