rapidtide Documentation

Release 2.2.6

Blaise Frederick

Jun 25, 2022

INTRODUCTION:

Rapidtide is a suite of python programs used to perform rapid time delay analysis on functional imaging data to find time lagged correlations between the voxelwise time series and other time series, both in the LFO band (rapditide2) and now in the cardiac band (happy).

CHAPTER

ONE

CITING RAPIDTIDE

Frederick, B, rapidtide [Computer Software] (2016-2022). Available from https://github.com/bbfrederick/rapidtide. doi:10.5281/zenodo.814990

CHAPTER

TWO

CONTENTS

2.1 The rapidtide package

Rapidtide is a suite of Python programs used to model, characterize, visualize, and remove time varying, physiological blood signals from fMRI and fNIRS datasets. The primary workhorses of the package are the rapidtide program, which characterizes bulk blood flow, and happy, which focusses on the cardiac band.

Full documentation is at: http://rapidtide.readthedocs.io/en/latest/

2.1.1 The rapidtide program

Rapidtide is also the name of the first program in the package, which is used to perform rapid time delay analysis on functional imaging data to find time lagged correlations between the voxelwise time series and other time series, primarily in the LFO band.

Why do I want to know about time lagged correlations?

This comes out of work by our group (The Opto-Magnetic group at McLean Hospital - http://www.nirs-fmri.net) looking at the correlations between neuroimaging data (fMRI) and NIRS data recorded simultaneously, either in the brain or the periphery. We found that a large fraction of the "noise" we found at low frequency in fMRI data was due to real, random[*] fluctuations of blood oxygenation and volume (both of which affect the intensity of BOLD fMRI images) in the blood passing through the brain. More interestingly, because these characteristics of blood move with the blood itself, this gives you a way to determine blood arrival time at any location in the brain. This is interesting in and of itself, but also, this gives you a method for optimally modelling (and removing) in band physiological noise from fMRI data (see references below).

After working with this for several years we've also found that you don't need to used simultaneous NIRS to find this blood borne signal - you can get it from blood rich BOLD voxels for example in the superior sagittal sinus, or bootstrap it out of the global mean signal in the BOLD data. You can also track exogenously applied waveforms, such as hypercarbic and/or hyperoxic gas challenges to really boost your signal to noise. So there are lots of times when you might want to do this type of correlation analysis.

As an aside, some of these tools are just generally useful for looking at correlations between timecourses from other sources – for example doing PPI, or even some seed based analyses.

[*] "random" in this context means "determined by something we don't have any information about" - maybe EtCO2 variation, or sympathetic nervous system activity - so not really random.

Correlation analysis is easy - why use this package?

The simple answer is "correlation analysis is easy, but using a prewritten package that handles file I/O, filtering, resampling, windowing, and the rest for you is even easier". A slightly more complex answer is that while correlation analysis is pretty easy to do, it's hard to do right; there are lots and lots of ways to do it incorrectly. Fortunately, I've made most of those mistakes for you over the last 8 years, and corrected my code accordingly. So rather than repeat my boring mistakes, why not make new, interesting mistakes? Explore your own, unique chunk of wrongspace…

2.1.2 Happy

More recently, inspired by Henning Voss' paper on hypersampling of cardiac signals in fMRI, we developed a method to extract and clean cardiac waveforms from fMRI data, even when the fMRI TR is far too long to properly sample cardiac waveforms. This cardiac waveform can then be to track the pulsatile cardiac pressure wave through the brain in somewhat the same way that we track the LFO signals. Among other things, this allows you to get cardiac waveforms during scans even when either 1) you didn't use a plethysmograph, or 2) you did, but the recording was of poor quality, which happens more than you might think.

What does "happy" have to do with any of this?

As to why happy is part of rapidtide, that's partially for practical reasons (the libraries in rapidtide have an awful lot of code that is reused in happy), and partially thematically - rapidtide has evolved from a "let's look at low frequency signals in fMRI data" package to a "let's look at everything in fMRI data EXCEPT neuronal activation", so happy fits right in.

2.2 Why are you releasing this package?

For a number of reasons.

- I want people to use it! I think if it were easier for people to do time delay analysis, they'd be more likely to do it. I don't have enough time or people in my group to do every experiment that I think would be interesting, so I'm hoping other people will, so I can read their papers and learn interesting things.
- It's the right way to do science – I can say lots of things, but if nobody can replicate my results, nobody will believe it (we've gotten that a lot, because some of the implications of what we've seen in resting state data can be a little uncomfortable). We've reached a stage in fMRI where getting from data to results involves a huge amount of processing, so part of confirming results involves being able to see how the data were processed. If you had to do everything from scratch, you'd never even try to confirm anybody's results.
- In any complicated processing scheme, it's quite possible (or in my case, likely) to make dumb mistakes, either coding errors or conceptual errors, and I almost certainly have made some (although hopefully the worst ones have been dealt with at this point). More users and more eyes on the code make it more likely that they will be found. As much as I'm queasy about somebody potentially finding a mistake in my code, I'd rather that they did so, so I can fix it[‡].
- It's giving back to the community. I benefit from the generosity of a lot of authors who have made the open source tools I use for work and play, so I figure I can pony up too.

[‡] or better yet, you, empowered user, can fix it, and push back a fix that benefits everybody…

2.3 Stability, etc.

This is an evolving code base. I'm constantly tinkering with it. That said, now that I've sent this off into to the world, I'm being somewhat more responsible about locking down stable release points. In between releases, however, I'll be messing with things, although for the most part this will be restricted to the dev branch. **It's very possible that at any given time the dev branch will be very broken, so stay away from it unless you have a good reason to be using it.** I've finally become a little more modern and started adding automated testing, so as time goes by hopefully the "in between" releases will be somewhat more reliable. That said, my tests routinely fail, even when things actually work. Probably should deal with that. Check back often for exciting new features and bug fixes!

2.4 Python version compatibility

I switched over a while ago to using Python 3 as my daily driver, so I know that everything works there. However, I know that a lot of people can't or won't switch from Python 2x, so I kept Python 2.7 compatibility for quite some time.

That said, the writing is on the wall, and since I depend on a number of packages that have dropped Python 2.x support, as of 2.0, so has rapidtide. However, as of version 1.9.0 I'm also releasing the code in a docker container (frederick-lab/rapidtide), which has everything nicely installed in a fully configured Python 3 environment, so there's really no need for me continue 2.x support. So now it's f-strings all the way, kids!

2.5 Ok, I'm sold. What's in here?

- **rapidtide** - This is the heart of the package - this is the workhorse program that will determine the time lagged correlations between all the voxels in a NIFTI file and a temporal "probe" regressor (which can come from a number of places, including the data itself) - it rapidly determines time delays… There are a truly bewildering array of options, and just about everything can be adjusted, however I've tried to pick a good set of default options for the most basic processing to get you going. At a minimum, it requires a 4D NIFTI file as input, and a root name for all of the output files. It generates a number of 3D NIFTI file maps of various parameters (lag time of maximum correlation, maximum correlation value, a mask of which voxels have valid fits, etc.) and some text files with useful information (significance thresholds, processing timing information, a list of values of configurable options).
- **happy** - This is a companion to rapidtide that focusses on cardiac signals. happy does three things - it attempts to determine the cardiac waveform over the time course of an fMRI dataset using slice selective averaging of fully unprocessed fMRI data. It also cleans up this initial estimate using a deep learning filter to infer what the simultaneously recorded plethysmogram would be. Finally, it uses either the derived or a supplied plethysmogram signal to construct a cardiac pulsation map over a single cycle of the cardiac waveform, a la Voss.
- **showxcorrx** - Like rapidtide, but for single time courses. Takes two text files as input, calculates and displays the time lagged cross correlation between them, fits the maximum time lag, and estimates the significance of the correlation. It has a range of filtering, windowing, and correlation options.
- **rapidtide2x_legacy**, **happy_legacy**, **showxcorr_legacy** - The older versions of the similarly named programs. These use the old calling conventions, for compatibility with older workflows. These will go away eventually, and they don't really get updates or bugfixes, so if you're using them, change to the new ones, and if you're not using them, don't.
- **rapidtide2std** - This is a utility for registering rapidtide output maps to standard coordinates. It's usually much faster to run rapidtide in native space then transform afterwards to MNI152 space. NB: this will only work if you have a working FSL installation.
- **happy2std** - Guess.

- **showtc** - A very simple command line utility that takes timecourses from text files and plots the data in it in a matplotlib window. That's it. A good tool for quickly seeing what's in a file. Has a number of options to make the plot prettier.
- **showxy** - Another simple command line utility that displays the the data contained in text files containing whitespace separated x-y pairs.
- **showhist** - Another simple command line utility that displays the histograms generated by rapidtide.
- **resamp1tc** - takes an input text file at some sample rate and outputs a text file resampled to the specified sample rate.
- **resamplenifti** - takes an input nifti file at some TR and outputs a nifti file resampled to the specified TR.
- **tidepool** - This is a GUI tool for displaying all of the various maps and timecourses generated by rapidtide in one place, overlaid on an anatomic image. This makes it a bit easier to see how all the maps are related to one another, how the probe regressor evolves over the run, and the effect of the filtering parameters. To use it, launch tidepool from the command line, and then select a lag time map - tidepool will figure out the root name and pull in all of the other associated data. Works in native or standard space.

2.6 Financial Support

This code base is being developed and supported by a grant from the US NIH (1R01 NS097512).

2.7 Additional packages used

Rapidtide would not be possible without many additional open source packages. These include:

2.7.1 numpy:

1) Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011) | https://doi.org/10.1109/MCSE.2011.37

2.7.2 scipy:

1) Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17, 261–272 (2020) | https://doi.org/10.1038/s41592-019-0686-2

2.7.3 matplotlib:

1) John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007) | https://doi.org/10.1109/MCSE.2007.55

2.7.4 nibabel:

1) https://github.com/nipy/nibabel | https://doi.org/10.5281/zenodo.591597

2.7.5 scikit-learn:

1) Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 2011. 12: p. 2825-2830. | https://scikit-learn.org

2.7.6 pandas:

1) McKinney, W., pandas: a foundational Python library for data analysis and statistics. Python for High Performance and Scientific Computing, 2011. 14.

2.7.7 nilearn:

1) https://github.com/nilearn/nilearn

2.8 References

Links to PDFs of all papers mentioned here can be found on the OMG website: https://www.nirs-fmri.net/home/publications

2.8.1 General overview of systemic low frequency oscillations in fMRI data

1) Tong Y, Hocke LM, Frederick BB. (2019) Low Frequency Systemic Hemodynamic "Noise" in Resting State BOLD fMRI: Characteristics, Causes, Implications, Mitigation Strategies, and Applications. Front. Neurosci., 14 August 2019 | https://doi.org/10.3389/fnins.2019.00787

2.8.2 Multimodal Cerebral Circulation Imaging

1) Tong Y, Frederick BD. (2010) Time lag dependent multimodal processing of concurrent fMRI and near-infrared spectroscopy (NIRS) data suggests a global circulatory origin for low-frequency oscillation signals in human brain. Neuroimage, 53(2), 553-64.

2) Tong Y, Hocke L, Frederick BD. (2011) Isolating the sources of widespread physiological fluctuations in fNIRS signals. J Biomed Opt. 16(10), 106005.

3) Tong Y, Bergethon PR, Frederick BD. (2011c) An improved method for mapping cerebrovascular reserve using concurrent fMRI and near-infrared spectroscopy with Regressor Interpolation at Progressive Time Delays (RIPTiDe). Neuroimage, 56(4), 2047-2057.

4) Tong Y, Frederick BD. (2012) Concurrent fNIRS and fMRI processing allows independent visualization of the propagation of pressure waves and bulk blood flow in the cerebral vasculature. Neuroimage, Jul 16;61(4): 1419-27.

5) Tong Y, Hocke LM, Licata SC, Frederick BD. (2012) Low frequency oscillations measured in the periphery with near infrared spectroscopy (NIRS) are strongly correlated with blood oxygen level-dependent functional magnetic resonance imaging (BOLD fMRI) signals. J Biomed Opt, 2012;17(10):106004. doi: 10.1117/1.JBO.17.10.106004. PubMed PMID: 23224003; PMCID: 3461094.

6) Tong Y, Hocke LM, Frederick BD. (2013) Short repetition time multiband EPI with simultaneous pulse recording allows dynamic imaging of the cardiac pulsation signal. Magn Reson Med 2014;72(5):1268-76. Epub Nov 22, 2013. doi: 10.1002/mrm.25041. PubMed PMID: 24272768.

7) Tong Y, Frederick B. (2014) Studying the Spatial Distribution of Physiological Effects on BOLD Signals using Ultrafast fMRI. Front Hum Neurosci 2014;5(196). doi: doi: 10.3389/fnhum.2014.00196.

8) Tong Y, Frederick B. (2014) Tracking cerebral blood flow in BOLD fMRI using recursively generated regressors. Hum Brain Mapp. 2014;35(11):5471-85. doi: 10.1002/hbm.22564. PubMed PMID: 24954380; PMCID: PMC4206590.

9) Donahue M, Strother M, Lindsey K, Hocke L, Tong Y, Frederick B. (2015) Time delay processing of hypercapnic fMRI allows quantitative parameterization of cerebrovascular reactivity and blood flow delays. Journal of Cerebral Blood Flow & Metabolism. 2015. PubMed PMID: 26661192. Epub October 19, 2015. doi: 10.1177/0271678X15608643.

10) Hocke L, Cayetano K, Tong Y, Frederick B. (2015) An optimized multimodal fMRI/NIRS probe for ultra-high resolution mapping. Neurophotonics. 2(4), 045004 (Oct-Dec 2015). doi: 10.1117/1.NPh.2.4.0450004.

11) Tong Y, Hocke LM, Fan X, Janes AC, Frederick B (2015). Can apparent resting state connectivity arise from systemic fluctuations? Frontiers in human neuroscience. 2015;9. doi: 10.3389/fnhum.2015.00285.

12) Tong Y, Lindsey KP, Hocke LM, Vitaliano G, Mintzopoulos D, Frederick B. (2016) Perfusion information extracted from resting state functional magnetic resonance imaging. Journal of cerebral blood flow and metabolism : official journal of the International Society of Cerebral Blood Flow and Metabolism. 2016. doi: 10.1177/0271678X16631755. PubMed PMID: 26873885.

2.8.3 Cardiac waveform extraction and refinement

1) Aslan S, Hocke L, Schwarz N, Frederick B. (2019) Extraction of the cardiac waveform from simultaneous multislice fMRI data using slice sorted averaging and a deep learning reconstruction filter. NeuroImage 198, 303–316 (2019).

2.8.4 Physiological noise identification and removal using time delay methods

1) Tong Y, Lindsey KP, Frederick BD. (2011b) Partitioning of physiological noise signals in the brain with concurrent near-infrared spectroscopy (NIRS) and fMRI. J Cereb Blood Flow Metab. 31(12), 2352-62.

2) Frederick BD, Nickerson LD, Tong Y. (2012) Physiological denoising of BOLD fMRI data using Regressor Interpolation at Progressive Time Delays (RIPTiDe) processing of concurrent fMRI and near-infrared spectroscopy (NIRS). Neuroimage, Apr 15;60(3): 1419-27.

3) Tong Y, Hocke LM, Nickerson LD, Licata SC, Lindsey KP, Frederick BB (2013) Evaluating the effects of systemic low frequency oscillations measured in the periphery on the independent component analysis results of resting state networks. NeuroImage. 2013;76C:202-15. doi: 10.1016/j.neuroimage.2013.03.019. PubMed PMID: 23523805; PMCID: PMC3652630.

4) Hocke LM, Tong Y, Lindsey KP, Frederick BB (2016). Comparison of peripheral near-infrared spectroscopy low-frequency oscillations to other denoising methods in resting state functional MRI with ultrahigh temporal

resolution. Magnetic resonance in medicine : official journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 2016. | http://dx.doi.org/10.1002/mrm.26038. PubMed PMID: 26854203.

5) Erdoğan S, Tong Y, Hocke L, Lindsey K, Frederick B (2016). Correcting resting state fMRI-BOLD signals for blood arrival time enhances functional connectivity analysis. Front. Hum. Neurosci., 28 June 2016 | http://dx.doi.org/10.3389/fnhum.2016.00311

6) Tong, Y, Hocke, LM, and Frederick, BB, Low Frequency Systemic Hemodynamic "Noise" in Resting State BOLD fMRI: Characteristics, Causes, Implications, Mitigation Strategies, and Applications. Front Neurosci, 2019. 13: p. 787. | http://dx.doi.org/10.3389/fnins.2019.00787

2.9 Bare metal installation

This gives you the maximum flexibility if you want to look at the code and/or modify things. It may seem a little daunting at first, but it's not that bad. And if you want a simpler path, skip down to the Docker installation instructions

2.9.1 Required dependencies

The processing programs in rapidtide require the following to be installed first:

- Python 3.x (I no longer support or test running in 2.x, but it does still work for the time being. But I use dependencies like nibabel that have dropped 2.x support, so this is not going to last long.)
- numpy>=1.16
- scipy
- pandas
- scikit-learn
- scikit-image
- nibabel
- nilearn
- matplotlib
- statsmodels

If you want to use tidepool for image display, you will also need to install the following:

- pyqt5
- pyqtgraph

2.9.2 Optional dependencies

The following optional dependencies will be used if present:

- numba (for faster performance)
- pyfftw (also for faster performance)
- mkl and mkl-service (again, faster performance)

If you want to use the deep learning filter in happy, you'll need Keras and some sort of backend. If you want to be able to train filters, you'll probably want GPU support. This is currently an annoying, non-trivial thing to set up, especially on a Mac, which is where I do things, because Apple and Nvidia aren't friends at the moment. If you are on a linux box (or maybe Windows - haven't tested), WITH an Nvidia GPU, install:

- keras
- tensorflow-gpu (This assumes you have all the necessary CUDA libraries. Making this all work together properly is a version dependent moving target. Ask The Google the best way to do it this week - anything I say here will probably be obsolete by the time you read this.)

If you are on linux (or Windows) WITHOUT an Nvidia GPU, install:

- keras
- tensorflow (and make sure it doesn't sneakily try to install the GPU version - that won't work)

If you are on a Mac, you almost certainly have a non-Nvidia GPU, so you should install

- plaidml-keras (it installs Keras and uses PlaidML as the backend rather than tensorflow). You will have to run a configuration step in plaidML to tell it what GPU to use and how. I use the "metal" option with the AMD GPU in my laptop - that seems to be the most stable. Currently, I think you have you have to do this from pypi - I haven't seen a conda version of this.

2.9.3 Installing Python

The simplest way BY FAR to get this all done is to use Anaconda python from Continuum Analytics. It's a free, curated scientific Python distribution that is easy to maintain and takes a lot of headaches out of maintaining a distribution. It also already comes with many of the dependancies for rapidtide installed by default. You can get it here: https://www.continuum.io. Rapidtide works with Python 2 or 3. If you are new to Python, you should probably just start at 3.

After installing Anaconda python, install the remaining dependencies (including some good optional ones:

```
conda install nibabel pyqtgraph pyfftw
```

For the deep learning filter in happy, also do:

```
conda install keras tensorflow-gpu
```

(for Linux or Windows WITH Nvidia GPU)

or:

```
conda install keras tensorflow
```

(for Linux or Windows WITHOUT Nvidia GPU)

or

```
pip install plaidml-keras
```

(on a Mac)

Done.

2.9.4 Installing the rapidtide library

Once you have installed the prerequisites, cd into the package directory, and type the following:

```
python setup.py install
```

to install all of the tools in the package. You should be able to run them from the command line then (after rehashing).

2.9.5 Updating

If you've previously installed rapidtide and want to update, cd into the package directory and do a git pull first:

```
git pull
python setup.py install
```

2.10 Docker installation

As of 1.9.0, there is now a Docker container with a full rapidtide installation. To use this, first make sure you have docker installed and properly configured, then run the following:

```
docker pull fredericklab/rapidtide:VERSIONNUMBER
```

This will download the docker container from dockerhub. It's around 4GB, so it may take some time, but it caches the file locally, so you won't have to do this again unless the container updates. To use a particular version, replace VERSIONNUMBER with the version of the container you want.

If you like to live on the edge, just use:

```
docker pull fredericklab/rapidtide:latest
```

This will use the most recent version on dockerhub.

Now that the file is downloaded, you can run and rapidtide command in the Docker container. For example, to run a simple rapidtide analysis, you would use the following command (you can do this all in one step - it will just integrate the first pull into the run time if the version you request hasn't already been downloaded).

Docker runs completely in it's own self-contained environment. If you want to be able to interact with disks outside of container, you map the volume to a mount point in the container using the –volume=EXTERNALDIR:MOUNTPOINT[,ANOTHERDIR:ANOTHERMOUNTPOINT] option to docker.

```
docker run \
    --mount type=bind,source=INPUTDIRECTORY,destination=/data_in \
    --mount type=bind,source=OUTPUTDIRECTORY,destination=/data_out \
    fredericklab/rapidtide:VERSIONNUMBER \
        rapidtide \
            /data_in/YOURNIFTIFILE.nii.gz \
            /data_out/outputname \
            --filterband lfo \
            --searchrange -15 15 \
            --passes 3
```

NOTE: If you want to run this on the test data, like the examples above for the bare metal installation, the example data is in the Docker container in the /src/rapidtide/rapidtide/data/examples/src directory. So to run the first example, you could just do:

```
docker run \
    --mount type=bind,source=OUTPUTDIRECTORY,destination=/data_out \
    fredericklab/rapidtide:latest \
        rapidtide \
            /src/rapidtide/rapidtide/data/examples/src/sub-RAPIDTIDETEST.nii.gz \
            /data_out/dgsr \
            --filterband lfo \
            --searchrange -15 15 \
            --passes 3
```

You can replace the `rapidtide blah blah blah` command with any program in the package - after the fredericklab/rapidtide:latest, just specify the command and arguments as you usually would. If you're running a program that displays anything, you'll have to add a few extra arguments to the docker call. Docker is a little weird about X forwarding - the easiest thing to do is find the IP address of the machine you're running on (lets call it MYIPADDRESS), and do the following:

```
xhost +
```

This disables X11 security - this is almost certainly not the best thing to do, but I don't have a better solution at this time, and it works.

If you're on a Mac using Xquartz, prior to this you'll also have to do three more things.

1) In Xquartz, go into the security preferences, and make sure "Allow connections from network hosts" is checked.

2) Tell Xquartz to listen for TCP connections (this is not the default). Go to a terminal window and type:

```
defaults write org.macosforge.xquartz.X11 nolisten_tcp 0
```

3) Log out and log back in again (you only need to do this once - it will stay that way until you change it.)

Then the following command will work (you can replace 'tidepool' with any of the rapidtide commands that put up windows):

```
docker run \
    --network host\
    --volume=INPUTDIRECTORY:/data_in,OUTPUTDIRECTORY:/data_out \
    -it \
    -e DISPLAY=MYIPADDRESS:0 \
    -u rapidtide \
    fredericklab/rapidtide:latest \
        tidepool
```

2.11 Singularity installation

Many times you can't use Docker, because of security concerns. Singularity, from LBL, offers containerized computing that runs entirely in user space, so the amount of mischief you can get up to is significantly less. Singularity containers can be created from Docker containers as follows (stealing from the fMRIprep documentation):

```
singularity build /my_images/rapidtide-VERSIONNUMBER.simg docker://fredericklab/
↪rapidtide:VERSIONNUMBER
```

Running the container is similar to Docker. The "-B" option is used to bind filesystems to mountpoints in the container. For example, to run the simple rapidtide2x analysis above, type the following:

```
singularity run \
    --cleanenv \
    -B INPUTDIRECTORY:/data_in,OUTPUTDIRECTORY:/data_out \
    rapidtide-VERSIONNUMBER.simg \
        rapidtide \
            /data_in/YOURNIFTIFILE.nii.gz \
            /data_out/outputname \
            --filterband lfo \
            --searchrange -15 15 \
            --passes 3
```

To run a GUI application, you need to disable x security on your host (see comment about this above):

```
xhost +
```

then set the display variable to import to the container:

```
setenv SINGULARITY_DISPLAY MYIPADDRESS:0   (if you are using csh)
```

or

```
export SINGULARITY_DISPLAY="MYIPADDRESS:0" (if you are using sh/bash)
```

then just run the gui command with the command given above.

2.12 References

1) Erdoğan S, Tong Y, Hocke L, Lindsey K, Frederick B (2016). Correcting resting state fMRI-BOLD signals for blood arrival time enhances functional connectivity analysis. Front. Hum. Neurosci., 28 June 2016 | http://dx.doi.org/10.3389/fnhum.2016.00311

For more information about how the rapidtide library can be used, please see the API page. Common rapidtide workflows can also be called from the command line.

2.13 Background

Before talking about the individual programs, in the 2.0 release and going forward, I've tried to adhere to some common principals, across all program, to make them easier to understand and maintain, and more interoperable with other programs, and to simplify using the outputs.

2.13.1 BIDS Outputs

By default, all outputs are in BIDS compatible formats (this is most true for rapidtide and happy, which get the majority of the work, but the goal is to eventually make all the programs in the package conform to this). The two major ramifications of this are that I have tried to follow BIDS naming conventions for NIFTI, json, and text files containing time series. Also, all text files are by default BIDS continuous timeseries files - data is in compressed, tab separated column format (.tsv.gz), with the column names, sample rate, and start time, in the accompanying .json sidecar file.

2.13.2 Text Inputs

A side effect of moving to BIDS is that I've now made a standardized interface for reading text data into programs in the package to handle many different types of file. In general, now if you are asked for a timeseries, you can supply it in any of the following ways:

A plain text file with one or more columns.

You can specify any subset of columns in any order by adding ":colspec" to the end of the filename. "colspec" is a column specification consisting of one or more comma separated "column ranges". A "column range" is either a single column number or a hyphen separated minimum and maximum column number. The first column in a file is column 0.

For example specifying, "mytextfile.txt:5-6,2,0,10-12"

would return an array containing all the timepoints from columns 5, 6, 2, 0, 10, 11, and 12 from mytextfile.txt, in that order. Not specifying ":colspec" returns all the columns in the file, in order.

If the program in question requires the actual sample rate, this can be specified using the `--samplerate` or `--sampletime` flags. Otherwise 1.0Hz is assumed.

A BIDS continuous file with one or more columns.

BIDS files have names for each column, so these are used in column specification. For these files, "colspec" is a comma separated list of one or more column names:

"thefile_desc-interestingtimeseries_physio.json:cardiac,respiration"

would return the two named columns "cardiac" and "respiration" from the accompanying .tsv.gz file. Not specifying ":colspec" returns all the columns in the file, in order.

Because BIDS continuous files require sample rate and start time to be specified in the sidecar file, these quantities will now already be set. Using the `--samplerate`, `--sampletime` or `--starttime` flags will override any header values, if specified.

2.13.3 Visualizing files

Any output NIFTI file can be visualized in your favorite NIFTI viewer. I like FSLeyes, part of FSL. It's flexible and fast, and has lots of options for displaying 3 and 4D NIFTI files.

While there may be nice, general graphing tools for BIDS timeseries files, I wrote "showtc" many years ago, a matplotlib based file viewer with lots of nice tweaks to make pretty and informative graphs of various rapidtide input and output files. It's part of rapidtide, and pretty easy to learn. Just type "showtc" with no arguments to get the options.

As an example, after running happy, if you want to see the derived cardiac waveform, you'd run:

```
showtc \
    happytest_desc-slicerescardfromfmri_timeseries.json:cardiacfromfmri,cardiacfromfmri_
↪dlfiltered \
    --format separate
```

2.14 rapidtide

2.14.1 Description:

The central program in this package is rapidtide. This is the program that calculates a similarity function between a "probe" signal and every voxel of a BOLD fMRI dataset. It then determines the peak value, time delay, and wi dth of the similarity function to determine when and how strongly that probe signal appears in each voxel.

At its core, rapidtide is simply performing a full crosscorrelation between a "probe" timecourse and every voxel in an fMRI dataset (by "full" I mean over a range of time lags that account for any delays between the signals, rather than only at zero lag, as in a Pearson correlation). As with many things, however, the devil is in the details, and so rapidtide provides a number of features which make it pretty good at this particular task. A few highlights:

- There are lots of ways to do something even as simple as a cross-correlation in a nonoptimal way (not windowing, improper normalization, doing it in the time rather than frequency domain, etc.). I'm pretty sure what rapidtide does by default is, if not the best way, at least a very good and very fast way.
- rapidtide has been optimized and profiled to speed it up quite a bit; it has an optional dependency on numba – if it's installed, some of the most heavily used routines will speed up significantly due to judicious use of @jit.
- The sample rate of your probe regressor and the fMRI data do not have to match - rapidtide resamples the probe regressor to an integral multiple of the fMRI data rate automatically.
- The probe and data can be temporally prefiltered to the LFO, respiratory, or cardiac frequency band with a command line switch, or you can specify any low, high, or bandpass range you want.
- The data can be spatially smoothed at runtime (so you don't have to keep smoothed versions of big datasets around). This is quite fast, so no reason not to do it this way.
- rapidtide can generate a probe regressor from the global mean of the data itself - no externally recorded timecourse is required. Optionally you can input both a mask of regions that you want to be included in the mean, and the voxels that you want excluded from the mean (there are situations when you might want to do one or the other or both).
- Determining the significance threshold for filtered correlations where the optimal delay has been selected is nontrivial; using the conventional formulae for the significance of a correlation leads to wildly inflated p values. rapidtide estimates the spurious correlation threshold by calculating the distribution of null correlation values obtained with a shuffling procedure at the beginning of each run (the default is to use 10000 shuffled correlations), and uses this value to mask the correlation maps it calculates. As of version 0.1.2 it will also handle two-tailed significance, which you need when using bipolar mode.
- rapidtide can do an iterative refinement of the probe regressor by aligning the voxel timecourses in time and regenerating the test regressor.
- rapidtide fits the peak of the correlation function, so you can make fine grained distinctions between close lag times. The resolution of the time lag discrimination is set by the length of the timecourse, not the timestep – this is a feature of correlations, not rapidtide.
- Once the time delay in each voxel has been found, rapidtide outputs a 4D file of delayed probe regressors for using as voxel specific confound regressors or to estimate the strength of the probe regressor in each voxel. This regression is performed by default, but these outputs let you do it yourself if you are so inclined.
- I've put a lot of effort into making the outputs as informative as possible - lots of useful maps, histograms, timecourses, etc.
- There are a lot of tuning parameters you can mess with if you feel the need. I've tried to make intelligent defaults so things will work well out of the box, but you have the ability to set most of the interesting parameters yourself.

2.14.2 Inputs:

At a minimum, rapidtide needs a data file to work on (space by time), which is generally thought to be a BOLD fMRI data file. This can be Nifti1 or Nifti2 (for fMRI data, in which case it is time by up to 3 spatial dimensions) or a whitespace separated text file (for NIRS data, each column is a time course, each row a separate channel); I can currently read (probably) but not write Cifti files, so if you want to use grayordinate files you need to convert them to nifti2 in workbench, run rapidtide, then convert back. As soon as nibabel finishes their Cifti support (EDIT: and I get around to figuring it out), I'll add that.

The file needs one time dimension and at least one spatial dimension. Internally, the array is flattened to a time by voxel array for simplicity.

The file you input here should be the result of any preprocessing you intend to do. The expectation is that rapidtide will be run as the last preprocessing step before resting state or task based analysis. So any slice time correction, motion correction, spike removal, etc. should already have been done. If you use FSL, this means that if you've run preprocessing, you would use the filtered_func_data.nii.gz file as input. Temporal and spatial filtering are the two (partial) exceptions here. Generally rapidtide is most useful for looking at low frequency oscillations, so when you run it, you usually use the `--filterband lfo` option or some other to limit the analysis to the detection and removal of low frequency systemic physiological oscillations. So rapidtide will generally apply it's own temporal filtering on top of whatever you do in preprocessing. Also, you have the option of doing spatial smoothing in rapidtide to boost the SNR of the analysis; the hemodynamic signals rapidtide looks for are often very smooth, so you rather than smooth your functional data excessively, you can do it within rapidtide so that only the hemodynamic data is smoothed at that level.

2.14.3 Outputs:

Outputs are space or space by time Nifti or text files, depending on what the input data file was, and some text files containing textual information, histograms, or numbers. File formats and naming follow BIDS conventions for derivative data for fMRI input data. Output spatial dimensions and file type match the input dimensions and file type (Nifti1 in, Nifti1 out). Depending on the file type of map, there can be no time dimension, a time dimension that matches the input file, or something else, such as a time lag dimension for a correlation map.

2.14.4 BIDS Outputs:

Name	Extension(s)	Content	When present
XXX_maxtime_map	.nii.gz, .json	Time of offset of the maximum of the similarity function	Always
XXX_desc-maxtime_hist	.tsv, .json	Histogram of the maxtime map	Always
XXX_maxcorr_map	.nii.gz, .json	Maximum similarity function value (usually the correlation coefficient, R)	Always
XXX_desc-maxcorr_hist	.tsv, .json	Histogram of the maxcorr map	Always
XXX_maxcorrsq_map	.nii.gz, .json	Maximum similarity function value, squared	Always
XXX_desc-maxcorrsq_hist	.tsv, .json	Histogram of the maxcorrsq map	Always
XXX_maxwidth_map	.nii.gz, .json	Width of the maximum of the similarity function	Always
XXX_desc-maxwidth_hist	.tsv, .json	Histogram of the maxwidth map	Always

continues on next page

Table 1 – continued from previous page

Name	Extension(s)	Content	When present
XXX_MTT_map	.nii.gz, .json	Mean transit time (estimated)	Always
XXX_corrfit_mask	.nii.gz	Mask showing where the similarity function fit succeeded	Always
XXX_corrfitfailreason_map	.nii.gz, .json	A numerical code giving the reason a peak could not be found (0 if fit succeeded)	Always
XXX_desc-corrfitwindow_info	.nii.gz	Values used for correlation peak fitting	Always
XXX_desc-runoptions_info	.json	A detailed dump of all internal variables in the program. Useful for debugging and data provenance	Always
XXX_desc-lfofilterCleaned_bold	.nii.gz, .json	Filtered BOLD dataset after removing moving regressor	If GLM filtering is enabled (default)
XXX_desc-lfofilterRemoved_bold	.nii.gz, .json	Scaled, voxelwise delayed moving regressor that has been removed from the dataset	If GLM filtering is enabled (default) and `--nolimitoutput` is selected
XXX_desc-lfofilterCoeff_map	.nii.gz	Magnitude of the delayed sLFO regressor from GLM filter	If GLM filtering is enabled (default)
XXX_desc-lfofilterMean_map	.nii.gz	Mean value over time, from GLM fit	If GLM filtering is enabled (default)
XXX_desc-lfofilterNorm_map	.nii.gz	GLM filter coefficient, divided by the voxel mean over time	If GLM filtering is enabled (default)
XXX_desc-lfofilterR2_map	.nii.gz	R value for the GLM fit in the voxel, squared	If GLM filtering is enabled (default)
XXX_desc-lfofilterR_map	.nii.gz	R value for the GLM fit in the voxel	If GLM filtering is enabled (default)
XXX_desc-processed_mask	.nii.gz	Mask of all voxels in which the similarity function is calculated	Always
XXX_desc-globalmean_mask	.nii.gz	Mask of voxels used to calculate the global mean signal	This file will exist if no external regressor is specified
XXX_desc-refine_mask	.nii.gz	Mask of voxels used in the last estimate a refined version of the probe regressor	Present if passes > 1
XXX_desc-despeckle_mask	.nii.gz	Mask of the last set of voxels that had their time delays adjusted due to autocorrelations in the probe regressor	Present if despecklepasses > 0
XXX_desc-corrout_info	.nii.gz	Full similarity function over the search range	Always

continues on next page

Table 1 – continued from previous page

Name	Extension(s)	Content	When present
XXX_desc-gaussout_info	.nii.gz	Gaussian fit to similarity function peak over the search range	Always
XXX_desc-autocorr_timeseries	.tsv, .json	Autocorrelation of the probe regressor for each pass	Always
XXX_desc-corrdistdata_info	.tsv, .json	Null correlations from the significance estimation for each pass	Present if `--numnull` > 0
XXX_desc-nullsimfunc_hist	.tsv, .json	Histogram of the distribution of null correlation values for each pass	Present if `--numnull` > 0
XXX_desc-plt0p050_mask	.nii.gz	Voxels where the maxcorr value exceeds the p < 0.05 significance level	Present if `--numnull` > 0
XXX_desc-plt0p010_mask	.nii.gz	Voxels where the maxcorr value exceeds the p < 0.01 significance level	Present if `--numnull` > 0
XXX_desc-plt0p005_mask	.nii.gz	Voxels where the maxcorr value exceeds the p < 0.005 significance level	Present if `--numnull` > 0
XXX_desc-plt0p001_mask	.nii.gz	Voxels where the maxcorr value exceeds the p < 0.001 significance level	Present if `--numnull` > 0
XXX_desc-globallag_hist	.tsv, .json	Histogram of peak correlation times between probe and all voxels, over all time lags, for each pass	Always
XXX_desc-initialmovingregressor_timeseries	.tsv, .json	The raw and filtered initial probe regressor, at the original sampling resolution	Always
XXX_desc-movingregressor_timeseries	.tsv, .json	The probe regressor used in each pass, at the time resolution of the data	Always
XXX_desc-oversampledmovingregressor_timeseries	.tsv, .json	The probe regressor used in each pass, at the time resolution used for calculating the similarity function	Always
XXX_desc-refinedmovingregressor_timeseries	.tsv, .json	The raw and filtered probe regressor produced by the refinement procedure, at the time resolution of the data	Present if passes > 1

2.14.5 Usage:

Perform a RIPTiDe time delay analysis on a dataset.

```
usage: rapidtide [-h] [--denoising | --delaymapping] [--venousrefine | --nirs]
                 [--datatstep TSTEP | --datafreq FREQ] [--noantialias]
                 [--invert] [--interptype {univariate,cubic,quadratic}]
                 [--offsettime OFFSETTIME] [--autosync]
                 [--filterband {None,vlf,lfo,resp,cardiac,lfo_legacy}]
                 [--filterfreqs LOWERPASS UPPERPASS]
                 [--filterstopfreqs LOWERSTOP UPPERSTOP]
```

(continues on next page)

(continued from previous page)

```
                [--filtertype {trapezoidal,brickwall,butterworth}]
                [--butterorder ORDER] [--padseconds SECONDS]
                [--permutationmethod {shuffle,phaserandom}] [--numnull NREPS]
                [--skipsighistfit]
                [--windowfunc {hamming,hann,blackmanharris,None}]
                [--nowindow] [--zeropadding PADVAL] [--detrendorder ORDER]
                [--spatialfilt GAUSSSIGMA] [--globalmean]
                [--globalmaskmethod {mean,variance}] [--globalpreselect]
                [--globalmeaninclude MASK[:VALSPEC]]
                [--globalmeanexclude MASK[:VALSPEC]] [--motionfile MOTFILE]
                [--motpos] [--motderiv] [--motdelayderiv]
                [--globalsignalmethod {sum,meanscale,pca}]
                [--globalpcacomponents VALUE] [--slicetimes FILE]
                [--numskip SKIP] [--timerange START END] [--nothresh]
                [--oversampfac OVERSAMPFAC] [--regressor FILE]
                [--regressorfreq FREQ | --regressortstep TSTEP]
                [--regressorstart START]
                [--corrweighting {None,phat,liang,eckart}]
                [--corrmaskthresh PCT | --corrmask MASK[:VALSPEC]]
                [--similaritymetric {correlation,mutualinfo,hybrid}]
                [--mutualinfosmoothingtime TAU]
                [--fixdelay DELAYTIME | --searchrange LAGMIN LAGMAX]
                [--sigmalimit SIGMALIMIT] [--bipolar] [--nofitfilt]
                [--peakfittype {gauss,fastgauss,quad,fastquad,COM,None}]
                [--despecklepasses PASSES] [--despecklethresh VAL]
                [--refineprenorm {None,mean,var,std,invlag}]
                [--refineweighting {None,NIRS,R,R2}] [--passes PASSES]
                [--refineinclude MASK[:VALSPEC]]
                [--refineexclude MASK[:VALSPEC]] [--norefinedespeckled]
                [--lagminthresh MIN] [--lagmaxthresh MAX] [--ampthresh AMP]
                [--sigmathresh SIGMA] [--norefineoffset] [--psdfilter]
                [--pickleft] [--pickleftthresh THRESH]
                [--refineupperlag | --refinelowerlag]
                [--refinetype {pca,ica,weighted_average,unweighted_average}]
                [--pcacomponents VALUE] [--convergencethresh THRESH]
                [--maxpasses MAXPASSES] [--nolimitoutput] [--savelags]
                [--histlen HISTLEN] [--glmsourcefile FILE] [--noglm]
                [--preservefiltering] [--saveintermediatemaps]
                [--legacyoutput] [--calccoherence] [--noprogressbar]
                [--checkpoint] [--wiener] [--spcalculation] [--dpoutput]
                [--cifti] [--simulate] [--displayplots] [--nonumba]
                [--nosharedmem] [--memprofile] [--mklthreads MKLTHREADS]
                [--nprocs NPROCS] [--version] [--echocancel] [--respdelete]
                [--negativegradient] [--cleanrefined] [--dispersioncalc]
                [--acfix] [--tmask FILE] [--debug] [--verbose]
                [--alwaysmultiproc] [--singleproc_getNullDist]
                [--singleproc_calcsimilarity] [--singleproc_peakeval]
                [--singleproc_fitcorr] [--singleproc_glm]
                in_file outputname
```

Positional Arguments

in_file The input data file (BOLD fMRI file or NIRS text file).

outputname The root name for the output files. For BIDS compliance, this can only contain valid BIDS entities from the source data.

Named Arguments

--permutationmethod Possible choices: shuffle, phaserandom

Permutation method for significance testing. Default is "shuffle".

Default: "shuffle"

--numnull Estimate significance threshold by running NREPS null correlations (default is 10000, set to 0 to disable).

Default: 10000

--skipsighistfit Do not fit significance histogram with a Johnson SB function.

Default: True

Analysis type

Single arguments that change default values for many arguments. Any parameter set by an analysis type can be overridden by setting that parameter explicitly. Analysis types are mutually exclusive with one another.

--denoising Preset for hemodynamic denoising - this is a macro that sets lagmin=-10.0, lagmax=10.0, passes=3, despeckle_passes=4, refineoffset=True, peakfittype=gauss, doglmfilt=True. Any of these options can be overridden with the appropriate additional arguments.

Default: False

--delaymapping Preset for delay mapping analysis - this is a macro that sets lagmin=-10.0, lagmax=30.0, passes=3, despeckle_passes=4, refineoffset=True, pickleft=True, limitoutput=True, doglmfilt=False. Any of these options can be overridden with the appropriate additional arguments.

Default: False

Macros

Single arguments that change default values for many arguments. Macros override individually set parameters. Macros are mutually exclusive with one another.

--venousrefine This is a macro that sets –lagminthresh=2.5, –lagmaxthresh=6.0, –ampthresh=0.5, and –refineupperlag to bias refinement towards voxels in the draining vasculature for an fMRI scan.

Default: False

--nirs This is a NIRS analysis - this is a macro that sets –nothresh, –preservefiltering, –refineprenorm=var, –ampthresh=0.7, and –lagminthresh=0.1.

Default: False

Preprocessing options

--datatstep Set the timestep of the data file to TSTEP. This will override the TR in an fMRI file. NOTE: if using data from a text file, for example with NIRS data, using one of these options is mandatory.

Default: auto

--datafreq Set the timestep of the data file to 1/FREQ. This will override the TR in an fMRI file. NOTE: if using data from a text file, for example with NIRS data, using one of these options is mandatory.

Default: auto

--noantialias Disable antialiasing filter.

Default: True

--invert Invert the sign of the regressor before processing.

Default: False

--interptype Possible choices: univariate, cubic, quadratic

Use specified interpolation type. Options are "cubic", "quadratic", and "univariate". Default is univariate.

Default: "univariate"

--offsettime Apply offset OFFSETTIME to the lag regressors.

Default: 0.0

--autosync Estimate and apply the initial offsettime of an external regressor using the global crosscorrelation. Overrides offsettime if present.

Default: False

--detrendorder Set order of trend removal (0 to disable). Default is 3.

Default: 3

--spatialfilt Spatially filter fMRI data prior to analysis using GAUSSSIGMA in mm. Set GAUSSSIGMA negative to have rapidtide set it to half the mean voxel dimension (a rule of thumb for a good value).

Default: 0.0

--globalmean Generate a global mean regressor and use that as the reference regressor. If no external regressor is specified, this is enabled by default.

Default: False

--globalmaskmethod Possible choices: mean, variance

Select whether to use timecourse mean or variance to mask voxels prior to generating global mean. Default is "mean".

Default: "mean"

--globalpreselect Treat this run as an initial pass to locate good candidate voxels for global mean regressor generation.

Default: False

--globalmeaninclude Only use voxels in mask file NAME for global regressor generation (if VALSPEC is given, only voxels with integral values listed in VALSPEC are used).

--globalmeanexclude Do not use voxels in mask file NAME for global regressor generation (if VALSPEC is given, only voxels with integral values listed in VALSPEC are excluded).

--motionfile Read 6 columns of motion regressors out of MOTFILE file (.par or BIDS .json) (with timepoints rows) and regress their derivatives and delayed derivatives out of the data prior to analysis.

--motpos Toggle whether displacement regressors will be used in motion regression. Default is False.

Default: False

--motderiv Toggle whether derivatives will be used in motion regression. Default is True.

Default: True

--motdelayderiv Toggle whether delayed derivative regressors will be used in motion regression. Default is False.

Default: False

--globalsignalmethod Possible choices: sum, meanscale, pca

The method for constructing the initial global signal regressor - straight summation, mean scaling each voxel prior to summation, or MLE PCA of the voxels in the global signal mask. Default is "sum."

Default: "sum"

--globalpcacomponents Number of PCA components used for estimating the global signal. If VALUE >= 1, will retain thismany components. If 0.0 < VALUE < 1.0, enough components will be retained to explain the fraction VALUE of the total variance. If VALUE is negative, the number of components will be to retain will be selected automatically using the MLE method. Default is 0.8.

Default: 0.8

--slicetimes Apply offset times from FILE to each slice in the dataset.

--numskip SKIP TRs were previously deleted during preprocessing (e.g. if you have done your preprocessing in FSL and set dummypoints to a nonzero value.) Default is 0.

Default: 0

--timerange Limit analysis to data between timepoints START and END in the fmri file. If END is set to -1, analysis will go to the last timepoint. Negative values of START will be set to 0. Default is to use all timepoints.

Default: (-1, -1)

--nothresh Disable voxel intensity threshold (especially useful for NIRS data).

Default: False

Filtering options

--filterband Possible choices: None, vlf, lfo, resp, cardiac, lfo_legacy

Filter data and regressors to specific band. Use "None" to disable filtering. Default is "lfo".

Default: "lfo"

--filterfreqs Filter data and regressors to retain LOWERPASS to UPPERPASS. If –filterstopfreqs is not also specified, LOWERSTOP and UPPERSTOP will be calculated automatically.

--filterstopfreqs Filter data and regressors to with stop frequencies LOWERSTOP and UPPERSTOP. LOWERSTOP must be <= LOWERPASS, UPPERSTOP must be >= UPPERPASS. Using this argument requires the use of –filterfreqs.

--filtertype Possible choices: trapezoidal, brickwall, butterworth

Filter data and regressors using a trapezoidal FFT, brickwall FFT, or butterworth bandpass filter. Default is "trapezoidal".

Default: "trapezoidal"

--butterorder Set order of butterworth filter (if used). Default is 6.

Default: 6

--padseconds The number of seconds of padding to add to each end of a filtered timecourse to reduce end effects. Default is 30.0.

Default: 30.0

Windowing options

--windowfunc Possible choices: hamming, hann, blackmanharris, None

Window function to use prior to correlation. Options are hamming, hann, blackmanharris, and None. Default is hamming

Default: "hamming"

--nowindow Disable precorrelation windowing.

Default: "hamming"

--zeropadding Pad input functions to correlation with PADVAL zeros on each side. A PADVAL of 0 does circular correlations, positive values reduce edge artifacts. Set PADVAL < 0 to set automatically. Default is 0.

Default: 0

Correlation options

--oversampfac Oversample the fMRI data by the following integral factor. Set to -1 for automatic selection (default).

Default: -1

--regressor Read the initial probe regressor from file FILE (if not specified, generate and use the global regressor).

--regressorfreq Probe regressor in file has sample frequency FREQ (default is 1/tr) NB: –regressorfreq and –regressortstep) are two ways to specify the same thing.

Default: auto

--regressortstep Probe regressor in file has sample frequency FREQ (default is 1/tr) NB: –regressorfreq and –regressortstep) are two ways to specify the same thing.

Default: auto

--regressorstart The time delay in seconds into the regressor file, corresponding in the first TR of the fMRI file (default is 0.0).

Default: 0.0

--corrweighting Possible choices: None, phat, liang, eckart

Method to use for cross-correlation weighting. Default is “None”.

Default: “None”

--corrmaskthresh Do correlations in voxels where the mean exceeds this percentage of the robust max. Default is 1.0.

Default: 1.0

--corrmask Only do correlations in nonzero voxels in NAME (if VALSPEC is given, only voxels with integral values listed in VALSPEC are used).

--similaritymetric Possible choices: correlation, mutualinfo, hybrid

Similarity metric for finding delay values. Choices are “correlation”, “mutualinfo”, and “hybrid”. Default is correlation.

Default: “correlation”

--mutualinfosmoothingtime Time constant of a temporal smoothing function to apply to the mutual information function. Default is 3.0 seconds. TAU <=0.0 disables smoothing.

Default: 3.0

Correlation fitting options

--fixdelay Don’t fit the delay time - set it to DELAYTIME seconds for all voxels.

--searchrange Limit fit to a range of lags from LAGMIN to LAGMAX. Default is -30.0 to 30.0 seconds.

Default: (-30.0, 30.0)

--sigmalimit Reject lag fits with linewidth wider than SIGMALIMIT Hz. Default is 1000.0 Hz.

Default: 1000.0

--bipolar Bipolar mode - match peak correlation ignoring sign.

Default: False

--nofitfilt Do not zero out peak fit values if fit fails.

Default: True

--peakfittype Possible choices: gauss, fastgauss, quad, fastquad, COM, None

Method for fitting the peak of the similarity function "gauss" performs a Gaussian fit, and is most accurate. "quad" and "fastquad" use a quadratic fit, which is faster, but not as well tested. Default is "gauss".

Default: "gauss"

--despecklepasses Detect and refit suspect correlations to disambiguate peak locations in PASSES passes. Default is to perform 4 passes. Set to 0 to disable.

Default: 4

--despecklethresh Refit correlation if median discontinuity magnitude exceeds VAL. Default is 5.0 seconds.

Default: 5.0

Regressor refinement options

--refineprenorm Possible choices: None, mean, var, std, invlag

Apply TYPE prenormalization to each timecourse prior to refinement. Default is "mean".

Default: "mean"

--refineweighting Possible choices: None, NIRS, R, R2

Apply TYPE weighting to each timecourse prior to refinement. Default is "R2".

Default: "R2"

--passes Set the number of processing passes to PASSES. Default is 3.

Default: 3

--refineinclude Only use voxels in file MASK for regressor refinement (if VALSPEC is given, only voxels with integral values listed in VALSPEC are used).

--refineexclude Do not use voxels in file MASK for regressor refinement (if VALSPEC is given, voxels with integral values listed in VALSPEC are excluded).

--norefinedespeckled Do not use despeckled pixels in calculating the refined regressor.

Default: True

--lagminthresh For refinement, exclude voxels with delays less than MIN. Default is 0.5 seconds.

Default: 0.5

--lagmaxthresh For refinement, exclude voxels with delays greater than MAX. Default is 5.0 seconds.

Default: 5.0

--ampthresh For refinement, exclude voxels with correlation coefficients less than AMP (default is 0.3). NOTE: ampthresh will automatically be set to the p<0.05 significance level determined by the –numnull option if NREPS is set greater than 0 and this is not manually specified.

Default: -1.0

--sigmathresh For refinement, exclude voxels with widths greater than SIGMA seconds. Default is 100.0 seconds.

Default: 100.0

--norefineoffset Disable realigning refined regressor to zero lag.

Default: True

--psdfilter Apply a PSD weighted Wiener filter to shifted timecourses prior to refinement.

Default: False

--pickleft Will select the leftmost delay peak when setting the refine offset.

Default: False

--pickleftthresh Threshhold value (fraction of maximum) in a histogram to be considered the start of a peak. Default is 0.33.

Default: 0.33

--refineupperlag Only use positive lags for regressor refinement.

Default: “both”

--refinelowerlag Only use negative lags for regressor refinement.

Default: “both”

--refinetype Possible choices: pca, ica, weighted_average, unweighted_average

Method with which to derive refined regressor. Default is “pca”.

Default: “pca”

--pcacomponents Number of PCA components used for refinement. If VALUE >= 1, will retain this many components. If 0.0 < VALUE < 1.0, enough components will be retained to explain the fraction VALUE of the total variance. If VALUE is negative, the number of components will be to retain will be selected automatically using the MLE method. Default is 0.8.

Default: 0.8

--convergencethresh Continue refinement until the MSE between regressors becomes <= THRESH. By default, this is not set, so refinement will run for the specified number of passes.

--maxpasses Terminate refinement after MAXPASSES passes, whether or not convergence has occured. Default is 15.

Default: 15

Output options

--nolimitoutput Save some of the large and rarely used files.

Default: True

--savelags Save a table of lagtimes used.

Default: False

--histlen Change the histogram length to HISTLEN. Default is 101.

Default: 101

--glmsourcefile Regress delayed regressors out of FILE instead of the initial fmri file used to estimate delays.

--noglm Turn off GLM filtering to remove delayed regressor from each voxel (disables output of fitNorm).

Default: True

--preservefiltering Don't reread data prior to performing GLM.

Default: False

--saveintermediatemaps Save lag times, strengths, widths, and mask for each pass.

Default: False

--legacyoutput Use legacy file naming and formats rather than BIDS naming and format conventions for output files.

Default: True

--calccoherence Calculate and save the coherence between the final regressor and the data.

Default: False

Miscellaneous options

--noprogressbar Will disable showing progress bars (helpful if stdout is going to a file).

Default: True

--checkpoint Enable run checkpoints.

Default: False

--wiener Do Wiener deconvolution to find voxel transfer function.

Default: False

--spcalculation Use single precision for internal calculations (may be useful when RAM is limited).

Default: "double"

--dpoutput Use double precision for output files.

Default: "single"

--cifti Data file is a converted CIFTI.

Default: False

--simulate Simulate a run - just report command line options.

Default: False

--displayplots Display plots of interesting timecourses.

Default: False

--nonumba Disable jit compilation with numba.

Default: False

--nosharedmem Disable use of shared memory for large array storage.

Default: True

--memprofile Enable memory profiling - warning: this slows things down a lot.

Default: False

--mklthreads Use no more than MKLTHREADS worker threads in accelerated numpy calls.

Default: 1

--nprocs Use NPROCS worker processes for multiprocessing. Setting NPROCS to less than 1 sets the number of worker processes to n_cpus - 1.

Default: 1

--version Print version information and exit.

Default: False

Experimental options (not fully tested, may not work)

--echocancel Attempt to perform echo cancellation.

Default: False

--respdelete Attempt to detect and remove respiratory signal that strays into the LFO band.

Default: False

--negativegradient Calculate the negative gradient of the fmri data after spectral filtering so you can look for CSF flow à la https://www.biorxiv.org/content/10.1101/2021.03.29.437406v1.full.

Default: False

--cleanrefined Perform additional processing on refined regressor to remove spurious components.

Default: False

--dispersioncalc Generate extra data during refinement to allow calculation of dispersion.

Default: False

--acfix Perform a secondary correlation to disambiguate peak location. Experimental.

Default: False

--tmask Only correlate during epochs specified in MASKFILE (NB: each line of FILE contains the time and duration of an epoch to include.

Debugging options. You probably don't want to use any of these unless I ask you to to help diagnose a problem

--debug Enable additional debugging output.

Default: False

--verbose Enable additional runtime information output.

Default: False

--alwaysmultiproc Use the multiprocessing code path even when nprocs=1.

Default: False

--singleproc_getNullDist Force single proc path for getNullDist.

Default: False

--singleproc_calcsimilarity Force single proc path for calcsimilarity.

Default: False

--singleproc_peakeval Force single proc path for peakeval.

Default: False

--singleproc_fitcorr Force single proc path for fitcorr.

Default: False

--singleproc_glm Force single proc path for glm.

Default: False

2.14.6 Legacy interface:

For compatibility with old workflows, rapidtide can be called using legacy syntax by using "rapidtide2x_legacy". Although the underlying code is the same, not all options are settable from the legacy interface. This interface is deprecated and will be removed in a future version of rapidtide, so please convert existing workflows.

```
usage:  rapidtide2x_legacy  datafilename outputname
[-r LAGMIN,LAGMAX] [-s SIGMALIMIT] [-a] [--nowindow] [--phat] [--liang] [--
↪eckart] [-f GAUSSSIGMA] [-O oversampfac] [-t TSTEP] [--datatstep=TSTEP] [--
↪datafreq=FREQ] [-d] [-b] [-V] [-L] [-R] [-C] [-F LOWERFREQ,UPPERFREQ[,
↪LOWERSTOP,UPPERSTOP]] [-o OFFSETTIME] [--autosync] [-T] [-p] [-P] [-B] [-h␣
↪HISTLEN] [-i INTERPTYPE] [-I] [-Z DELAYTIME] [--nofitfilt] [--
↪searchfrac=SEARCHFRAC] [-N NREPS] [--motionfile=MOTFILE] [--pickleft] [--
↪numskip=SKIP] [--refineweighting=TYPE] [--refineprenorm=TYPE] [--
↪passes=PASSES] [--refinepasses=PASSES] [--excluderefine=MASK] [--
↪includerefine=MASK] [--includemean=MASK] [--excludemean=MASK][--
↪lagminthresh=MIN] [--lagmaxthresh=MAX] [--ampthresh=AMP] [--
↪sigmathresh=SIGMA] [--corrmask=MASK] [--corrmaskthresh=PCT] [--refineoffset]␣
↪[--pca] [--ica] [--weightedavg] [--avg] [--psdfilter] [--noprogressbar] [--
↪despecklethresh=VAL] [--despecklepasses=PASSES] [--dispersioncalc] [--
↪refineupperlag] [--refinelowerlag] [--nosharedmem] [--tmask=MASKFILE] [--
↪limitoutput] [--motionfile=FILENAME[:COLSPEC] [--softlimit] [--
↪timerange=START,END] [--skipsighistfit] [--accheck] [--acfix][--
↪numskip=SKIP] [--slicetimes=FILE] [--glmsourcefile=FILE] [--
↪regressorfreq=FREQ] [--regressortstep=TSTEP][--regressor=FILENAME] [--
↪regressorstart=STARTTIME] [--usesp] [--peakfittype=FITTYPE] [--
↪mklthreads=NTHREADS] [--nprocs=NPROCS] [--nirs] [--venousrefine]
```

(continues on next page)

(continued from previous page)

```
Required arguments:
    datafilename                - The input data file (BOLD fmri file or NIRS)
    outputname                  - The root name for the output files

Optional arguments:
    Arguments are processed in order of appearance.  Later options can␣
↪override ones earlier on
    the command line

Macros:
    --venousrefine                  - This is a macro that sets --
↪lagminthresh=2.5, --lagmaxthresh=6.0,
                                      --ampthresh=0.5, and --refineupperlag to␣
↪bias refinement towards
                                      voxels in the draining vasculature for an␣
↪fMRI scan.
    --nirs                          - This is a NIRS analysis - this is a macro␣
↪that sets --nothresh,
                                      --preservefiltering, --refinenorm=var, --
↪ampthresh=0.7,
                                      and --lagminthresh=0.1.

Preprocessing options:
    -t TSTEP,                       - Set the timestep of the data file to␣
↪TSTEP (or 1/FREQ)
      --datatstep=TSTEP,            This will override the TR in an fMRI file.
      --datafreq=FREQ               NOTE: if using data from a text file, for␣
↪example with
                                      NIRS data, using one of these options is␣
↪mandatory.
    -a                              - Disable antialiasing filter
    --detrendorder=ORDER            - Set order of trend removal (0 to disable,␣
↪default is 1 - linear)
    -I                              - Invert the sign of the regressor before␣
↪processing
    -i                              - Use specified interpolation type (options␣
↪are 'cubic',
                                      'quadratic', and 'univariate (default)')
    -o                              - Apply an offset OFFSETTIME to the lag␣
↪regressors
    --autosync                      - Calculate and apply offset time of an␣
↪external regressor from
                                      the global crosscorrelation.  Overrides␣
↪offsettime if specified.
    -b                              - Use butterworth filter for band splitting␣
↪instead of
                                      trapezoidal FFT filter
    -F  LOWERFREQ,UPPERFREQ[,LOWERSTOP,UPPERSTOP]
                                    - Filter data and regressors from LOWERFREQ␣
↪to UPPERFREQ.
                                      LOWERSTOP and UPPERSTOP can be specified,␣
↪or will be
```

(continues on next page)

(continued from previous page)

```
                                    calculated automatically
    -V                              - Filter data and regressors to VLF band
    -L                              - Filter data and regressors to LFO band
    -R                              - Filter data and regressors to respiratory␣
↪band
    -C                              - Filter data and regressors to cardiac band
    --padseconds=SECONDS            - Set the filter pad time to SECONDS␣
↪seconds.  Default
                                    is 30.0
    -N NREPS                        - Estimate significance threshold by␣
↪running NREPS null
                                    correlations (default is 10000, set to 0␣
↪to disable).  If you are
                                    running multiple passes, 'ampthresh' will␣
↪be set to the 0.05 significance.
                                    level unless it is manually specified␣
↪(see below).
    --permutationmethod=METHOD      - Method for permuting the regressor for␣
↪significance estimation.  Default
                                    is shuffle
    --skipsighistfit                - Do not fit significance histogram with a␣
↪Johnson SB function
    --windowfunc=FUNC               - Use FUNC window funcion prior to␣
↪correlation.  Options are
                                    hamming (default), hann, blackmanharris,␣
↪and None
    --nowindow                      - Disable precorrelation windowing
    -f GAUSSSIGMA                   - Spatially filter fMRI data prior to␣
↪analysis using
                                    GAUSSSIGMA in mm
    -M                              - Generate a global mean regressor and use␣
↪that as the
                                    reference regressor
    --globalmeaninclude=MASK[:VALSPEC]
                                    - Only use voxels in NAME for global␣
↪regressor generation (if VALSPEC is
                                    given, only voxels with integral values␣
↪listed in VALSPEC are used.)
    --globalmeanexclude=MASK[:VALSPEC]
                                    - Do not use voxels in NAME for global␣
↪regressor generation (if VALSPEC is
                                    given, only voxels with integral values␣
↪listed in VALSPEC are used.)
    -m                              - Mean scale regressors during global mean␣
↪estimation
    --slicetimes=FILE               - Apply offset times from FILE to each␣
↪slice in the dataset
    --numskip=SKIP                  - SKIP tr's were previously deleted during␣
↪preprocessing (e.g. if you
                                    have done your preprocessing in FSL and␣
↪set dummypoints to a
                                    nonzero value.) Default is 0.
```

(continues on next page)

(continued from previous page)

```
    --timerange=START,END          - Limit analysis to data between timepoints␣
↪START
                                     and END in the fmri file. If END is set␣
↪to -1,
                                     analysis will go to the last timepoint. ␣
↪Negative values
                                     of START will be set to 0. Default is to␣
↪use all timepoints.
    --nothresh                     - Disable voxel intensity threshold␣
↪(especially useful
                                     for NIRS data)
    --motionfile=MOTFILE[:COLSPEC] - Read 6 columns of motion regressors out␣
↪of MOTFILE text file.
                                     (with timepoints rows) and regress their␣
↪derivatives
                                     and delayed derivatives out of the data␣
↪prior to analysis.
                                     If COLSPEC is present, use the comma␣
↪separated list of ranges to
                                     specify X, Y, Z, RotX, RotY, and RotZ, in␣
↪that order.  For
                                     example, :3-5,7,0,9 would use columns 3,␣
↪4, 5, 7, 0 and 9
                                     for X, Y, Z, RotX, RotY, RotZ,␣
↪respectively
    --motpos                       - Toggle whether displacement regressors␣
↪will be used in motion regression.
                                     Default is False.
    --motderiv                     - Toggle whether derivatives will be used␣
↪in motion regression.
                                     Default is True.
    --motdelayderiv                - Toggle whether delayed derivative ␣
↪regressors will be used in motion regression.
                                     Default is False.

Correlation options:
    -O OVERSAMPFAC                 - Oversample the fMRI data by the following␣
↪integral
                                     factor.  Setting to -1 chooses the factor␣
↪automatically (default)
    --regressor=FILENAME           - Read probe regressor from file FILENAME␣
↪(if none
                                     specified, generate and use global␣
↪regressor)
    --regressorfreq=FREQ           - Probe regressor in file has sample␣
↪frequency FREQ
                                     (default is 1/tr) NB: --regressorfreq and␣
↪--regressortstep
                                     are two ways to specify the same thing
    --regressortstep=TSTEP         - Probe regressor in file has sample time␣
↪step TSTEP
                                     (default is tr) NB: --regressorfreq and --
↪regressortstep
```

(continues on next page)

(continued from previous page)

```
                                    are two ways to specify the same thing
    --regressorstart=START          - The time delay in seconds into the␣
↪regressor file, corresponding
                                    in the first TR of the fmri file (default␣
↪is 0.0)
    --phat                          - Use generalized cross-correlation with␣
↪phase alignment
                                    transform (PHAT) instead of correlation
    --liang                         - Use generalized cross-correlation with␣
↪Liang weighting function
                                    (Liang, et al, doi:10.1109/IMCCC.2015.283)
    --eckart                        - Use generalized cross-correlation with␣
↪Eckart weighting function
    --corrmaskthresh=PCT            - Do correlations in voxels where the mean␣
↪exceeeds this
                                    percentage of the robust max (default is␣
↪1.0)
    --corrmask=MASK                 - Only do correlations in voxels in MASK␣
↪(if set, corrmaskthresh
                                    is ignored).
    --accheck                       - Check for periodic components that␣
↪corrupt the autocorrelation

Correlation fitting options:
    -Z DELAYTIME                    - Don't fit the delay time - set it to␣
↪DELAYTIME seconds
                                    for all voxels
    -r LAGMIN,LAGMAX                - Limit fit to a range of lags from LAGMIN␣
↪to LAGMAX
    -s SIGMALIMIT                   - Reject lag fits with linewidth wider than␣
↪SIGMALIMIT
    -B                              - Bipolar mode - match peak correlation␣
↪ignoring sign
    --nofitfilt                     - Do not zero out peak fit values if fit␣
↪fails
    --searchfrac=FRAC               - When peak fitting, include points with␣
↪amplitude > FRAC * the
                                    maximum amplitude.
                                    (default value is 0.5)
    --peakfittype=FITTYPE           - Method for fitting the peak of the␣
↪similarity function
                                    (default is 'gauss'). 'quad' uses a␣
↪quadratic fit. Other options are
                                    'fastgauss' which is faster but not as␣
↪well tested, and 'None'.
    --despecklepasses=PASSES        - detect and refit suspect correlations to␣
↪disambiguate peak
                                    locations in PASSES passes
    --despecklethresh=VAL           - refit correlation if median discontinuity␣
↪magnitude exceeds
                                    VAL (default is 5s)
    --softlimit                     - Allow peaks outside of range if the␣
↪maximum correlation is
```

(continues on next page)

(continued from previous page)

```
                                    at an edge of the range.

Regressor refinement options:
    --refineprenorm=TYPE            - Apply TYPE prenormalization to each␣
↪timecourse prior
                                    to refinement (valid weightings are 'None
↪',
                                    'mean' (default), 'var', and 'std'
    --refineweighting=TYPE          - Apply TYPE weighting to each timecourse␣
↪prior
                                    to refinement (valid weightings are 'None
↪',
                                    'R', 'R2' (default)
    --passes=PASSES,                - Set the number of processing passes to␣
↪PASSES
     --refinepasses=PASSES          (default is 1 pass - no refinement).
                                    NB: refinepasses is the wrong name for␣
↪this option -
                                    --refinepasses is deprecated, use --
↪passes from now on.
    --refineinclude=MASK[:VALSPEC] - Only use nonzero voxels in MASK for␣
↪regressor refinement (if VALSPEC is
                                    given, only voxels with integral values␣
↪listed in VALSPEC are used.)
    --refineexclude=MASK[:VALSPEC] - Do not use nonzero voxels in MASK for␣
↪regressor refinement (if VALSPEC is
                                    given, only voxels with integral values␣
↪listed in VALSPEC are used.)
    --lagminthresh=MIN              - For refinement, exclude voxels with␣
↪delays less
                                    than MIN (default is 0.5s)
    --lagmaxthresh=MAX              - For refinement, exclude voxels with␣
↪delays greater
                                    than MAX (default is 5s)
    --ampthresh=AMP                 - For refinement, exclude voxels with␣
↪correlation
                                    coefficients less than AMP (default is 0.
↪3).  NOTE: ampthresh will
                                    automatically be set to the p<0.05␣
↪significance level determined by
                                    the -N option if -N is set greater than 0␣
↪and this is not
                                    manually specified.
    --sigmathresh=SIGMA             - For refinement, exclude voxels with␣
↪widths greater
                                    than SIGMA (default is 100s)
    --refineoffset                  - Adjust offset time during refinement to␣
↪bring peak
                                    delay to zero
    --pickleft                      - When setting refineoffset, always select␣
↪the leftmost histogram peak
    --pickleftthresh=THRESH         - Set the threshold value (fraction of␣
↪maximum) to decide something is a
```

(continues on next page)

(continued from previous page)

```
                                peak in a histogram.  Default is 0.33.
    --refineupperlag            - Only use positive lags for regressor␣
↪refinement
    --refinelowerlag            - Only use negative lags for regressor␣
↪refinement
    --pca                       - Use pca to derive refined regressor␣
↪(default is
                                unweighted averaging)
    --ica                       - Use ica to derive refined regressor␣
↪(default is
                                unweighted averaging)
    --weightedavg               - Use weighted average to derive refined␣
↪regressor
                                (default is unweighted averaging)
    --avg                       - Use unweighted average to derive refined␣
↪regressor
                                (default)
    --psdfilter                 - Apply a PSD weighted Wiener filter to␣
↪shifted
                                timecourses prior to refinement

Output options:
    --limitoutput               - Don't save some of the large and rarely␣
↪used files
    -T                          - Save a table of lagtimes used
    -h HISTLEN                  - Change the histogram length to HISTLEN␣
↪(default is
                                100)
    --glmsourcefile=FILE        - Regress delayed regressors out of FILE␣
↪instead of the
                                initial fmri file used to estimate delays
    --noglm                     - Turn off GLM filtering to remove delayed␣
↪regressor
                                from each voxel (disables output of␣
↪fitNorm)
    --preservefiltering         - don't reread data prior to GLM

Miscellaneous options:
    --noprogressbar             - Disable progress bars - useful if saving␣
↪output to files
    --wiener                    - Perform Wiener deconvolution to get voxel␣
↪transfer functions
    --usesp                     - Use single precision for internal␣
↪calculations (may
                                be useful when RAM is limited)
    -c                          - Data file is a converted CIFTI
    -S                          - Simulate a run - just report command line␣
↪options
    -d                          - Display plots of interesting timecourses
    --nonumba                   - Disable jit compilation with numba
    --nosharedmem               - Disable use of shared memory for large␣
↪array storage
```

(continues on next page)

(continued from previous page)

```
    --memprofile                    - Enable memory profiling for debugging -␣
↪warning:
                                      this slows things down a lot.
    --multiproc                     - Enable multiprocessing versions of key␣
↪subroutines.  This
                                      speeds things up dramatically.  Almost␣
↪certainly will NOT
                                      work on Windows (due to different forking␣
↪behavior).
    --mklthreads=NTHREADS           - Use no more than NTHREADS worker threads␣
↪in accelerated numpy calls.
    --nprocs=NPROCS                 - Use NPROCS worker processes for␣
↪multiprocessing.  Setting NPROCS
                                      less than 1 sets the number of worker␣
↪processes to
                                      n_cpus - 1 (default).  Setting NPROCS␣
↪enables --multiproc.
    --debug                         - Enable additional information output
    --saveoptionsasjson             - Save the options file in json format␣
↪rather than text.  Will eventually
                                      become the default, but for now I'm just␣
↪trying it out.

Experimental options (not fully tested, may not work):
    --cleanrefined                  - perform additional processing on refined␣
↪regressor to remove spurious
                                      components.
    --dispersioncalc                - Generate extra data during refinement to␣
↪allow calculation of
                                      dispersion.
    --acfix                         - Perform a secondary correlation to␣
↪disambiguate peak location
                                      (enables --accheck).  Experimental.
    --tmask=MASKFILE                - Only correlate during epochs specified in
                                      MASKFILE (NB: if file has one colum, the␣
↪length needs to match
                                      the number of TRs used.  TRs with nonzero␣
↪values will be used
                                      in analysis.  If there are 2 or more␣
↪columns, each line of MASKFILE
                                      contains the time (first column) and␣
↪duration (second column) of an
                                      epoch to include.)
```

These options are somewhat self-explanatory. I will be expanding this section of the manual going forward, but I want to put something here to get this out here.

When using the legacy interface, file names will be output using the old, non-BIDS names and formats. rapidtide can be forced to use the old style outputs with the `--legacyoutput` flag.

2.14.7 Equivalence between BIDS and legacy outputs:

BIDS style name	Legacy name
XXX_maxtime_map(.nii.gz, .json)	XXX_lagtimes.nii.gz
XXX_desc-maxtime_hist(.tsv, .json)	XXX_laghist.txt
XXX_maxcorr_map(.nii.gz, .json)	XXX_lagstrengths.nii.gz
XXX_desc-maxcorr_hist(.tsv, .json)	XXX_strengthhist.txt
XXX_maxcorrsq_map(.nii.gz, .json)	XXX_R2.nii.gz
XXX_desc-maxcorrsq_hist(.tsv, .json)	XXX_R2hist.txt
XXX_maxwidth_map(.nii.gz, .json)	XXX_lagsigma.nii.gz
XXX_desc-maxwidth_hist(.tsv, .json)	XXX_widthhist.txt
XXX_MTT_map(.nii.gz, .json)	XXX_MTT.nii.gz
XXX_corrfit_mask.nii.gz	XXX_fitmask.nii.gz
XXX_corrfitfailreason_map(.nii.gz, .json)	XXX_failreason.nii.gz
XXX_desc-corrfitwindow_info.nii.gz	XXX_windowout.nii.gz
XXX_desc-runoptions_info.json	XXX_options.json
XXX_desc-lfofilterCleaned_bold(.nii.gz, .json)	XXX_filtereddata.nii.gz
XXX_desc-lfofilterRemoved_bold(.nii.gz, .json)	XXX_datatoremove.nii.gz
XXX_desc-lfofilterCoeff_map.nii.gz	XXX_fitcoeff.nii.gz
XXX_desc-lfofilterMean_map.nii.gz	XXX_meanvalue.nii.gz
XXX_desc-lfofilterNorm_map.nii.gz	XXX_fitNorm.nii.gz
XXX_desc-lfofilterR2_map.nii.gz	XXX_r2value.nii.gz
XXX_desc-lfofilterR_map.nii.gz	XXX_rvalue.nii.gz
XXX_desc-processed_mask.nii.gz	XXX_corrmask.nii.gz
XXX_desc-globalmean_mask.nii.gz	XXX_meanmask.nii.gz
XXX_desc-refine_mask.nii.gz	XXX_refinemask.nii.gz
XXX_desc-despeckle_mask.nii.gz	XXX_despecklemask.nii.gz
XXX_desc-corrout_info.nii.gz	XXX_corrout.nii.gz
XXX_desc-gaussout_info.nii.gz	XXX_gaussout.nii.gz
XXX_desc-autocorr_timeseries(.tsv, .json)	XXX_referenceautocorr_passN.txt
XXX_desc-corrdistdata_info(.tsv, .json)	XXX_corrdistdata_passN.txt
XXX_desc-nullsimfunc_hist(.tsv, .json)	XXX_nullsimfunchist_passN.txt
XXX_desc-plt0p050_mask.nii.gz	XXX_p_lt_0p050_mask.nii.gz
XXX_desc-plt0p010_mask.nii.gz	XXX_p_lt_0p010_mask.nii.gz
XXX_desc-plt0p005_mask.nii.gz	XXX_p_lt_0p005_mask.nii.gz
XXX_desc-plt0p001_mask.nii.gz	XXX_p_lt_0p001_mask.nii.gz
XXX_desc-globallag_hist(.tsv, .json)	XXX_globallaghist_passN.txt
XXX_desc-initialmovingregressor_timeseries(.tsv, .json)	XXX_reference_origres.txt, XXX_reference_origres_prefilt.txt
XXX_desc-movingregressor_timeseries(.tsv, .json)	XXX_reference_fmrires_passN.txt
XXX_desc-oversampledmovingregressor_timeseries(.tsv, .json)	XXX_reference_resampres_passN.txt
XXX_desc-refinedmovingregressor_timeseries(.tsv, .json)	XXX_unfilteredrefinedregressor_passN.txt, XXX_refinedregressor_passN.txt
XXX_commandline.txt	XXX_commandline.txt
XXX_formattedcommandline.txt	XXX_formattedcommandline.txt
XXX_memusage.csv	XXX_memusage.csv
XXX_runtimings.txt	XXX_runtimings.txt

2.14.8 Examples:

Rapidtide can do many things - as I've found more interesting things to do with time delay processing, it's gained new functions and options to support these new applications. As a result, it can be a little hard to know what to use for a new experiment. To help with that, I've decided to add this section to the manual to get you started. It's broken up by type of data/analysis you might want to do.

Removing low frequency physiological noise from resting state data

This is what I thought most people would use rapidtide for - finding and removing the low frequency (LFO) signal from an existing dataset. This presupposes you have not made a simultaneous physiological recording (well, you may have, but it assumes you aren't using it). For this, you can use a minimal set of options, since the defaults are mostly right.

The base command you'd use would be:

```
rapidtide inputfmrifile outputname --frequencyband lfo --passes 3
```

This will do a fairly simple analysis. First, the -L option means that rapidtide will prefilter the data to the LFO band (0.009-0.15Hz). It will then construct a regressor from the global mean of the signal in inputfmrifile (default behavior if no regressor is specified), and then use crosscorrelation to determine the time delay in each voxel. The `--passes=3` option directs rapidtide to to perform the delay analysis 3 times, each time generating a new estimate of the global noise signal by aligning all of the timecourses in the data to bring the global signal in phase prior to averaging. The `--refineoffset` flag recenters the peak of the delay distribution on zero during the refinement process, which should make datasets easier to compare. After the three passes are complete, it will then use a GLM filter to remove a lagged copy of the final mean regressor that from the data - this denoised data will be in the file "outputname_filtereddata.nii.gz". There will also a number of maps output with the prefix *"outputname_"* of delay, correlation strength and so on.

Mapping long time delays in response to a gas challenge experiment:

Processing this sort of data requires a very different set of options from the previous case. Instead of the distribution of delays you expect in healthy controls (a slightly skewed, somewhat normal distribution with a tail on the positive side, ranging from about -5 to 5 seconds), in this case, the maximum delay can be extremely long (100-120 seconds is not uncommon in stroke, moyamoya disesase, and atherosclerosis). To do this, you need to radically change what options you use, not just the delay range, but a number of other options having to do with refinement and statistical measures.

For this type of analysis, a good place to start is the following:

```
rapidtide inputfmrifile outputname --numnull 0 --searchrange -10 140 --
↪filterfreqs 0.0 0.1 --ampthresh 0.2 --noglm --nofitfilt
```

The first option (`--numnull 0`), shuts off the calculation of the null correlation distribution. This is used to determine the significance threshold, but the method currently implemented in rapidtide is a bit simplistic - it assumes that all the time points in the data are exchangable. This is certainly true for resting state data (see above), but it is very much NOT true for block paradigm gas challenges. To properly analyze those, I need to consider what time points are 'equivalent', and up to now, I don't, so setting the number of iterations in the Monte Carlo analysis to zero omits this step.

The second option (`--searchrange -10 140`) is fairly obvious - this extends the detectable delay range out to 140 seconds. Note that this is somewhat larger than the maximum delays we frequently see, but to find the correlation peak with maximum precision, you need sufficient additional delay values so that the correlation can come to a peak and then come down enough that you can properly fit it.

Setting `--filterfreqs 0.0 0.1` is VERY important. By default, rapidtide assumes you are looking at endogenous low frequency oscillations, which typically between 0.09 and 0.15 Hz. However, gas challenge paradigms are usually MUCH lower frequency (90 seconds off, 90 seconds on corresponds to 1/180s = ~0.006Hz). So if you use the default

frequency settings, you will completely filter out your stimulus, and presumably, your response. If you are processing one of these experiments and get no results whatsoever, this is almost certainly the problem.

The `--noglm` option disables data filtering. If you are using rapidtide to estimate and remove low frequency noise from resting state or task fMRI data, the last step is to use a glm filter to remove this circulatory signal, leaving "pure" neuronal signal, which you'll use in further analyses. That's not relevant here - the signal you'd be removing is the one you care about. So this option skips that step to save time and disk space.

`--nofitfilt` skips a step after peak estimation. Estimating the delay and correlation amplitude in each voxel is a two step process. First you make a quick estimate (where is the maximum point of the correlation function, and what is its amplitude?), then you refine it by fitting a Gaussian function to the peak to improve the estimate. If this step fails, which it can if the peak is too close to the end of the lag range, or strangely shaped, the default behavior is to mark the point as bad and zero out the parameters for the voxel. The nofitfilt option means that if the fit fails, output the initial estimates rather than all zeros. This means that you get some information, even if it's not fully refined. In my experience it does tend to make the maps for the gas challenge experiments a lot cleaner to use this option since the correlation function is pretty well behaved.

Denoising NIRS data (NEW)

When we started this whole research effort, I waw originally planning to denoise NIRS data, not fMRI data. But one thing led to another, and the NIRS got derailed for the fMRI effort. Now that we have some time to catch our breaths, and more importantly, we have access to some much higher quality NIRS data, this moved back to the front burner. The majority of the work was already done, I just needed to account for a few qualities that make NIRS data different from fMRI data:

- NIRS data is not generally stored in NIFTI files. There is not as yet a standard NIRS format. In the absence of one, you could do worse than a multicolumn text file, with one column per data channel. That's what I did here - if the file has a '.txt' extension rather than '.nii.', '.nii.gz', or no extension, it will assume all I/O should be done on multicolumn text files.
- NIRS data is often zero mean. This turned out to mess with a lot of my assumptions about which voxels have significant data, and mask construction. This has led to some new options for specifying mask threshholds and data averaging.
- NIRS data is in some sense "calibrated" as relative micromolar changes in oxy-, deoxy-, and total hemoglobin concentration, so mean and/or variance normalizing the timecourses may not be right thing to do. I've added in some new options to mess with normalizations.

2.15 happy

2.15.1 Description:

happy is a new addition to the rapidtide suite. It's complementary to rapidtide - it's focussed on fast, cardiac signals in fMRI, rather than the slow, LFO signals we are usually looking at. It's sort of a Frankenprogram - it has three distinct jobs, which are related, but are very distinct.

The first thing happy does is try to extract a cardiac waveform from the fMRI data. This is something I've been thinking about for a long time. Words go here

The second task is to take this raw estimate of the cardiac waveform, and clean it up using a deep learning filter. The original signal is useful, but pretty gross, but I figured you should be able to exploit the pseudoperiodic nature of the signal to greatly improve it. This is also a testbed to work on using neural nets to process time domain signals. It seemed like a worthwhile project, so it got grafted in.

The final task (which was actually the initial task, and the reason I wrote happy to begin with) is to implement Henning Voss' totally cool hypersampling with analytic phase projection (guess where the name "happy" comes from). This is fairly straightforward, as Voss describes his method very clearly. But I have lots of data with no simultaneously recorded cardiac signals, and I was too lazy to go find datasets with pleth data to play with, so that's why I did the cardiac waveform extraction part.

2.15.2 Inputs:

Happy needs a 4D BOLD fMRI data file (space by time) as input. This can be Nifti1 or Nifti2. If you have a simultaneously recorded cardiac waveform, it will happily use it, otherwise it will try to construct (and refine) one. NOTE: the 4D input dataset needs to be completely unpreprocessed - gradient distortion correction and motion correction can destroy the relationship between slice number and actual acquisition time, and slice time correction does not behave as expected for aliased signals (which the cardiac component in fMRI most certainly is), and in any case we need the slice time offsets to construct our waveform.

2.15.3 Outputs:

Outputs are space or space by time Nifti or text files, depending on what the input data file was, and some text files containing textual information, histograms, or numbers. File formats and naming follow BIDS conventions for derivative data for fMRI input data. Output spatial dimensions and file type match the input dimensions and file type (Nifti1 in, Nifti1 out). Depending on the file type of map, there can be no time dimension, a time dimension that matches the input file, or something else, such as a time lag dimension for a correlation map.

2.15.4 BIDS Outputs:

Name	Extension(s)	Content	When present
XXX_commandline	.txt	The command line used to run happy	Always
XXX_formattedcommandline	.txt	The command line used to run happy, attractively formatted	Always
XXX_desc-rawapp_info	.nii.gz	The analytic phase projection map of the cardiac waveform	Always
XXX_desc-app_info	.nii.gz	The analytic phase projection map of the cardiac waveform, voxelwise minimum subtracted	Always
XXX_desc-normapp_info	.nii.gz	The analytic phase projection map of the cardiac waveform, voxelwise minimum subtracted and normalized	Always
XXX_desc-apppeaks_hist	.tsv.gz, .json	Not sure	Always
XXX_desc-apppeaks_hist_centerofmass	.txt	Not sure	Always
XXX_desc-apppeaks_hist_peak	.txt	Not sure	Always
XXX_desc-slicerescardfromfmri_timeseries	.tsv.gz, .json	Cardiac timeseries at the time resolution of slice acquisition ((1/TR * number of slices / multiband factor	Always
XXX_desc-stdrescardfromfmri_timeseries	.tsv.gz, .json	Cardiac timeseries at standard time resolution (25.O Hz)	Always
XXX_desc-cardpulsefromfmri_timeseries	.tsv.gz, .json	The average (over time from minimum) of the cardiac waveform over all voxels	Always
XXX_desc-cardiaccyclefromfmri_timeseries	.tsv.gz, .json	The average (over a single cardiac cycle) of the cardiac waveform over all voxels	Always
XXX_desc-cine_info	.nii.gz	Average image of the fMRI data over a single cardiac cycle	Always
XXX_desc-cycleaverage_timeseries	.tsv.gz, .json	Not sure	Always
XXX_desc-maxphase_map	.nii.gz	Map of the average phase where the maximum amplitude occurs for each voxel	Always
XXX_desc-minphase_map	.nii.gz	Map of the average phase where the minimum amplitude occurs for each voxel	Always
XXX_desc-processvoxels_mask	.nii.gz	Map of all voxels used for analytic phase projection	Always
XXX_desc-vessels_map	.nii.gz	Amplitude of variance over a cardiac cycle (large values are assumed to be vessels)	Always
XXX_desc-vessels_mask	.nii.gz	Locations of voxels with variance over a cardiac cycle that exceeds a threshold (assumed to be vessels)	Always
XXX_desc-arteries_map	.nii.gz	High variance vessels with early maximum values within the cardiac cycle	Always
XXX_desc-veins_map	.nii.gz	High variance vessels with late maximum values within the cardiac cycle	Always
XXX_info	.json	Run parameters and derived values found during the run (quality metrics, derived thresholds, etc.)	Always
XXX_memusage	.csv	Memory statistics at multiple checkpoints over the course of the run	Always
XXX_runtimings	.txt	Detailed timing information	Always

2.15.5 Usage:

2.15.6 Example:

Extract the cardiac waveform and generate phase projections

Case 1: When you don't have a pleth recording

There are substantial improvements to the latest versions of happy. In the old versions, you actually had to run happy twice - the first time to estimate the vessel locations, and the second to actually derive the waveform. Happy now combines these operations interpolation a single run with multiple passes - the first pass locates voxels with high variance, labels them as vessels, then reruns the derivation, restricting the cardiac estimation to these high variance voxels. This gives substantially better results.

Using the example data in the example directory, try the following:

```
happy \
    rapidtide/data/examples/src/sub-HAPPYTEST.nii.gz \
    rapidtide/data/examples/src/sub-HAPPYTEST.json \
    rapidtide/data/examples/dst/happytest
```

This will perform a happy analysis on the example dataset. To see the extracted cardiac waveform (original and filtered), you can use showtc (also part of them rapidtide package):

```
showtc \
    rapidtide/data/examples/src/happytest_desc-slicerescardfromfmri_timeseries.
↪json:cardiacfromfmri,cardiacfromfmri_dlfiltered \
    --format separate
```

2.16 rapidtide2std

2.16.1 Description:

This is a utility for registering rapidtide output maps to standard coordinates. It's usually much faster to run rapidtide in native space then transform afterwards to MNI152 space. NB: this will only work if you have a working FSL installation.

2.16.2 Inputs:

2.16.3 Outputs:

New versions of the rapidtide output maps, registered to either MNI152 space or to the hires anatomic images for the subject. All maps are named with the specified root name with '_std' appended.

2.16.4 Usage:

```
usage: rapidtide2std INPUTFILEROOT OUTPUTDIR FEATDIRECTORY [--all] [--hires]

required arguments:
    INPUTFILEROOT      - The base name of the rapidtide maps up to but not␣
↪including the underscore
    OUTPUTDIR          - The location for the output files
    FEADDIRECTORY      - A feat directory (x.feat) where registration to␣
↪standard space has been performed

optional arguments:
    --all              - also transform the corrout file (warning - file may␣
↪be huge)
    --hires            - transform to match the high resolution anatomic image␣
↪rather than the standard
    --linear           - only do linear transformation, even if warpfile exists
```

2.17 showxcorr_legacy

2.17.1 Description:

Like rapidtide, but for single time courses. Takes two text files as input, calculates and displays the time lagged crosscorrelation between them, fits the maximum time lag, and estimates the significance of the correlation. It has a range of filtering, windowing, and correlation options. This is the old interface - for new analyses you should use showxcorrx.

2.17.2 Inputs:

showxcorr requires two text files containing timecourses with the same sample rate, one timepoint per line, which are to be correlated, and the sample rate.

2.17.3 Outputs:

showxcorr outputs everything to standard out, including the Pearson correlation, the maximum cross correlation, the time of maximum cross correlation, and estimates of the significance levels (if specified). There are no output files.

2.17.4 Usage:

```
usage: showxcorr timecourse1 timecourse2 samplerate [-l LABEL] [-s STARTTIME]␣
↪[-D DURATION] [-d] [-F LOWERFREQ,UPPERFREQ[,LOWERSTOP,UPPERSTOP]] [-V] [-L]␣
↪[-R] [-C] [-t] [-w] [-f] [-z FILENAME] [-N TRIALS]

required arguments:
        timcoursefile1: text file containing a timeseries
        timcoursefile2: text file containing a timeseries
```

(continues on next page)

(continued from previous page)

```
        samplerate:     the sample rate of the timecourses, in Hz

optional arguments:
    -t              - detrend the data
    -w              - prewindow the data
    -l LABEL        - label for the delay value
    -s STARTTIME    - time of first datapoint to use in seconds in the first file
    -D DURATION     - amount of data to use in seconds
    -r RANGE        - restrict peak search range to +/- RANGE seconds (default is
                      +/-15)
    -d              - turns off display of graph
    -F              - filter data and regressors from LOWERFREQ to UPPERFREQ.
                      LOWERSTOP and UPPERSTOP can be specified, or will be
                      calculated automatically
    -V              - filter data and regressors to VLF band
    -L              - filter data and regressors to LFO band
    -R              - filter data and regressors to respiratory band
    -C              - filter data and regressors to cardiac band
    -T              - trim data to match
    -A              - print data on a single summary line
    -a              - if summary mode is on, add a header line showing what␣
↪values
                      mean
    -f              - negate (flip) second regressor
    -z FILENAME     - use the columns of FILENAME as controlling variables and
                      return the partial correlation
    -N TRIALS       - estimate significance thresholds by Monte Carlo with TRIALS
                      repetition
```

2.18 showxcorrx

2.18.1 Description:

This is the newest, most avant-garde version of showxcorr. Because it's an x file, it's more fluid and I don't guarantee that it will keep a stable interface (or even work at any given time). But every time I add something new, it goes here. The goal is eventually to make this the "real" version. Unlike rapidtide, however, I've let it drift quite a bit without syncing it because some people here actually use showxcorr and I don't want to disrupt workflows...

2.18.2 Inputs:

showxcorrx requires two text files containing timecourses with the same sample rate, one timepoint per line, which are to be correlated, and the sample rate.

2.18.3 Outputs:

showxcorrx outputs everything to standard out, including the Pearson correlation, the maximum cross correlation, the time of maximum cross correlation, and estimates of the significance levels (if specified). There are no output files.

2.18.4 Usage:

2.19 showtc

2.19.1 Description:

A very simple command line utility that takes a text file and plots the data in it in a matplotlib window. That's it. A good tool for quickly seeing what's in a file. Has some options to make the plot prettier.

2.19.2 Inputs:

Text files containing time series data

2.19.3 Outputs:

None

2.19.4 Usage:

Plots the data in text files.

```
usage: showtc texfilename[:col1,col2...,coln] [textfilename]... [options]
```

Positional Arguments

textfilenames One or more input files, with optional column specifications

Named Arguments

--samplerate Set the sample rate of the data file to FREQ. If neither samplerate or sampletime is specified, sample rate is 1.0.

Default: auto

--sampletime Set the sample rate of the data file to 1.0/TSTEP. If neither samplerate or sampletime is specified, sample rate is 1.0.

Default: auto

--displaytype Possible choices: time, power, phase

Display data as time series (default), power spectrum, or phase spectrum.

Default: "time"

--format Possible choices: overlaid, separate, separatelinked

Display data overlaid (default), in individually scaled windows, or in separate windows with linked scaling.

Default: "overlaid"

--waterfall Display multiple timecourses in a waterfall plot.

Default: False

--voffset Plot multiple timecourses with OFFSET between them (use negative OFFSET to set automatically).

Default: 0.0

--transpose Swap rows and columns in the input files.

Default: False

--starttime Start plotting at START seconds (default is the start of the data).

--endtime Finish plotting at END seconds (default is the end of the data).

--numskip Skip NUM lines at the beginning of each file (to get past header lines).

Default: 0

--debug Output additional debugging information.

Default: False

General plot appearance options

--title Use TITLE as the overall title of the graph.

Default: ""

--xlabel Label for the plot x axis.

Default: ""

--ylabel Label for the plot y axis.

Default: ""

--legends Comma separated list of legends for each timecourse.

--legendloc Integer from 0 to 10 inclusive specifying legend location. Legal values are: 0: best, 1: upper right, 2: upper left, 3: lower left, 4: lower right, 5: right, 6: center left, 7: center right, 8: lower center, 9: upper center, 10: center. Default is 2.

Default: 2

--colors Comma separated list of colors for each timecourse.

--nolegend Turn off legend label.

Default: True

--noxax	Do not show x axis.
	Default: True
--noyax	Do not show y axis.
	Default: True
--linewidth	A comma separated list of linewidths (in points) for plots. Default is 1.
--tofile	Write figure to file FILENAME instead of displaying on the screen.
--fontscalefac	Scaling factor for annotation fonts (default is 1.0).
	Default: 1.0
--saveres	Write figure to file at DPI dots per inch (default is 1000).
	Default: 1000

2.20 glmfilt

2.20.1 Description:

Uses a GLM filter to remove timecourses (1D text files or 4D NIFTI files) from 4D NIFTI files.

2.20.2 Inputs:

2.20.3 Outputs:

2.20.4 Usage:

Fits and removes the effect of voxel specific and/or global regressors.

```
usage: glmfilt inputfile numskip outputroot --evfile FILE [FILE [FILE...]]
```

Positional Arguments

inputfile	The name of the 3 or 4 dimensional nifti file to fit.
numskip	The number of points to skip at the beginning of the timecourse when fitting.
outputroot	The root name for all output files.

Named Arguments

--evfile One or more files (text timecourse or 4D NIFTI) containing signals to regress out.

2.21 temporaldecomp

2.21.1 Description:

2.21.2 Inputs:

2.21.3 Outputs:

2.21.4 Usage:

2.22 spatialdecomp

2.22.1 Description:

2.22.2 Inputs:

2.22.3 Outputs:

2.22.4 Usage:

2.23 polyfitim

2.23.1 Description:

2.23.2 Inputs:

2.23.3 Outputs:

2.23.4 Usage:

Fit a spatial template to a 3D or 4D NIFTI file.

```
usage: polyfitim datafile datamask templatefile templatmask outputroot [options]
```

Positional Arguments

datafile	The name of the 3 or 4 dimensional nifti file to fit.
datamask	The name of the 3 or 4 dimensional nifti file valid voxel mask (must match datafile).
templatefile	The name of the 3D nifti template file (must match datafile).
templatemask	The name of the 3D nifti template mask (must match datafile).
outputroot	The root name for all output files.

Named Arguments

--regionatlas	Do individual fits to every region in ATLASFILE (3D NIFTI file).
--order	Perform fit to ORDERth order (default (and minimum) is 1) Default: 1

2.24 histnifti

2.24.1 Description:

A command line tool to generate a histogram for a nifti file

2.24.2 Inputs:

A nifti file

2.24.3 Outputs:

A text file containing the histogram information

None

2.24.4 Usage:

```
usage: histnifti inputfile outputroot

required arguments:
        inputfile       - the name of the input nifti file
        outputroot      - the root of the output nifti names
```

2.25 showhist

2.25.1 Description:

Another simple command line utility that displays the histograms generated by rapidtide.

2.25.2 Inputs:

A textfile generated by rapidtide containing histogram information

2.25.3 Outputs:

None

2.25.4 Usage:

```
usage: showhist textfilename
        plots xy histogram data in text file

required arguments:
        textfilename    - a text file containing one timepoint per line
```

2.26 resamp1tc

2.26.1 Description:

This takes an input text file at some sample rate and outputs a text file resampled to the specified sample rate.

2.26.2 Inputs:

2.26.3 Outputs:

2.26.4 Usage:

```
resamp1tc - resample a timeseries file

usage: resamp1tc infilename insamplerate outputfile outsamplerate [-s]

required arguments:
        inputfile       - the name of the input text file
        insamplerate    - the sample rate of the input file in Hz
        outputfile      - the name of the output text file
        outsamplerate   - the sample rate of the output file in Hz
```

(continues on next page)

(continued from previous page)

```
options:
       -s               - split output data into physiological bands (LFO,␣
↪respiratory, cardiac)
```

2.27 resamplenifti

2.27.1 Description:

This takes an input nifti file at some TR and outputs a nifti file resampled to the specified TR.

2.27.2 Inputs:

2.27.3 Outputs:

2.27.4 Usage:

```
usage: resamplenifti inputfile inputtr outputname outputtr [-a]

required arguments:
        inputfile       - the name of the input nifti file
        inputtr         - the tr of the input file in seconds
        outputfile      - the name of the output nifti file
        outputtr        - the tr of the output file in seconds

options:
        -a              - disable antialiasing filter (only relevant if you␣
↪are downsampling in time)
```

2.28 tcfrom3col

2.28.1 Description:

A simple command line that takes an FSL style 3 column regressor file and generates a time course (waveform) file. FSL 3 column files are text files containing one row per "event". Each row has three columns: start time in seconds, duration in seconds, and waveform value. The output waveform is zero everywhere that is not covered by an "event" in the file.

2.28.2 Inputs:

A three column text file

2.28.3 Outputs:

A single column text file containing the waveform

2.28.4 Usage:

```
tcfrom3col - convert a 3 column fsl style regressor into a one column␣
↪timecourse

usage: tcfrom3col infile timestep numpoints outfile

required arguments:
        infile:      a text file containing triplets of start time, duration,␣
↪and value
        timestep:    the time step of the output time coures in seconds
        numpoints:   the number of output time points
        outfile:     the name of the output time course file
```

2.29 pixelcomp

2.29.1 Description:

A program to compare voxel values in two 3D NIFTI files. You give pixelcomp two files, each with their own mask. Any voxel that has a nonzero mask in both files gets added to a list of xy pairs, with the value from the first file being x, and the value from the second file being y. Pixelcomp then: 1) Makes and displays a 2D histogram of all the xy values. 2) Does a linear fit to x and y, and outputs the coefficients (slope and offset) to a XXX_linfit.txt file. 3) Writes all the xy pairs to a tab separated text file, and 4) Makes a Bland-Altman plot of x vs y

2.29.2 Inputs:

Two 3D NIFTI image files, the accompanying mask files, and the root name for the output files.

2.29.3 Outputs:

None

2.29.4 Usage:

```
showtc - plots the data in text files

usage: showtc texfilename[:col1,col2...,coln] [textfilename]... [--nolegend] [-
↪-pspec] [--phase] [--samplerate=Fs] [--sampletime=Ts]

required arguments:
    textfilename        - a text file containing whitespace separated␣
↪timecourses, one timepoint per line
                        A list of comma separated numbers following the␣
↪filename and preceded by a colon is used to select columns to plot

optional arguments:
    --nolegend              - turn off legend label
    --pspec                 - show the power spectra magnitudes of the input␣
↪data instead of the timecourses
    --phase                 - show the power spectra phases of the input data␣
↪instead of the timecourses
    --transpose             - swap rows and columns in the input files
    --waterfall             - plot multiple timecourses as a waterfall
    --voffset=VOFFSET       - plot multiple timecourses as with VOFFSET␣
↪between them (use negative VOFFSET to set automatically)
    --samplerate=Fs         - the sample rate of the input data is Fs Hz␣
↪(default is 1Hz)
    --sampletime=Ts         - the sample time (1/samplerate) of the input␣
↪data is Ts seconds (default is 1s)
    --colorlist=C1,C2,..    - cycle through the list of colors specified by CN
    --linewidth=LW          - set linewidth to LW points (default is 1)
    --fontscalefac=FAC      - scale all font sizes by FAC (default is 1.0)
    --legendlist=L1,L2,..   - cycle through the list of legends specified by␣
↪LN
    --tofile=FILENAME       - write figure to file FILENAME instead of␣
↪displaying on the screen
    --title=TITLE           - use TITLE as the overall title of the graph
    --separate              - use a separate subplot for each timecourse
    --separatelinked        - use a separate subplot for each timecourse, but␣
↪use a common y scaling
    --noxax                 - don't show x axis
    --noyax                 - don't show y axis
    --starttime=START       - start plot at START seconds
    --endtime=END           - end plot at END seconds
    --legendloc=LOC         - Integer from 0 to 10 inclusive specifying␣
↪legend location.  Legal values are:
                              0: best, 1: upper right, 2: upper left, 3:␣
↪lower left, 4: lower right,
                              5: right, 6: center left, 7: center right, 8:␣
↪lower center, 9: upper center,
                              10: center.  Default is 2.
    --debug                 - print debugging information
```

2.30 glmfilt

2.30.1 Description:

Uses a GLM filter to remove timecourses (1D text files or 4D NIFTI files) from 4D NIFTI files.

2.30.2 Inputs:

2.30.3 Outputs:

2.30.4 Usage:

```
usage: glmfilt datafile numskip outputroot evfile [evfile_2...evfile_n]
    Fits and removes the effect of voxel specific and/or global regressors
```

2.31 ccorrica

2.31.1 Description:

Find temporal crosscorrelations between all the columns in a text file (for example the timecourse files output by MELODIC.)

2.31.2 Inputs:

2.31.3 Outputs:

2.31.4 Usage:

```
ccorrica - find temporal crosscorrelations between ICA components

        usage: ccorrica timecoursefile TR
                timcoursefile:  text file containing multiple timeseries, one␣
↪per column, whitespace separated
                TR:             the sample period of the timecourse, in seconds
```

2.32 showstxcorr

2.32.1 Description:

Calculate and display the short term crosscorrelation between two timeseries (useful for dynamic correlation).

2.32.2 Inputs:

2.32.3 Outputs:

2.32.4 Usage:

```
showstxcorr - calculate and display the short term crosscorrelation between␣
↪two timeseries

usage: showstxcorr -i timecoursefile1 [-i timecoursefile2] --samplefreq=FREQ -
↪o outputfile [-l LABEL] [-s STARTTIME] [-D DURATION] [-d] [-F LOWERFREQ,
↪UPPERFREQ[,LOWERSTOP,UPPERSTOP]] [-V] [-L] [-R] [-C] [--nodetrend] [-
↪nowindow] [-f] [--phat] [--liang] [--eckart] [-z FILENAME]

required arguments:
    -i, --infile= timcoursefile1     - text file containing one or more␣
↪timeseries
    [-i, --infile= timcoursefile2]   - text file containing a timeseries
                                       NB: if one timecourse file is specified,
↪ each column
                                       is considered a timecourse, and there␣
↪must be at least
                                       2 columns in the file.  If two␣
↪filenames are given, each
                                       file must have only one column of data.

    -o, --outfile=OUTNAME:           - the root name of the output files

    --samplefreq=FREQ                - sample frequency of all timecourses is␣
↪FREQ
          or
    --sampletime=TSTEP               - time step of all timecourses is TSTEP
                                       NB: --samplefreq and --sampletime are␣
↪two ways to specify
                                       the same thing.

optional arguments:
    --nodetrend   - do not detrend the data before correlation
    --nowindow    - do not prewindow data before corrlation
    --phat        - perform phase alignment transform (PHAT) rather than
                    standard crosscorrelation
    --liang       - perform phase alignment transform with Liang weighting␣
↪function rather than
                    standard crosscorrelation
    --eckart      - perform phase alignment transform with Eckart weighting␣
↪function rather than
                    standard crosscorrelation
    -s STARTTIME  - time of first datapoint to use in seconds in the first file
    -D DURATION   - amount of data to use in seconds
    -d            - turns off display of graph
    -F            - filter data and regressors from LOWERFREQ to UPPERFREQ.
                    LOWERSTOP and UPPERSTOP can be specified, or will be␣
↪calculated automatically
```

(continues on next page)

(continued from previous page)

```
    -V            - filter data and regressors to VLF band
    -L            - filter data and regressors to LFO band
    -R            - filter data and regressors to respiratory band
    -C            - filter data and regressors to cardiac band
    -W WINDOWLEN  - use a window length of WINDOWLEN seconds (default is 50.0s)
    -S STEPSIZE   - timestep between subsequent measurements (default is 25.
↪0s).  Will be rounded to the nearest sample time
    -f            - negate second regressor
```

2.33 tidepool

2.33.1 Description:

This is a very experimental tool for displaying all of the various maps generated by rapidtide in one place, overlayed on an anatomic image. This makes it a bit easier to see how all the maps are related to one another. To use it, launch tidepool from the command line, and then select a lag time map - tidpool will figure out the root name and pull in all of the other associated maps. Works in native or standard space.

2.33.2 Inputs:

2.33.3 Outputs:

2.33.4 Usage:

```
usage: tidepool [-h] [-o OFFSETTIME] [-r] [-n] [-t TRVAL] [-d DATAFILEROOT]
                        [-a ANATNAME] [-m GEOMASKNAME]

A program to display the results of a time delay analysis

optional arguments:
  -h, --help       show this help message and exit
  -o OFFSETTIME    Set lag offset
  -r               enable risetime display
  -n               enable movie mode
  -t TRVAL         Set correlation TR
  -d DATAFILEROOT  Use this dataset (skip initial selection step)
  -a ANATNAME      Set anatomic mask image
  -m GEOMASKNAME   Set geometric mask image
```

2.34 API

2.34.1 rapidtide.workflows: Rapidtide workflows

Common rapidtide workflows.

rapidtide.workflows.rapidtide.rapidtide_main(...)	

rapidtide.workflows.rapidtide.rapidtide_main

rapidtide.workflows.rapidtide.**rapidtide_main**(*argparsingfunc*)

2.34.2 rapidtide.correlate: Correlation functions

Functions for calculating correlations and similar metrics between arrays.

rapidtide.correlate.check_autocorrelation(...)	Check for autocorrelation in an array.
rapidtide.correlate.shorttermcorr_1D(data1, ...)	Calculate short-term sliding-window correlation between two 1D arrays.
rapidtide.correlate.shorttermcorr_2D(data1, ...)	Calculate short-term sliding-window correlation between two 2D arrays.
rapidtide.correlate.calc_MI(x, y[, bins])	Calculate mutual information between two arrays.
rapidtide.correlate.mutual_info_2d(x, y[, ...])	Compute (normalized) mutual information between two 1D variate from a joint histogram.
rapidtide.correlate.cross_mutual_info(x, y)	Calculate cross-mutual information between two 1D arrays.
rapidtide.correlate.mutual_info_to_r(themi)	Convert mutual information to Pearson product-moment correlation.
rapidtide.correlate.delayedcorr(data1, ...)	Calculate correlation between two 1D arrays, at specific delay.
rapidtide.correlate.cepstraldelay(data1, ...)	Estimate delay between two signals using Choudhary's cepstral analysis method.
rapidtide.correlate.arbcorr(input1, Fs1, ...)	Calculate something.
rapidtide.correlate.faststcorrelate(input1, ...)	Perform correlation between short-time Fourier transformed arrays.
rapidtide.correlate.fastcorrelate(input1, input2)	Perform a fast correlation between two arrays.
rapidtide.correlate._centered(arr, newsize)	Return the center newsize portion of the array.
rapidtide.correlate._check_valid_mode_shapes(...)	Check that two shapes are 'valid' with respect to one another.
rapidtide.correlate.convolve_weighted_fft(...)	Convolve two N-dimensional arrays using FFT.
rapidtide.correlate.gccproduct(fft1, fft2, ...)	Calculate product for generalized crosscorrelation.

rapidtide.correlate.check_autocorrelation

rapidtide.correlate.**check_autocorrelation**(*corrscale*, *thexcorr*, *delta=0.1*, *acampthresh=0.1*, *aclagthresh=10.0*, *displayplots=False*, *detrendorder=1*)

Check for autocorrelation in an array.

Parameters

- **corrscale** –
- **thexcorr** –
- **delta** –
- **acampthresh** –
- **aclagthresh** –
- **displayplots** –
- **windowfunc** –
- **detrendorder** –

Returns

- *sidelobetime*
- *sidelobeamp*

rapidtide.correlate.shorttermcorr_1D

rapidtide.correlate.**shorttermcorr_1D**(*data1*, *data2*, *sampletime*, *windowtime*, *samplestep=1*, *detrendorder=0*, *windowfunc='hamming'*)

Calculate short-term sliding-window correlation between two 1D arrays.

Parameters

- **data1** –
- **data2** –
- **sampletime** –
- **windowtime** –
- **samplestep** –
- **detrendorder** –
- **windowfunc** –

Returns

- *times*
- *corrpertime*
- *ppertime*

rapidtide.correlate.shorttermcorr_2D

rapidtide.correlate.**shorttermcorr_2D**(*data1*, *data2*, *sampletime*, *windowtime*, *samplestep=1*, *laglimits=None*, *weighting='None'*, *zeropadding=0*, *windowfunc='None'*, *detrendorder=0*, *display=False*)

Calculate short-term sliding-window correlation between two 2D arrays.

Parameters

- **data1** –
- **data2** –
- **sampletime** –
- **windowtime** –
- **samplestep** –
- **laglimits** –
- **weighting** –
- **zeropadding** –
- **windowfunc** –
- **detrendorder** –
- **display** –

Returns

- *times*
- *xcorrpertime*
- *Rvals*
- *delayvals*
- *valid*

rapidtide.correlate.calc_MI

rapidtide.correlate.**calc_MI**(*x*, *y*, *bins=50*)

Calculate mutual information between two arrays.

Notes

From https://stackoverflow.com/questions/20491028/ optimal-way-to-compute-pairwise-mutual-information-using-numpy/ 20505476#20505476

rapidtide.correlate.mutual_info_2d

rapidtide.correlate.**mutual_info_2d**(*x*, *y*, *sigma=1*, *bins=(256, 256)*, *fast=False*, *normalized=True*, *EPS=1e-06*, *debug=False*)

Compute (normalized) mutual information between two 1D variate from a joint histogram.

Parameters

- **x** (*1D array*) – first variable
- **y** (*1D array*) – second variable
- **sigma** (*float, optional*) – Sigma for Gaussian smoothing of the joint histogram. Default = 1.
- **bins** (*tuple, optional*) –
- **fast** (*bool, optional*) –
- **normalized** (*bool*) – If True, this will calculate the normalized mutual information from [1]. Default = False.
- **EPS** (*float, optional*) – Default = 1.0e-6.

Returns
nmi – the computed similarity measure

Return type
float

Notes

From Ionnis Pappas BBF added the precaching (fast) option

References

rapidtide.correlate.cross_mutual_info

rapidtide.correlate.**cross_mutual_info**(*x*, *y*, *returnaxis=False*, *negsteps=- 1*, *possteps=- 1*, *locs=None*, *Fs=1.0*, *norm=True*, *madnorm=False*, *windowfunc='None'*, *bins=-1*, *prebin=True*, *sigma=0.25*, *fast=True*)

Calculate cross-mutual information between two 1D arrays. :param x: first variable :type x: 1D array :param y: second variable. The length of y must by >= the length of x :type y: 1D array :param returnaxis: set to True to return the time axis :type returnaxis: bool :param negstaps: :type negstaps: int :param possteps: :type possteps: int :param locs: a set of offsets at which to calculate the cross mutual information :type locs: list :param Fs=1.0: :param : :param norm: calculate normalized MI at each offset :type norm: bool :param madnorm: set to True to normalize cross MI waveform by it's median average deviate :type madnorm: bool :param windowfunc: name of the window function to apply to input vectors prior to MI calculation :type windowfunc: str :param bins: number of bins in each dimension of the 2D histogram. Set to -1 to set automatically :type bins: int :param prebin: set to true to cache 2D histogram for all offsets :type prebin: bool :param sigma: histogram smoothing kernel :type sigma: float :param fast: apply speed optimizations :type fast: bool

Returns

- *if returnaxis is True* –

  thexmi_x
  [1D array] the set of offsets at which cross mutual information is calcuated

thexmi_y
[1D array] the set of cross mutual information values

len(thexmi_x): int
the number of cross mutual information values returned

- *else* –

thexmi_y
[1D array] the set of cross mutual information values

rapidtide.correlate.mutual_info_to_r

rapidtide.correlate.**mutual_info_to_r**(*themi*, *d=1*)

Convert mutual information to Pearson product-moment correlation.

rapidtide.correlate.delayedcorr

rapidtide.correlate.**delayedcorr**(*data1*, *data2*, *delayval*, *timestep*)

Calculate correlation between two 1D arrays, at specific delay.

Parameters

- **data1** –
- **data2** –
- **delayval** –
- **timestep** –

Return type
corr

rapidtide.correlate.cepstraldelay

rapidtide.correlate.**cepstraldelay**(*data1*, *data2*, *timestep*, *displayplots=True*)

Estimate delay between two signals using Choudhary's cepstral analysis method.

Parameters

- **data1** –
- **data2** –
- **timestep** –
- **displayplots** –

Return type
arr

References

- Choudhary, H., Bahl, R. & Kumar, A. Inter-sensor Time Delay Estimation using cepstrum of sum and difference signals in underwater multipath environment. in 1-7 (IEEE, 2015). doi:10.1109/UT.2015.7108308

rapidtide.correlate.arbcorr

rapidtide.correlate.**arbcorr**(*input1*, *Fs1*, *input2*, *Fs2*, *start1=0.0*, *start2=0.0*, *windowfunc='hamming'*, *method='univariate'*, *debug=False*)

Calculate something.

rapidtide.correlate.faststcorrelate

rapidtide.correlate.**faststcorrelate**(*input1*, *input2*, *windowtype='hann'*, *nperseg=32*, *weighting='None'*, *displayplots=False*)

Perform correlation between short-time Fourier transformed arrays.

rapidtide.correlate.fastcorrelate

rapidtide.correlate.**fastcorrelate**(*input1*, *input2*, *usefft=True*, *zeropadding=0*, *weighting='None'*, *displayplots=False*, *debug=False*)

Perform a fast correlation between two arrays.

Parameters

- **input1** –
- **input2** –
- **usefft** –
- **zeropadding** –
- **weighting** –
- **displayplots** –
- **debug** –

Return type
corr

Notes

From http://stackoverflow.com/questions/12323959/fast-cross-correlation-method-in-python.

rapidtide.correlate._centered

rapidtide.correlate.**_centered**(*arr*, *newsize*)

Return the center newsize portion of the array.

Parameters

- **arr** –
- **newsize** –

Return type

arr

rapidtide.correlate._check_valid_mode_shapes

rapidtide.correlate.**_check_valid_mode_shapes**(*shape1*, *shape2*)

Check that two shapes are 'valid' with respect to one another.

Specifically, this checks that each item in one tuple is larger than or equal to corresponding item in another tuple.

Parameters

- **shape1** –
- **shape2** –

Raises

ValueError – If at least one item in the first shape is not larger than or equal to the corresponding item in the second one.

rapidtide.correlate.convolve_weighted_fft

rapidtide.correlate.**convolve_weighted_fft**(*in1*, *in2*, *mode='full'*, *weighting='None'*, *displayplots=False*)

Convolve two N-dimensional arrays using FFT.

Convolve *in1* and *in2* using the fast Fourier transform method, with the output size determined by the *mode* argument. This is generally much faster than *convolve* for large arrays (n > ~500), but can be slower when only a few output values are needed, and can only output float arrays (int or object array inputs will be cast to float).

Parameters

- **in1** (*array_like*) – First input.
- **in2** (*array_like*) – Second input. Should have the same number of dimensions as *in1*; if sizes of *in1* and *in2* are not equal then *in1* has to be the larger array.
- **mode** (*str {'full', 'valid', 'same'}, optional*) – A string indicating the size of the output: `full`

  The output is the full discrete linear convolution of the inputs. (Default)

  valid
  The output consists only of those elements that do not rely on the zero-padding.

  same
  The output is the same size as *in1*, centered with respect to the 'full' output.

Returns
out – An N-dimensional array containing a subset of the discrete linear convolution of *in1* with *in2*.

Return type
array

rapidtide.correlate.gccproduct

rapidtide.correlate.**gccproduct**(*fft1*, *fft2*, *weighting*, *threshfrac=0.1*, *displayplots=False*)

Calculate product for generalized crosscorrelation.

Parameters

- **fft1** –
- **fft2** –
- **weighting** –
- **threshfrac** –
- **displayplots** –

Return type
product

2.34.3 rapidtide.filter: Filters

This module contains all the filtering operations for the rapidtide package.

rapidtide.filter.padvec(inputdata[, padlen, ...])	Returns a padded copy of the input data; padlen points of reflected data are prepended and appended to the input data to reduce end effects when the data is then filtered.
rapidtide.filter.unpadvec(inputdata[, padlen])	Returns a input data with the end pads removed (see padvec); padlen points of reflected data are removed from each end of the array.
rapidtide.filter.ssmooth(xsize, ysize, ...)	Applies an isotropic gaussian spatial filter to a 3D array
rapidtide.filter.dolpfiltfilt(Fs, upperpass, ...)	Performs a bidirectional (zero phase) Butterworth lowpass filter on an input vector and returns the result.
rapidtide.filter.dohpfiltfilt(Fs, lowerpass, ...)	Performs a bidirectional (zero phase) Butterworth highpass filter on an input vector and returns the result.
rapidtide.filter.dobpfiltfilt(Fs, lowerpass, ...)	Performs a bidirectional (zero phase) Butterworth bandpass filter on an input vector and returns the result.
rapidtide.filter.transferfuncfilt(inputdata, ...)	Filters input data using a previously calculated transfer function.
rapidtide.filter.getlpfftfunc(Fs, upperpass, ...)	Generates a brickwall lowpass transfer function.
rapidtide.filter.dolpfftfilt(Fs, upperpass, ...)	Performs an FFT brickwall lowpass filter on an input vector and returns the result.
rapidtide.filter.dohpfftfilt(Fs, lowerpass, ...)	Performs an FFT brickwall highpass filter on an input vector and returns the result.
rapidtide.filter.dobpfftfilt(Fs, lowerpass, ...)	Performs an FFT brickwall bandpass filter on an input vector and returns the result.
rapidtide.filter.getlptrapfftfunc(Fs, ...[, ...])	Generates a trapezoidal lowpass transfer function.
rapidtide.filter.dolptrapfftfilt(Fs, ...[, ...])	Performs an FFT filter with a trapezoidal lowpass transfer function on an input vector and returns the result.
rapidtide.filter.dohptrapfftfilt(Fs, ...[, ...])	Performs an FFT filter with a trapezoidal highpass transfer function on an input vector and returns the result.
rapidtide.filter.dobptrapfftfilt(Fs, ...[, ...])	Performs an FFT filter with a trapezoidal bandpass transfer function on an input vector and returns the result.
rapidtide.filter.wiener_deconvolution(...)	lambd is the SNR in the fourier domain
rapidtide.filter.pspec(inputdata)	Calculate the power spectrum of an input signal
rapidtide.filter.spectrum(inputdata[, Fs, ...])	Performs an FFT of the input data, and returns the frequency axis and spectrum of the input signal.
rapidtide.filter.csdfilter(obsdata, commondata)	Cross spectral density filter - makes a filter transfer function that preserves common frequencies.
rapidtide.filter.arb_pass(Fs, inputdata, ...)	Filters an input waveform over a specified range.
rapidtide.filter.blackmanharris(length[, debug])	Returns a Blackman Harris window function of the specified length.
rapidtide.filter.hann(length[, debug])	Returns a Hann window function of the specified length.
rapidtide.filter.hamming(length[, debug])	Returns a Hamming window function of the specified length.
rapidtide.filter.windowfunction(length[, ...])	Returns a window function of the specified length and type.
rapidtide.filter.NoncausalFilter([...])	

Methods

rapidtide.filter.padvec

rapidtide.filter.**padvec**(*inputdata*, *padlen=20*, *cyclic=False*)

Returns a padded copy of the input data; padlen points of reflected data are prepended and appended to the input data to reduce end effects when the data is then filtered.

Parameters

- **inputdata** (*1D array*) – An array of any numerical type. :param inputdata:
- **padlen** (*int, optional*) – The number of points to remove from each end. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:

Returns

paddeddata – The input data, with padlen reflected points added to each end

Return type

1D array

rapidtide.filter.unpadvec

rapidtide.filter.**unpadvec**(*inputdata*, *padlen=20*)

Returns a input data with the end pads removed (see padvec); padlen points of reflected data are removed from each end of the array.

Parameters

- **inputdata** (*1D array*) – An array of any numerical type. :param inputdata:
- **padlen** (*int, optional*) – The number of points to remove from each end. Default is 20. :param padlen:

Returns

unpaddeddata – The input data, with the padding data removed

Return type

1D array

rapidtide.filter.ssmooth

rapidtide.filter.**ssmooth**(*xsize*, *ysize*, *zsize*, *sigma*, *inputdata*)

Applies an isotropic gaussian spatial filter to a 3D array

Parameters

- **xsize** (*float*) – The array x step size in spatial units :param xsize:
- **ysize** (*float*) – The array y step size in spatial units :param ysize:
- **zsize** (*float*) – The array z step size in spatial units :param zsize:
- **sigma** (*float*) – The width of the gaussian filter kernel in spatial units :param sigma:
- **inputdata** (*3D numeric array*) – The spatial data to filter :param inputdata:

Returns

filtereddata – The filtered spatial data

Return type
3D float array

rapidtide.filter.dolpfiltfilt

rapidtide.filter.**dolpfiltfilt**(*Fs*, *upperpass*, *inputdata*, *order*, *padlen=20*, *cyclic=False*, *debug=False*)

Performs a bidirectional (zero phase) Butterworth lowpass filter on an input vector and returns the result. Ends are padded to reduce transients.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **upperpass** (*float*) – Upper end of passband in Hz :param upperpass:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **order** (*int*) – Order of Butterworth filter. :param order:
- **padlen** (*int, optional*) – Amount of points to reflect around each end of the input vector prior to filtering. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns
filtereddata – The filtered data

Return type
1D float array

rapidtide.filter.dohpfiltfilt

rapidtide.filter.**dohpfiltfilt**(*Fs*, *lowerpass*, *inputdata*, *order*, *padlen=20*, *cyclic=False*, *debug=False*)

Performs a bidirectional (zero phase) Butterworth highpass filter on an input vector and returns the result. Ends are padded to reduce transients.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **lowerpass** (*float*) – Lower end of passband in Hz :param lowerpass:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **order** (*int*) – Order of Butterworth filter. :param order:
- **padlen** (*int, optional*) – Amount of points to reflect around each end of the input vector prior to filtering. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns
filtereddata – The filtered data

Return type
1D float array

rapidtide.filter.dobpfiltfilt

rapidtide.filter.**dobpfiltfilt**(*Fs*, *lowerpass*, *upperpass*, *inputdata*, *order*, *padlen=20*, *cyclic=False*, *debug=False*)

Performs a bidirectional (zero phase) Butterworth bandpass filter on an input vector and returns the result. Ends are padded to reduce transients.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **lowerpass** (*float*) – Lower end of passband in Hz :param lowerpass:
- **upperpass** (*float*) – Upper end of passband in Hz :param upperpass:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **order** (*int*) – Order of Butterworth filter. :param order:
- **padlen** (*int, optional*) – Amount of points to reflect around each end of the input vector prior to filtering. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns
filtereddata – The filtered data

Return type
1D float array

rapidtide.filter.transferfuncfilt

rapidtide.filter.**transferfuncfilt**(*inputdata*, *transferfunc*)

Filters input data using a previously calculated transfer function.

Parameters

- **inputdata** (*1D float array*) – Input data to be filtered :param inputdata:
- **transferfunc** (*1D float array*) – The transfer function :param transferfunc:

Returns
filtereddata – Filtered input data

Return type
1D float array

rapidtide.filter.getlpfftfunc

rapidtide.filter.**getlpfftfunc**(*Fs*, *upperpass*, *inputdata*, *debug=False*)

Generates a brickwall lowpass transfer function.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **upperpass** (*float*) – Upper end of passband in Hz :param upperpass:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns

transferfunc – The transfer function

Return type

1D float array

rapidtide.filter.dolpfftfilt

rapidtide.filter.**dolpfftfilt**(*Fs*, *upperpass*, *inputdata*, *padlen=20*, *cyclic=False*, *debug=False*)

Performs an FFT brickwall lowpass filter on an input vector and returns the result. Ends are padded to reduce transients.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **upperpass** (*float*) – Upper end of passband in Hz :param upperpass:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **padlen** (*int, optional*) – Amount of points to reflect around each end of the input vector prior to filtering. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns

filtereddata – The filtered data

Return type

1D float array

rapidtide.filter.dohpfftfilt

rapidtide.filter.**dohpfftfilt**(*Fs*, *lowerpass*, *inputdata*, *padlen=20*, *cyclic=False*, *debug=False*)

Performs an FFT brickwall highpass filter on an input vector and returns the result. Ends are padded to reduce transients.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **lowerpass** (*float*) – Lower end of passband in Hz :param lowerpass:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **padlen** (*int, optional*) – Amount of points to reflect around each end of the input vector prior to filtering. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns

filtereddata – The filtered data

Return type

1D float array

rapidtide.filter.dobpfftfilt

rapidtide.filter.**dobpfftfilt**(*Fs*, *lowerpass*, *upperpass*, *inputdata*, *padlen=20*, *cyclic=False*, *debug=False*)

Performs an FFT brickwall bandpass filter on an input vector and returns the result. Ends are padded to reduce transients.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **lowerpass** (*float*) – Lower end of passband in Hz :param lowerpass:
- **upperpass** (*float*) – Upper end of passband in Hz :param upperpass:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **padlen** (*int, optional*) – Amount of points to reflect around each end of the input vector prior to filtering. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns

filtereddata – The filtered data

Return type

1D float array

rapidtide.filter.getlptrapfftfunc

rapidtide.filter.**getlptrapfftfunc**(*Fs*, *upperpass*, *upperstop*, *inputdata*, *debug=False*)

Generates a trapezoidal lowpass transfer function.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **upperpass** (*float*) – Upper end of passband in Hz :param upperpass:
- **upperstop** (*float*) – Lower end of stopband in Hz :param upperstop:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns

transferfunc – The transfer function

Return type

1D float array

rapidtide.filter.dolptrapfftfilt

rapidtide.filter.**dolptrapfftfilt**(*Fs*, *upperpass*, *upperstop*, *inputdata*, *padlen=20*, *cyclic=False*, *debug=False*)

Performs an FFT filter with a trapezoidal lowpass transfer function on an input vector and returns the result. Ends are padded to reduce transients.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **upperpass** (*float*) – Upper end of passband in Hz :param upperpass:
- **upperstop** (*float*) – Lower end of stopband in Hz :param upperstop:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **padlen** (*int, optional*) – Amount of points to reflect around each end of the input vector prior to filtering. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns

filtereddata – The filtered data

Return type

1D float array

rapidtide.filter.dohptrapfftfilt

rapidtide.filter.**dohptrapfftfilt**(*Fs*, *lowerstop*, *lowerpass*, *inputdata*, *padlen=20*, *cyclic=False*, *debug=False*)

Performs an FFT filter with a trapezoidal highpass transfer function on an input vector and returns the result. Ends are padded to reduce transients.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **lowerstop** (*float*) – Upper end of stopband in Hz :param lowerstop:
- **lowerpass** (*float*) – Lower end of passband in Hz :param lowerpass:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **padlen** (*int, optional*) – Amount of points to reflect around each end of the input vector prior to filtering. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns
filtereddata – The filtered data

Return type
1D float array

rapidtide.filter.dobptrapfftfilt

rapidtide.filter.**dobptrapfftfilt**(*Fs*, *lowerstop*, *lowerpass*, *upperpass*, *upperstop*, *inputdata*, *padlen=20*, *cyclic=False*, *debug=False*)

Performs an FFT filter with a trapezoidal bandpass transfer function on an input vector and returns the result. Ends are padded to reduce transients.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **lowerstop** (*float*) – Upper end of stopband in Hz :param lowerstop:
- **lowerpass** (*float*) – Lower end of passband in Hz :param lowerpass:
- **upperpass** (*float*) – Upper end of passband in Hz :param upperpass:
- **upperstop** (*float*) – Lower end of stopband in Hz :param upperstop:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **padlen** (*int, optional*) – Amount of points to reflect around each end of the input vector prior to filtering. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns
filtereddata – The filtered data

Return type
1D float array

rapidtide.filter.wiener_deconvolution

rapidtide.filter.**wiener_deconvolution**(*signal*, *kernel*, *lambd*)

lambd is the SNR in the fourier domain

rapidtide.filter.pspec

rapidtide.filter.**pspec**(*inputdata*)

Calculate the power spectrum of an input signal

Parameters
inputdata (*1D numpy array*) – Input data

Returns
spectrum – The power spectrum of the input signal.

Return type
1D numpy array

rapidtide.filter.spectrum

rapidtide.filter.**spectrum**(*inputdata*, *Fs=1.0*, *mode='power'*, *trim=True*)

Performs an FFT of the input data, and returns the frequency axis and spectrum of the input signal.

Parameters

- **inputdata** (*1D numpy array*) – Input data :param inputdata:
- **Fs** (*float*) – Sample rate in Hz. Defaults to 1.0 :param Fs:
- **mode** (*{'real', 'imag', 'complex', 'mag', 'phase', 'power'}*) – The type of spectrum to return. Default is 'power'. :param mode:
- **trim** (*bool*) – If True (default) return only the positive frequency values
- **Fs** – Sample rate in Hz. Defaults to 1.0 :param Fs:
- **mode** – The type of spectrum to return. Legal values are 'real', 'imag', 'mag', 'phase', and 'power' (default) :param mode:

Returns

- **specaxis** (*1D float array*) – The frequency axis.
- **specvals** (*1D float array*) – The spectral data.

rapidtide.filter.csdfilter

rapidtide.filter.**csdfilter**(*obsdata*, *commondata*, *padlen=20*, *cyclic=False*, *debug=False*)

Cross spectral density filter - makes a filter transfer function that preserves common frequencies.

Parameters

- **obsdata** (*1D numpy array*) – Input data
- **commondata** (*1D numpy array*) – Shared data
- **padlen** (*int, optional*) – Number of reflected points to add on each end of the input data. Default is 20.
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends
- **debug** (*bool, optional*) – Set to True for additiona information on function internals. Default is False.

Returns

filtereddata – The filtered data

Return type

1D numpy array

rapidtide.filter.arb_pass

rapidtide.filter.**arb_pass**(*Fs*, *inputdata*, *lowerstop*, *lowerpass*, *upperpass*, *upperstop*, *transferfunc='trapezoidal'*, *butterorder=6*, *padlen=20*, *cyclic=False*, *debug=False*)

Filters an input waveform over a specified range. By default it is a trapezoidal FFT filter, but brickwall and butterworth filters are also available. Ends are padded to reduce transients.

Parameters

- **Fs** (*float*) – Sample rate in Hz :param Fs:
- **inputdata** (*1D numpy array*) – Input data to be filtered :param inputdata:
- **lowerstop** (*float*) – Upper end of lower stopband in Hz :param lowerstop:
- **lowerpass** (*float*) – Lower end of passband in Hz :param lowerpass:
- **upperpass** (*float*) – Upper end of passband in Hz :param upperpass:
- **upperstop** (*float*) – Lower end of upper stopband in Hz :param upperstop:
- **butterorder** (*int, optional*) – Order of Butterworth filter, if used. Default is 6. :param butterorder:
- **padlen** (*int, optional*) – Amount of points to reflect around each end of the input vector prior to filtering. Default is 20. :param padlen:
- **cyclic** (*bool, optional*) – If True, pad by wrapping the data in a cyclic manner rather than reflecting at the ends :param cyclic:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns

filtereddata – The filtered data

Return type
1D float array

rapidtide.filter.blackmanharris

rapidtide.filter.**blackmanharris**(*length*, *debug=False*)

Returns a Blackman Harris window function of the specified length. Once calculated, windows are cached for speed.

Parameters

- **length** (*int*) – The length of the window function :param length:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns
windowfunc – The window function

Return type
1D float array

rapidtide.filter.hann

rapidtide.filter.**hann**(*length*, *debug=False*)

Returns a Hann window function of the specified length. Once calculated, windows are cached for speed.

Parameters

- **length** (*int*) – The length of the window function :param length:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns
windowfunc – The window function

Return type
1D float array

rapidtide.filter.hamming

rapidtide.filter.**hamming**(*length*, *debug=False*)

Returns a Hamming window function of the specified length. Once calculated, windows are cached for speed.

Parameters

- **length** (*int*) – The length of the window function :param length:
- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns
windowfunc – The window function

Return type
1D float array

rapidtide.filter.windowfunction

rapidtide.filter.**windowfunction**(*length*, *type='hamming'*, *debug=False*)

Returns a window function of the specified length and type. Once calculated, windows are cached for speed.

Parameters

- **length** (*int*) –

  The length of the window function

  param length

- **type** (*{'hamming', 'hann', 'blackmanharris'}, optional*) – Window type. Choices are 'hamming' (default), 'hann', and 'blackmanharris'. :param type:

- **debug** (*boolean, optional*) – When True, internal states of the function will be printed to help debugging. :param debug:

Returns

windowfunc – The window function

Return type

1D float array

rapidtide.filter.NoncausalFilter

class rapidtide.filter.**NoncausalFilter**(*filtertype='None'*, *transitionfrac=0.05*, *transferfunc='trapezoidal'*, *initlowerstop=None*, *initlowerpass=None*, *initupperpass=None*, *initupperstop=None*, *butterworthorder=6*, *correctfreq=True*, *padtime=30.0*, *cyclic=False*, *debug=False*)

Methods

apply(Fs, data)	Apply the filter to a dataset.

getcyclic	
getfreqs	
getpadtime	
gettype	
setbutterorder	
setcyclic	
setdebug	
setfreqs	
setpadtime	
settransferfunc	
settype	

__init__(*filtertype='None'*, *transitionfrac=0.05*, *transferfunc='trapezoidal'*, *initlowerstop=None*, *initlowerpass=None*, *initupperpass=None*, *initupperstop=None*, *butterworthorder=6*, *correctfreq=True*, *padtime=30.0*, *cyclic=False*, *debug=False*)

A zero time delay filter for one dimensional signals, especially physiological ones.

Parameters

- **filtertype** (*{'None' 'vlf', 'lfo', 'resp', 'card', 'vlf_stop', 'lfo_stop', 'resp_stop', 'card_stop', 'arb', 'arb_stop', 'ringstop'}, optional*) – The type of filter.
- **butterworthorder** (*int, optional*) – Butterworth filter order. Default is 6.
- **correctfreq** (*boolean, optional*) – Fix pass frequencies that are impossible. Default is True.
- **padtime** (*float, optional*) – Amount of time to end pad to reduce edge effects. Default is 30.0 seconds
- **cyclic** (*boolean, optional*) – If True, pad vectors cyclicly rather than reflecting data around the ends
- **debug** (*boolean, optional*) – Enable extended debugging messages. Default is False.

settype(*thetype*)

Set the filter type. Options are 'None' (default), 'vlf', 'lfo', 'resp', 'card', 'vlf_stop', 'lfo_stop', 'resp_stop', 'card_stop', 'arb', 'arb_stop', 'ringstop'.

gettype()

Return the current filter type.

getfreqs()

Return the current frequency limits.

setbutterorder(*order=self.butterworthorder*)

Set the order for Butterworth filter

setpadtime(*padtime*)

Set the end pad time in seconds.

setdebug(*debug*)

Turn debugging on and off with the debug flag.

getpadtime()

Return the current end pad time.

setfreqs(*lowerstop*, *lowerpass*, *upperpass*, *upperstop*)

Set the frequency parameters of the 'arb' and 'arb_stop' filter.

2.34.4 rapidtide.fit: Fitting functions

rapidtide.fit.gaussresidualssk(p, y, x)	**param p**
rapidtide.fit.gaussskresiduals(p, y, x)	**param p**
rapidtide.fit.gaussresiduals(p, y, x)	**param p**
rapidtide.fit.trapezoidresiduals(p, y, x, ...)	**param p**
rapidtide.fit.risetimeresiduals(p, y, x)	**param p**
rapidtide.fit.gausssk_eval(x, p)	**param x**
rapidtide.fit.gauss_eval(x, p)	**param x**
rapidtide.fit.trapezoid_eval_loop(x, ...)	**param x**
rapidtide.fit.risetime_eval_loop(x, p)	**param x**
rapidtide.fit.trapezoid_eval(x, toplength, p)	**param x**
rapidtide.fit.risetime_eval(x, p)	**param x**
rapidtide.fit.locpeak(data, samplerate, ...)	**param data**
rapidtide.fit.trendgen(thexvals, ...)	**param thexvals**
rapidtide.fit.detrend(inputdata[, order, demean])	**param inputdata**
rapidtide.fit.findfirstabove(theyvals, thevalue)	**param theyvals**
rapidtide.fit.findtrapezoidfunc(thexvals, ...)	**param thexvals**
rapidtide.fit.findrisetimefunc(thexvals, ...)	**param thexvals**
rapidtide.fit.findmaxlag_gauss(thexcorr_x, ...)	**param thexcorr_x**
rapidtide.fit.maxindex_noedge(thexcorr_x, ...)	**param thexcorr_x**

rapidtide.fit.gaussresidualssk

rapidtide.fit.**gaussresidualssk**(*p*, *y*, *x*)

Parameters

- **p** –
- **y** –
- **x** –

rapidtide.fit.gaussskresiduals

rapidtide.fit.**gaussskresiduals**(*p*, *y*, *x*)

Parameters

- **p** –
- **y** –
- **x** –

rapidtide.fit.gaussresiduals

rapidtide.fit.**gaussresiduals**(*p*, *y*, *x*)

Parameters

- **p** –
- **y** –
- **x** –

rapidtide.fit.trapezoidresiduals

rapidtide.fit.**trapezoidresiduals**(*p*, *y*, *x*, *toplength*)

Parameters

- **p** –
- **y** –
- **x** –
- **toplength** –

rapidtide.fit.risetimeresiduals

rapidtide.fit.**risetimeresiduals**(*p*, *y*, *x*)

Parameters

- **p** –
- **y** –
- **x** –

rapidtide.fit.gausssk_eval

rapidtide.fit.**gausssk_eval**(*x*, *p*)

Parameters

- **x** –
- **p** –

rapidtide.fit.gauss_eval

rapidtide.fit.**gauss_eval**(*x*, *p*)

Parameters

- **x** –
- **p** –

rapidtide.fit.trapezoid_eval_loop

rapidtide.fit.**trapezoid_eval_loop**(*x*, *toplength*, *p*)

Parameters

- **x** –
- **toplength** –
- **p** –

rapidtide.fit.risetime_eval_loop

rapidtide.fit.**risetime_eval_loop**(*x*, *p*)

Parameters

- **x** –
- **p** –

rapidtide.fit.trapezoid_eval

rapidtide.fit.**trapezoid_eval**(*x*, *toplength*, *p*)

Parameters

- **x** –
- **toplength** –
- **p** –

rapidtide.fit.risetime_eval

rapidtide.fit.**risetime_eval**(*x*, *p*)

Parameters

- **x** –
- **p** –

rapidtide.fit.locpeak

rapidtide.fit.**locpeak**(*data*, *samplerate*, *lastpeaktime*, *winsizeinsecs=5.0*, *thresh=0.75*, *hysteresissecs=0.4*)

Parameters

- **data** –
- **samplerate** –
- **lastpeaktime** –
- **winsizeinsecs** –
- **thresh** –
- **hysteresissecs** –

rapidtide.fit.trendgen

rapidtide.fit.**trendgen**(*thexvals*, *thefitcoffs*, *demean*)

Parameters

- **thexvals** –
- **thefitcoffs** –
- **demean** –

rapidtide.fit.detrend

rapidtide.fit.**detrend**(*inputdata*, *order=1*, *demean=False*)

Parameters

- **inputdata** –
- **order** –
- **demean** –

rapidtide.fit.findfirstabove

rapidtide.fit.**findfirstabove**(*theyvals*, *thevalue*)

Parameters

- **theyvals** –
- **thevalue** –

rapidtide.fit.findtrapezoidfunc

rapidtide.fit.**findtrapezoidfunc**(*thexvals*, *theyvals*, *thetoplength*, *initguess=None*, *debug=False*, *minrise=0.0*, *maxrise=200.0*, *minfall=0.0*, *maxfall=200.0*, *minstart=-100.0*, *maxstart=100.0*, *refine=False*, *displayplots=False*)

Parameters

- **thexvals** –
- **theyvals** –
- **thetoplength** –
- **initguess** –
- **debug** –
- **minrise** –
- **maxrise** –
- **minfall** –
- **maxfall** –
- **minstart** –
- **maxstart** –
- **refine** –
- **displayplots** –

rapidtide.fit.findrisetimefunc

rapidtide.fit.**findrisetimefunc**(*thexvals*, *theyvals*, *initguess=None*, *debug=False*, *minrise=0.0*, *maxrise=200.0*, *minstart=- 100.0*, *maxstart=100.0*, *refine=False*, *displayplots=False*)

Parameters

- **thexvals** –
- **theyvals** –
- **initguess** –
- **debug** –
- **minrise** –
- **maxrise** –
- **minstart** –
- **maxstart** –
- **refine** –
- **displayplots** –

rapidtide.fit.findmaxlag_gauss

rapidtide.fit.**findmaxlag_gauss**(*thexcorr_x*, *thexcorr_y*, *lagmin*, *lagmax*, *widthlimit*, *edgebufferfrac=0.0*, *threshval=0.0*, *uthreshval=30.0*, *debug=False*, *tweaklims=True*, *zerooutbadfit=True*, *refine=False*, *maxguess=0.0*, *useguess=False*, *searchfrac=0.5*, *fastgauss=False*, *lagmod=1000.0*, *enforcethresh=True*, *absmaxsigma=1000.0*, *absminsigma=0.1*, *displayplots=False*)

Parameters

- **thexcorr_x** –
- **thexcorr_y** –
- **lagmin** –
- **lagmax** –
- **widthlimit** –
- **edgebufferfrac** –
- **threshval** –
- **uthreshval** –
- **debug** –
- **tweaklims** –
- **zerooutbadfit** –
- **refine** –
- **maxguess** –
- **useguess** –

- **searchfrac** –
- **fastgauss** –
- **lagmod** –
- **enforcethresh** –
- **displayplots** –

rapidtide.fit.maxindex_noedge

rapidtide.fit.**maxindex_noedge**(*thexcorr_x*, *thexcorr_y*, *bipolar=False*)

Parameters

- **thexcorr_x** –
- **thexcorr_y** –
- **bipolar** –

rapidtide.fit.findmaxlag_gauss_rev

rapidtide.fit.**findmaxlag_gauss_rev**(*thexcorr_x*, *thexcorr_y*, *lagmin*, *lagmax*, *widthlimit*, *absmaxsigma=1000.0*, *hardlimit=True*, *bipolar=False*, *edgebufferfrac=0.0*, *threshval=0.0*, *uthreshval=1.0*, *debug=False*, *tweaklims=True*, *zerooutbadfit=True*, *refine=False*, *maxguess=0.0*, *useguess=False*, *searchfrac=0.5*, *fastgauss=False*, *lagmod=1000.0*, *enforcethresh=True*, *displayplots=False*)

Parameters

- **thexcorr_x** (*1D float array*) – The time axis of the correlation function
- **thexcorr_y** (*1D float array*) – The values of the correlation function
- **lagmin** (*float*) – The minimum allowed lag time in seconds
- **lagmax** (*float*) – The maximum allowed lag time in seconds
- **widthlimit** (*float*) – The maximum allowed peak halfwidth in seconds
- **absmaxsigma** –
- **hardlimit** –
- **bipolar** (*boolean*) – If true find the correlation peak with the maximum absolute value, regardless of sign
- **edgebufferfrac** –
- **threshval** –
- **uthreshval** –
- **debug** –
- **tweaklims** –
- **zerooutbadfit** –
- **refine** –

- **maxguess** –
- **useguess** –
- **searchfrac** –
- **fastgauss** –
- **lagmod** –
- **enforcethresh** –
- **displayplots** –

rapidtide.fit.findmaxlag_quad

rapidtide.fit.**findmaxlag_quad**(*thexcorr_x*, *thexcorr_y*, *lagmin*, *lagmax*, *widthlimit*, *edgebufferfrac=0.0*, *threshval=0.0*, *uthreshval=30.0*, *debug=False*, *tweaklims=True*, *zerooutbadfit=True*, *refine=False*, *maxguess=0.0*, *useguess=False*, *fastgauss=False*, *lagmod=1000.0*, *enforcethresh=True*, *displayplots=False*)

Parameters

- **thexcorr_x** –
- **thexcorr_y** –
- **lagmin** –
- **lagmax** –
- **widthlimit** –
- **edgebufferfrac** –
- **threshval** –
- **uthreshval** –
- **debug** –
- **tweaklims** –
- **zerooutbadfit** –
- **refine** –
- **maxguess** –
- **useguess** –
- **fastgauss** –
- **lagmod** –
- **enforcethresh** –
- **displayplots** –

rapidtide.fit.gaussfitsk

rapidtide.fit.**gaussfitsk**(*height*, *loc*, *width*, *skewness*, *xvals*, *yvals*)

Parameters

- **height** –
- **loc** –
- **width** –
- **skewness** –
- **xvals** –
- **yvals** –

rapidtide.fit.gaussfit

rapidtide.fit.**gaussfit**(*height*, *loc*, *width*, *xvals*, *yvals*)

Parameters

- **height** –
- **loc** –
- **width** –
- **xvals** –
- **yvals** –

rapidtide.fit.mlregress

rapidtide.fit.**mlregress**(*x*, *y*, *intercept=True*)

Parameters

- **x** –
- **y** –
- **intercept** –

rapidtide.fit.parabfit

rapidtide.fit.**parabfit**(*x_axis*, *y_axis*, *peakloc*, *points*)

Parameters

- **x_axis** –
- **y_axis** –
- **peakloc** –
- **peaksize** –

rapidtide.fit._datacheck_peakdetect

rapidtide.fit.**_datacheck_peakdetect**(*x_axis*, *y_axis*)

Parameters

- **x_axis** –
- **y_axis** –

rapidtide.fit.peakdetect

rapidtide.fit.**peakdetect**(*y_axis*, *x_axis=None*, *lookahead=200*, *delta=0.0*)

Converted from/based on a MATLAB script at: http://billauer.co.il/peakdet.html

function for detecting local maxima and minima in a signal. Discovers peaks by searching for values which are surrounded by lower or larger values for maxima and minima respectively

keyword arguments: y_axis – A list containing the signal over which to find peaks

x_axis – A x-axis whose values correspond to the y_axis list and is used
in the return to specify the position of the peaks. If omitted an index of the y_axis is used. (default: None)

lookahead – distance to look ahead from a peak candidate to determine if
it is the actual peak (default: 200) '(samples / period) / f' where '4 >= f >= 1.25' might be a good value

delta – this specifies a minimum difference between a peak and
the following points, before a peak may be considered a peak. Useful to hinder the function from picking up false peaks towards to end of the signal. To work well delta should be set to delta >= RMSnoise * 5. (default: 0)

When omitted delta function causes a 20% decrease in speed. When used Correctly it can double the speed of the function

return: two lists [max_peaks, min_peaks] containing the positive and
negative peaks respectively. Each cell of the lists contains a tuple of: (position, peak_value) to get the average peak value do: np.mean(max_peaks, 0)[1] on the results to unpack one of the lists into x, y coordinates do: x, y = zip(*max_peaks)

2.34.5 rapidtide.io: Input/output functions

rapidtide.io.readfromnifti(inputfile)	Open a nifti file and read in the various important parts
rapidtide.io.readfromcifti(inputfile[, debug])	Open a cifti file and read in the various important parts
rapidtide.io.getciftitr(cifti_hdr)	
rapidtide.io.parseniftidims(thedims)	Split the dims array into individual elements
rapidtide.io.parseniftisizes(thesizes)	Split the size array into individual elements
rapidtide.io.savetonifti(thearray, ...)	Save a data array out to a nifti file
rapidtide.io.savetocifti(thearray, ...[, ...])	Save a data array out to a cifti
rapidtide.io.checkifnifti(filename)	Check to see if a file name is a valid nifti name.
rapidtide.io.niftisplitext(filename)	Split nifti filename into name base and extensionn.
rapidtide.io.niftisplit(inputfile, outputroot)	
rapidtide.io.niftimerge(inputlist, outputname)	

continues on next page

Table 3 – continued from previous page

rapidtide.io.niftiroi(inputfile, outputfile, ...)	
rapidtide.io.checkifcifti(filename[, debug])	Check to see if the specified file is CIFTI format
rapidtide.io.checkiftext(filename)	Check to see if the specified filename ends in '.txt'
rapidtide.io.getniftiroot(filename)	Strip a nifti filename down to the root with no extensions
rapidtide.io.fmriheaderinfo(niftifilename)	Retrieve the header information from a nifti file
rapidtide.io.fmritimeinfo(niftifilename)	Retrieve the repetition time and number of timepoints from a nifti file
rapidtide.io.checkspacematch(hdr1, hdr2[, ...])	Check the headers of two nifti files to determine if the cover the same volume at the same resolution (within tolerance)
rapidtide.io.checkspaceresmatch(sizes1, sizes2)	Check the spatial pixdims of two nifti files to determine if they have the same resolution (within tolerance)
rapidtide.io.checkspacedimmatch(dims1, dims2)	Check the dimension arrays of two nifti files to determine if the cover the same number of voxels in each dimension
rapidtide.io.checktimematch(dims1, dims2[, ...])	Check the dimensions of two nifti files to determine if the cover the same number of timepoints
rapidtide.io.checkifparfile(filename)	Checks to see if a file is an FSL style motion parameter file
rapidtide.io.readparfile(filename)	Checks to see if a file is an FSL style motion parameter file
rapidtide.io.readmotion(filename)	Reads motion regressors from filename (from the columns specified in colspec, if given)
rapidtide.io.calcmotregressors(motiondict[, ...])	Calculates various motion related timecourses from motion data dict, and returns an array
rapidtide.io.sliceinfo(slicetimes, tr)	
rapidtide.io.getslicetimesfromfile(slicetimename)	
rapidtide.io.readbidssidecar(inputfilename)	Read key value pairs out of a BIDS sidecar file
rapidtide.io.writedicttojson(thedict, ...)	Write key value pairs to a json file
rapidtide.io.readdictfromjson(inputfilename)	Read key value pairs out of a json file
rapidtide.io.readlabelledtsv(inputfilename)	Read time series out of an fmriprep confounds tsv file
rapidtide.io.readoptionsfile(inputfileroot)	
rapidtide.io.writebidstsv(outputfileroot, ...)	NB: to be strictly valid, a continuous BIDS tsv file (i.e.
rapidtide.io.readvectorsfromtextfile(...[, ...])	Read one or more time series from some sort of text file
rapidtide.io.readbidstsv(inputfilename[, ...])	Read time series out of a BIDS tsv file
rapidtide.io.readcolfrombidstsv(inputfilename)	**param inputfilename**
rapidtide.io.parsefilespec(filespec[, debug])	
rapidtide.io.colspectolist(colspec[, debug])	
rapidtide.io.processnamespec(maskspec, ...)	
rapidtide.io.readcolfromtextfile(inputfilespec)	**param inputfilename**

continues on next page

Table 3 – continued from previous page

rapidtide.io.readvecs(inputfilename[, ...]) **param inputfilename**	
rapidtide.io.readvec(inputfilename[, numskip])	Read an array of floats in from a text file.
rapidtide.io.readtc(inputfilename[, colnum, ...])	
rapidtide.io.readlabels(inputfilename) **param inputfilename**	
rapidtide.io.writedict(thedict, outputfile)	Write all the key value pairs from a dictionary to a text file.
rapidtide.io.readdict(inputfilename)	Read key value pairs out of a text file
rapidtide.io.writevec(thevec, outputfile[, ...])	Write a vector out to a text file.
rapidtide.io.writenpvecs(thevecs, outputfile)	Write out a two dimensional numpy array to a text file

rapidtide.io.readfromnifti

rapidtide.io.**readfromnifti**(*inputfile*)

Open a nifti file and read in the various important parts

Parameters
inputfile (*str*) – The name of the nifti file.

Returns

- **nim** (*nifti image structure*)
- **nim_data** (*array-like*)
- **nim_hdr** (*nifti header*)
- **thedims** (*int array*)
- **thesizes** (*float array*)

rapidtide.io.readfromcifti

rapidtide.io.**readfromcifti**(*inputfile*, *debug=False*)

Open a cifti file and read in the various important parts

Parameters
inputfile (*str*) – The name of the cifti file.

Returns

- **nim** (*nifti image structure*)
- **nim_data** (*array-like*)
- **nim_hdr** (*nifti header*)
- **thedims** (*int array*)
- **thesizes** (*float array*)

rapidtide.io.getciftitr

rapidtide.io.**getciftitr**(*cifti_hdr*)

rapidtide.io.parseniftidims

rapidtide.io.**parseniftidims**(*thedims*)

Split the dims array into individual elements

Parameters
thedims (*int array*) – The nifti dims structure

Returns
nx, ny, nz, nt – Number of points along each dimension

Return type
int

rapidtide.io.parseniftisizes

rapidtide.io.**parseniftisizes**(*thesizes*)

Split the size array into individual elements

Parameters
thesizes (*float array*) – The nifti voxel size structure

Returns
dimx, dimy, dimz, dimt – Scaling from voxel number to physical coordinates

Return type
float

rapidtide.io.savetonifti

rapidtide.io.**savetonifti**(*thearray*, *theheader*, *thename*)

Save a data array out to a nifti file

Parameters

- **thearray** (*array-like*) – The data array to save.
- **theheader** (*nifti header*) – A valid nifti header
- **thename** (*str*) – The name of the nifti file to save

rapidtide.io.savetocifti

rapidtide.io.**savetocifti**(*thearray*, *theciftiheader*, *theniftiheader*, *thename*, *isseries=False*, *names=['placeholder']*, *start=0.0*, *step=1.0*, *debug=False*)

Save a data array out to a cifti

Parameters

- **thearray** (*array-like*) – The data array to save.
- **theciftiheader** (*cifti header*) – A valid cifti header

- **theniftiheader** (*nifti header*) – A valid nifti header
- **thename** (*str*) – The name of the cifti file to save
- **isseries** (*bool*) – True if output is a dtseries, False if dtscalar
- **start** (*float*) – starttime in seconds
- **step** (*float*) – timestep in seconds
- **debug** (*bool*) – Print extended debugging information

rapidtide.io.checkifnifti

rapidtide.io.**checkifnifti**(*filename*)

Check to see if a file name is a valid nifti name.

Parameters
filename (*str*) – The file name

Returns
isnifti – True if name is a valid nifti file name.

Return type
bool

rapidtide.io.niftisplitext

rapidtide.io.**niftisplitext**(*filename*)

Split nifti filename into name base and extensionn.

Parameters
filename (*str*) – The file name

Returns

- **name** (*str*) – Base name of the nifti file.
- **ext** (*str*) – Extension of the nifti file.

rapidtide.io.niftisplit

rapidtide.io.**niftisplit**(*inputfile*, *outputroot*, *axis=3*)

rapidtide.io.niftimerge

rapidtide.io.**niftimerge**(*inputlist*, *outputname*, *writetodisk=True*, *axis=3*, *returndata=False*, *debug=False*)

rapidtide.io.niftiroi

rapidtide.io.**niftiroi**(*inputfile*, *outputfile*, *startpt*, *numpoints*)

rapidtide.io.checkifcifti

rapidtide.io.**checkifcifti**(*filename*, *debug=False*)

Check to see if the specified file is CIFTI format

Parameters
filename (*str*) – The file name

Returns
iscifti – True if the file header indicates this is a CIFTI file

Return type
bool

rapidtide.io.checkiftext

rapidtide.io.**checkiftext**(*filename*)

Check to see if the specified filename ends in '.txt'

Parameters
filename (*str*) – The file name

Returns
istext – True if filename ends with '.txt'

Return type
bool

rapidtide.io.getniftiroot

rapidtide.io.**getniftiroot**(*filename*)

Strip a nifti filename down to the root with no extensions

Parameters
filename (*str*) – The file name to strip

Returns
strippedname – The file name without any nifti extensions

Return type
str

rapidtide.io.fmriheaderinfo

rapidtide.io.**fmriheaderinfo**(*niftifilename*)

Retrieve the header information from a nifti file

Parameters
niftifilename (*str*) – The name of the nifti file

Returns

- **tr** (*float*) – The repetition time, in seconds
- **timepoints** (*int*) – The number of points along the time axis

rapidtide.io.fmritimeinfo

rapidtide.io.**fmritimeinfo**(*niftifilename*)

Retrieve the repetition time and number of timepoints from a nifti file

Parameters
niftifilename (*str*) – The name of the nifti file

Returns

- **tr** (*float*) – The repetition time, in seconds
- **timepoints** (*int*) – The number of points along the time axis

rapidtide.io.checkspacematch

rapidtide.io.**checkspacematch**(*hdr1*, *hdr2*, *tolerance=0.0001*)

Check the headers of two nifti files to determine if the cover the same volume at the same resolution (within tolerance)

Parameters

- **hdr1** (*nifti header structure*) – The header of the first file
- **hdr2** (*nifti header structure*) – The header of the second file

Returns
ismatched – True if the spatial dimensions and resolutions of the two files match.

Return type
bool

rapidtide.io.checkspaceresmatch

rapidtide.io.**checkspaceresmatch**(*sizes1*, *sizes2*, *tolerance=0.0001*)

Check the spatial pixdims of two nifti files to determine if they have the same resolution (within tolerance)

Parameters

- **sizes1** (*float array*) – The size array from the first nifti file
- **sizes2** (*float array*) – The size array from the second nifti file
- **tolerance** (*float*) – The fractional difference that is permissible between the two sizes that will still match

Returns
ismatched – True if the spatial resolutions of the two files match.

Return type
bool

rapidtide.io.checkspacedimmatch

rapidtide.io.**checkspacedimmatch**(*dims1*, *dims2*)

Check the dimension arrays of two nifti files to determine if the cover the same number of voxels in each dimension

Parameters

- **dims1** (*int array*) – The dimension array from the first nifti file
- **dims2** (*int array*) – The dimension array from the second nifti file

Returns
ismatched – True if the spatial dimensions of the two files match.

Return type
bool

rapidtide.io.checktimematch

rapidtide.io.**checktimematch**(*dims1*, *dims2*, *numskip1=0*, *numskip2=0*)

Check the dimensions of two nifti files to determine if the cover the same number of timepoints

Parameters

- **dims1** (*int array*) – The dimension array from the first nifti file
- **dims2** (*int array*) – The dimension array from the second nifti file
- **numskip1** (*int, optional*) – Number of timepoints skipped at the beginning of file 1
- **numskip2** (*int, optional*) – Number of timepoints skipped at the beginning of file 2

Returns
ismatched – True if the time dimension of the two files match.

Return type
bool

rapidtide.io.checkifparfile

rapidtide.io.**checkifparfile**(*filename*)

Checks to see if a file is an FSL style motion parameter file

Parameters
filename (*str*) – The name of the file in question.

Returns
isparfile – True if filename ends in '.par', False otherwise.

Return type
bool

rapidtide.io.readparfile

rapidtide.io.**readparfile**(*filename*)

Checks to see if a file is an FSL style motion parameter file

Parameters
filename (*str*) – The name of the file in question.

Returns
motiondict – All the timecourses in the file, keyed by name

Return type
dict

rapidtide.io.readmotion

rapidtide.io.**readmotion**(*filename*)

Reads motion regressors from filename (from the columns specified in colspec, if given)

Parameters

- **filename** (*str*) – The name of the file in question.
- **colspec** (*str, optional*) – The column numbers from the input file to use for the 6 motion regressors

Returns
motiondict – All the timecourses in the file, keyed by name

Return type
dict

rapidtide.io.calcmotregressors

rapidtide.io.**calcmotregressors**(*motiondict*, *start=0*, *end=- 1*, *position=True*, *deriv=True*, *derivdelayed=False*)

Calculates various motion related timecourses from motion data dict, and returns an array

Parameters
motiondict (*dict*) – A dictionary of the 6 motion direction vectors

Returns
motionregressors – All the derivative timecourses to use in a numpy array

Return type
array

rapidtide.io.sliceinfo

rapidtide.io.**sliceinfo**(*slicetimes*, *tr*)

rapidtide.io.getslicetimesfromfile

rapidtide.io.**getslicetimesfromfile**(*slicetimename*)

rapidtide.io.readbidssidecar

rapidtide.io.**readbidssidecar**(*inputfilename*)

Read key value pairs out of a BIDS sidecar file

Parameters
inputfilename (*str*) – The name of the sidecar file (with extension)

Returns
thedict – The key value pairs from the json file

Return type
dict

rapidtide.io.writedicttojson

rapidtide.io.**writedicttojson**(*thedict*, *thefilename*)

Write key value pairs to a json file

Parameters

- **thedict** (*dict*) – The key value pairs from the json file
- **thefilename** (*str*) – The name of the json file (with extension)

rapidtide.io.readdictfromjson

rapidtide.io.**readdictfromjson**(*inputfilename*)

Read key value pairs out of a json file

Parameters
inputfilename (*str*) – The name of the json file (with extension)

Returns
thedict – The key value pairs from the json file

Return type
dict

rapidtide.io.readlabelledtsv

rapidtide.io.**readlabelledtsv**(*inputfilename*)

Read time series out of an fmriprep confounds tsv file

Parameters
inputfilename (*str*) – The root name of the tsv (no extension)

Returns
confounddict – All the timecourses in the file, keyed by the first row

Return type
dict

NOTE: If file does not exist or is not valid, return an empty dictionary

rapidtide.io.readoptionsfile

rapidtide.io.**readoptionsfile**(*inputfileroot*)

rapidtide.io.writebidstsv

rapidtide.io.**writebidstsv**(*outputfileroot*, *data*, *samplerate*, *compressed=True*, *columns=None*, *starttime=0.0*, *append=False*, *colsinjson=True*, *colsintsv=False*, *omitjson=False*, *debug=False*)

NB: to be strictly valid, a continuous BIDS tsv file (i.e. a "_physio" or "_stim" file) requires: 1) The .tsv is compressed (.tsv.gz) 2) "SamplingFrequency", "StartTime", "Columns" must exist and be in the .json file 3) The tsv file does NOT have column headers. 4) "_physio" or "_stim" has to be at the end of the name, although this seems a little flexible

The first 3 are the defaults, but if you really want to override them, you can.

Parameters

- **outputfileroot** –
- **data** –
- **samplerate** –
- **compressed** –
- **columns** –
- **starttime** –
- **append** –
- **colsinjson** –
- **colsintsv** –
- **omitjson** –
- **debug** –

Returns

rapidtide.io.readvectorsfromtextfile

rapidtide.io.**readvectorsfromtextfile**(*fullfilespec*, *onecol=False*, *debug=False*)

Read one or more time series from some sort of text file

Parameters

- **fullfilespec** (*str*) –

  The file name. If extension is .tsv or .json, it will be assumed to be either a BIDS tsv, or failing that,
  a non-BIDS tsv. If any other extension or no extension, it will be assumed to be a text file.

- **colspec** (*A valid list and/or range of column numbers, or list of column names, or None*) –

- **debug** (*bool*) – Output additional debugging information

Returns

- **samplerate** (*float*) – Sample rate in Hz. None if not knowable.
- **starttime** (*float*) – Time of first point, in seconds. None if not knowable.
- **columns** (*str array*) – Names of the timecourses contained in the file. None if not knowable.
- **data** (*2D numpy array*) – Timecourses from the file
- **compressed** (*bool*) – True if time data is gzipped (as in a .tsv.gz file).
- **filetype** (*str*) – One of "text", "plaintsv", "bidscontinuous".

NOTE: If file does not exist or is not valid, all return values are None

rapidtide.io.readbidstsv

rapidtide.io.**readbidstsv**(*inputfilename*, *colspec=None*, *warn=True*, *debug=False*)

Read time series out of a BIDS tsv file

Parameters

- **inputfilename** (*str*) – The root name of the tsv and accompanying json file (no extension)
- **colspec** (*list*) – A comma separated list of column names to return
- **debug** (*bool*) – Output additional debugging information

Returns

- **samplerate** (*float*) – Sample rate in Hz
- **starttime** (*float*) – Time of first point, in seconds
- **columns** (*str array*) – Names of the timecourses contained in the file
- **data** (*2D numpy array*) – Timecourses from the file

NOTE: If file does not exist or is not valid, all return values are None

rapidtide.io.readcolfrombidstsv

rapidtide.io.**readcolfrombidstsv**(*inputfilename*, *columnnum=0*, *columnname=None*, *debug=False*)

Parameters

- **inputfilename** –
- **columnnum** –
- **columnname** –

rapidtide.io.parsefilespec

rapidtide.io.**parsefilespec**(*filespec*, *debug=False*)

rapidtide.io.colspectolist

rapidtide.io.**colspectolist**(*colspec*, *debug=False*)

rapidtide.io.processnamespec

rapidtide.io.**processnamespec**(*maskspec*, *spectext1*, *spectext2*)

rapidtide.io.readcolfromtextfile

rapidtide.io.**readcolfromtextfile**(*inputfilespec*)

Parameters

- **inputfilename** –
- **colspec** –

rapidtide.io.readvecs

rapidtide.io.**readvecs**(*inputfilename*, *colspec=None*, *numskip=0*, *debug=False*, *thedtype=<class 'float'>*)

Parameters
inputfilename –

rapidtide.io.readvec

rapidtide.io.**readvec**(*inputfilename*, *numskip=0*)

Read an array of floats in from a text file.

Parameters
inputfilename (*str*) – The name of the text file

Returns
inputdata – The data from the file

Return type
1D numpy float array

rapidtide.io.readtc

rapidtide.io.**readtc**(*inputfilename*, *colnum=None*, *colname=None*, *debug=False*)

rapidtide.io.readlabels

rapidtide.io.**readlabels**(*inputfilename*)

Parameters
inputfilename –

rapidtide.io.writedict

rapidtide.io.**writedict**(*thedict*, *outputfile*, *lineend=''*, *machinereadable=False*)

Write all the key value pairs from a dictionary to a text file.

Parameters

- **thedict** (*dict*) – A dictionary
- **outputfile** (*str*) – The name of the output file
- **lineend** (*{ 'mac', 'win', 'linux' }, optional*) – Line ending style to use. Default is 'linux'.

rapidtide.io.readdict

rapidtide.io.**readdict**(*inputfilename*)

Read key value pairs out of a text file

Parameters
inputfilename (*str*) – The name of the json file (with extension)

Returns
thedict – The key value pairs from the json file

Return type
dict

rapidtide.io.writevec

rapidtide.io.**writevec**(*thevec*, *outputfile*, *lineend=''*)

Write a vector out to a text file. :param thevec: The array to write. :type thevec: 1D numpy or python array :param outputfile: The name of the output file :type outputfile: str :param lineend: Line ending style to use. Default is 'linux'. :type lineend: { 'mac', 'win', 'linux' }, optional

rapidtide.io.writenpvecs

rapidtide.io.**writenpvecs**(*thevecs*, *outputfile*, *lineend=''*)

Write out a two dimensional numpy array to a text file

Parameters

- **thevecs** (*1D or 2D numpy array*) – The data to write to the file
- **outputfile** (*str*) – The name of the output file
- **lineend** (*{ 'mac', 'win', 'linux' }, optional*) – Line ending style to use. Default is 'linux'.

2.34.6 rapidtide.miscmath: Miscellaneous math functions

rapidtide.miscmath.phase(mcv)	Return phase of complex numbers.
rapidtide.miscmath.polarfft(invec, samplerate)	**param invec**
rapidtide.miscmath.complex_cepstrum(x)	**param x**
rapidtide.miscmath.real_cepstrum(x)	**param x**
rapidtide.miscmath.thederiv(y)	**param y**
rapidtide.miscmath.primes(n)	**param n**
rapidtide.miscmath.largestfac(n)	**param n**
rapidtide.miscmath.znormalize(vector)	**param vector**
rapidtide.miscmath.stdnormalize(vector)	**param vector**
rapidtide.miscmath.varnormalize(vector)	**param vector**
rapidtide.miscmath.pcnormalize(vector)	**param vector**
rapidtide.miscmath.ppnormalize(vector)	**param vector**
rapidtide.miscmath.corrnormalize(thedata[, ...])	**param thedata**
rapidtide.miscmath.rms(vector)	**param vector**
rapidtide.miscmath.envdetect(Fs, inputdata)	**param Fs** Sample frequency in Hz.

rapidtide.miscmath.phase

rapidtide.miscmath.**phase**(*mcv*)

Return phase of complex numbers.

Parameters
mcv (*complex array*) – A complex vector

Returns
phase – The phase angle of the numbers, in radians

Return type
float array

rapidtide.miscmath.polarfft

rapidtide.miscmath.**polarfft**(*invec*, *samplerate*)

Parameters

- **invec** –
- **samplerate** –

rapidtide.miscmath.complex_cepstrum

rapidtide.miscmath.**complex_cepstrum**(*x*)

Parameters
x –

rapidtide.miscmath.real_cepstrum

rapidtide.miscmath.**real_cepstrum**(*x*)

Parameters
x –

rapidtide.miscmath.thederiv

rapidtide.miscmath.**thederiv**(*y*)

Parameters
y –

rapidtide.miscmath.primes

rapidtide.miscmath.**primes**(*n*)

Parameters
n –

rapidtide.miscmath.largestfac

rapidtide.miscmath.**largestfac**(*n*)

Parameters
n –

rapidtide.miscmath.znormalize

rapidtide.miscmath.**znormalize**(*vector*)

Parameters
vector –

rapidtide.miscmath.stdnormalize

rapidtide.miscmath.**stdnormalize**(*vector*)

Parameters
vector –

rapidtide.miscmath.varnormalize

rapidtide.miscmath.**varnormalize**(*vector*)

Parameters
vector –

rapidtide.miscmath.pcnormalize

rapidtide.miscmath.**pcnormalize**(*vector*)

Parameters
vector –

rapidtide.miscmath.ppnormalize

rapidtide.miscmath.**ppnormalize**(*vector*)

Parameters
vector –

rapidtide.miscmath.corrnormalize

rapidtide.miscmath.**corrnormalize**(*thedata*, *detrendorder=1*, *windowfunc='hamming'*)

Parameters

- **thedata** –
- **detrendorder** –
- **windowfunc** –

rapidtide.miscmath.rms

rapidtide.miscmath.**rms**(*vector*)

Parameters
vector –

rapidtide.miscmath.envdetect

rapidtide.miscmath.**envdetect**(*Fs*, *inputdata*, *cutoff=0.25*)

Parameters

- **Fs** (*float*) – Sample frequency in Hz.
- **inputdata** (*float array*) – Data to be envelope detected
- **cutoff** (*float*) – Highest possible modulation frequency

Returns
envelope – The envelope function

Return type
float array

2.34.7 rapidtide.resample: Resampling functions

rapidtide.resample.congrid(xaxis, loc, val, ...)	Perform a convolution gridding operation with a Kaiser-Bessel or Gaussian kernel of width 'width'.
rapidtide.resample.doresample(orig_x, ...[, ...])	Resample data from one spacing to another.
rapidtide.resample.arbresample(inputdata, ...)	**param inputdata**
rapidtide.resample.dotwostepresample(orig_x, ...)	**param orig_x**
rapidtide.resample.calcsliceoffset(sotype, ...)	**param sotype**
rapidtide.resample.timeshift(inputtc, ...[, ...])	**param inputtc**
rapidtide.resample.FastResampler(timeaxis, ...)	

Methods

rapidtide.resample.congrid

rapidtide.resample.**congrid**(*xaxis*, *loc*, *val*, *width*, *kernel='kaiser'*, *cyclic=True*, *debug=False*)

Perform a convolution gridding operation with a Kaiser-Bessel or Gaussian kernel of width 'width'. Grid parameters are cached for performance.

Parameters

- **xaxis** (*array-like*) – The target axis for resampling
- **loc** (*float*) – The location, in x-axis units, of the sample to be gridded
- **val** (*float*) – The value to be gridded
- **width** (*float*) – The width of the gridding kernel in target bins
- **kernel** (*{'old', 'gauss', 'kaiser'}, optional*) – The type of convolution gridding kernel. Default is 'kaiser'.
- **cyclic** (*bool, optional*) – When True, gridding wraps around the endpoints of xaxis. Default is True.
- **debug** (*bool, optional*) – When True, output additional information about the gridding process

Returns

- **vals** (*array-like*) – The input value, convolved with the gridding kernel, projected on to x axis points
- **weights** (*array-like*) – The values of convolution kernel, projected on to x axis points (used for normalization)
- **indices** (*array-like*) – The indices along the x axis where the vals and weights fall.

Notes

See IEEE TRANSACTIONS ON MEDICAL IMAGING. VOL. IO.NO. 3, SEPTEMBER 1991

rapidtide.resample.doresample

rapidtide.resample.**doresample**(*orig_x*, *orig_y*, *new_x*, *method='cubic'*, *padlen=0*, *antialias=False*, *debug=False*)

Resample data from one spacing to another. By default, does not apply any antialiasing filter.

Parameters

- **orig_x** –
- **orig_y** –
- **new_x** –
- **method** –
- **padlen** –

rapidtide.resample.arbresample

rapidtide.resample.**arbresample**(*inputdata*, *init_freq*, *final_freq*, *intermed_freq=0.0*, *method='univariate'*, *antialias=True*, *decimate=False*, *debug=False*)

Parameters

- **inputdata** –
- **init_freq** –
- **final_freq** –
- **intermed_freq** –
- **method** –
- **antialias** –
- **decimate** –
- **debug** –

rapidtide.resample.dotwostepresample

rapidtide.resample.**dotwostepresample**(*orig_x*, *orig_y*, *intermed_freq*, *final_freq*, *method='univariate'*, *antialias=True*, *debug=False*)

Parameters

- **orig_x** –
- **orig_y** –
- **intermed_freq** –
- **final_freq** –
- **method** –

- **debug** –

Return type
resampled_y

rapidtide.resample.calcsliceoffset

rapidtide.resample.**calcsliceoffset**(*sotype*, *slicenum*, *numslices*, *tr*, *multiband=1*)

Parameters

- **sotype** –
- **slicenum** –
- **numslices** –
- **tr** –
- **multiband** –

rapidtide.resample.timeshift

rapidtide.resample.**timeshift**(*inputtc*, *shifttrs*, *padtrs*, *doplot=False*, *debug=False*)

Parameters

- **inputtc** –
- **shifttrs** –
- **padtrs** –
- **doplot** –

rapidtide.resample.FastResampler

class rapidtide.resample.**FastResampler**(*timeaxis*, *timecourse*, *padtime=30.0*, *upsampleratio=100*, *doplot=False*, *debug=False*, *method='univariate'*)

Methods

yfromx	

__init__(*timeaxis*, *timecourse*, *padtime=30.0*, *upsampleratio=100*, *doplot=False*, *debug=False*, *method='univariate'*)

2.34.8 rapidtide.stats: Statistical functions

rapidtide.stats.fitjsbpdf(thehist, histlen, ...)	**param thehist**
rapidtide.stats.getjohnsonppf(percentile, ...)	**param percentile**
rapidtide.stats.sigFromDistributionData(...)	**param vallist**
rapidtide.stats.rfromp(fitfile, thepercentiles)	**param fitfile**
rapidtide.stats.tfromr(r, nsamps[, ...])	**param r**
rapidtide.stats.zfromr(r, nsamps[, ...])	**param r**
rapidtide.stats.fisher(r)	**param r**
rapidtide.stats.gethistprops(indata, histlen)	**param indata**
rapidtide.stats.makehistogram(indata, histlen)	**param indata**
rapidtide.stats.makeandsavehistogram(indata, ...)	**param indata**
rapidtide.stats.symmetrize(a[, ...])	**param a**
rapidtide.stats.getfracval(datamat, thefrac)	**param datamat**
rapidtide.stats.makepmask(rvals, pval, ...)	**param rvals**
rapidtide.stats.getfracvals(datamat, thefracs)	**param datamat**
rapidtide.stats.getfracvalsfromfit_old(...)	**param histfit**
rapidtide.stats.getfracvalsfromfit(histfit, ...)	**param histfit**
rapidtide.stats.makemask(image[, threshpct, ...])	**param image** The image data to generate the mask for.

rapidtide.stats.fitjsbpdf

rapidtide.stats.**fitjsbpdf**(*thehist*, *histlen*, *thedata*, *displayplots=False*, *nozero=False*)

Parameters

- **thehist** –
- **histlen** –
- **thedata** –
- **displayplots** –
- **nozero** –

rapidtide.stats.getjohnsonppf

rapidtide.stats.**getjohnsonppf**(*percentile*, *params*, *zeroterm*)

Parameters

- **percentile** –
- **params** –
- **zeroterm** –

rapidtide.stats.sigFromDistributionData

rapidtide.stats.**sigFromDistributionData**(*vallist*, *histlen*, *thepercentiles*, *displayplots=False*, *twotail=False*, *nozero=False*, *dosighistfit=True*)

Parameters

- **vallist** –
- **histlen** –
- **thepercentiles** –
- **displayplots** –
- **twotail** –
- **nozero** –
- **dosighistfit** –

rapidtide.stats.rfromp

rapidtide.stats.**rfromp**(*fitfile*, *thepercentiles*)

Parameters

- **fitfile** –
- **thepercentiles** –

rapidtide.stats.tfromr

rapidtide.stats.**tfromr**(*r*, *nsamps*, *dfcorrfac=1.0*, *oversampfactor=1.0*, *returnp=False*)

Parameters

- **r** –
- **nsamps** –
- **dfcorrfac** –
- **oversampfactor** –
- **returnp** –

rapidtide.stats.zfromr

rapidtide.stats.**zfromr**(*r*, *nsamps*, *dfcorrfac=1.0*, *oversampfactor=1.0*, *returnp=False*)

Parameters

- **r** –
- **nsamps** –
- **dfcorrfac** –
- **oversampfactor** –
- **returnp** –

rapidtide.stats.fisher

rapidtide.stats.**fisher**(*r*)

Parameters

r –

rapidtide.stats.gethistprops

rapidtide.stats.**gethistprops**(*indata*, *histlen*, *refine=False*, *therange=None*, *pickleft=False*, *peakthresh=0.33*)

Parameters

- **indata** –
- **histlen** –
- **refine** –
- **therange** –
- **pickleftpeak** –

rapidtide.stats.makehistogram

rapidtide.stats.**makehistogram**(*indata*, *histlen*, *binsize=None*, *therange=None*, *refine=False*, *normalize=False*)

Parameters

- **indata** –
- **histlen** –
- **binsize** –
- **therange** –
- **refine** –
- **normalize** –

rapidtide.stats.makeandsavehistogram

rapidtide.stats.**makeandsavehistogram**(*indata*, *histlen*, *endtrim*, *outname*, *binsize=None*, *displaytitle='histogram'*, *displayplots=False*, *refine=False*, *therange=None*, *normalize=False*, *dictvarname=None*, *thedict=None*, *saveasbids=False*, *append=False*, *debug=False*)

Parameters

- **indata** –
- **histlen** –
- **endtrim** –
- **outname** –
- **displaytitle** –
- **displayplots** –
- **refine** –
- **therange** –
- **normalize** –
- **dictvarname** –
- **thedict** –

rapidtide.stats.symmetrize

rapidtide.stats.**symmetrize**(*a*, *antisymmetric=False*, *zerodiagonal=False*)

Parameters

- **a** –
- **antisymmetric** –
- **zerodiagonal** –

rapidtide.stats.getfracval

rapidtide.stats.**getfracval**(*datamat*, *thefrac*, *nozero=False*)

Parameters

- **datamat** –
- **thefrac** –

rapidtide.stats.makepmask

rapidtide.stats.**makepmask**(*rvals*, *pval*, *sighistfit*, *onesided=True*)

Parameters

- **rvals** –
- **pval** –
- **sighistfit** –
- **onesided** –

rapidtide.stats.getfracvals

rapidtide.stats.**getfracvals**(*datamat*, *thefracs*, *nozero=False*, *debug=False*)

Parameters

- **datamat** –
- **thefracs** –
- **displayplots** –
- **nozero** –
- **debug** –

rapidtide.stats.getfracvalsfromfit_old

rapidtide.stats.**getfracvalsfromfit_old**(*histfit*, *thefracs*, *numbins=2000*, *displayplots=False*)

Parameters

- **histfit** –
- **thefracs** –
- **numbins** –
- **displayplots** –

rapidtide.stats.getfracvalsfromfit

rapidtide.stats.**getfracvalsfromfit**(*histfit*, *thefracs*)

Parameters

- **histfit** –
- **thefracs** –
- **displayplots** –

rapidtide.stats.makemask

rapidtide.stats.**makemask**(*image*, *threshpct=25.0*, *verbose=False*, *nozero=False*, *noneg=False*)

Parameters

- **image** (*array-like*) – The image data to generate the mask for.
- **threshpct** (*float*) – Voxels with values greater then threshpct of the 98th percentile of voxel values are preserved.
- **verbose** (*bool*) – If true, print additional debugging information.
- **nozero** (*bool*) – If true, exclude zero values when calculating percentiles
- **noneg** (*bool*) – If true, exclude negative values when calculating percentiles

Returns

themask – An int16 mask with dimensions matching the input. 1 for voxels to preserve, 0 elsewhere

Return type

array-like

2.34.9 rapidtide.util: Utility functions

rapidtide.util.logmem([msg])	Log memory usage with a logging object.
rapidtide.util.findexecutable(command)	**param command**
rapidtide.util.isexecutable(command)	**param command**
rapidtide.util.savecommandline(theargs, thename)	**param theargs**
rapidtide.util.valtoindex(thearray, thevalue)	**param thearray** An ordered list of values (does not need to be equally spaced)
rapidtide.util.progressbar(thisval, end_val)	**param thisval**
rapidtide.util.makelaglist(lagstart, lagend, ...)	**param lagstart**
rapidtide.util.version()	
rapidtide.util.timefmt(thenumber)	**param thenumber**
rapidtide.util.proctiminginfo(thetimings[, ...])	**param thetimings**

rapidtide.util.logmem

rapidtide.util.**logmem**(*msg=None*)

Log memory usage with a logging object.

Parameters

msg (*str or None, optional*) – A message to include in the first column. If None, the column headers are logged. Default is None.

rapidtide.util.findexecutable

rapidtide.util.**findexecutable**(*command*)

Parameters

command –

rapidtide.util.isexecutable

rapidtide.util.**isexecutable**(*command*)

Parameters
command –

rapidtide.util.savecommandline

rapidtide.util.**savecommandline**(*theargs*, *thename*)

Parameters

- **theargs** –
- **thename** –

rapidtide.util.valtoindex

rapidtide.util.**valtoindex**(*thearray*, *thevalue*, *evenspacing=True*, *discrete=True*, *discretization='round'*, *debug=False*)

Parameters

- **thearray** (*array-like*) – An ordered list of values (does not need to be equally spaced)
- **thevalue** (*float*) – The value to search for in the array
- **evenspacing** (*boolean, optional*) – If True (default), assume data is evenly spaced for faster calculation.
- **discrete** (*boolean, optional*) – If True make the index an integer (round by default).
- **discretization** (*string, optional*) – Select rounding method - floor, ceiling, or round(default)

Returns
closestidx – The index of the sample in thearray that is closest to val

Return type
int

rapidtide.util.progressbar

rapidtide.util.**progressbar**(*thisval*, *end_val*, *label='Percent'*, *barsize=60*)

Parameters

- **thisval** –
- **end_val** –
- **label** –
- **barsize** –

rapidtide.util.makelaglist

rapidtide.util.**makelaglist**(*lagstart*, *lagend*, *lagstep*)

Parameters

- **lagstart** –
- **lagend** –
- **lagstep** –

rapidtide.util.version

rapidtide.util.**version**()

rapidtide.util.timefmt

rapidtide.util.**timefmt**(*thenumber*)

Parameters
thenumber –

rapidtide.util.proctiminginfo

rapidtide.util.**proctiminginfo**(*thetimings*, *outputfile=''*, *extraheader=None*)

Parameters

- **thetimings** –
- **outputfile** –
- **extraheader** –

2.35 Example gallery

2.35.1 Run showxcorrx workflow

Calculate and display crosscorrelation between two random timeseries.

```
# sphinx_gallery_thumbnail_number = 2
```

Start with the necessary imports

```
from os import remove

import matplotlib.pyplot as plt
import numpy as np

from rapidtide.scripts import showxcorrx
```

Arguments

```
np.random.seed(314)
dat1 = np.random.random(500)
f1 = "f1.txt"
np.savetxt(f1, dat1)

np.random.seed(42)
dat2 = np.random.random(500)
f2 = "f2.txt"
np.savetxt(f2, dat2)

samplerate = 2.0
```

Let's plot the timeseries first

```
fig, ax = plt.subplots(figsize=(16, 8))
ax.plot(dat1, color="red", alpha=0.7)
ax.plot(dat2, color="blue", alpha=0.7)
fig.show()
```

Now let's run it

```
showxcorrx.main(f1, f2, samplerate)
```

Clean up

```
remove(f1)
remove(f2)
```

Total running time of the script: (0 minutes 0.000 seconds)

2.36 Contributing to rapidtide

This document explains how to set up a development environment for contributing to rapidtide and code style conventions we follow within the project. For a more general guide to rapidtide development, please see our contributing guide. Please also remember to follow our code of conduct.

2.36.1 Style Guide

Code

Docstrings should follow numpydoc convention. We encourage extensive documentation.

The code itself should follow PEP8 convention as much as possible, with at most about 500 lines of code (not including docstrings) per file*.

* obviously some of the existing files don't conform to this - working on it...

Pull Requests

We encourage the use of standardized tags for categorizing pull requests. When opening a pull request, please use one of the following prefixes:

- **[ENH]** for enhancements
- **[FIX]** for bug fixes
- **[TST]** for new or updated tests
- **[DOC]** for new or updated documentation
- **[STY]** for stylistic changes
- **[REF]** for refactoring existing code

Pull requests should be submitted early and often! If your pull request is not yet ready to be merged, please also include the **[WIP]** prefix. This tells the development team that your pull request is a "work-in-progress", and that you plan to continue working on it.

2.36.2 A note on current coding quality and style

This code has been in active development since June of 2012. This has two implications. The first is that it has been tuned and refined quite a bit over the years, with a lot of optimizations and bug fixes - most of the core routines have been tested fairly extensively to get rid of the stupidest bugs. I find new bugs all the time, but most of the showstoppers seem to be gone. The second result is that the coding style is all over the place. When I started writing this, I had just moved over from C, and it was basically a mental port of how I would write it in C (and I do mean just "C". Not C++, C#, or anything like that. You can literally say my old code has no Class (heh heh)), and was extremely unpythonic (I've been told by a somewhat reliable source that looking over some of my early python efforts "made his eyes bleed"). Over the years, as I've gone back and added functions, I periodically get embarrassed and upgrade things to a somewhat more modern coding style. I even put in some classes - that's what the cool kids do, right? But the pace of that effort has to be balanced with the fact that when I make major architectural changes, I tend to break things. So be patient with me, and keep in mind that you get what you pay for, and this cost you nothing! Function before form.

2.37 Theory of operation

If you're bored enough or misguided enough to be reading this section, you are my intended audience!

2.37.1 rapidtide

What is rapidtide trying to do?

Rapidtide attempts to separate an fMRI or NIRS dataset into two components - a single timecourse that appears throughout the dataset with varying time delays and intensities in each voxel, and everything else. We and others have observed that a large proportion of the "global mean signal", commonly referred to as "physiological noise" seen throughout in vivo datasets that quantify time dependant fluctuations in hemodynamic measures can be well modelled by a single timecourse with a range of time shifts. This has been seen in fMRI and NIRS data recorded througout the brain and body, with time lags generally increasing at locations farther from the heart along the vasculature. This appears to be a signal carried by the blood, as changes in blood oxygenation and/or volume that propagate with bulk blood flow. The source of the signal is not known, being variously attributed to cardiac and respiratory changes over time, changes in blood CO2, gastric motility, and other sources (for a survey, see [Tong2019].) As biology is complicated, it's probably some mixture of these sources and others that we may not have considered. No matter what the source of the signal, this model can be exploited for a number of purposes.

If you're interested in hemodynamics, using rapidtide to get the time delay in every voxel gives you a lot of information that's otherwise hard or impossible to obtain noninvasively, namely the arrival time of blood in each voxel, and the fraction of the variance in that voxel that's accounted for by that moving signal, which is related to regional CBV (however there's also a factor that's due to blood oxygenation, so you have to interpret it carefully). You can use this information to understand the blood flow changes arising from vascular pathology, such as stroke or moyamoya disease, or to potentially see changes in blood flow due to a pharmacological intervention. In this case, the moving signal is not noise - it's the signal of interest. So the various maps rapidtide produces can be used to describe hemodynamics.

However, if you are interested in local rather than global hemodynamics, due to, say, neuronal activation, then this moving signal is rather pernicious in-band noise. Global mean regression is often used to remove it, but this is not optimal - in fact it can generate spurious anticorrelations, which are not helpful. Rapidtide will regress out the moving signal, appropriately delayed in each voxel. This gives you better noise removal, and also avoids generating spurious correlations. For a detailed consideration of this, look here [Erdogan2016].

How does rapidtide work?

In order to perform this task, rapidtide does a number of things:

1. Obtain some initial estimate of the moving signal.
2. Preprocess this signal to emphasize the bloodborne component.
3. Analyze the signal to find and correct, if possible, non-ideal properties that may confound the estimation of time delays.
4. Preprocess the incoming dataset to determine which voxels are suitable for analysis, and to emphasize the bloodborne component.
5. Determine the time delay in each voxel by finding the time when the voxel timecourse has the maximum similarity to the moving signal.
6. Optionally use this time delay information to generate a better estimate of the moving signal.
7. Repeat steps 3-7 as needed.
8. Parametrize the similarity between the moving signal and each voxels' timecourse, and save these metrics.
9. Optionally regress the voxelwise time delayed moving signal out of the original dataset.

Each of these steps has nuances which will be discussed below.

Initial Moving Signal Estimation

Moving Signal Preprocessing

Moving Signal Massaging

Dataset Preprocessing

Time delay determination

Generating a Better Moving Signal Estimate

Lather, Rinse, Repeat

Save Useful Parameters

Regress Out the Moving Signal

2.38 Release history

2.38.1 Version 2.2.6 (5/17/22)

- (fingerprint) Various fixes to mask handling.
- (package) Staged some prep work on updating the setup/installation files.

2.38.2 Version 2.2.5 (4/26/22)

- (rapidtide) Postprocess timing information to make it more useful.
- (rapidtide) Reenabled numba by default.
- (fingerprint) Fixed handling of 4D atlases, empty regions, and 4D masks. Added "constant" template, and allow 0th order processing (mean).
- (atlastood) Fixed 4D atlas handling. Now mask atlas after collapsing to 3D.
- (histnifti) Added –transform flag to map values to percentiles.

2.38.3 Version 2.2.4 (4/11/22)

- (fingerprint) Now works properly for 3D input files.
- (tidepool) Turned the default level of verbosity way down, but gave you the ability to crank it back up.
- (RapidtideDataset.py) Fixed the default type of "numberofpasses".

2.38.4 Version 2.2.3 (4/1/22)

- (rapidtide) Added a new feature, –globalmeanselect, to try to locate a good, uniform, short delay pool of voxels to use for the initial global mean signal. This is an attempt to fix the "poison regressor" problem - if the initial regressor contains data from multiple, distinct pools of voxels with different delays, the initial global regressor is strongly autocorrelated, and delay fits become ambiguous. This cannot be corrected by refinement, so better to avoid it altogether. This option selects only voxels with clear, short delays, after a single pass with despeckling disabled. The result is a mask (XXXdesc-globalmeanpreselect_mask.nii.gz) that can be used with –globalmeanincludemask for a subsequent run.
- (rapidtide) Fixed a nasty bug that caused offsettime and lagminthresh to interact incorrectly, sometimes leading to almost no voxels for refinement.
- (happy) Moved some code around, changed some internal names, and added secret bits to support future, secret, features.
- (tidepool) Trying to add a little more clarity to the user about image orientation (the image's affine transform is correct, so the mapping between voxel and MNI coordinate is correct, but currently it's not clear if displayed images are radiological or neurological orientation.
- (fingerprint) Added the JHU atlases as options.
- (package) Added slightly modified version of the JHU arterial territorial atlases to the reference section (Paper: https://doi.org/10.1101/2021.05.03.442478, Download: https://www.nitrc.org/projects/arterialatlas).
- (Docker) Fixed a dependency problem for pyfftw (resolves https://github.com/bbfrederick/rapidtide/issues/79)
- (pony) One time offer, today only - every user gets a pony upon request!

2.38.5 Version 2.2.2 (3/16/22)

- (happy, happy_legacy, simdata) This release corrects a flaw (or perhaps more accurately an ambiguity) in slice time specification. In FSL slicetime files, slicetimes are specified in fractions of a TR. In .json sidecars, they are specified in seconds. This is now detected on file read, and slicetime files are now converted to seconds. Until now, happy and simdata assumed all slice times were in seconds. This will fix behavior when FSL-style (fractional TR) slicetime files are used. Behavior with .json sidecars is not changed. Non-json files are assumed to be the FSL style (fractions of a TR) UNLESS the –slicetimesareinseconds flag is used.

2.38.6 Version 2.2.1 (3/16/22)

- (rapidtide) Tweaked mask checking logic to address a bug introduced by despeckling changes.
- (histtc, histnifti) Harmonized options between the programs.
- (Docker) Updated Dockerfile to fix a bug that caused automatic build to fail.

2.38.7 Version 2.2.0 (3/11/22)

- (rapidtide) Major rethink of despeckling. Despeckling no longer causes negative correlation values when bipolar fitting is not enabled, and voxel parameters are only updated in a despeckled voxel if the correlation fit succeeds. This results in better fits without mysteriously unfit voxels.
- (showxy) Bland-Altman plots can now use alternative formatting, as per Krouwer, J. S. Why Bland–Altman plots should use X, not (Y+X)/2 when X is a reference method. Stat Med 27, 778–780 (2008).
- (fingerprint) This program is now substantially more useful, working on 4D input files. Output files are more convenient as well.

- (cleandirs) Cleandirs now keeps cleaning until it runs out of old installations to remove.

2.38.8 Version 2.1.2 (1/10/22)

- (calcicc, calctexticc) Some fixes to indexing.

2.38.9 Version 2.1.1 (11/4/21)

- (spatialmi, calcicc) Major improvements in performance, stability, and flexibility.
- (showtc) Added support for files with large star time offsets.
- (showxy) Some appearance tweaks.
- (niftidecomp) Improved mask generation.
- (variabilityizer) New progam to transform fMRI datasets to variability measures.

2.38.10 Version 2.1.0 (9/21/21)

- (spatialmi) Added new program to calculate local mutual information between 3D images.
- (calcicc) Tool to calculate ICC(3,1) - quickly - for a set of 3D images.
- (correlate.py) Fixed the reference for mutual_info_2d.
- (package) Simplified and cleaned up release process.

2.38.11 Version 2.0.9 (8/26/21)

- (rapidtide) Fixed a strange edge case that could lead to "hot pixels" in the maxcorr map.
- (io) Added a "tolerance" for spatial mapping of niftis to account for rounding errors in header creation.

2.38.12 Version 2.0.8 (8/20/21)

- (rapidtide) Disabled processing of abbreviated arguments.
- (showtc) Suppressed some unnecessary output when not in debug mode.
- (Docker) Added automatic build and push as a github action.

2.38.13 Version 2.0.7 (8/19/21)

- (reference) Include the new JHU digital arterial territory atlas.
- (Docker) Updated container to Python 3.9.
- (package) General cleanup of imports.

2.38.14 Version 2.0.6 (8/16/21)

- (package) Merged Taylor Salo's PR that fixes the documentation builds on readthedocs (THANK YOU!) and cleans up and centralizes requirement specifications.

2.38.15 Version 2.0.5 (8/9/21)

- (package) Further Windows compatibility fixes.
- (documentation) Updated USAGE.rst for the current naming and syntax of rapidtide.

2.38.16 Version 2.0.4 (7/28/21)

- (package) Fixed a problem where any program using util.py wouldn't run on Windows.
- (roisummarize) New addition to the package.
- (CI) Fixed a bug in the document environment.

2.38.17 Version 2.0.3 (7/16/21)

- (spatialdecomp, temporaldecomp) Improved the consistency between the programs.
- (showxcorrx) Fixed some command line options.
- (package) Began to clean up and unify text output formatting.
- (package) Addressed some numpy deprecation warnings.
- (all scripts) Corrected file permissions (this may matter on Windows installations).
- (docs) Fixed some typos.
- (showtc) Enhanced debugging output.
- (testing) Tweaked circleci configuration file to update environment prior to installation.

2.38.18 Version 2.0.2 (6/10/21)

- (rapidtide) Did you know that in python 3.8 and above, the default multiprocessing method is "spawn" rather than "fork"? Did you know the subtle differences? Do you know that that breaks rapidtide? I didn't, now I do, and now it doesn't.
- (rapidtide) Made some tweaks to the timing logger to improve output formatting.
- (rapidtide, happy) Tested on M1. The tests run more than twice as fast on an M1 mac mini with 8GB of RAM as on a 2017 MBP with a 2.9 GHz Quad-Core Intel Core i7. Emulated. Yow. When I get a native anaconda installation going, watch out.
- (happy, Docker) Now require tensorflow 2.4.0 or above to address a security issue.

2.38.19 Version 2.0.1 (6/8/21)

- (showxcorrx, plethquality, resamp1tc, simdata, happy, rapidtide) Cleaned up, improved, and unified text file reading and writing.
- (showxcorrx) Various functionality improvements.
- (package) Added options for timecourse normalization and zeropadding correlations.
- (documentation) Further cleanup.
- (Docker) Various fixes to versioning and other internals.

2.38.20 Version 2.0 (6/2/21)

Much thanks to Taylor Salo for his continuing contributions, with several substantive improvements to code, documentation, and automatic testing, and generally helping devise a sensible release roadmap that made this version possible.

This release is a big one - there are many new programs, new capabilities in existing programs, and workflow breaking syntax changes. However, this was all with the purpose of making a beter package with much more consistent interfaces that allow you to figure out pretty quickly how to get the programs to do exactly what you want. The biggest change is to rapidtide itself. For several years, there have been two versions of rapidtide; rapidtide2 (the traditional version), and rapidtide2x (the experimental version for testing new features). When features became stable, I migrated them back to rapidtide2, more and more quickly as time went by, so they became pretty much the same. I took the 2.0 release as an opportunity to do some cleanup. As of now, there is only one version of rapidtide, with two parsers. If you call "rapidtide", you get the spiffy new option parser and much more rational and consistent option naming and specification. This is a substantial, but simple, change. For compatibility with old workflows, I preserved the old parser, which is called "rapidtide2x_legacy". This accepts options just as rapidtide2 and rapidtide2x did in version 1.9.6. There is only one rapidtide routine. Once the arguments are all read in and processed, "rapidtide" and "rapidtide2x_legacy" call the same processing workflow. However, in addition to the new parser, there are completely new options and capabilities in rapidtide, but you can only get to them using the new parser. This is my way of subtly forcing you to change your workflows if you want the new shiny, without pulling the rug out from under you. "rapidtide2x_legacy" WILL be going away though, so change over when you can. Also - all outputs now conform to BIDS naming conventions to improve compatibility with other packages. Use the "–legacyoutput" flag to get the old output file names.

- (rapidtide2x_legacy): Added deprecation warning.
- (rapidtide): The correlation function has been replaced by a more flexible "similarity function". There are currently 3 options: "correlation" (the old method), "mutualinfo", which uses a cross mutual information function, and "hybrid", which uses the correlation function, but disambiguates which peak to use by comparing the mutual information for each peak.
- (rapidtide) Pulled a number of default values into global variables so that defaults and help strings will stay in sync.
- (rapidtide) Fixed text file (nirs) processing.
- (rapidtide) Fixed a search range setting error.
- (rapidtide) Fixed the default method for global mean signal generation.
- (rapidtide) Added the '–negativegradient' option in response to https://github.com/bbfrederick/rapidtide/issues/67
- (rapidtide) Added flexibility to regressor input (can use multicolumn and BIDS text files).
- (rapidtide, tidepool) Fixed reading and writing the globalmean mask.
- (rapidtide) Gracefully handle refinement failure.
- (rapidtide) Added a new method for generating global signal using PCA.

- (rapidtide) Did some prep work to implement echo cancellation.
- (rapidtide) Added workaround for occasional MLE PCA component estimation failure (this seems to be an unresolved scikit-learn problem as of 0.23.2)
- (rapidtide) Significant enhancement to PCA refinement options.
- (rapidtide) Rapidtide can now run refinement passes until the change in the probe regressor falls below a specified mean square difference. Set –convergencethresh to a positive number to invoke this (0.0005 is good). Rapidtide will refine until the M.S.D. falls below this value, or you hit maxpasses (use –maxpasses NUM to set - default is 15). This implements the procedure used in Champagne, A. A., et al., NeuroImage 187, 154–165 (2019).
- (rapidtide) The PCA refinement algorithm has been improved to match the method described in Champagne, et al., and is now the default.
- (rapidtide, io) Significant improvement to CIFTI handling - now properly read and write parcellated scalars and time series.
- (rapidtide) Completely revamped CIFTI I/O. Should now read and write native CIFTI2 files (do not need to convert to NIFTI-2 in workbench).
- (rapidtide) Better handling of motion files.
- (rapidtide) Added coherence calculation. Not quite working right yet.
- (rapidtide, happy) Switched to using nilearn's mask generator for automatic mask generation, since it's much more sophisticated. It seems to be a big improvement, and handles data processed by fmriprep and SPM with no fiddling.
- (rapidtide, happy) General improvement of output of floating point numbers. Limit to 3 decimal places.
- (rapidtide) Use logging module for output.
- (rapidtide, rapidtide_legacy) Options file is now always saved as a json.
- (rapidtide) Added ability to autochoose an appropriate spatial filter by setting –spatialfilt to a negative value.
- (rapidtide, rapidtide2x_legacy) The options file is now always saved in .json format.
- (rapidtide) BIDS format output naming and file structures have been updated to be more compliant with the standard.
- (rapidtide) Fixed a longstanding bug which used an unnecessarily stringent amplitude threshold for selecting voxels to use for refinement.
- (rapidtide) Improvements to processing in "bipolar" mode.
- (rapidtide): The getopt argument parser has been completely rewritten using argparse. The way you specify many (most?) options has changed.
- (rapidtide): Any option that takes additional values (numbers, file names, etc.) is now specified as '–option VALUE [VALUE [VALUE...]]' rather than as '–option=VALUE[,VALUE[,VALUE...]]'.
- (rapidtide): After a lot of use over the years, I've reset a lot of defaults to reflect typical usage. You can still do any analysis you were doing before, but it may now require changes to scripts and workflows to get the old default behavior. For most cases you can get good analyses with a minimum set of command line options now.
- (rapidtide): There are two new macros, –denoise and –delaymapping, which will set defaults to good values for those use cases in subjects without vascular pathology. Any of the preset values for these macros can be overridden with command line options.
- (rapidtide, rapidtide2x_legacy): Regressor and data filtering has been changed significantly. While the nominal filter passbands are the same, the transitions to the stopbands have been tightened up quite a bit. This is most noticable in the LFO band. The pasband is still from 0.01-0.15Hz with a trapezoidal rolloff, but the upper stopband now starts at 0.1575Hz instead of 0.20Hz. The wide transition band was letting in a significant amount

of respiratory signal for subjects with low respiratory rates (about half of my subjects seem to breath slower than the nominal adult minimum rate of 12 breaths/minute).

- (rapidtide): The -V, -L, -R and -C filter band specifiers have been retired. Filter bands are now specified with '–filterband XXX', where XXX is vlf, lfo, lfo_legacy, resp, cardiac, or None. 'lfo' is selected by default (LFO band with sharp transition bands). To skip filtering, use '–filterband None'. '–filterband lfo_legacy' will filter to the LFO band with the old, wide transition bands.
- (rapidtide): To specify an arbitrary filter, specify the pass freqs with –filterfreqs, and then optionally the stop freqs with –filterstopfreqs (otherwise the stop freqs will be calculated automatically from the pass freqs).
- (rapidtide): The method for specifying the lag search range has changed. '-r LAGMIN,LAGMAX' has been removed. You now use '–searchrange LAGMIN LAGMAX'
- (rapidtide): The method for specifying bipolar correlation search has changed. '-B' is replaced by '–bipolar'.
- (rapidtide): The method for specifying a fixed delay (no correlation lag search) has changed. '-Z DELAYVAL' is replaced by '–fixdelay DELAYVAL'.
- (rapidtide,rapidtide2x_legacy): The 'timerange' option is now handled properly. This can be used to restrict processing to a portion of the datafile. This is useful to get past initial transients if you didn't remove them in preprocessing, or to see if parameters change over the course of a long acquisition.
- (rapidtide): The multiprocessing code path can be forced on, even on a single processor.
- (rapidtide): Multiprocessing can be disabled on a per-routine basis.

Happy also got a new parser and BIDS outputs. You can call happy with the old interface by calling "happy_legacy".

- (happy) Output files now follow BIDS naming convention.
- (happy) Code cleanup, improved tensorflow selection.
- (happy) Fixed logmem calls to work with new logging structure (and not crash immediately).
- (happy) Fixed a very subtle bug when an externally supplied pleth waveform doesn't start at time 0.0 (fix to issue #59).
- (happy) General formatting improvements.
- (happy) Added new tools for slice time generation.
- (happy) Added support for scans where there is circulating contrast.

General Changes to the entire package:

- (package) Python 2.7 support is now officially ended. Cleaned out compatiblity code.
- (package) Dropped support for python 3.3-3.5 and added 3.9.
- (package) Made pyfftw and numba requirements.
- (package) Significantly increased test coverage by including smoke tests (exercise as many code paths as possible to find crashers in neglected code - this is how the above bugs were found).
- (package) Automated consistent formatting. black now runs automatically on file updates.
- (package) General cleanup and rationalization of imports. isort now runs automatically on file updates.
- (package) Fixed a stupid bug that surfaced when reading in all columns of a text file as input.
- (package) Merged tsalo's PR starting transition to new logging output.
- (package) Started to phase out sys.exit() calls in favor of raising exceptions.
- (package) Updated all headers and copyright lines.
- (package) Trimmed the size of the installation bundle to allow deployment on pypi.

- (package) Copied Taylor Salo's improvements to build and deployment from the master branch.
- (package) Renamed some test data for consistency.
- (package) Began effort with T. Salo to address linter errors and generally improve PEP8 conformance - remove dead code, rationalize imports, improve docstrings, convert class names to CamelCase, use snake_case for functions.
- (package) Cleaned up imports and unresolved references
- (package) Addressed many linter issues, updated deprecated numpy and scipy calls.
- (package) readvectorsfromtextfile now handles noncompliant BIDS timecourse files.
- (package) Implemented new, generalized text/tsv/bids text file reader with column selection (readvectorsfromtextfile).
- (package) Significant internal changes to noncausalfilter.
- (package) CircleCI config files changed to keep tests from stepping on each other's caches (thanks to Taylor Salo).

Changes to the Docker container builds:

- (Docker) Fixed some problems with task dispatch and container versioning.
- (Docker) Improvements to version specification, entry point.
- (Docker) Update package versions.
- (Docker) Install python with mamba for ~10x speedup.
- (Docker) Switched to current method for specifying external mounts in the documentation.
- (Docker) Improved build scripts.
- (Docker) Containers are now automatically pushed to dockerhub after build.

Documentation changes:

- (documentation) Fixed errors in documentation files that caused errors building in readthedocs.
- (documentation) New "theory of operation" section for rapidtide. Still working on it.
- (documentation) The documentation has been expanded and revised to reflect the current state of rapidtide with significant reorganization, reformatting, cleanup and additions

Tidepool:

- (tidepool) Reorganized interface, ability to show dynamic correlation movies, much clearer histogram graphs.
- (tidepool) Tidepool now gracefully handles runs with more than 4 passes. The timecourses displayed are prefilt, postfilt, pass1, pass2, pass(N-1) and pass(N).
- (tidepool) Fixed to work with Big Sur (macOS 11).
- (tidepool) Corrected sample rate for regressor timecourses.
- (tidepool) Revised to properly handle new BIDS naming conventions for output files.
- (tidepool) You can now turn out-of-range map transparency on and off (it's off by default).
- (tidepool) Internal colortable code is now much cleaner.
- (tidepool): Now properly handles missing timecourses properly. Some cosmetic fixes.

Miscellaneous changes:

- (aligntcs, applydlfilter, pixelcomp, plethquality, resamp1tc, showstxcorr, showxcorrx) Fixed matplotlib backend initialization to allow headless operation.
- (glmfilt, linfit, temporaldecomp, spatialdecomp): Argument parsers were rewritten in argparse, main routines were moved into workflows.
- (applydlfilter, atlasaverage, ccorrica, filttc, happy2std, histnifti, histtc, pixelcomp, plethquality, rapidtide2std, resamp1tc, showarbcorr, showtc, showxcorrx, simdata, spatialfit) Argument parsers were rewritten in argparse.
- (rapidtide, showxcorrx, showarbcorr, showstxcorr, ccorrica, aligntcs, filttc) Changed argument handling for arbitrary filters. Specify pass freqs with –filterfreqs, stop freqs with –filterstopfreqs.
- (happy, rapidtide) Now properly handle negative mklthreads specification.
- (physiofreq): New program to get the average frequency of a physiological waveform.
- (showtc): Some cleanup in option specification.
- (showtc) Converted text inputs to standardized code.
- (atlastool) Major fixes to functionality with 4D template files.
- (atlastool) Fixed some import and syntax issues with numpy.
- (showxcorrx) Significant cleanup for maximum flexibility and utility.
- (showxcorr) Renamed to showxcorr_legacy
- (linfit) Renamed to polyfitim to better reflect it's function.
- (histnifti) Major upgrade to functionality.
- (showarbcorr) New program to do crosscorrelations on timecourses with different samplerates.
- (polyfitim) New program to fit a spatial template to a 3D or 4D NIFTI file.
- (endtidalproc) New program to extract end tidal CO2 or O2 waveforms from a gas exhalation recording.
- (dlfilter) Made diagnostics more informative to help get dlfilter enabled.
- (simdata) Complete overhaul - new parser better checks, more flexible input formats.
- (io.py) Vastly improved reading in arbitrarily large text files.
- (stats.py) Fixed a bug in getfracvals when you try to find the maximum value.
- (RapidtideDataset.py) Better handling of different file names.

2.38.21 Version 2.0alpha29 (6/1/21)

- (Docker) Fixed some problems with task dispatch and container versioning.
- (rapidtide) Pulled a number of default values into global variables so that defaults and help strings will stay in sync.
- (documentation) Reorganization and significant updates.
- (documentation) Fixed errors in documentation files that caused errors building in readthedocs.
- (package) Updated versioneer.

2.38.22 Version 1.9.6 (6/1/21)

- (Docker) ACTUALLY fixed some problems with task dispatch and container versioning.
- (package) Updated versioneer.

2.38.23 Version 2.0alpha28 (5/27/21)

- (testing) Synced function calls with some internal changes to Correlator to fix the tests crashing.

2.38.24 Version 1.9.5 (5/27/21)

- (Docker) Fixed some problems with task dispatch and container versioning.

2.38.25 Version 2.0alpha27 (5/27/21)

- (rapidtide, showxcorrx, showarbcorr, showstxcorr, ccorrica, aligntcs, filttc) Changed argument handling for arbitrary filters. Specify pass freqs with –filterfreqs, stop freqs with –filterstopfreqs.
- (happy) Code cleanup, improved tensorflow selection.
- (Docker) Improvements to version specification, entry point.
- (testing) Increased coverage.
- (packaging) Multiple corrections in support files.

2.38.26 Version 2.0alpha26 (5/5/21)

- (happy, rapidtide) Now properly handle negative mklthreads specification.
- (Dockerfile) Update package versions.
- (Docker container) Added happy test.
- (package) Further increased test coverage.

2.38.27 Version 2.0alpha25 (5/3/21)

- (rapidtide) Fixed text file (nirs) processing.
- (rapidtide) Fixed a search range setting error.
- (rapidtide) Fixed the default method for global mean signal generation.
- (rapidtide) Fixed a crash when using the mutualinfo similarity metric.
- (rapidtide, io) Significant improvement to CIFTI handling - now properly read and write parcellated scalars and time series.
- (io) Vastly improved reading in arbitrarily large text files.
- (stats) Fixed a bug in getfracvals when you try to find the maximum value.
- (package) Began aggressive implementation of smoke tests (excercise as many code paths as possible to find crashers in neglected code - this is how the above bugs were found).
- (package) More logging refinement.

2.38.28 Version 2.0alpha24 (4/14/21)

- (rapidtide) Added the '–negativegradient' option in response to https://github.com/bbfrederick/rapidtide/issues/67
- (rapidtide) Added flexibility to regressor input (can use multicolumn and BIDS text files).

2.38.29 Version 2.0alpha23 (4/14/21)

- (happy) Fixed logmem calls to work with new logging structure (and not crash immediately).

2.38.30 Version 2.0alpha22 (4/13/21)

- (rapidtide, tidepool) Fixed reading and writing the globalmean mask.
- (package) Fixed a stupid bug that surfaced when reading in all columns of a text file as input (really, this time).

2.38.31 Version 2.0alpha21 (4/12/21)

- (rapidtide) Gracefully handle refinement failure.
- (happy) Fixed some output timecourse naming.
- (atlastool) Major fixes to functionality with 4D template files.
- (aligntcs, applydlfilter, pixelcomp, plethquality, resamp1tc, showstxcorr, showxcorrx) Fixed matplotlib backend initialization to allow headless operation.
- (package) General cleanup and rationalization of imports. isort now used by default.
- (package) Dropped support for python 3.3-3.5
- (package) Fixed a stupid bug that surfaced when reading in all columns of a text file as input.
- (package) Merged tsalo's PR starting transition to new logging output.
- (package) Fixed environment for py39 testing.
- (package) Started to phase out sys.exit() calls in favor of raising exceptions.
- (rapidtide) Corrected BIDS naming of intermediate maps.

2.38.32 Version 2.0alpha20 (3/28/21)

- (package) Python 2.7 support is now officially ended. Cleaned out compatiblity code.
- (package) Made pyfftw and numba requirements.
- (docs) Wrote general description of text input functions, enhanced description of happy, include examples.
- (style) Began effort with T. Salo to address linter errors and generally improve PEP8 conformance - remove dead code, rationalize imports, improve docstrings, convert class names to CamelCase, use snake_case for functions.
- (showtc) Converted text inputs to standardized code.

2.38.33 Version 2.0alpha19 (3/26/21)

- (showxcorrx) Significant cleanup for maximum flexibility and utility.
- (showxcorr) Renamed to showxcorr_legacy
- (linfit) Renamed to polyfitim to better reflect it's function.
- (histnifti) Major upgrade to functionality.
- (showxcorrx, atlasaverage, happy2std, rapidtide2std, spatialfit, applydlfilter, plethquality, histnifti) Moved to argparse.
- (package) Updated all headers and copyright lines.

2.38.34 Version 2.0alpha18 (3/17/21)

- (package) Trimmed the size of the installation bundle (really, truly, correctly this time).

2.38.35 Version 1.9.4 (3/17/21)

- (package) T. Salo made a number of changes to allow pypi deployment.
- (package) Fixed a problem in setup.py that causes installation to fail.
- (rapidtide2x) Backported a critical fix from the dev version so that the refinement threshold is properly set with null correlations (the result is that the refine mask rejects fewer voxels, and gives a better regressor estimate).

2.38.36 Version 2.0alpha17 (3/17/21)

- (package) Trimmed the size of the installation bundle (maybe even correctly this time).

2.38.37 Version 2.0alpha16 (3/17/21)

- (package) Trimmed the size of the installation bundle.
- (various) Cleaned up imports and unresolved references

2.38.38 Version 2.0alpha15 (3/17/21)

- (rapidtide) Added a new method for generating global signal using PCA.
- (all) Further imports from master branch to improve deployment.
- (simdata) Complete overhaul - new parser better checks, more flexible input formats.
- (io.py) Improvements to readvecs to handle very large files.
- (ccorrica, filttc, histtc, pixelcomp, resamp1tc) Facelift, cleanup, new parsers.
- (testing) Removed python 2.7 CI build.
- (all) Addressed many linter issues, updated deprecated numpy and scipy calls.

2.38.39 Version 2.0alpha14 (3/1/21)

- (all) readvectorsfromtextfile now handles noncompliant BIDS timecourse files.
- (happy) Fixed a very subtle bug when an externally supplied pleth waveform doesn't start at time 0.0 (fix to issue #59).
- (filttc, histtc, showarbcorr) parser improvements.

2.38.40 Version 2.0alpha13 (2/22/21)

- (package) Copied Taylor Salo's improvements to build and deployment from the master branch.
- (all) Ran all python files through Black to give consistent formatting (really, truly this time).
- (all) Implemented new, generalized text/tsv/bids text file reader with column selection (readvectorsfromtextfile).
- (atlastool) Fixed some import and syntax issues with numpy.
- (showarbcorr) New program to do crosscorrelations on timecourses with different samplerates.
- (happy) Fixed column selection bug with BIDS files.
- (happy) General formatting improvements.
- (dlfilter) Made diagnostics more informative to help get dlfilter enabled.

2.38.41 Version 2.0alpha12 (2/5/21)

- (all) Fixed readbidstsv calls.
- (all) Beginning a rethink of a universal text timecourse reader.
- (happy) Added new tools for slice time generation.

2.38.42 Version 2.0alpha11 (1/5/21)

- (rapidtide) Rolled back default similarity metric to 'correlation' from 'hybrid'. 'hybrid' works very well most of the time, and fails strangely occasionally. When 'correlation' fails, it does so in more predictable and explicable ways.
- (happy) Restored functionality and options for motion regression that I broke when separating out the command parser.
- (tests) CircleCI config files changed to keep tests from stepping on each other's caches (thanks to Taylor Salo).

2.38.43 Version 2.0alpha10 (12/21/20)

- (package) Ran all python files through Black to give consistent formatting.
- (rapidtide) Did some prep work to implement echo cancellation.

2.38.44 Version 2.0alpha9 (12/9/20)

- (rapidtide) Added workaround for occasional MLE PCA component estimation failure (this seems to be an unresolved scikit-learn problem as of 0.23.2)

2.38.45 Version 2.0alpha8 (12/9/20)

- (rapidtide) Significant enhancement to PCA refinement options.
- (tidepool) Tidepool now gracefully handles runs with more than 4 passes. The timecourses displayed are prefilt, postfilt, pass1, pass2, pass(N-1) and pass(N).
- (happy) Added support for scans where there is circulating contrast.
- (happy, rapidtide2x, rapidtide) The parsers are now being properly installed during setup.
- (package) Renamed some test data for consistency.

2.38.46 Version 2.0alpha7 (12/1/20)

- (rapidtide) Rapidtide can now run refinement passes until the change in the probe regressor falls below a specified mean square difference. Set –convergencethresh to a positive number to invoke this (0.0005 is good). Rapidtide will refine until the M.S.D. falls below this value, or you hit maxpasses (use –maxpasses NUM to set - default is 15). This implements the procedure used in Champagne, A. A., et al., NeuroImage 187, 154–165 (2019).
- (rapidtide) The PCA refinement algorithm has been improved to match the method described in Champagne, et al., and is now the default.

2.38.47 Version 2.0alpha6 (11/30/20)

- (rapidtide) Completely revamped CIFTI I/O. Should now read and write native CIFTI2 files (do not need to convert to NIFTI-2 in workbench).
- (rapidtide) Better handling of motion files.
- (rapidtide) Added coherence calculation. Not quite working right yet.
- (happy) Started adding BIDS output.
- (tidepool) Fixed to work with Big Sur (macOS 11).

2.38.48 Version 2.0alpha5 (10/29/20)

Much thanks to Taylor Salo for his continuing contributions, with several substantive improvements to code, documentation, and automatic testing, and generally helping devise a sensible release roadmap.

- (rapidtide, happy) Switched to using nilearn's mask generator for automatic mask generation, since it's much more sophisticated. It seems to be a big improvement, and handles data processed by fmriprep and SPM with no fiddling.
- (rapidtide, happy) General improvement of output of floating point numbers. Limit to 3 decimal places.
- (rapidtide) Use logging module for output.
- (rapidtide, rapidtide_legacy) Options file is now always saved as a json.
- (rapidtide) Added ability to autochoose an appropriate spatial filter by setting –spatialfilt to a negative value.

- (rapidtide_parser) Code cleanup and formatting fixes.
- (documentation) Much reorganization, reformatting and cleanup.
- (documentation) New "theory of operation" section for rapidtide. Still working on it.

2.38.49 Version 2.0alpha4 (10/26/20)

- rapidtide2x has been renamed to rapidtide2x_legacy
- (rapidtide, rapidtide2x) The options file is now always saved in .json format.
- (rapidtide) BIDS format output naming and file structures have been updated to be more compliant with the standard.
- (RapidtideDataset.py) Better handling of different file names.
- (documentation) The documentation has been expanded and revised to reflect the current state of rapidtide.
- (all) Code cleanup.

2.38.50 Version 2.0alpha3 (10/19/20)

- (rapidtide) Fixed sample rate on BIDS regressor outputs.
- (tidepool) Corrected sample rate for regressor timecourses.
- (Docker) Switched to current method for specifying external mounts in the documentation.
- (tests) Fixed test_filter.py to remove bad test.
- (tests) Added test_delayestimation.py to try to get end to end validation on the core of rapidtide.

2.38.51 Version 2.0alpha2 (10/19/20)

- (all) Significant internal changes to noncausalfilter.
- (rapidtide) Fixed a longstanding bug which used an unnecessarily stringent amplitude threshold for selecting voxels to use for refinement.
- (rapidtide) Improvements to processing in "bipolar" mode.
- (rapidtide) Internal improvements to mutual information normalization.
- (rapidtide) New –bidsoutput option to make all output files BIDS compatible in naming and format.
- (tidepool) Revised to properly handle new naming conventions for output files.
- (tidepool) You can now turn out-of-range map transparency on and off (it's off by default).
- (tidepool) Internal colortable code is now much cleaner.
- (Docker) Improved build scripts.

2.38.52 Version 2.0alpha1 (8/24/20)

- (all): Python 2.x is no longer supported. To be fair, I've done nothing to break 2.x compatibility on purpose, so it probably still works, but I'm expending no effort to keep it working.
- (documentation): General updates and cleanups.
- (rapidtide2): rapidtide2 has been eliminated. If you used it before, you can use rapidtide2x as a dropin replacement (but you really should start moving to using rapidtide, the new version that is actively being developed).
- (rapidtide2x): rapidtide2x has been deprecated and replaced by rapidtide (which is basically rapidtide2x v1.9.3 with a different argument parser and default option values).
- (rapidtide2x): Added deprecation warning.
- (rapidtide): The correlation function has been replaced by a more flexible "similarity function". There are currently 3 options: "correlation" (the old method), "mutualinfo", which uses a cross mutual information function, and "hybrid", the new default, which uses the correlation function, but disambiguates which peak to use by comparing the mutual information for each peak.
- (rapidtide): Changed the default peak fit type to "fastquad", which does a parabolic fit to the peaks to refine location.
- (rapidtide): The getopt argument parser has been completely rewritten using argparse. The way you specify many (most?) options has changed.
- (rapidtide): Any option that takes additional values (numbers, file names, etc.) is now specified as '–option VALUE [VALUE [VALUE...]]' rather than as '–option=VALUE[,VALUE[,VALUE...]]'.
- (rapidtide): After a lot of use over the years, I've reset a lot of defaults to reflect typical usage. You can still do any analysis you were doing before, but it may now require changes to scripts and workflows to get the old default behavior. For most cases you can get good analyses with a minimum set of command line options now.
- (rapidtide): There are two new macros, –denoise and –delaymapping, which will set defaults to good values for those use cases in subjects without vascular pathology. Any of the preset values for these macros can be overridden with command line options.
- (rapidtide, rapidtide2x): Regressor and data filtering has been changed significantly. While the nominal filter passbands are the same, the transitions to the stopbands have been tightened up quite a bit. This is most noticable in the LFO band. The pasband is still from 0.01-0.15Hz with a trapezoidal rolloff, but the upper stopband now starts at 0.1575Hz instead of 0.20Hz. The wide transition band was letting in a significant amount of respiratory signal for subjects with low respiratory rates (about half of my subjects seem to breath slower than the nominal adult minimum rate of 12 breaths/minute).
- (rapidtide): The -V, -L, -R and -C filter band specifiers have been retired. Filter bands are now specified with '–filterband XXX', where XXX is vlf, lfo, lfo_legacy, resp, cardiac, or None. 'lfo' is selected by default (LFO band with sharp transition bands). To skip filtering, use '–filterband None'. '–filterband lfo_legacy' will filter to the LFO band with the old, wide transition bands.
- (rapidtide): To specify an arbitrary filter, use '–filterfreqs LOWERPASS UPPERPASS [LOWERSTOP UPPERSTOP]'. If you don't specify the stop bands, the stop frequencies are set to 5% below and above LOWERPASS and UPPERPASS, respectively.
- (rapidtide): The method for specifying the lag search range has changed. '-r LAGMIN,LAGMAX' has been removed. You now use '–searchrange LAGMIN LAGMAX'
- (rapidtide): The method for specifying bipolar correlation search has changed. '-B' is replaced by '–bipolar'.
- (rapidtide): The method for specifying a fixed delay (no correlation lag search) has changed. '-Z DELAYVAL' is replaced by '–fixdelay DELAYVAL'.
- (rapidtide): Options file is saved in json by default now.

- (rapidtide,rapidtide2x): The 'timerange' option is now handled properly. This can be used to restrict processing to a portion of the datafile. This is useful to get past initial transients if you didn't remove them in preprocessing, or to see if parameters change over the course of a long acquisition.
- (physiofreq): New program to get the average frequency of a physiological waveform.
- (tidepool): Now properly handles missing timecourses properly. Some cosmetic fixes.
- (showtc): Converted to argparse, some cleanup in option specification.
- (glmfilt, linfit, temporaldecomp, spatialdecomp): Argument parsers were rewritten, main routines were moved into workflows.
- (docker container): Fixed some build errors, now pushes container to dockerhub.
- (rapidtide): Multiprocessing can be forced on, even on a single processor.
- (rapidtide): Multiprocessing can be disabled on a per-routine basis.

2.38.53 Version 1.9.3 (7/30/20)

- Bumped version number because I forgot to commit a file

2.38.54 Version 1.9.2 (7/30/20)

- (all): Changed over to using versioneer to handle version numbers.
- (rapidtide2, rapidtide2x, rapidtide_2x_trans, rapidtideX) Runtimings file now has additional version information.

2.38.55 Version 1.9.1 (6/17/20)

- (all): Documentation improvements.
- (all): Many internal changes to support future argument specifications.
- (all): Backported bugfixes from the development version.
- (rapidtide2x) Fixed specification of timerange.
- (docker): Fixed an incompatibility in versions between pyfftw and scipy (thank you to Niranjana Shashikumar for reporting the bug and providing the solution!)
- (docker): Improved container labelling.
- (docker): Cleaned up container build.
- (tidepool): Various fixes and improvements backported from the development version.

2.38.56 Version 1.9 (1/6/20)

- (all): Now compatible with nibabel 3.x
- (all): Significantly expanded test suite. Code coverage is now at 47%.
- (documentation): Added instructions for installing the deep learning filter code
- (documentation): Numerous tweaks and fixes
- (docker): There is now a containerized version of the rapidtide package, which avoids a lot of installation woes

- (rapidtide2x, showxcorrx): Completely replaced correlation and fitting routines with faster, more robust (and more rigorously tested) versions
- (rapidtide2x, showxcorrx): Enhancements to the permutation methods
- (rapidtide2x, showxcorrx): Revised internals to guarantee xcorr scale matches values
- (rapidtide2x, showxcorrx): Improved fitter performance in edge cases (thin peaks, symmetric around max)
- (rapidtide2x, showxcorrx): Changed limits to avoid crash when peak is at edge of range
- (rapidtide2x, showxcorrx): Fixed some (but apparantly not all) dumb errors in calls to null correlation calculations.
- (rapidtide2x): Implemented workaround for unknown crasher in GLM filtering when nprocs != 1
- (rapidtide2x): Added experimental respiration processing
- (rapidtide2x): Fixed an uncaught bug in bipolar processing.
- (rapidtide2x): Setting ampthresh to a negative number between 0 and 1 sets the percentile of voxels to use for refinement
- (rapidtide2x): Support for new minimum sigma limit in correlation fit
- (rapidtide2x): Fixed a bug that caused fit fails on very narrow peaks, added diagnostic info
- (rapidtide2x): Putting in scaffolding to support phase randomization for null correlation calculation.
- (rapidtide2x): Properly specify correlation limits in null correlation and accheck
- (rapidtide2x): Slight modifications to pickleft routine to avoid a rare bug
- (rapidtide2x): Masks should now be properly generated for zero mean and non-positive definite data.
- (rapidtide2x): Tweaked the autocorrelation correction
- (rapidtide2x): Added an error check to avoid crashing when no significance tests are nonzero
- (rapidtide2x): Added upper and lower sigma limits to peak fitting uss to match new class
- (showxcorrx): Updated to match rapidtide2x fitting
- (showxcorrx): Enhanced error reporting
- (showxcorrx): Added width range tests
- (showxcorrx): Added norefine option, output more debugging info
- (showxcorrx): Set validity limits for gaussian fit
- (happy, rapidtide2x): Fixed a bug in output nifti header parameters (copy headers by value, not reference!)
- (happy): New multipass architecture allows much better results - initial passes set estimation masks and find potential arterial voxels.
- (happy): Aliased correlation integrated into happy as experimental feature
- (happy): Fixed a conceptual error in how I normalized projected timecourses
- (happy): Added brain cine output
- (happy): If estmask is supplied, us it. If not, generate a vessel mask and repeat final steps.
- (happy): Added option to detect and invert arterial signals, improved time output.
- (happy): Shorten pulse recon step size to 10 ms
- (happy): Better envelope handling, fixed bug in timecourse phase projection.

- (happy): Some changes to improve performance with long TRs
- (happy): Added happy paper reference
- (happy): Increased gridbins (used in phase projection): default to 2 after testing showed lower noise than 1.5 bins
- (happy): Added ability to pad cyclically rather than reflecting around the ends
- (happy): Added ability to smooth projection in the phase direction (on by default)
- (happy): Significant improvements to GLM processing (spatial and temporal versions, aliased temporal correlation)
- (happy): Added "checkpoint" option to dump more intermediate data for debugging.
- (happy): Added more progress bars, and the ability to turn them off.
- (happy): Print out version info at runtime.
- (tidepool): Major update with new functionality
- (tidepool): The probe regressor, it's spectrum, and how it was filtered are now shown in the main window
- (tidepool): Properly disable atlas buttons when no atlas is loaded, avoiding crashes
- (tidepool): Removed support for pyqt4
- (tidepool): Some UI tweaks
- (tidepool): Added some infrastructure for future support for loading multiple runs
- (tidepool): New atlases to suport fmriprep default coordinates
- (tidepool): Numerous bug fixes
- (ccorrica): Added the ability to oversample the data prior to crosscorrelation
- (showtc): Added ability to select a column from a multicolumn file as input.
- (showtc): Can now select a column from multicolumn input text files for each vector.
- (showtc): Changed the specification of colors and legends. Internal code cleanup.
- (happy2std): New tool to align happy maps
- (happywarp): Improvements in filtering
- (aligntcs): You can now specify single columns out of multicolumn files
- (showxy): Initial support for specifying color names
- (spectrogram): Cleanup, added some configuration options
- (simdata): Some reformatting, updates, and improvements
- (simdata): Put some data in the example directory to use with simdata
- (fingerprint): New addition to the library to decompose delay maps using vascular territory templates
- (fingerprint): Added canonical HCP template maps to the distribution
- (helper_classes): Added freqtrack class
- (correlate.py): Added rudimentary mutual information calculation
- (correlate.py): Multiple aliased correlation methods added, depending on demeaning.
- (io.py): Fixed support for named columns in BIDS tsvs
- (io.py): Relaxed requirements for required fields in BIDS jsons

- (io.py): Added a few convenience routines for dealing with NIFTI files
- (io.py): Fixed import of parser_funcs

2.38.57 Version 1.8 (5/10/19)

- (fit.py): The following fixes to both variants of findmaxlag_gauss affect rapidtide2, rapidtide2x, showxcorr, showxcorrx, happy, and happyx.
- (fit.py): CRITICAL FIX - edge behavior in both versions of findmaxlag_gauss was very broken.
- (fit.py): Fixed a rare failure in findmaxlag_gauss when the correlation peak is very narrow.
- (fit.py): Constrain initial gaussian fit values to be rational.
- (fit.py): Always return rational (if wrong) values when zerooutbadfit is False.
- (fit.py): Fixed a rare problem where no peaks were found in autocorrcheck, causing crash.
- (fit.py): Fixed a pernicious bug that sometimes caused mayhem when –nofitfilt was set.
- (rapidtide2, rapidtide2x): There is now sanity checking on lagmin and lagmax input using -r.
- (rapidtide2x): Masking logic has been completely redone, with numerous bugfixes, error checks, and new capabilities.
- (rapidtide2x): Added option to refine offset on leftmost lag peak (–pickleft). This helps a lot with people with vascular pathology.
- (rapidtide2x): Added ability to save options file in json.
- (rapidtide2x): Fixed a bug when timerange was used in conjunction with glm filtering.
- (rapidtide2x): Added fixes to crash where there were bad significance estimates.
- (showtc): Allow multiple linewidths.
- (showtc): Added ability to set x and y axis labels.
- (showtc): Added DPI option to set resolution of file output.
- (showxy): Changed Bland-Altman plot to use open circles.
- (showhist): Add bar histograms.
- (showhist): Added the option to set binsize to makehistogram.
- (happy, happyx): All changes in happyx have now been synced to happy - at this moment, they are identical. New changes will be tested in happyx.
- (happy, happyx): Fixed starttime and endtime selection.
- (happy, happyx): Lowered maximum heart rate to 140 by default.
- (happy, happyx): Output of info as json is optional, not default.
- (happy, happyx): Save info file as json rather than text.
- (happy, happyx): writedictasjson now supports numpy objects, added readdict function.
- (happy, happyx): Cardiac filter and search range are now specified independently.
- (happy, happyx): Removed lowerpass from cardiac estimation.
- (happy, happyx): Retrained and optimized model after revising happy paper.
- (happy, happyx): Use explicit copies to avoid variable changing out from under me.

- (happy, happyx): Added pleth/filtpleth correlation.
- (happy, happyx): Turn off variance masking by default, correlate raw and filtered waveforms.
- (happy, happyx): Numerous tweaks to resampling to perform better in edge cases.
- (happy, happyx): Fixed problem reading json physio files.
- (happy, happyx): Added ability to force the use of raw cardiac waveform for phase projection.
- (happy, happyx): Allow varmasking by volume or slice, filter prior to cardiac correlation.
- (happy, happyx): Resolved problem with definition of notch filter width.
- (happy, happyx): Corrected a type coercion error for estimation masks.
- (happy, happyx): Reduced verbosity of notch filter.
- (happy, happyx): Altered estimation mask logic.
- (happy, happyx): Added sanity checks to lag range.
- (happy, happyx): Now properly handle uncompressed bids tsv files.
- (happy, happyx): Variance and projection masks are separate, can set variance thresh percent.
- (happy, happyx): Changes in response to paper review.
- (happy, happyx): Filter cardiac waveform to slice samplerate Nyquist frequency when upsampling.
- (happy, happyx): Also added choice of centric or noncentric phase reconstruction..
- (happy, happyx): Fixed implementation of Hilbert transform phase analysis.
- (happy, happyx): Made robust to missing anatomics, started adding ants support.
- (happy, happyx): harmonic notch filter notch pct was not properly scaled.
- (happy, happyx): Now align pleth regressor with cardfromfmri.
- (happy, happyx): Fixed convolution gridding.
- (happy, happyx): Changed default gridding kernel to 3.0 wide Kaiser-Bessel.
- (happy, happyx): Reordered usage to functionally separate flags.
- (happy, happyx): Implemented workaround for strange interaction of tensorflow and MKL.
- (happy, happyx): Lots of little bugs fixed, print statements cleaned up.
- (tidepool): Added support for files in MNI152NLin2009cAsym space (fmriprep output).
- (tidepool): Fixed a crash when no atlas exists.
- (ccorrica): Modernized ccorrica to use new library calls.
- (atlasaverage, filttc, histtc, aligntcs, highresmotion): Added to the distro.
- (tests): Numerous maintenance fixes. test_findmaxlag is much more sophisticated now.
- (whole project): Code cleanup, reformatting.

2.38.58 Version 1.7 (12/5/18)

- (whole project) Stopped pretending happy doesn't exist - adding to the changelog and will start writing docs.
- (whole project) Tried to generate some workflows.
- (whole project) Update issue templates.
- (whole project) Put back some critical information that got lost in the docs reorganization.
- (happyx) Changed default minhr to 40 BPM, cleaned up specification of min/max HR.
- (happyx) Put a lower limit on the envelope function in cleancardiac to limit gain..
- (happyx) Can use weighted masks, calculate envelop normalized cardiac waveforms..
- (happyx) Fixed notch filter to always filter at least one frequency bin.
- (happyx) Added ability to skip trs in fmrifile and motion file.
- (happyx) Mad normalize slice timecourses, refactor code, add some test data.
- (happy, happyx) Moved some routines out of happy(x) into libraries, added trendfilter.
- (happy, happyx, rapidtide2, rapidtide2x) Added motion regressor filtering.
- (happyx, rapidtide2, rapidtide2x) Add high order polynomial detrending.
- (happyx) Added deep learning filter for refining cardiac waveform (off by default).
- (rapidtide2, rapidtide2x) Oversample factor was erroneously set to 0 if TR <=0.5 seconds.
- (showxcorrx) Added file output capability.
- (showxcorrx) Set verbose to False by default.
- (showxcorrx) Trimming extraneous output.
- (tidepool) First crack at fixing atlas averaging.
- (tidepool) Initialize atlasniftiname.
- (showxy) Added Bland-Altman plots with annotations, range specifications, font scaling.
- (showxy) Updated for new matplotlib interface, enabled legends.
- (showtc) Now can specify legend location.
- (showtc) Added fontscalefac option.
- (resample.py) Fixed cutoff frequency on upsample filter.
- (resample.py) Lowpass filter after upsampling.
- (fit.py) Limit peakstart and peakend to stay within legal range.
- (io.py) New additions to readvecs to specify columns.
- (dlfilter.py) added.

2.38.59 Version 1.6 (9/19/18)

- (whole project) Cleanup and reorganization (tsalo).
- (documentation) Major revisions to clean things up (tsalo).
- (workflows) Initial creation (work in progress) (tsalo).
- (testing) Reorganized and fixed - now it actually works! (tsalo).
- (coverage) Code coverage for testing is now tracked (21% - we can improve significantly with workflows) (tsalo).
- (rapidtide2, 2x, happy) Finally found (and fixed) the reason for a range of random stalls and slowdowns when running on a cluster. MKL extensions were silently distributing some numpy calculations over all cores (which means running N jobs running on a cluster tried to use N^2 cores - not good at all…). The maxiumum number of MKL threads is now settable on the command line, and defaults to 1 (no multiprocessor numpy). Strangely, this makes everything a little faster in single processor mode, and A LOT faster in multiprocessor mode.
- (tide_funcs.py) tide_funcs.py has been split into filter.py, fit.py, io.py, miscmath.py, resample.py, stats.py, and util.py. All executables fixed to match.
- (rapidtide2, 2x) Oversample factor is now set automatically by default to make the correlation timestep 0.5 or less. This dramatically improves fits for longer TRs (> 1.5 seconds).
- (rapidtide2, 2x) Moved the major passes (null correlation, correlation, correlation fit, refine, wiener filter and glm) into separate modules for maintainability and to simplify tinkering.
- (rapidtide2, 2x) Isolated multiprocessing code to make speeding up new routines easier and avoid massive code duplication.
- (rapidtide2, 2x) Fixed some bugs in correlation mask reading and saving include and exclude masks.
- (rapidtide2, 2x) Improved tmask, fixed a bug.
- (rapidtide2, 2x, glmfilt) Made glmpass more general so it could be used in other scripts)
- (resamp1tc, resample.py) Added arbresample, modified dotwostepresample.
- (fit.py) Added Kaiser Bessel window function.
- (io.py) savetonifti now properly sets output data type in header.
- (io.py) Added routine to read in motion timecourses.
- (filter.py) Consolidated doprecalcfftfilt and xfunc into transferfuncfilt.
- (filter.py) Added new “complex” spectrum option.
- (filter.py) Added docstrings, code cleanup and regularization.
- (filter.py) Added new ‘spectrum’ routine.
- (filter.py) Initial support for precalculated arb filtering.
- (resample.py) Adjusted gridding to be symmetric around output value.
- (util.py) Reimplemented valtoindex to make it faster.
- (showtc) Updated to use new spectrum routine.
- (spectrogram) added to distro.
- (rapidtide2, 2x, resamp1tc showxcorr, showxcorrx, showstxcorr) eliminated all use of Butterworth filters by default.
- (spatialfit) Fixed numpy imports

2.38.60 Version 1.5 (6/11/18)

- (documentation) Added description of rapidtide output files.
- (tide_funcs) Fixed a VERY old bug in detrend that added an offset to the detrended timecourse. The effect of the bug on rapidtide2(x) is probably small, because we almost always use data that has already been detrended. The effect in some very particular use cases, though, was large enough that I finally noticed it after 3 years.
- (rapidtide2, 2x) Added option to disable progress bars (good when saving output to a file).
- (rapidtide2, 2x) Improved output to memusage file.
- (rapidtide2, 2x) Report fit errors with better granularity.
- (rapidtide2, 2x) Allow specification of external correlation mask.
- (rapidtide2, 2x) Added "MTT" map to hopefully remove the effect of autocorrelation.
- (rapidtide2, 2x) Added some additional diagnostic information to significance estimation.
- (rapidtide2x, tide_funcs) Major changes to peak fitting to try to improve stability using new findmaxlag_gauss_rev function.
- (rapidtide2x, tide_funcs) Moved logmem function to tide_funcs so that it's available for other programs.
- (rapidtide2x) Fixed bug when running despeckling on a single processor.
- (rapidtide2, 2x) Attempting to stabilize the lagsigma measurement with better initial parameter estimates.
- (tide_funcs) Cast timecourse index as a long to avoid an overflow in NIRS timecourses.
- (rapidtide2, 2x, tide_funcs) Added ability to set searchfrac in fits.
- (rapidtide2, 2x) Disable threshold during significance estimation, simplify internal logic for turning on estimation.
- (rapitdide2) Fixed bug in mask generation
- (showtc) Added the ability to select columns to plot, and to read BIDS style json/tsv.gz physiological files
- (showtc, showxy) Updated colormap names for compatibility with matplotlib 2.2+
- (rapidtide2std) Initial support for warping with ANTs (does not work yet)
- (rapidtide2std) Added the ability to align a single file.
- (tidepool) Support for MTT map.
- (ccorrica, showstxcorr) PEP 8 reformatting.
- (testing) Added test for findmaxlag versions.
- (testing) Added test for stxcorr functions.
- (temporaldecomp) Allow 3D masks.
- (atlastool) Changed method for generating 3D files.
- (atlastool) Various bug fixes in 3D atlas generation.
- (resamp1tc) Modernized option selection, Added nodisplay option.
- (showstxcorr) Explicit integer cast of indices.
- (showstxcorr) Removed initial Hamming window.
- (showstxcorr) Added csv output of matrices.
- (linfit) Added to distribution.

- (tide_funcs) Changed value of rcond in leastsq to be compatible over multiple versions of bumpy (least. comprehensible. note. ever.)
- (tide_funcs) Added findexecutable and isexecutable functions
- (tide_funcs) Added a convolution gridding function
- (tide_funcs) Fixed readvec(s) so that is now works properly if a text file ends with an empty line.
- (tide_funcs) Added function to read slice times from a BIDS sidecar file.
- (spatialdecomp, temporaldecomp) Command line is now saved.
- (threeD) Added

2.38.61 Version 1.4.2 (2/21/18)

- (documentation) Fixed some formatting.
- (showxcorrx) Cleaned up usage statement.

2.38.62 Version 1.4.0 (2/21/18)

- (rapidtide2, 2x) Added macros to support setting multiple options at once.
- (rapidtide2, 2x) –nirs macro sets a number of parameters to be appropriate for NIRS data processing.
- (rapidtide2, 2x) –venousrefine macro sets refinement parameters to use data only from large draining veins (only works reliably in healthy subjects ATM).
- (rapidtide2, 2x) Now tabulate maximum correlation times without range limit for my own secret, diabolical purposes.
- (rapidtide2, 2x) Fixed a bug that was not shifting all of the timecourses if they were not in the refinement mask.
- (rapidtide2, 2x) Improved memory usage tracking.
- (rapidtide2, 2x) Reduced memory footprint.
- (rapidtide2, 2x) Move large arrays into shared memory for multiprocessor jobs to avoid duplicating RAM.
- (rapidtide2, 2x) You can now set the number of processors used (rather than always using all of them) when multiprocessing.
- (rapidtide2, 2x) Properly shut down worker procedures to free RAM earlier.
- (rapidtide2, 2x) Fixed a bug in the initialization of the dispersion calculation.
- (rapidtide2, 2x) Fixed a maddening bug in output of the refinement mask.
- (rapidtide2, 2x) Fixed the range on the Gaussian filtering progress bar.
- (rapidtide2, 2x) Made some improvements to the despeckling procedure.
- (rapidtide2, 2x) –refinepasses is now deprecated - use –passes instead.
- (rapidtide2, 2x) Added new methods to specify the sample rate (or sample time) of the input data file.
- (rapidtide2, 2x) Revised usage statement to make parameter names better reflect their function.
- (rapidtide2, 2x) A lot of internal code cleanup and dead code removal.
- (rapidtide2, 2x, showxcorrx) Allow specification of the correlation window function (hamming (default), hann, blackmanharris, or None).

- (showtc) Cleaned up some bugs introduced during the last overhaul.
- (tcfrom3col) Added to package (generates a timecourse from an FSL style 3 column regressor file.
- (tidepool) Default to displaying using the valid mask rather than the p<0.05 mask.
- (tidepool) Enabled usage of the refine mask.

2.38.63 Version 1.3.0 (12/15/17)

- (rapidtide2, 2x) Added new option, '–despeckle', which uses a spatial median filter to find and correct points where the correlation fit picked the wrong autocorrelation lobe. This dramatically improves the quality of the output maps. This will probably be turned on by default in the next release.
- (tidepool) FINALLY fixed the click positioning bug. Worth the update just for this. That was driving me crazy.
- (tidepool) Formatting improvements.
- (tidepool) Preliminary support for multiple territory atlases and averaging modes in tidepool.
- (tidepool) Atlas averaging is now (mostly) working.
- (rapidtide2, 2x) Now support text format NIRS datasets (2D text files) in addition to NIFTI fMRI files.
- (rapidtide2, 2x) Substantial internal changes to reduce memory footprint, improve speed.
- (rapidtide2x, showxcorrx) Initial support added for Choudry's cepstral analysis method for delay calculation.
- (showtc) Substantial improvements (improved formatting, ability to specify separate subplots, transpose input, line colors, waterfall plots, offsets between lines, titles, etc).
- (rapidtide2std) Internal code cleanup.
- (showxy) Now supports multiple input files.
- Added glmfilt to package to filter 1D or 4D data out of 4D datasets.
- Added spatialdecomp to do spatial PCA decomposition of 4D NIFTI files.
- Added temporaldecomp to do temporal PCA decomposition of 4D NIFTI files.
- Added ccorrica to the distribution to provide cross correlation matrices between all timeseries in a 2D text files.
- Added atlastool to aid in preparation of atlas files for tidepool.

2.38.64 Version 1.2.0 (6/20/17)

- New release to trigger a Zenodo DOI.
- Fully tested for python 3.6 compatibility.
- Added linfit to the distribution.
- Set a limit of 25 dispersion regressors.
- Reformatted the documentation somewhat.
- Added some recipes to the documentation for common use cases.
- Cleaned up and fixed the resampling code.
- Minor quality and speed improvement to timeshift.
- No longer output "datatoremove" to save space.
- Removed some redundant screen refreshes from tidepool.

- Reorganized and removed dead code.
- Changed default mode for calculating refined regressors to "unweighted_average".
- Synced changes in rapidtide2x to rapidtide2

2.38.65 Version 1.1.0 (4/3/17)

- I have now synced all of the changes in rapidtide2x back to rapidtide2.
- rapidtide now has multiprocessing support using the –multiproc flag. This can cause a dramatic increase in processing speed on multicore/processor machines.
- Dispersion calculation is now off by default.
- Tidepool is now compatible with both PyQt4 and 5.
- Reordered some memory allocation and deallocation to keep the RAM footprint down.
- Some additional significant speedups (support for ffftw if present, caching hamming windows).
- Added an optional cross-spectral density filter for adaptive timecourse filtering. This is a work in progress - not really ready for general use.
- Skeleton of support for Wiener deconvolution to sharpen the correlation function (off by default, not ready for general use).

2.38.66 Version 1.0.0 (2/13/17)

- I decided to stop hedging and actually commit myself - this is version 1.0.0 - out of beta!
- To promote stability, new features will be put into test versions (the name of the program will have an "x" appended). This way I can do major internal changes and have them available to users without breaking something they might rely on. The "x" versions will sync with the "normal" versions after extensive testing.
- Major new feature (rapidtide2x only for now). Multiprocessing! Significant speedup when using the –multiproc option on machines with multiple cores.
- showxcorrx has new features and defaults.
- Memory allocation has been reorganized to reduce footprint (rapidtide2x).
- Changed imports for better compatibility when running in the NITRC-CE environment (rapidtide2x).
- rapidtide2std now supports nonlinear alignment.
- histnifti is added to the distribution.
- I've added some additional outputs to rapidtide2 and rapidtide2x during refinement to help figure out if the brain is a dispersive filter. This doesn't change how rapidtide2 does the refinement - it's just informational at this point.
- Added spatialfit to the distribution. I use this to analyze delay maps. More on this later.
- Fully implemented samplerate and sampletime options (rapidtide2)
- Corrected and enhanced the use of alternative correlation weighting functions (PHAT, Liang, and Eckart weighting) (rapidtide).
- Updated all scripts for compatibility with matplotlib 2.0.
- Fixed tidepool for compatibility with the new version of pyqtgraph.
- Significant enhancements to showstxcorr (this is a work in progress).

- Example data is no longer installed in the python directory (this never made sense to do).
- Added code to do matched filtering of regressors with mean PSD and/or cross-spectral density. It works, but doesn't do much (rapidtide2x).

2.38.67 Version 0.1.9 (12/19/16)

- Added code to allow runtime memory profiling if memory_profile library is present.
- Extensive casting of variables to lower memory footprint and allow future optimizations.
- Added explicit garbage collection to reduce memory usage.
- Added optional single precision calculation mode to lower memory footprint.
- Added a second script, "rapidtide2x" where I can distribute and test new features without breaking the main code branch.
- Did some speed optimizations in findmaxlag, including faster gaussian fitting and new MUCH faster parabolic fitting (still experimental).
- Minor bug fixes, code reorganization and cleanup.

2.38.68 Version 0.1.8 (11/30/16)

- Fixed a bug in the GLM filtering code - if spatial filtering was applied in rapidtide2, the smoothed data was filtered rather than the original data.
- Added an option in rapidtide2 ("–glmsourcefile=FILE") to apply GLM filter to a different dataset than the one used to estimate the delays (this is used for HCP data - the "hp2000_clean" data has had LFO signals partially removed and may compromise delay estimation, so that should be done on the un-"FIX"ed data).
- Added the ability to detect autocorrelation properties of the test regressor that may cause delay estimation to fail with the "–accheck" flag.
- Added an option "–acfix" to try to correct for bad test regressor autocorrelation properties. This is not yet working correctly.
- Added the ability to specify a slicetimes file ("–slicetimes=FILE") if slicetime correction has not yet been applied to the dataset. Not fully tested.
- (rapidtide2std, tidepool) Added the ability to transform and display functional data to highres anatomic space in addition to MNI152 space.
- Various small bugfixes and format cleanups.

2.38.69 Version 0.1.7 (11/15/16)

- I think I've resolved the issue of crashes due to numba functioning differently on machines other than mine.
- Fixed a masking bug in tidepool that was due to numbers being very close to, but not exactly, 1.
- Made a number of internal changes to rapidtide2 and tidepool to allow dynamic significance masking (not yet working - actually failing spectacularly for the most part, but it's currently commented out).
- Added showstxcorr to the distribution.
- Added the ability to set the mask threshold for correlation and global mask inclusion (this turns out to be needed if you use non-skull-stripped data.)

- Put in some code to start to figure out how to account for dispersion in the delay function.
- Moved the "start movie" button in tidepool to better align with the numerical spin boxes.
- showtc has gotten a significant upgrade in functionality, adding the ability to display power spectra, phase spectra, and set the sample rate to make the x-axis correct.
- Lots of internal formatting/style fixes, and fixed some formatting in the usage statements and documentation.

2.38.70 Version 0.1.6 (10/15/16)

- Fixed a critical bug that had been introduced in the last round of changes to findmaxlag.
- Disabled numba for findmaxlag (it seems to cause problems for some users).
- New option –skipsighistfit to omit fitting a Johnson SB function to the significance histogram.
- Fixed the usage statement.
- Fixed a bug that set ampthresh to zero when not doing significance estimation.

2.38.71 Version 0.1.5 (10/11/16)

- Fixed a bug that made it impossible to specify –regressortstep.
- Added undocumented option –nonumba to turn off just in time compilation if there's a problem with it.
- Print rapidtide version on launch.
- Made pandas import explicit (sklearn requires it).

2.38.72 Version 0.1.4 (10/10/16)

- Some fixes to usage output.
- Added functions for fitting trapezoids (for risetime calculations).
- Changed argument parsing and option output to avoid conflicts
- Added an option to not zero out bad fits (so as not to skew lag statistics)
- Improved fitting of probability distributions.
- Better handling of failed correlation peak fits.
- Now installations should work properly if not installed using git (fixed _gittag import problem).

2.38.73 Version 0.1.3 (9/2/16)

- Added a tool (rapidtide2std) to register all output maps to MNI152 coordinates (requires FSL).
- Made a 3mm resolution ASPECTS map for use in tidepool.
- Reference data is now properly installed, and tidepool can find it reliably.
- Redid the version information. Rapidtide now records both the release version and the git hash in in the output data to help with data provenance.
- Reorganized the distribution into what seems to be a more canonical layout.
- Resolved the issues I seem to have introduced with Python 3 compatibility.

- Significantly cleaned up resampling and filtering code and improved reliability.
- Added some unit tests for critical routines. Strangely, they all fail on Travis-CI, but work on my local machine. It seems to be a numerical precision issue. The answers are rightish, just not right on Travis.

2.38.74 Version 0.1.2 (8/5/16)

- Some bug fixes in filtering and resampling code.
- Beginning to add automated tests.
- Biphasic mode is now fully implemented, including two-tailed significance calculation.

2.38.75 Version 0.1.1 (7/8/16)

- First release

2.38.76 Version 1.3.0 (12/15/17)

- (rapidtide2, 2x) Added new option, '–despeckle', which uses a spatial median filter to find and correct points where the correlation fit picked the wrong autocorrelation lobe. This dramatically improves the quality of the output maps. This will probably be turned on by default in the next release.
- (tidepool) FINALLY fixed the click positioning bug. Worth the update just for this. That was driving me crazy.
- (tidepool) Formatting improvements.
- (tidepool) Preliminary support for multiple territory atlases and averaging modes in tidepool.
- (tidepool) Atlas averaging is now (mostly) working.
- (rapidtide2, 2x) Now support text format NIRS datasets (2D text files) in addition to NIFTI fMRI files.
- (rapidtide2, 2x) Substantial internal changes to reduce memory footprint, improve speed.
- (rapidtide2x, showxcorrx) Initial support added for Choudry's cepstral analysis method for delay calculation.
- (showtc) Substantial improvements (improved formatting, ability to specify separate subplots, transpose input, line colors, waterfall plots, offsets between lines, titles, etc).
- (rapidtide2std) Internal code cleanup.
- (showxy) Now supports multiple input files.
- Added glmfilt to package to filter 1D or 4D data out of 4D datasets.
- Added spatialdecomp to do spatial PCA decomposition of 4D NIFTI files.
- Added temporaldecomp to do temporal PCA decomposition of 4D NIFTI files.
- Added ccorrica to the distribution to provide cross correlation matrices between all timeseries in a 2D text files.
- Added atlastool to aid in preparation of atlas files for tidepool.

2.38.77 Version 1.2.0 (6/20/17)

- New release to trigger a Zenodo DOI.
- Fully tested for python 3.6 compatibility.
- Added linfit to the distribution.
- Set a limit of 25 dispersion regressors.
- Reformatted the documentation somewhat.
- Added some recipes to the documentation for common use cases.
- Cleaned up and fixed the resampling code.
- Minor quality and speed improvement to timeshift.
- No longer output "datatoremove" to save space.
- Removed some redundant screen refreshes from tidepool.
- Reorganized and removed dead code.
- Changed default mode for calculating refined regressors to "unweighted_average".
- Synced changes in rapidtide2x to rapidtide2

2.38.78 Version 1.1.0 (4/3/17)

- I have now synced all of the changes in rapidtide2x back to rapidtide2.
- rapidtide now has multiprocessing support using the –multiproc flag. This can cause a dramatic increase in processing speed on multicore/processor machines.
- Dispersion calculation is now off by default.
- Tidepool is now compatible with both PyQt4 and 5.
- Reordered some memory allocation and deallocation to keep the RAM footprint down.
- Some additional significant speedups (support for fftw if present, caching hamming windows).
- Added an optional cross-spectral density filter for adaptive timecourse filtering. This is a work in progress - not really ready for general use.
- Skeleton of support for Wiener deconvolution to sharpen the correlation function (off by default, not ready for general use).

2.38.79 Version 1.0.0 (2/13/17)

- I decided to stop hedging and actually commit myself - this is version 1.0.0 - out of beta!
- To promote stability, new features will be put into test versions (the name of the program will have an "x" appended). This way I can do major internal changes and have them available to users without breaking something they might rely on. The "x" versions will sync with the "normal" versions after extensive testing.
- Major new feature (rapidtide2x only for now). Multiprocessing! Significant speedup when using the –multiproc option on machines with multiple cores.
- showxcorrx has new features and defaults.
- Memory allocation has been reorganized to reduce footprint (rapidtide2x).

- Changed imports for better compatibility when running in the NITRC-CE environment (rapidtide2x).
- rapidtide2std now supports nonlinear alignment.
- histnifti is added to the distribution.
- I've added some additional outputs to rapidtide2 and rapidtide2x during refinement to help figure out if the brain is a dispersive filter. This doesn't change how rapidtide2 does the refinement - it's just informational at this point.
- Added spatialfit to the distribution. I use this to analyze delay maps. More on this later.
- Fully implemented samplerate and sampletime options (rapidtide2)
- Corrected and enhanced the use of alternative correlation weighting functions (PHAT, Liang, and Eckart weighting) (rapidtide).
- Updated all scripts for compatibility with matplotlib 2.0.
- Fixed tidepool for compatibility with the new version of pyqtgraph.
- Significant enhancements to showstxcorr (this is a work in progress).
- Example data is no longer installed in the python directory (this never made sense to do).
- Added code to do matched filtering of regressors with mean PSD and/or cross-spectral density. It works, but doesn't do much (rapidtide2x).

2.38.80 Version 0.1.9 (12/19/16)

- Added code to allow runtime memory profiling if memory_profile library is present.
- Extensive casting of variables to lower memory footprint and allow future optimizations.
- Added explicit garbage collection to reduce memory usage.
- Added optional single precision calculation mode to lower memory footprint.
- Added a second script, "rapidtide2x" where I can distribute and test new features without breaking the main code branch.
- Did some speed optimizations in findmaxlag, including faster gaussian fitting and new MUCH faster parabolic fitting (still experimental).
- Minor bug fixes, code reorganization and cleanup.

2.38.81 Version 0.1.8 (11/30/16)

- Fixed a bug in the GLM filtering code - if spatial filtering was applied in rapidtide2, the smoothed data was filtered rather than the original data.
- Added an option in rapidtide2 ("–glmsourcefile=FILE") to apply GLM filter to a different dataset than the one used to estimate the delays (this is used for HCP data - the "hp2000_clean" data has had LFO signals partially removed and may compromise delay estimation, so that should be done on the un-"FIX"ed data).
- Added the ability to detect autocorrelation properties of the test regressor that may cause delay estimation to fail with the "–accheck" flag.
- Added an option "–acfix" to try to correct for bad test regressor autocorrelation properties. This is not yet working correctly.
- Added the ability to specify a slicetimes file ("–slicetimes=FILE") if slicetime correction has not yet been applied to the dataset. Not fully tested.

- (rapidtide2std, tidepool) Added the ability to transform and display functional data to highres anatomic space in addition to MNI152 space.
- Various small bugfixes and format cleanups.

2.38.82 Version 0.1.7 (11/15/16)

- I think I've resolved the issue of crashes due to numba functioning differently on machines other than mine.
- Fixed a masking bug in tidepool that was due to numbers being very close to, but not exactly, 1.
- Made a number of internal changes to rapidtide2 and tidepool to allow dynamic significance masking (not yet working - actually failing spectacularly for the most part, but it's currently commented out).
- Added showstxcorr to the distribution.
- Added the ability to set the mask threshold for correlation and global mask inclusion (this turns out to be needed if you use non-skull-stripped data.)
- Put in some code to start to figure out how to account for dispersion in the delay function.
- Moved the "start movie" button in tidepool to better align with the numerical spin boxes.
- showtc has gotten a significant upgrade in functionality, adding the ability to display power spectra, phase spectra, and set the sample rate to make the x-axis correct.
- Lots of internal formatting/style fixes, and fixed some formatting in the usage statements and documentation.

2.38.83 Version 0.1.6 (10/15/16)

- Fixed a critical bug that had been introduced in the last round of changes to findmaxlag.
- Disabled numba for findmaxlag (it seems to cause problems for some users).
- New option –skipsighistfit to omit fitting a Johnson SB function to the significance histogram.
- Fixed the usage statement.
- Fixed a bug that set ampthresh to zero when not doing significance estimation.

2.38.84 Version 0.1.5 (10/11/16)

- Fixed a bug that made it impossible to specify –regressortstep.
- Added undocumented option –nonumba to turn off just in time compilation if there's a problem with it.
- Print rapidtide version on launch.
- Made pandas import explicit (sklearn requires it).

2.38.85 Version 0.1.4 (10/10/16)

- Some fixes to usage output.
- Added functions for fitting trapezoids (for risetime calculations).
- Changed argument parsing and option output to avoid conflicts
- Added an option to not zero out bad fits (so as not to skew lag statistics)
- Improved fitting of probability distributions.
- Better handling of failed correlation peak fits.
- Now installations should work properly if not installed using git (fixed _gittag import problem).

2.38.86 Version 0.1.3 (9/2/16)

- Added a tool (rapidtide2std) to register all output maps to MNI152 coordinates (requires FSL).
- Made a 3mm resolution ASPECTS map for use in tidepool.
- Reference data is now properly installed, and tidepool can find it reliably.
- Redid the version information. Rapidtide now records both the release version and the git hash in in the output data to help with data provenance.
- Reorganized the distribution into what seems to be a more canonical layout.
- Resolved the issues I seem to have introduced with Python 3 compatibility.
- Significantly cleaned up resampling and filtering code and improved reliability.
- Added some unit tests for critical routines. Strangely, they all fail on Travis-CI, but work on my local machine. It seems to be a numerical precision issue. The answers are rightish, just not right on Travis.

2.38.87 Version 0.1.2 (8/5/16)

- Some bug fixes in filtering and resampling code.
- Beginning to add automated tests.
- Biphasic mode is now fully implemented, including two-tailed significance calculation.

2.38.88 Version 0.1.1 (7/8/16)

- First release

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Colin Studholme, David John Hawkes, Derek L.G. Hill (1998). “Normalized entropy measure for multi-modality image alignment”. in Proc. Medical Imaging 1998, vol. 3338, San Diego, CA, pp. 132-143.

[Tong2019] Tong, Y., Hocke, L.M., and Frederick, B.B., Low Frequency Systemic Hemodynamic “Noise” in Resting State BOLD fMRI: Characteristics, Causes, Implications, Mitigation Strategies, and Applications. Front Neurosci, 2019. 13: p. 787. | http://dx.doi.org/10.3389/fnins.2019.00787

[Erdogan2016] Erdoğan S, Tong Y, Hocke L, Lindsey K, Frederick B. Correcting resting state fMRI-BOLD signals for blood arrival time enhances functional connectivity analysis. Front. Hum. Neurosci., 28 June 2016 | http://dx.doi.org/10.3389/fnhum.2016.00311

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

P

R

S

T

U

V

W

Z