NL-SAR Toolbox Manual

Charles-Alban Deledalle
charles-alban.deledalle@math.u-bordeaux.fr

February 7, 2018

NL-SAR Toolbox provides a collection of tools for the estimation of multi-modal SAR images with non-local filters. Beyond estimation, NL-SAR Toolbox provides a suite of tools to manipulate SAR images. There are 5 ways to interact with NL-SAR:

- in command line
- with Matlab
- with Python
- with IDL
- as a dynamic library (*e.g.* to use it from a C program).

Feel free to develop plugins for PolSARPro, OTB, NEST, SNAP...

So far, the command line version is the most stable one while others can crash, for instance, if you do not provide the good inputs in arguments. Feel free to fix such bugs or contribute to NL-SAR Toolbox as you wish under the term of the license (see Section 1).

NL-SAR Toolbox has been compiled and tested successfully on GNU/Linux (Ubuntu 12.04 Precise Pangolin & Linux Mint 13 Maya), Max OS X (10.7.5 Lion) and Windows 64bits (for cli and mex with MinGW-64 and for cli only with Cygwin-gcc).

Contents

1 License

This software is a computer program whose purpose is to provide a suite of tools to manipulate SAR images.

This software is governed by the CeCILL license under French law and abiding by the rules of distribution of free software. You can use, modify and/ or redistribute the software under the terms of the CeCILL license as circulated by CEA, CNRS and INRIA at the following URL "http://www.cecill.info".

As a counterpart to the access to the source code and rights to copy, modify and redistribute granted by the license, users are provided only with a limited warranty and the software's author, the holder of the economic rights, and the successive licensors have only limited liability.

In this respect, the user's attention is drawn to the risks associated with loading, using, modifying and/or developing or reproducing the software by the user in light of its specific status of free software, that may mean that it is complicated to manipulate, and that also therefore means that it is reserved for developers and experienced professionals having in-depth computer knowledge. Users are therefore encouraged to load and test the software's suitability as regards their requirements in conditions enabling the security of their systems and/or data to be ensured and, more generally, to use and operate it in the same conditions as regards security.

The fact that you are presently reading this means that you have had knowledge of the CeCILL license and that you accept its terms.

2 Installation

In the following, we are assuming that you are using `bash`. In this case, make sure that the configuration file `.bashrc` is sourced when opening a new shell instance. This is the default behavior, but not everytime (for instance on Mac OS X). In this case type once:

```
> echo 'source $HOME/.bashrc' >> $HOME/.bash_profile
```

If you are using another shell (like `csh` or `zsh`) replace in the followings `.bashrc` with the corresponding configuration file and adjust the commands accordingly.

2.1 Dependencies

First check the following dependencies:

software	feature	status
gcc	to compile the project	required
gcc version > 4.2	to enable parallelization with OpenMP	optional
lapack	to enable non-local filtering with covariance matrices higher than 3 x 3	optional
libtiff	to read TIFF images	optional
idl	to enable IDL interface	optional
matlab	to enable Matlab interface	optional
python	to enable Python interface	optional
numpy	to enable Python interface	optional
pdflatex	to create the documentation	optional

The above pieces of software have to be present in your environment variable `PATH` (for binaries) or `LD_LIBRARY_PATH` (for libraries) otherwise their associated feature will be disabled. For instance, if you want to enable the MATLAB interface, typing in your shell:

```
> which matlab
```

should give you the path where MATLAB is installed (e.g., `/Applications/MATLAB_R2013a.app/bin/matlab`). If it is not the case, type something like:

```
> echo 'export PATH=/Applications/MATLAB_R2013a.app/bin/:$PATH' >> $HOME/.bashrc
> source $HOME/.bashrc
```

Or, if you are using aliases, you can instead do the following:

```
> echo 'export PATH=$(dirname `echo ${BASH_ALIASES[matlab]}`):$PATH' >> $HOME/.bashrc
> source $HOME/.bashrc
```

Once you have checked your dependencies, you can compile and install NL-SAR in two ways: as a super user or as a non super user.

2.2 Installation for super users

First configure and compile NL-SAR by typing in a shell prompt:

```
> ./configure
> make
> sudo make install
```

This will install NL-SAR's command line interface and dynamic library interface in `/usr/`, NL-SAR's Matlab interface in Matlab's subdirectory `toolbox/nlsar/`, NL-SAR's Python interface in Python's site-packages directory and NL-SAR's IDL interface in IDL's subdirectory `external/lib/nlsar/`.

2.3 Installation for non super users

First configure and compile NL-SAR by typing in a shell prompt:

```
> ./configure --prefix=<NLSAR_PATH> --prefix-matlab=<MATLAB_PATH>
              --prefix-python=<PYTHON_PATH> --prefix-idl=<IDL_PATH>
> make
> make install
```

This will install NL-SAR's command line interface and dynamic library interface in `<NLSAR_PATH>`, NL-SAR's Matlab interface in `<MATLAB_PATH>/toolbox/nlsar/`, NL-SAR's Python module in `<PYTHON_PATH>` and NL-SAR's IDL interface in `<IDL_PATH>/lib/nlsar/`.

You will need to update your environment paths variables. Make sure you are placed in the NL-SAR's directory and type the followings:

```
> echo 'export PATH=<NLSAR_PATH>/bin:$PATH' >> $HOME/.bashrc
> echo' export LD_LIBRARY_PATH=<NLSAR_PATH>/lib:$LD_LIBRARY_PATH' >> $HOME/.bashrc
> source $HOME/.bashrc
```

2.4 Update Matlab environment

If you are using Matlab and you have specified a different prefix during configuration, you will need to update the Matlab environment as follows:

```
> echo 'export MATLABPATH=<MATLAB_PATH>/toolbox/nlsar/:$MATLABPATH' >> $HOME/.bashrc
> source $HOME/.bashrc
```

2.5 Update Python environment

If you are using Python and you have specified a different prefix during configuration, you will need to update the Python environment as follows:

```
> echo 'export PYTHONPATH=<PYTHON_PATH>:$PYTHONPATH' >> $HOME/.bashrc
> source $HOME/.bashrc
```

2.6 Update IDL environment

If you are using IDL and you have specified a different prefix during configuration, you will need to update the IDL environment as follows:

```
> echo "PREF_SET, 'IDL_PATH', '<PREFIX_IDL>/lib/nlsar/:<IDL_DEFAULT>', /COMMIT" | idl
> echo "PREF_SET, 'IDL_DLM_PATH', '<IDL_PATH>/dlm/:<IDL_DLM_DEFAULT>', /COMMIT" | idl
> source $HOME/.bashrc
```

3 Prerequisites

- From command line: works out of the box
- From Matlab: works out of the box
- From Python: add the following before doing anything

  ```
  import nlsartoolbox as nlsartb
  ```

- From IDL: works out of the box
- From C: add the following before everything

  ```
  #include <nlsartoolbox.h>
  ```

  Link to NL-SAR Toolbox with the option `-lnlsartoolbox`

4 Images formats

Supported formats:

- RAT formats [read and write],
- PolSARpro formats [read and write],
- XIMA formats [read all and write `cxf`].
- TIFF formats (include Sentinel-1) [read only and write mono-channel].
- ENVI formats [read only and write mono-channel].
- CEOS formats (include ALOS-PALSAR) [read only].

Note that NL-SAR deals only with images of intensity or of covariance matrices. Other inputs will not produce what you want. If you have amplitude or complex images, use the program `sarjoin` which build an intensity image or an image of covariance matrices from amplitude or complex images (see Section 5.4). Use `sarexplode` to recover each channel of the image of matrices stored in `RAT` or `PolSARpro` format, into mono-channel images in `XIMA`, `TIFF` or `ENVI` formats (see Section 5.5).

Files can be encoded in little or big endian. The defaults behavior of NL-SAR is to interpret binary data according to your own architecture. You can specify NL-SAR that the data uses the other endianness by adding '[SWAP]' at the end of the file name (see Section 5.6)

4.1 RAT formats

- NL-SAR can read RAT files of version 1 and 2, but it writes only RAT files in version 1.
- A RAT file is assumed to be an image of complex covariance matrices. A RAT file containing vectorial data will produce an error message. Only the arrays of the following types are implemented so far:
  - float (`var` = 4)
  - float complex (`var` = 6)
  - double complex (`var` = 9)

  Rat files with other types will produce an error message. Fell free to contact me to extend to other modalities.

4.2 PolSARpro formats

- NL-SAR can read Sinclar, Coherency and Complex matrices, but it writes only Coherency and Complex matrices. Fell free to contact me to extend to other modalities.
- A PolSARPRo data is a directory containing binary files (with extensions `.bin`) and a `config.txt` file, You can find more details about this format there: http://earth.eo.esa.int/polsarpro/Manuals/PolSARpro_DataFormat.pdf
- Note that NL-SAR can deal with a specific mode (`PolarType: pp0`) for reading 1-dimensional data.

4.3 XIMA formats

- An XIMA data is a binary file without header which comes with a file with same name and extension .dim. You can find more details about this format there: http://perso.telecom-paristech.fr/~nicolas/XIMA/index.html.
- NL-SAR can read all XIMA formats (NB: all formats haven't been tested, contact me if you have any troubles) but write only in `cxf` (complex float) from a 1-dimensional SAR image.

4.4 TIFF formats (experimental)

- NL-SAR can read and write mono dimensionnal images in TIFF formats with extensions `.tif` or `.tiff` (NB: all formats haven't been tested, contact me if you have any troubles). Written TIFF images will be stored in IEEE Complex Single Precision format.
- This will require you to install `libtiff`.

4.5 ENVI formats (experimental)

- NL-SAR can read and write mono dimensionnal images in ENVI formats with extensions `.hdr` (NB: all formats haven't been tested, contact me if you have any troubles). Written ENVI images will be stored in IEEE Complex Single Precision format.

4.6 CEOS formats (experimental)

- NL-SAR can read mono dimensionnal images in CEOS formats without extension (NB: all formats haven't been tested, contact me if you have any troubles) but cannot write CEOS files.

5 Interfaces and their basic commands

This section describes the different basic commands/functions that are available to manipulate SAR images in different languages: command line, Matlab, Python, IDL or in C.

5.1 Reading information

- From command line:

```
> sarinfo file.rat
dimensions:
       M = 512
       N = 256
       D = 3
```

- From Matlab:

```
> [M, N, D] = sarinfo('file.rat')
M =
     512
N =
     256
D =
       3
```

- From Python:

```
In [1]: M, N, D = nlsartb.sarinfo('file.rat')

In [2]: M, N, D
Out[2]: (512, 256, 3)
```

- From IDL:

```
> PRINT, sarinfo('file.rat')
          512         256          3
```

- From C:

```
sardata* sarimage;

// anything

if (!(sarimage = sardata_alloc()))
    // treat error
if (!(sarimage = sarread_header("file.rat", sarimage)))
    // treat error
printf("M=%d N=%d D=%d\n", sarimage->M, sarimage->N, sarimage->D);

// anything

sardata_free(sarimage);
```

5.2 Reading data

The following commands import a SAR image from disk to memory

- From Matlab:

```
> sarimage = sarread('file.rat');
```

  Look at the matrix dimensions:

```
> size(sarimage)
ans =
     3      3    256    512
```

- From Python:

```
In [1]: sarimage = nlsartb.sarread('file.rat')
```

Look at the matrix dimensions:

```
In [2]: shape(sarimage)
Out[2]: (512, 256, 3, 3)
```

- From IDL:

```
> sarimage = sarread('file.rat')
```

Look at the matrix dimensions:

```
> PRINT, size(sarimage, /DIMENSIONS)
       3           3         256         512
```

- From C:

```
sardata* sarimage;

// anything

if (!(sarimage = sardata_alloc()))
    // treat error
if (!(sarimage = sarread("file.rat", sarimage)))
    // treat error

// anything

sardata_free(sarimage);
```

Note that a command line version would be meaningless.

5.3 Writing data

The following commands export a SAR image from memory to disk.

- From Matlab:

```
> sarwrite(sarimage, 'newfile.rat');
```

- From Python:

```
In [1]: nlsartb.sarwrite(sarimage, 'newfile.rat')
Out[1]: True
```

- From IDL:

```
> sarwrite, sarimage, 'newfile.rat'
```

- From C:

```
sardata*    sarimage;

// anything

if (!(sarwrite(sarimage, "newfile.rat")))
    // treat error

// anything

sardata_free(sarimage);
```

Note that a command line version would be meaningless.

Remark: to export your image into TIFF, ENVI or XIMA formats, please use `sarexplode` (see Section 5.5).

5.4 Join

The following commands creates an intensity image or a covariance matrix from a list of amplitude images or a single look complex images:

- From command line:

```
> sarjoin file1 [file2 ... fileN] newfile
```

Note that it is the only command of NL-SAR which deals with amplitude or single look complex data as input. If you provide only one file in input, this function basically compute the intensity image from the amplitude or complex image.

5.5 Explode

The following commands creates D^2 complex images from an image of $N = D \times D$ covariance matrices

- From command line:

```
> sarexplode file newfile1 [newfile2 ... newfileN]
```

This command is the only command allowing you to convert/export an image of $D \times D$ complex covariance matrices stored in `PolSARPro` or `RAT` format into D^2 mono-dimensional images in `TIFF`, `ENVI` or `XIMA` formats.

Remark: in the following sequence of command:

```
> sarjoin in1.tiff in2.tiff tmp.rat
> sarexplode tmp.rat out11.tiff out12.tiff out21.tiff out22.tiff
```

the pixel values in the file `out11.tiff` will be the square modulus of the ones of `in1.tiff`, `out12.tiff` will be the product of `in1.tiff` and the complex conjugate of `in2.tiff`, `out21.tiff` will be the product of `in2.tiff` and the complex conjugate of `in1.tiff`, `out22.tiff` will be the square modulus of `in2.tiff`.

5.6 Conversion

The following example convert a RAT file to PolSARpro format:

- From command line

```
> sarconvert file.rat newfile
```

- From Matlab:

```
> sarimage = sarread('file.rat');
> sarwrite(sarimage, 'newfile');
```

- From Python:

```
In [1]: sarimage = nlsartb.sarread('file.rat')

In [2]: nlsartb.sarwrite(sarimage, 'newfile')
Out[2]: True
```

- From IDL:

```
> sarimage = sarread('file.rat')
> sarwrite, sarimage, 'newfile'
```

- From C:

```
sardata*    sarimage;

// anything

if (!(sarimage = sardata_alloc()))
    // treat error
if (!(sarimage = sarread("file.rat", sarimage)))
    // treat error
if (!(sarwrite(sarimage, "newfile")))
    // treat error

// anything

sardata_free(sarimage);
```

The following example convert a PolSARpro format to the same PolSARpro format except it switches the endianness of the output:

- From command line

```
> sarconvert file newfile[SWAP]
```

- Same principle from other interfaces.

Remark: to convert your image into `TIFF`, `ENVI` or `XIMA` formats, please use `sarexplode` (see Section 5.5).

5.7 Extraction

The following commands extract a subarea from position (x, y) to position $(x + width - 1, y + height - 1)$ with a decimation step:

- From command line

```
> sarextract file.rat newfile.rat x y width height step
```

- From Matlab:

```
> sarimage_new = sarimage(:, :, y + (1:step:height)), x + (1:step:width));
```

- From Python:

```
In [1]: sarimage_new = sarimage[x:x + width:step, y:y + height:step, :, :]
```

- From IDL:

```
> sarimage_new = sarimage[*, *, y:(y + height - 1):step, x:(x + width - 1):step]
```

- From C:

```
sardata*    sarimage;
sardata*    sarimage_new;
long int    xoffset, yoffset, width, height, step;

// anything

if (!(sarimage_new = sardata_alloc()))
    // treat error
if (!(sarimage_new = sardata_extract(sarimage, sarimage_new,
                                     x, y, width, height, step)))
    // treat error

// anything

sardata_free(sarimage_new);
```

5.8 RGB export

- From command line:

```
> sar2png file.rat rgbexport.png [alpha]
```

where `alpha` is an optional parameter to enhance contrast (default 3)

- From Matlab:

```
> rgbexport = sar2rgb(sarimage [, alpha]);
```

The argument `alpha` is the same as for the command line version.
Look at the matrix dimensions:

```
> size(sarimage)
ans =
     3      3    256    512
> size(rgbexport)
ans =
   512    256      3
```

The storing convention for the RGB image is reversed compared to our usual convention to ensure compatibility with the Matlab Image Toolbox.

- From Python:

```
In [1]: rgbexport = nlsartb.sar2rgb(sarimage [, alpha]);
```

The argument `alpha` is the same as for the command line version.
Look at the matrix dimensions:

```
In [2]: shape(sarimage)
Out[2]: (512, 256, 3, 3)
In [2]: shape(rgbexport)
Out[2]: (512, 256, 3)
```

- From IDL:

```
> rgbexport = sar2rgb(sarimage [, alpha])
```

The argument `alpha` is the same as for the command line version.
Look at the matrix dimensions:

```
> PRINT, SIZE(sarimage, /DIMENSIONS)
       3           3         256         512
> PRINT, SIZE(rgbexport, /DIMENSIONS)
       3         256         512
```

- From C:

```
sardata*    sarimage;
rgbdata*    rgbexport;

// anything

if (!(rgbexport = rgbdata_alloc()))
    // treat error
if (!(rgbexport = sar2rgb(sarimage, rgbexport, alpha, gamma)))
    // treat error

// anything

rgbdata_free(rgbexport);
```

5.9 Viewer

- From command line:

```
> sarshow file.rat [alpha]
```

The argument `alpha` is the same as for the RGB export.
The first time, you will probably have the following message:

```
Please set your environment variable NLSAR_VIEWER
```

You need to define an environment variable `NLSAR_VIEWER` pointing to your favorite image viewer. For instance for Linux, if you like Eye Of Gnome, type the following:

```
> echo 'export NLSAR_VIEWER="eog -n"' >> $HOME/.bashrc
> source $HOME/.bashrc
```

Or, if you prefer Konqueror

```
> echo 'export NLSAR_VIEWER=konqueror' >> $HOME/.bashrc
> source $HOME/.bashrc
```

For Mac OS X, you can use

```
> echo 'export NLSAR_VIEWER="open -W"' >> $HOME/.bashrc
> source $HOME/.bashrc
```

- From Matlab:

```
> sarshow(sarimage [, alpha]);
```

The argument `alpha` is the same as for the RGB export.

- From Python:

```
In [1]: nlsartb.sarshow(sarimage [, alpha]);
Out[1]: True
```

The argument `alpha` is the same as for the RGB export.

- From IDL:

```
> sarshow, sarimage [, alpha]
```

The argument `alpha` is the same as for the RGB export.

6 Interfaces for image filtering

6.1 Car filters

NL-SAR Toolbox implements the boxcar, diskcar and gausscar filters. Examples:

- From command line:

```
> sarboxcar file.rat newfile.rat [hWx hWy]
```

where `hWx` and `hWy` are the half-width and hal-height of the box (default 1).

- From Matlab:

```
> sarimage_new = sarboxcar(sarimage [, hWx, hWy]);
```

The arguments are the same as for the command line version.

- From Python:

```
In [1]: sarimage_new = nlsartb.sarboxcar(sarimage [, hWx, hWy]);
```

The arguments are the same as for the command line version.

- From IDL:

```
> sarimage_new = sarboxcar(sarimage [, hWx, hWy])
```

The arguments are the same as for the command line version.

- From C:

```
sardata*    sarimage;
sardata*    sarimage_new;
int         hWx, hWy;

// anything

if (!(sarimage_new = sardata_alloc()))
    // treat error
if (!(sarimage_new = sarboxcar(sarimage, sarimage_new, hWx, hWy)))
    // treat error

// anything

sardata_free(sarimage_new);
```

6.2 NL-SAR filter

NL-SAR Toolbox implements the NL-SAR filter.

simfunc	Name	Formula
'glr'	Generalized likelihood ratio	$-\log \frac{\lvert C_1 + C_2 \rvert^2}{\lvert C_1 \rvert \lvert C_2 \rvert} - 2\sqrt{2}D$
'kl'	Kullback-Leibler	$\mathrm{tr}(C_1^{-1}C_2 + C_2^{-1}C_1) - 2D$
'skl'	Square-root Kullback-Leibler	$\sqrt{\mathrm{tr}(C_1^{-1}C_2 + C_2^{-1}C_1) - 2D}$
'geo'	Geodesic	$\left\lVert \log(C_1^{-1/2} C_2 C_1^{-1/2}) \right\rVert_F^2 = \sum_{i=1}^{D} \log \lambda_i^2$

Table 1: Description of the implemented similarity criteria. C_1 and C_2 are two $D \times D$ compared covariance matrices and λ_i is the i-th eigenvalue of $C_1^{-1}C_2$. Whan comparing patches, these quantities are summed over the corresponding pixels within the patches.

Precomputation NL-SAR requires to learn its internal kernel function from statistics of homogeneous patches. This learning can be performed as follow:

- From command line:

```
> sarnlstats noise.rat noise.stats L [verbose hWmin hWmax hPmin hPmax simfunc]
```

  where `L` is the equivalent number of look of the input noisy image, `hWmin`/`hWmax` is the radius of the smallest/largest search window size (default 1/12) and `hPmin`/`hPmax` the half-width of the smallest/largest patches (default 0/5), `simfunc` is the name of the patch similarity criterion which can be 'glr', 'kl', 'skl' or 'geo' (default 'glr') see Table 1. If `verbose` = 1, steps and progressing bars are displayed on the standard output (default 1).

- From Matlab:

```
> stats = sarnlstats(sarnoise, L [, verbose, hWmin, hWmax, hPmin, hPmax, simfunc]);
```

  The arguments are the same as for the command line version. Your program should finish by `sarnlstats_free(stats)`. NB: If you look at the content of stats you'll see that it only contains a weird value. This is completely fine, it's just an handle to an internal memory block.

- From Python:

```
In [1]: stats = nlsartb.sarnlstats(sarnoise, L
                                   [, verbose, hWmin, hWmax, hPmin, hPmax])
```

  The arguments are the same as for the command line version. Your program should finish by `nlsartb.sarnlstats_free(stats)`.

- From IDL:

```
> stats = sarnlstats(sarnoise, L [, verbose, hWmin, hWmax, hPmin, hPmax])
```

  The arguments are the same as for the command line version. Your program should finish by `sarnlstats_free, stats)`.

- From C:

```
sardata*    sarnoise;
void*       stats;
int         L, verbose, hWmin, hWmax;

// anything

if (!(stats = sarnlstats(sarnoise, L, 3, verbose, hWmin, hWmax)))
    // treat error
```

```
// anything

sarsimstats_free(stats);
```

The arguments are the same as for the command line version, except the third one which is the number of optional arguments. In this examples only three optionals arguments are given.

Computation NL-SAR requires to learn its internal kernel function from statistics of homogeneous patches. This learning can be performed as follow:

- From command line:

```
> sarnlsar image.rat image_new.rat noise.stats [verbose rho h enl.pgm]
```

where `rho` is a filtering parameter acting on the bias reduction step (default 1.0) and `h` is a filtering parameter acting on the kernel function (default 1.0) If `verbose` = 1, steps and progressing bars are displayed on the standard output (default 1). If `enl.pgm` is provided, it creates an image of localy equivalent number of looks.

- From Matlab:

```
> [ sarimage_new, enl_map ] = sarnlsar(sarimage, stats [, verbose, rho, h]);
```

The arguments are the same as for the command line version.

- From Python:

```
In [1]: sarimage_new = nlsartb.sarnlsar(sarimage, stats [, verbose, rho, h])
```

The arguments are the same as for the command line version.

- From IDL:

```
> sarimage_new = sarnlsar(sarimage, stats [, verbose, rho, h])
```

The arguments are the same as for the command line version.

- From C:

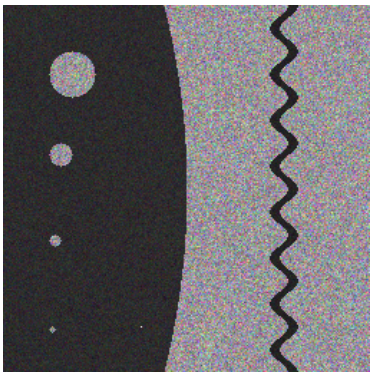
```
sardata     *sarimage, *sarimage_new;
void*       stats;
int         verbose;
float       rho = 1.;
float       h   = 1.;

// anything

if (!(sarimage_new = sardata_alloc()))
    // treat error
if (!(sarimage_new = sarnlsar(sarimage, sarimage_new, stats, 3, verbose, rho, h)))
    // treat error

// anything

sardata_free(sarimage_new);
```

The arguments are the same as for the command line version.

Example

- From command line:

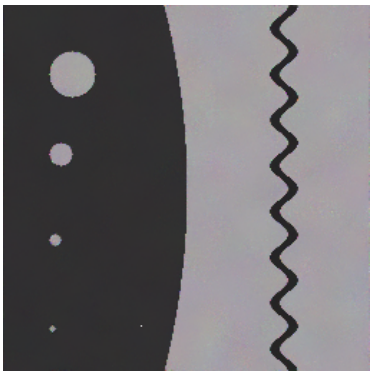
```
# Create a simulated polsar image with 3-looks Wishart speckle
> sarmire example.rat 256 256 3 3
> sarshow example.rat
```



```
# Create a homogeneous polsar image with same statistics
> sarnoise noise.rat 48 48 3 3
> sarshow noise.rat
```

```
# Run the precomputation
> sarnlstats noise.rat noise.stats 3
|==================================================| 100%
eta2=0.35 [theoretical=0.33] corr=0.06
=> #scales=3 step=1
s=1 hP= 1 mean=9.718527 std=1.616482
s=1 hP= 5 mean=9.718513 std=0.413537
s=2 hP= 1 mean=1.128147 std=0.214467
s=2 hP= 5 mean=1.128150 std=0.069138
s=3 hP= 1 mean=0.375986 std=0.109126
s=3 hP= 5 mean=0.375987 std=0.037603
```

```
# Run the non-local estimation
> sarnlsar example.rat result.rat noise.stats
Computation (#proc=4, #windows=4)
|==================================================| 100%
> sarshow result.rat
```



- From other interfaces: See provided examples in directory `examples`.

7 Frequently asked questions (FAQ)

How can I participate to the development of NL-SAR Toolbox? Send me an email!

Why does Matlab crash with the following message "Invalid MEX-file [...]: undefined symbol: fftwf_cleanup"? Matlab might use its own precompiled version of FFTW3. NL-SAR Toolbox uses the version usually placed in the directory `/usr/lib/`. In order to use the right version, run matlab from command line as follow (or something similar):

```
LD_PRELOAD=/usr/bin/libfftw3f.so:$LD_PRELOAD matlab
```

To avoid to type this long command line each time you run matlab, you can also create an alias by typing something like:

```
> echo 'alias matlab="LD_PRELOAD=/usr/bin/libfftw3f.so:$LD_PRELOAD matlab"'
                                                                    >> $HOME/.bashrc
> source $HOME/.bashrc
```

Why does Matlab crash with the following message "Invalid MEX-file '...sarnlsar.mexa64': ...libgfortran.so.3: version 'GFORTRAN_1.4' not found (required by /usr/lib/liblapack.so.3)"? Matlab might use its own precompiled version of the gnu Fortran library which is required by the Lapack library (linked wtih the NL-SAR Toolbox) wich uses the version usually placed in the directory `/usr/lib/`. One possible option is to disable the use of Lapack (wich would limit NL-SAR to be used only on covariance matrices of size 3 x 3 or smaller). For doing so, just reconfigure and recompile NL-SAR Toolbox using:

```
./configure --disable-lapack
make clean all
sudo make install
```

If you really need NL-SAR for covariance matrices higher than 3 x 3, you will need to indicate to Matlab to use the right version. This can be done by runnig matlab from command line as follow (or something similar):

```
LD_PRELOAD=/usr/lib/libgfortran.so.3:$LD_PRELOAD matlab
```

To avoid to type this long command line each time you run matlab, you can also create an alias by typing something like:

```
> echo 'alias matlab="LD_PRELOAD=/usr/lib/libgfortran.so.3:$LD_PRELOAD matlab"'
                                                                    >> $HOME/.bashrc
> source $HOME/.bashrc
```

Why `mex` compilation failed with message `ld: library not found -lgomp`? Matlab might not use `gcc` to compile `C` programs. To change this behavior, do the following

```
cp <PREFIX_MATLAB>/bin/mexopts.sh $HOME/.matlab/<version>/mexopts.sh
chmod 755 $HOME/.matlab/<version>/mexopts.sh
```

where you replace `<PREFIX_MATLAB>` by the directory where MATLAB is installed and `<version>` by the version identifier (something like `R2013a`). Then edit the file `$HOME/.matlab/<version>/mexopts.sh`. Locate the lines

```
CC='<some compiler>'
```

where `<some compiler>` might be for instance "`xcrun -sdk macosx10.7 clang`" and replace them by

```
CC='gcc'
```

Then rerun compilation.

I've just installed some dependencies (for instance `libfftw3f`) but when I run `./configure` it cannot find it, why? NL-SAR Toolbox uses the command `locate` to find your dependencies. You may need to update your locate database. On Linux run

```
sudo updatedb
```

On Mac OS X run

```
sudo /usr/libexec/locate.updatedb
```

The configuration step stops with: "Checking for idl ./configure: ligne 49 : cd: [...]/jacorb-2.3.1/bin/../external: No such file or directory [ok]" Another program than the Exelisvis IDL one is available on your system and called `idl`. Here, it corresponds to a program of the free Java implementation of the OMG's CORBA standard (JacORB). If you don't have the Exelisvis version installed or you don't want to use the IDL interface, a simple solution consists in disabling IDL explicitely before the configuration by running instead:

```
./configure --disable-idl
```

If both version of IDL are installed (the Exelisvis one and the JacORB one) and you want to use the IDL interface, you have to do some manipulations in your PATH environment variable to ensure that Exelisvis IDL starts when you type '`idl`' on command line.

The compilation step stops with: "mex: unrecognized option '-cxx' mex: unrecognized option '-largeArrayDims' This is pdfTeX, Version 3.14 [...] (Press Enter to retry, or Control-D to exit) Please type another input file name:" Another program than the Mathworks mex compiler is available on your system and called `mex`. Here, it corresponds to a program of the pdfTeX project. If you don't want to use the Matlab interface, a simple solution consists in disabling Matlab explicitely before the configuration by running instead:

```
./configure --disable-matlab
```

If you want to use the Matlab interface, you have to do some manipulations in your PATH environment variable to ensure that the Mathworks mex compiler starts when you type '`mex`' on command line.

Most of your examples are based on covariance matrices. Could you give an example of using the NL-SAR filter with an image of intensity? Intensity images are particular cases of 1×1 matrices. Hence, every examples with covariance matrices can be used identically with intensity images. File with RAT or XIMA formats can be used for intensity images. A modified PolSARpro format can be used for intensity image with a specific mode `PolarType: pp0`. If you are using another specific format, an alternative is to load your file from Matlab, Python, IDL or C in a single precision floating matrix that you give as an argument to NL-SAR functions. Recall that NL-SAR Toolbox assumes that the content of a scalar image corresponds to intensity values (not the amplitude). You can use `sarjoin` (see Section 5.4) to convert amplitude images to intensity images.

I've exported an intensity image in PolSARpro format from the Next ESA SAR Toolbox (NEST), but NL-SAR Toolbox cannot read it, why? When you export an intensity image from NEST to PolSARpro format, the directory contains a file `config.txt` and `Intensity_HH.bin`. Rename `Intensity_HH.bin` to `C11.bin` and change in the file `config.txt` the field `PolarType` from `pp1` to `pp0`. NL-SAR Toolbox will then read your file successfully, otherwise contact me.

I am trying to get the NLSAR filtering done with Sentinel-1 intensity image, exported from SNAP as PolSARpro format, and renamed to `C11.bin` as well as PolarType to pp0 within the `config.txt` file. I systematically get the error message: `Cannot open file C11.bin: Cannot open config.txt file: Not a directory`. Do you have any idea/suggestion to solve this issue? First of all, with the latest version of NL-SAR, you could have load directly the Sentinel-1 image in TIFF format, without converting it to PolSARPro format. Considering the PolSARPro image is stored as

```
directory/C11.bin
directory/config.txt
```

you should access the image as

```
sarinfo directory           # CORRECT
```

not as

```
sarinfo directory/C11.bin   # INCORRECT
```

I want to produce interferograms with NL-SAR toolbox but the input and output are images of covariance matrices. What should I do? From a pair of single look complex images (SLC), you can build an image of 2×2 covariance matrices with the command `sarjoin` (see Section 5.4). Then run NL-SAR on this image. After you can retrieve the interferometric phase or coherence (normalized correlation) from the produced image. For instance, you can do the following using Matlab:

```
> insar_phase     = squeeze(angle(sarimage(1,2,:,:)));
> insar_coherence = squeeze(abs(sarimage(1,2,:,:)) ./ ...
                        sqrt(abs(sarimage(1,1,:,:)) .* abs(sarimage(2,2,:,:))));
```

If you moreover assume the same reflectivity (radar cross-section) for each pair of corresponding pixels in both images, the coherence can be refined as

```
> insar_coherence = squeeze(2 * abs(sarimage(1,2,:,:)) ./ ...
                        (abs(sarimage(1,1,:,:)) + abs(sarimage(2,2,:,:))));
```

Look at the matrix dimensions:

```
> size(sarimage)
ans =
     2      2    256    512
> size(insar_phase)
ans =
   256    512
> size(insar_coherence)
ans =
   256    512
```

I know that the flat earth and orbital inaccuracies of the interferogram should be removed by a pre-processing step. If I do the interferometry with the command like 'sarjoin a.rat b.rat ab.rat', the output would contain an orbital component. If I do the interferometry and removed the orbital component in another sortware, the interferogram is not bi-dimensional. So what is your suggestion about how to remove the orbital component? You can build an image of 2×2 covariance matrices with removed flat earth and orbital inaccuracies. Using Matlab, you can proceed with the following instructions:

```
> insar_phase                 = squeeze(angle(sarimage(1,2,:,:)));
> insar_phase_corrected       = remove_orbital_component(insar_phase);
> sarimage_corrected          = sarimage;
> sarimage_corrected(1,2,:,:) = abs(sarimage_corrected(1,2,:,:)) .* ...
                      exp(i * reshape(insar_phase_corrected, size(sarimage(1,2,:,:))));
> sarimage_corrected(2,1,:,:) = conj(sarimage_corrected(1,2,:,:));
```

where `i = sqrt(-1)` and `remove_orbital_component` is a function using your favourite algorithm to remove the flat earth and orbital inaccuracies.

As you said in your paper, NLSAR is also effective for InSAR data. But I don't know that how to calculate the dissimilarity between two patches when involving interferometric phase. Should I just use the interferometric phase (real data)? Or should I use the real and imaginary parts to calculate the dissimilarity? Or should I use the complex interferometric phase? NLSAR requires at each pixel to access a 2x2 "covariance" matrix obtained from the pair (z_1, z_2) of your interferometric images as $C = \begin{pmatrix} |z_1|^2 & z_1 z_2^* \\ z_1^* z_2 & |z_2|^2 \end{pmatrix}$. This operation can be performed with the command `sarjoin z1.rat z2.rat C.rat`. Then, NLSAR (`sarsimstats` and `sarnlstats`) loads these matrices, performs some diagonal loading if required, and compares them using the generalized likelihood ratio (`glr`). Other criterion can be used see Section 6.2.

How can I process a dual-pol Sentinel-1 image using NLSAR Toolbox? You will have to create a covariance matrix representation of the dual pol image, and then process this image in that format. In command line:

```
# Create covariance matrices form dual-pol Sentinel files
sarjoin 's1a-iw1-slc-vv.tiff' 's1a-iw1-slc-vh.tiff' 's1a-iw1-slc.rat'

# Filter covariance matrices
sarnlsar 's1a-iw1-slc.rat' 's1a-iw1-slc.new.rat' noise.stats

# Optionally go back to TIFF format (extract the 4 channels of the covariance matrix)
sarexplode 's1a-iw1-slc.new.rat' 's1a-iw1-slc-vv-vv.new.tiff' 's1a-iw1-slc-vv-vh.new.tiff'
                                 's1a-iw1-slc-vh-vv.new.tiff' 's1a-iw1-slc-vh-vh.new.tiff'
```

From Matlab:

```
% Load dual pol Sentinel files VV and VH
vv = sarread('s1a-iw1-slc-vv.tiff');
vh = sarread('s1a-iw1-slc-vh.tiff');

% Create 2x2 covariance matrices
sarimage(1, 1, :, :) = vv.^2;
sarimage(1, 2, :, :) = vv .* conj(vh);
sarimage(2, 1, :, :) = vh .* conj(vv);
sarimage(2, 2, :, :) = vh.^2;

% Filter covariance matrices
sarimage_new = sarnlsar(sarimage, stats);
```

8 Not documented yet

- `sarnlsar_dispatch` for distributing NL-SAR on a cluster.
- `sarcat` for concatenating files.
- `sarmire` to generate a simulated SAR image with multi-scale structure.
- `sarnoise` to generate an homogeneous SAR image with noise only.