NLSAM Documentation

Release 0.6.1

Samuel St-Jean

Jul 17, 2020

CONTENTS

This is the documentation detailing the internal of the non local spatial and angular matching (NLSAM) denoising algorithm for diffusion MRI, which is available at https://github.com/samuelstjean/nlsam.

You can find the original paper and full details of the algorithm as presented in

```
St-Jean, S., Coupé, P., & Descoteaux, M. (2016)
"Non Local Spatial and Angular Matching :
Enabling higher spatial resolution diffusion MRI datasets through adaptive denoising."
Medical Image Analysis, 32(2016), 115-130.
```

Which you can grab a copy from the publisher website or from arxiv.

You can find below the documentation for each modules and a few instructions on topics such as noise estimation and detailed installation instructions.

CHAPTER

ONE

NLSAM

1.1 Submodules

1.1.1 nlsam.angular_tools

Module Contents

Functions

angular_neighbors(vec, n)	Returns the indices of the n closest neighbors (excluding the vector itself)
_angle(vec)	Inner function that finds the angle between all vectors of the input.

nlsam.angular_tools.**angular_neighbors**(*vec*, *n*)

Returns the indices of the n closest neighbors (excluding the vector itself) given an array of m points with x, y and z coordinates.

Input : A m x 3 array, with m being the number of points, one per line. Each column has x, y and z coordinates for each vector.

Output : A m x n array. Each line has the n indices of the closest n neighbors amongst the m input vectors.

Note : Symmetries are not considered here so a vector and its opposite sign counterpart will be considered far apart, even though in dMRI we consider (x, y, z) and -(x, y, z) to be practically identical.

nlsam.angular_tools.**_angle**(*vec*)

Inner function that finds the angle between all vectors of the input. The diagonal is the angle between each vector and itself, thus 0 everytime. It should not be called as is, since it serves mainly as a shortcut for other functions.

arccos(0) = pi/2, so b0s are always far from everyone in this formulation.

1.1.2 nlsam.bias_correction

Module Contents

Functions

stabilization(data, m_hat, sigma, N, mask=None, clip_eta=True, return_eta=False, n_cores=-1, verbose=False)	
multiprocess_stabilization(data, m_hat, mask, sigma, N, clip_eta)	Helper function for multiprocessing the stabilization part.
corrected_sigma(eta, sigma, N, mask=None)	
root_finder_sigma(data, sigma, N, mask=None)	Compute the local corrected standard deviation for the adaptive nonlocal

nlsam.bias_correction.**logger**

nlsam.bias_correction.**vec_fixed_point_finder**

nlsam.bias_correction.**vec_chi_to_gauss**

nlsam.bias_correction.**vec_xi**

nlsam.bias_correction.**vec_root_finder**

nlsam.bias_correction.**stabilization**(*data*, *m_hat*, *sigma*, *N*, *mask=None*, *clip_eta=True*, *return_eta=False*, *n_cores=- 1*, *verbose=False*)

nlsam.bias_correction.**multiprocess_stabilization**(*data*, *m_hat*, *mask*, *sigma*, *N*, *clip_eta*)

Helper function for multiprocessing the stabilization part.

nlsam.bias_correction.**corrected_sigma**(*eta*, *sigma*, *N*, *mask=None*)

nlsam.bias_correction.**root_finder_sigma**(*data*, *sigma*, *N*, *mask=None*)

Compute the local corrected standard deviation for the adaptive nonlocal means according to the correction factor xi.

data [ndarray] Signal intensity

sigma [ndarray] Noise magnitude standard deviation

N [ndarray or double] Number of coils of the acquisition (N=1 for Rician noise)

mask [ndarray, optional] Compute only the corrected sigma value inside the mask.

output, ndarray Corrected sigma value, where sigma_gaussian = sigma / sqrt(xi)

1.1.3 nlsam.denoiser

Module Contents

Functions

nlsam_denoise(data, sigma, bvals, bvecs, block_size, mask=None, is_symmetric=False, n_cores=-1, split_b0s=False, subsample=True, n_iter=10, b0_threshold=10, dtype=np.float64, verbose=False)	Main nlsam denoising function which sets up everything nicely for the local
local_denoise(data, block_size, overlap, variance, n_iter=10, mask=None, dtype=np.float64, n_cores=-1, verbose=False)	
greedy_set_finder(sets)	Returns a list of subsets that spans the input sets with a greedy algorithm
processer(data, mask, variance, block_size, overlap, param_alpha, param_D, dtype=np.float64, n_iter=10, gamma=3.0, tau=1.0, tolerance=1e-05)	

nlsam.denoiser.**logger**

nlsam.denoiser.**nlsam_denoise**(*data*, *sigma*, *bvals*, *bvecs*, *block_size*, *mask=None*, *is_symmetric=False*, *n_cores=-1*, *split_b0s=False*, *subsample=True*, *n_iter=10*, *b0_threshold=10*, *dtype=np.float64*, *verbose=False*)

Main nlsam denoising function which sets up everything nicely for the local block denoising.

data [ndarray] Input volume to denoise.

sigma [ndarray] Noise standard deviation estimation at each voxel. Converted to variance internally.

bvals [1D array] the N b-values associated to each of the N diffusion volume.

bvecs [N x 3 2D array] the N 3D vectors for each acquired diffusion gradients.

block_size [tuple, length = data.ndim] Patch size + number of angular neighbors to process at once as similar data.

mask [ndarray, default None] Restrict computations to voxels inside the mask to reduce runtime.

is_symmetric [bool, default False] If True, assumes that for each coordinate (x, y, z) in bvecs, (-x, -y, -z) was also acquired.

n_cores [int, default -1] Number of processes to use for the denoising. Default is to use all available cores.

split_b0s [bool, default False] If True and the dataset contains multiple b0s, a different b0 will be used for each run of the denoising. If False, the b0s are averaged and the average b0 is used instead.

subsample [bool, default True] If True, find the smallest subset of indices required to process each dwi at least once.

n_iter [int, default 10] Maximum number of iterations for the reweighted l1 solver.

b0_threshold [int, default 10] A b-value below b0_threshold will be considered as a b0 image.

dtype [np.float32 or np.float64, default np.float64] Precision to use for inner computations. Note that np.float32 should only be used for very, very large datasets (that is, your ram starts swappping) as it can lead to numerical precision errors.

verbose [bool, default False] print useful messages.

data_denoised [ndarray] The denoised dataset

nlsam.denoiser.**local_denoise**(*data*, *block_size*, *overlap*, *variance*, *n_iter=10*, *mask=None*, *dtype=np.float64*, *n_cores=-1*, *verbose=False*)

nlsam.denoiser.**greedy_set_finder**(*sets*)
Returns a list of subsets that spans the input sets with a greedy algorithm http://en.wikipedia.org/wiki/Set_cover_problem#Greedy_algorithm

nlsam.denoiser.**processer**(*data*, *mask*, *variance*, *block_size*, *overlap*, *param_alpha*, *param_D*, *dtype=np.float64*, *n_iter=10*, *gamma=3.0*, *tau=1.0*, *tolerance=1e-05*)

1.1.4 nlsam.smoothing

Module Contents

Functions

sh_smooth(data, bvals, bvecs, sh_order=4, similarity_threshold=50, regul=0.006)	Smooth the raw diffusion signal with spherical harmonics.
_local_standard_deviation(arr)	Standard deviation estimation from local patches.
local_standard_deviation(arr, n_cores=-1, verbose=False)	Standard deviation estimation from local patches.

nlsam.smoothing.**logger**

nlsam.smoothing.**sh_smooth**(*data*, *bvals*, *bvecs*, *sh_order=4*, *similarity_threshold=50*, *regul=0.006*)
Smooth the raw diffusion signal with spherical harmonics. data : ndarray

The diffusion data to smooth.

gtab [gradient table object] Corresponding gradients table object to data.

sh_order [int, default 8] Order of the spherical harmonics to fit.

similarity_threshold [int, default 50] All b-values such that **|b_1 - b_2|** < similarity_threshold will be considered as identical for smoothing purpose. Must be lower than 200.

regul [float, default 0.006] Amount of regularization to apply to sh coefficients computation.

pred_sig [ndarray] The smoothed diffusion data, fitted through spherical harmonics.

nlsam.smoothing.**_local_standard_deviation**(*arr*)
Standard deviation estimation from local patches.

Estimates the local variance on patches by using convolutions to estimate the mean. This is the multiprocessed function.

arr [3D or 4D ndarray] The array to be estimated

sigma [ndarray] Map of standard deviation of the noise.

nlsam.smoothing.**local_standard_deviation**(*arr*, *n_cores=- 1*, *verbose=False*)
Standard deviation estimation from local patches.

The noise field is estimated by subtracting the data from it's low pass filtered version, from which we then compute the variance on a local neighborhood basis.

arr [3D or 4D ndarray] The array to be estimated

n_cores [int] Number of cores to use for multiprocessing, default : all of them

verbose: int If True, prints progress information. A higher number prints more often

sigma [ndarray] Map of standard deviation of the noise.

Advanced techniques for estimating degrees of freedom in a non central chi distribution

This section is mostly personal recommendations based on some literature, stuff I have played with and stuff I have seen in MR physics classes. It is probably not exhaustive nor perfectly accurate, but should give the interested reader a feeling of what is happening and why.

The problem

Noise estimation in MR highly depends on the reconstruction algorithm implemented by your vendor (*Dietrich et al.*). Unfortunately, due to interference between adjacent receiver coils as used in modern parallel imaging (i.e. pretty much always unless you are doing fancy specialized acquisitions), the real noise distribution is slightly different than a pure Rician or Noncentral chi distribution (*Aja-Fernandez et al., Constantinides et al.*).

It is still possible to estimate the distribution, but the values of the 'standard deviation' and degrees of freedom of that distribution depends on the parameters of the acquisition (i.e. SENSE maps, GRAPPA weights), which are probably hard to acquire if you do not have a friendly MR physicist at hand (they are always nice guys anyway, so don't be afraid to ask for help). The authors of (*Aja-Fernandez et al.*) still offer a way to do a blind estimation of these values for those interested to dig a bit more.

Why it is hard to find a surefire correction method

Also based on my MR physics class understanding, due to the way the (closed source) algorithm in each vendor's scanner software work, they are likely to discard the signal coming from far away receiver elements from the imaged body region. As an example, near the top of the head, the coil elements placed near the neck are very likely to measure little relevant signal and mostly contribute noise.

This means that during the k-space reconstruction, the signal contribution from these coils will get thrown away, and thus the number of effectively used coils will vary per region and will be lower than the number of coils on you receiver array. Add noise correlation into that due to electrical interference and proximity of your receiver elements, and you are looking at some (hard to figure out) distribution which is different from what you expect according to the theory.

What other people suggest

The authors of (*Veraart et al.*) also provided another way to estimate those relevant parameters based on constructing synthetic noise maps for those interested in trying another method.

As a final tl;dr advice, some other studies have found that for GRAPPA reconstruction, with a 12 channels head coils N=4 (*Brion et al.*) (that's what we use in Sherbrooke also for the 1.5T Siemens scanner with a 12 channels head coil) works well and for a 32 channels head coils (*Varadarajan et al.*), a value around N=9 seems to work (remember that N varies spatially, but it seems to be fairly homogeneous/vary slowly).

The authors of (*Becker et al.*) also indicates that in the worst case, using N=1 for Rician noise is better than doing nothing.

Same observation from (*Sakaie et al.*) in real data; they fitted the background distribution and found out that for a sum of square (SoS) reconstruction with 12 coils, N = 3.76 $\pm$ 0.07 in 5 subjects (well, it should be an integer since it represents the number of degrees of freedom, so let's say N=4). As expected, it is much lower than 12 because of the correlation in each adjacent coils and produces DTI metrics (FA, MD, RD, AD) with a stronger bias than an adaptive combine (N = 1.03 $\pm$ 0.01) reconstruction.

Take home message

In all cases, the take home message would be that estimating the real value for N is still challenging, but it will most likely be lower than the number of coils present on your receiver coil due to the way MRI scanners reconstruct and combine images.

From my personal recommendations (well, don't quote me too much on it though), a good rule of thumb would be for 12 coils -> N=4, for 32 coils -> N=8 and for 64 coils (which are separated as a 24 coils for the upper part and 40 coils for the lower part), I would try out N=6 as dictated by the upper part, N=4 if it does not produce satisfactory results.

References

Aja-Fernandez, S., Vegas-Sanchez-Ferrero, G., Tristan-Vega, A., 2014. Noise estimation in parallel MRI: GRAPPA and SENSE. Magnetic resonance imaging

Becker, S. M. A., Tabelow, K., Mohammadi, S., Weiskopf, N., & Polzehl, J. (2014). Adaptive smoothing of multi-shell diffusion weighted magnetic resonance data by msPOAS. NeuroImage

Brion, V., Poupon, C., Riff, O., Aja-Fernández, S., Tristán-Vega, A., Mangin, J.-F., Le Bihan D, Poupon, F. (2013). Noise correction for HARDI and HYDI data obtained with multi-channel coils and sum of squares reconstruction: an anisotropic extension of the LMMSE. Magnetic Resonance Imaging

Constantinides, C. D., Atalar, E., & McVeigh, E. R. (1997). Signal-to-Noise Measurements in Magnitude Images from NMR Phased Arrays. Magnetic Resonance in Medicine, 38(5), 852–857.

Dietrich, O., Raya, J. G., Reeder, S. B., Ingrisch, M., Reiser, M. F., & Schoenberg, S. O. (2008). Influence of multichannel combination, parallel imaging and other reconstruction techniques on MRI noise characteristics. Magnetic Resonance Imaging

Sakaie, M. & Lowe, M., Retrospective correction of bias in diffusion tensor imaging arising from coil combination mode, Magnetic Resonance Imaging, Volume 37, April 2017

Varadarajan, D., & Haldar, J. (2015). A Majorize-Minimize Framework for Rician and Non-Central Chi MR Images. IEEE Transactions on Medical Imaging

Veraart, J., Rajan, J., Peeters, R. R., Leemans, A., Sunaert, S., & Sijbers, J. (2013). Comprehensive framework for accurate diffusion MRI parameter estimation. Magnetic Resonance in Medicine

Installation for Linux

For a linux installation, you should find everything you need in your package manager. These are the gcc compilers, python headers and a blas/lapack implementation such as atlas/openblas intel mkl/etc.

Debian/Ubuntu and derivatives

```shell sudo apt-get install build-essential python-dev libopenblas-dev libopenblas-base liblapack-dev ```

Red hat/Cent OS/Fedora and derivatives

```shell sudo yum install python-devel atlas-devel blas-devel lapack-devel gcc gcc-c++ gcc-gfortran ```

The gsl lib is now included as a precompiled static library, so no need to install it anymore.

Of course feel free to use your favorite blas/lapack implementation (such as intel MKL), but I got 5x faster runtimes out of openblas vs atlas for NLSAM just by switching libraries.

Installing NLSAM

For the python dependencies themselves, I recommend a fresh pip install since versions from the repositories tend to get old quite fast. You will need numpy, scipy, cython, nibabel, dipy and spams.

Get a [release archive](https://github.com/samuelstjean/nlsam/releases) and install it directly from the downloaded file

```shell pip install file_you_just_downloaded.tar.gz --user ```

and it should grab all the required dependencies if they are not already installed. If you encounter some errors (e.g. spams needs numpy and blas/lapack headers), install the missing package with pip first and continue the installation afterwards.
```

Now you can run the main script from your terminal, be sure to have a look at the [example](https://github.com/samuelstjean/nlsam/tree/master/example) for more information about the usage.

`shell nlsam_denoising --help `

You may also build and install the package from a local git clone instead of installing stuff with

`shell pip install -e . `

After updating your local git copy, you can rebuild the cython files by running

`shell python setup.py build_ext --inplace `

Installation for Mac OSX

You have two options

- Easy way - Use the binary release for Mac, no installation needed and perfect for trying it out quickly on your data, see https://github.com/samuelstjean/nlsam/releases
- Build from source with the instructions below. These steps will walk you through getting [Anaconda](https://www.continuum.io/downloads#_macosx) and a few other things, but will let you easily use a non-released version and other fancy features.

Python prerequisite

- While some Mac Os versions come with their own python, it seems [discouraged to use it](https://github.com/MacPython/wiki/wiki/Which-Python) and people suggest to use another distribution. For example, [Anaconda](https://www.continuum.io/downloads#_macosx) will get you an easy way to install it.
- You will also need a compiler (which might already be installed). If not, people suggest getting [XCode](https://developer.apple.com/xcode/download/) from the app store.

Spams and openmp support

A. The lazy way, using conda forge channel

The old installation instruction would have you install a non openmp version, but instead it is much simpler to use an already built one after all. This can be done by adding the conda forge channel, which has a working openmp build of spams.

~~~bash conda config –add channels conda-forge conda install python-spams ~~~

Of course, while it is easy, it also has a downside. Adding the conda forge channel will also replace all of your existing conda package with their own version which is using openblas instead of apple veclib. This might not be a problem in general, but you might need to rebuild/compile other python packages installed prior to adding this channel.

B. The slower way, building spams *without* openmp support

As the apple version of clang does not support openmp, we would need to install another compiler. The old instructions would have you do *brew install llvm* and a few compiler paths shenanigans, but it is much easier to use Anaconda and gcc. It is also possible to install spams without openmp, but some parts will not be multithreaded in that case, making the overall runtime of the algorithm longer.

Building spams with openmp deactivated can be done in one line using

~~~bash pip install https://github.com/samuelstjean/spams-python/releases/download/0.1/spams-2.6.zip ~~~

which downloads an archive hosted by yours truly. You can also go grab a newer version on the original authors [website](http://spams-devel.gforge.inria.fr/downloads.html) if needed/available or if you want to build it with openmp support.

Installing NLSAM

For the python dependencies themselves, I recommend a fresh pip install since versions from the repositories tend to get old quite fast. You will need numpy, scipy, cython, nibabel, dipy and spams.

Get a [release archive](https://github.com/samuelstjean/nlsam/releases) and install it directly from the downloaded file

`shell pip install file_you_just_downloaded.tar.gz --user `

and it should grab all the required dependencies if they are not already installed. If you encounter some errors (e.g. spams needs numpy and blas/lapack headers), install the missing package with pip first and continue the installation afterwards.

Now you can run the main script from your terminal, be sure to have a look at the [example](https://github.com/samuelstjean/nlsam/tree/master/example) for more information about the usage.

`shell nlsam_denoising --help `

You may also build and install the package from a local git clone instead of installing stuff with

`shell pip install -e . `

After updating your local git copy, you can rebuild the cython files by running

`shell python setup.py build_ext --inplace `

Installation for Windows

Easy way - grab a binary release

- To simply run the algorithm, the easiest way is definitely to download the windows binary from the [release](https://github.com/samuelstjean/nlsam/releases) section as you won't need to install python or anything else.

Just unzip the archive and start the program from a command line prompt, that's it!

- If you want to try out a precompiled dev version from master/another branch, you can find automatic builds [here](https://ci.appveyor.com/project/samuelstjean/nlsam/build/artifacts).

Unless you need a fancy feature which is not yet released, I would stick to the released version for simplicity as the automatic builds are subject to frequent changes.

- If you would like to study/modify the code, you will need a python distribution and a compiler as outlined below.

Installing Visual C++ compiler for python 2.7

I suggest using python 2.7 as the installation is much easier on windows. You can install a lightweight version of [Visual C++ Compiler for python 2.7](https://www.microsoft.com/en-us/download/details.aspx?id=44266).

If you would like to install python 3, you will need to install the full visual studio appropriate for your version of python as explained [here](https://wiki.python.org/moin/WindowsCompilers). As the whole thing is at least around 20 GB, I would strongly recommend to stick with the easy python 2.7 version for now.

Installing python and dependencies

You will need to get a python distribution and some other stuff. For starters, grabbing a complete distribution such as [Anaconda](https://www.continuum.io/downloads#_windows) is the easiest way to go as it comes with all the usual scientific packages.

From an anaconda terminal, we can also install a prebuilt version of spams for windows 2.7

`shell pip install https://github.com/samuelstjean/spams-python/releases/download/0.1/spams-2.4-cp27-none-win_amd64.whl `

Installing NLSAM

For the python dependencies themselves, I recommend a fresh pip install since versions from the repositories tend to get old quite fast. You will need numpy, scipy, cython, nibabel, dipy and spams.

Get a [release archive](https://github.com/samuelstjean/nlsam/releases) and install it directly from the downloaded file

`shell pip install file_you_just_downloaded.tar.gz --user `

and it should grab all the required dependencies if they are not already installed. If you encounter some errors (e.g. spams needs numpy and blas/lapack headers), install the missing package with pip first and continue the installation afterwards.

Now you can run the main script from your terminal, be sure to have a look at the [example](https://github.com/samuelstjean/nlsam/tree/master/example) for more information about the usage.

```shell
nlsam_denoising --help
```

You may also build and install the package from a local git clone instead of installing stuff with

```shell
pip install -e .
```

After updating your local git copy, you can rebuild the cython files by running

```shell
python setup.py build_ext --inplace
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX

Symbols

A

C

G

L

M

N

P

R

S

V