This quick-start guide covers getting the code and running basic tasks.
For a document describing the implementation see here.

1. Installation prerequisites

I'm assuming you're running on cortex. If so, you don't need to install anything.
If not (e.g. on your laptop), you should make sure to have the following installed:

- git.
    - On linux: `sudo apt-get install git`
    - On windows I recommend using gitextensions. It's a great git GUI and installs the windows port of git itself as well.
- python 2.7+ (**not** python 3) with the following libraries: numpy, scipy, matplotlib, sklearn, jinja2.
    - On linux:
      >> `sudo apt-get install python-numpy python-scipy python-matplotlib ipython ipython-notebook python-pandas python-sympy python-nose`
      >> `sudo apt-get install python-pip`
      >> `pip install -U scikit-learn`
      >> `pip install jinja2`
    - On windows a great way to start is to install winpython, which comes with the spyder IDE and all the above libraries.
    - If you work with virtualenv, then after you get the code using git (see below), you can use the file requirements.txt in the project's root folder to install all the required libraries, by running
      `pip install -r requirements.txt`

2. Getting the code and setting up the project

Start by creating a directory for the project, e.g. `~/projects/timefit`
>> `mkdir ~/projects/timefit`
>> `cd ~/projcets/timefit`

Now get the code from github to a subdirectory called `code`:
projdir>> `git clone git@github.com:ronniemaor/timfit.git code`

Create a sibling directory `data`, which holds the input data files. This includes the gene expression data and pathway/category lists.
The easiest way to do this is to link to some canonical location:
projdir>> `ln -s /cortex/ronniemaor/HTR/data data`

Two other directories - `cache` and `results` will be created for you if they don't exist, but I suggest creating them on /cortex, since they will contain large files and creating links under the project directory.

```
projdir>> mkdir /cortex/$USER/timefits/cache
projdir>> ln -s /cortex/$USER/timefits/cache
projdir>> mkdir /cortex/$USER/timefits/results
projdir>> ln -s /cortex/$USER/timefits/results
```

It is also recommended to copy the initial contents of the cache and results from other people to save time in reproducing previous runs. For example:

```
projdir>> cp -r /cortex/ronniemaor/HTR/cache/* cache
projdir>> cp -r /cortex/ronniemaor/HTR/results/* results
(this might take a while)
```

3. Running the package

TL;DR

```
python <projdir>/code/scripts/compute_fits.py -v --shape sigslope
--sigma_prior normal --priors sigslope80 --from_age postnatal --pathway all
--scale none --html --mat
```

3.1 Creating a single fit

The script `do_one_fit.py` fits a shape for a single gene/region:

```
>> python <projdir>/code/scripts/do_one_fit.py
```

This should create the file <projdir>/results/fit.png
For more options:
>> python do_one_fit.py --help

3.2 The main script - compute_fits.py

code/scripts/compute_fits.py is the main script used to create fits for a whole pathway or the whole genome and also to create the html and figures, change distributions and/or export the results to matlab. It has accumulated quite a few switches and options over time. To see the various options and usage, run it with `--help`:

```
projdir/code/scripts>> python compute_fits.py --help
```

The following command runs the sigslope80 shape on the whole genome, using only postnatal data points and no age scaling. It also computes the change distributions and exports the fit and change distribution files to matlab format.

```
projdir/code/scripts>> python compute_fits.py -v --shape sigslope --sigma_prior
normal --priors sigslope80 --from_age postnatal --pathway all --mat
--change_dist
```

3.3 Specifying the set of genes (pathway)

Use --pathway PATHWAY
When PATHWAY=***serotonin*** or ***cannabinoids*** or ***all***, a predefined set of strings is used. Otherwise PATHWAY is treated as a file path (relative to the data directory), containing the list of genes. Files accepted formats are:

- Files ending in `.mat` that contain a matlab file with one variable that is a cell array of strings
- Files not ending in `.mat` are expected to be text files with gene names separated by whitespace or commas.

3.4 Note on parallelization

Currently for each gene, regions are fit in parallel using N-1 processes, where N is the number of cores on your machine.
If you're fitting many genes, use the `--part k/n` option to split the work on several machines, e.g.

```
ctx03>> python compute_fits.py --shape poly1 --part 1/3
ctx04>> python compute_fits.py --shape poly1 --part 2/3
ctx05>> python compute_fits.py --shape poly1 --part 3/3
```

Each of these will compute part of the genes and write the fits to files like e.g. <base filename>.pkl.2_of_3
After all the partial jobs are done, you should run again without `--part`. The “main” run will now detect the cached partial results and consolidate all the parts into one file.

3.5 Specifying command-line arguments

The programs provide a standard options parser, so the normal things will work and you can get help by running with `--help`.

Some things that are non standard or otherwise worth mentioning:
You can read arguments from a text file by using @, e.g.:

```
>> python compute_fits.py -v @compute_fits.args
```

will read a set of arguments from compute_fits.args.

Most options accept one value. If they are specified twice then the last value specified is used. This behavior allows you to use the file as defaults and override some of the values on the command line.

Lines starting with # in the text file are ignored (treated as comments).

3.6 Translate the output From PKL to Matlab format

code/timing/script_export_to_matlab.py

3.7 Other scripts

There are quite a number of other scripts in the `scripts`, `timing` and `writing` subdirectories (most are in `scripts`. `timing` contains code related to analyzing change distribution and some of it was ported to compute_fits.py. `writing` contains scripts with hardcoded values to produce the figures for my RP and thesis).

These scripts use the main library for specific ad-hoc needs. They are in varying states of disrepair - most should work or require minor fixes, but those that I haven't used for a while will probably require at least some work to get running.

4. Adding a new dataset

1. Add a file named <DATASET>_allGenes.mat to the data directory. The default data directory is a sibling of the .../code directory.
   That matlab file should have the same structure as the kang2011_allGenes.mat file.
   The following variables in that file are used:
   - **expression**: N x G x R (double)
   - **gene_names**: G x 1 (cell array of strings)
   - **region_names**: R x 1 (cell array of strings)
   - **ages**: N x 1 (double)
   - optional: genders: N x 1 (cell array of strings which are either 'M' or 'F'). However, this field is not used in the current analysis besides being loaded.

   Importantly, entrez_ids are not used by the package, only gene_names.
2. Edit **`command_line.py`**, add DATASET to the variable *known_datasets*.
3. Edit **`config.py`**, add the dataset and ordered list of regions to the dictionary *sorted_regions*.
   In the future we could change the code so it sets the last two automatically.
4. run
   ```
   compute_fits.py --dataset DATASET --shape sigslope --scaling
   log --sigma_prior normal  --priors sigslope80
   ```

5. Editing the code

1. open a user in github: https://github.com/
2. Two options:
   a. Option A: get permission to push to the main branch by asking gal (or ronnie)

    b. Option B: create a fork of the code, test your changes, and email the main person that owns the main branch so they merge your changes into the main branch.
3. run cd code; git config -e , this alolows your to edit the git config file located in code/.git/config.
   Varify that the url line is: url = git@github.com/ronniemaor/timefit.git
4. Follow the instructions on github to generate a public/private key pair and add the public key to your account on github. This will relieve you of the need to enter your username and password every time you push code to github.

Now, to push changes:

1. Make some changes to the code.
2. commit your changes
    a. git add <filename1> <filename2> ...
    b. git commit -m “your commit message here”

   This will create a commit on your local repository with the changes you made to the above files. Note that files that were edited/added but not “git added” will not be part of the commit.
3. You can commit several times. Once you want to share your changes with other, push them to the github repository:
    a. git push origin
       If you configured the ssh keys correctly, you shouldn’t be asked for a password. If someone else pushed changes in parallel to you (since the last time you pulled) you will get an error saying “non fast-forward merge”. This means that you need to “git pull” and merge any conflicts. Once that is done you will be abel to push to the server. Merging and conflict resolution (in the git and in the broad sense) are outside the scope of this document.