1 Installation

Clone with submodules:

```
git clone http://www.sternwarte.uni-erlangen.de/gitlab/collischon/litchi.git
cd litchi; git submodule update --init --recursive
cd ..
```

Create a build directory and create the Makefile if your standard compiler can handle C++20 (e.g. g++-10, can be checked with g++ -v):

```
mkdir litchi-build; cd litchi-build;
cmake ../litchi ./
```

Some setups may enable you to load a module that changes your compiler version beforehand, (e.g. module load gcc/10 on Remeis). Ask your administrator about this.

If you need to specify a different compiler (such as g++-10), name it before calling cmake:

```
CC=gcc-10 CXX=g++-10 cmake ../litchi ./
```

Compile:

```
make
```

Litchi requires Healpix to be installed. Its folder should either be specified in an environment variable called HEALPIX (Healpix does this if you compile it yourself, check with `echo $HEALPIX`). Alternatively, it should be in the system's include/library paths (e.g. after installing with the package manager or adding it manually).

Optional: create code documentation with doxygen by going to litchi/doc and then calling `doxygen doxygen.config`

2 How to use litchi

2.1 Command line arguments

`--infile, -i` Filename of input file

`--outfile, -o` Filename of output file, parameters are by default appended to name and to Fits-header (see also –forceOutname)

`--mask, -m` Filename of maskfile, everything touching the masked area will be set to NAN. Must have same Nside and scheme as input file. Default: Empty (no masking)

`--maskThresh` Threshold to be applied to mask, every pixel below this in the mask will be set to NAN in the input image (higher threshold leads to stricter mask). Default: 0.9

`--rankA, -A` First rank of Tensor, exponent of r. Default: 0

`--rankB, -B` Second rank of Tensor, exponent of n. Default: 0

`-curvI, -C, -c` Third index of tensor/functional. Default: 0

`--mint` minimal Threshold. Default: 0

`--maxt` maximal threshold. Default: 1

`--numt` Number of thresholds. Default: 1 (mint)

`--nside` Set Nside for input image if it should be degraded (simple downscaling without Fourier transform etc.). Must be power of 2 and smaller than Nside of the original image. Default: 0 (no downscaling)

`--smooth` Downscale output by given factor before calculating scalar from Minkowskis to get information at larger scales. One output image pixel will contain information from input image pixels in a circle with radius $\sim 1.5\times$distance pixel center-corner. Overwrites smoothRad and NsideOut. Default: 0 (no downscaling)

`--NsideOut` Set Nside for output image if smooth is not set. Combine with use in combination with smoothRad. Must be power of 2 or zero. Default: 0 (will be set equal to Nside)

`--smoothRad` One output image pixel will contain information from input image pixels in a circle with radius smoothRad (in rad). Overwritten by smooth. Default: 0

`--linThresh` Set linear spacing of thresholds. Default: active.

`--logThresh` Set logarithmic spacing of thresholds. Default: inactive.

`--forceOutname` Writes file with exactly given file name without appending the parameters to name (parameters are always in Fits-header). Default: inactive (append parameters)

`--trace` Calculate trace of tensors for outputmap. Default: active

`uotient, --evq` Calculate quotient of eigenvalues of tensors for outputmap instead of trace. Default: inactive

`rection, --evd` Calculate direction of vector functional/Eigenvector with largest eigenvalue for outputmap instead of trace. Default: inactive

`--sequence` Generate a sequence of single-threshold Minkowski maps instead of one map averaged over several thresholds

2.2 Quick guide

- Enter input filename with `-i / --infile`
- For masking the input file, give a mask file with `-m / --maskThresh`. By default, mask pixels > 0.90 are treated as unmasked. For different thresholds, set `--maskThresh`
- Enter ranks of the desired Minkowski tensor/functional with -A, -B, -C. A and B are tensor indices (set to zero for functionals), C is the type of Minkowski functional/tensor (often denoted as ν; 0 for area-like, 1 for boundary without curvature, 2 for curvature)
- Give the desired thresholds with `--mint, --maxt, --numt`. By setting `--linThresh` (default), they will be spaced linearly, and logarithmically with `--logThresh`. The boundaries (mint and maxt) are included.
- Do you want only one output map where maps for all thresholds have been averaged (default) or a sequence of Minkowski maps for a single threshold each? For the latter, set `--sequence`
- By default, the maximum resolution of the input file is used, which might be resource intensive. For lowering the resolution set –Nside to simply downscale the resolution. Warning: this is a simple downscaling function for quick tests that does not use spherical harmonics, pixel window functions, or the required smoothing needed for a serious analysis.
- The default output gives information on the scale of the 2x2 px marching square window. For larger scales, there are two options:

  - Enter a downscaling factor with `--smooth` that will be applied before calculating eigenvalues/trace. One output image pixel will contain information from input image pixels in a circle with radius $1.5 \times d_{max}$ where d_{max} is the maximum distance between a pixel center and its corners at output resolution.
  - Enter a smoothing radius with `--smoothRad` and an output Nside with `--NsideOut`. One output image pixel will contain information from input pixels within a circle around its center with radius smoothRad

- `--trace` (default), `--EVquotient`, or `--EVDirection` determine whether the trace, the quotient of eigenvalues, or the direction of the vector pointing along the largest eigenvalue of the tensors is calculated for the output map

The output path and file prefix is entered with `-o / --outfile`. Several modifications are applied as necessary:

- By default, all relevant parameters are appended to the output name (before .fits ending if given. If not, .fits is appended)
- By using `--forceOutname`, the given name is used as is
- When using `--forceOutname` and `--sequence`, a three digit counter is added to the output name (before .fits ending if given)
- Independent from these settings, all parameters are written into the primary file header
- If the output directory does not exist yet, it is created

2.3 Examples

Generate Minkmap of trace of $W_1^{0,2}$ with one threshold at 1e-5, and then smooth output to Nside/4, appending all relevant parameters to output filename and using a mask where only pixels with values > 0.95 are unmasked:

```
./litchi -i myfile.fits -A 0 -B 2 -C 1 --smooth 4 --mint 1e-5 --numt 1 \
 -o ../litchi_output/myoutput.fits -m mymask.fits --maskThresh 0.95
```

Generate sequence of perimeter length at 19 linearly spaced thresholds between -9e-5 and 9e-5, using the filename as given, appending only a counter:

```
./litchi -i myfile.fits -A 0 -B 0 -C 1 --EVquotient --mint -9e-5 \
 --maxt 9e-5 --numt 19 -o ../litchi_output/mysequence --forceOutname --sequence
```

Generate Minkmap of eigenvalue quotient of $W_1^{0,2}$ with one threshold at 0, calculate the output map with Nside 256, using a window with radius 0.02 rad, applying a mask where only pixels with values larger than 0.9 are unmasked:

```
./litchi -i myfile.fits --evq -A 0 -B 2 -C 1 --NsideOut 256 --smoothRad 2e-2 \
 --mint 0 --numt 1  -o ../litchi_output/myoutput.fits -m mymask.fits
```

3 Using litchi's python bindings: litchieat

Compiling according to the above instructions will provide a python package in a file called `litchieat.cpython<version>.so`. It can be imported by adding its path using sys and then importing as usual:

```
import sys
sys.path.insert(0, '../path/to/litchi-build')
import litchieat as li
```

It provides the `paramStruct` class (create with li.paramStruct()) to contain the parameters for calling litchi. Its members stand for the same parameters as the command line arguments described above. After construction, the default values are the same as above.

The member names are: `Nside, NsideOut, rankA, rankB, curvIndex, mint, maxt, numt, smooth, smoothRad, linThresh, function, forceOutname, sequence, maskname, maskThresh`

`function` is a string and can be any of `trace, EVQuo, EVDir`

`linThresh`, `forceOutname` and `sequence` are boolean

For creating maps, the function
`makeMinkmap(string inname, paramStruct params, string outname)`
is provided. It creates either a single Minkmap or a sequence of the file given by `inname` according to the parameters given in `params` and writes the single map/the sequence to `outname`.

4 The header files: A quick overview

As Litchi is a header-only library, its parts can easily be used in other projects. This is a general overview on which file does what; for details see the doxygen-generated documentation.

Litchi's header files are divided in three general categories:

- `geometryHelpers.hpp` provides general funcions for geometry on the sphere, many of which are standard textbook functions. Uses the healpix classes `pointing` and `vec3`
- The files related to tensors themselves and operations on them (`minkTensorIntegrand.hpp` and `tensorOperations.hpp`)
- Everything related to the structure of actual minkmaps including file output is found in the `litchi*.hpp` files

4.1 Tensors

`minkTensorIntegrand.hpp` contains all functionality related to the $r^{(a)} \otimes n^{(b)}$ part of the tensor calculation, including accessing an element at a given tensor index. Since the integration is performed at zeroth order, these `minkTensorIntegrand`s are turned into the actual Minkowski tensors by multiplication with their respective contour/area segment at small enough scales and then adding them up (see next paragraph).

`tensorOperations.hpp` contains everything related to sums and products of `minkTensorIntegrand`s and functions that turn them into scalars for the final Minkmap pixels. `minkTensorStack` handles linear combinations of `minkTensorIntegrand`s including parallel transporting them to the same spot upon addition. Additionally, the functions `eigenValueQuotient`, `trace`, and `eigenVecDir` are implemented here.

4.2 Minkmaps

Minkmap-related files are ordered from background structure to actual calculations to data output according to their name. Each file includes the lower files.

`litchi_kernel.hpp` contains the virtual base class `minkmapFamily` and everything related to operations of derived classes: multiplications with scalars, addition, and `minkmapStack` which uses a `std::vector` of Minkmaps to be treated as a sum. The latter must be used if the number of Minkmaps to be summed up is not known at compile time.

`litchi_pulp.hpp` contains the `minkmapSphere` class, which calculates the desired tensor at a given pixel position. This happens on-demand to save memory; already calculated pixels are not saved by this class. The pixels are not located at Healpix-pixel centers but rather at the vertex east of the pixel. As the tensors are always calculated for a window of 3 or 4 Healpix-pixels, this is the true center of the Minkmap-pixel. North and south poles are handled with specific negative pixel numbers.

`litchi_peel.hpp` mainly contains the `normalHealpixInterface` class, which generates regular Healpix-maps from Minkmaps. It can smooth the Minkmaps before calculating the desired scalar from the tensors. This should be done to catch sufficient structure in each output pixel as the zeroth-order integration of the Minkowski tensors may diverge. The file also includes functions to create vectors of numbers linearly or logarithmically spaced between two values and a function to mask Healpix maps.

`litchi_eat.hpp` contains `paramStruct` to handle all input parameters, and the functions that bring the other code together to create and write the desired output map to a file. This includes reading the input file, sanity checking the parameters, and creating the output name according to the given parameters. Everything included in the python bindings is taken from here.