Anisotropic Centre Vortex Gluon Propagator

Handover Notes

Benjamin Page
benjaminpage.acer@gmail.com
b.t.page@swansea.ac.uk

April 20, 2024

Contents

Preface

These notes constitute a brief overview of the gluon propagator codebase and results for the anisotropic (Gen-2L) centre vortex gluon propagator. Any questions may be freely directed to either my primary email (benjamin-page.acer@gmail.com) or my institutional email (currently b.t.page@swansea.ac.uk).

This work was performed primarily in conjunction with Chris Allton (FASTSUM Swansea), Ryan Bignell (FASTSUM Trinity College Dublin), Jackson Mickley (University of Adelaide) and Derek Leinweber (University of Adelaide). Also credited are Jonivar Skullerud (FASTSUM Maynooth University) and Jesuel Marques (University São Paolo).

1 Dependencies

- Lyncs-io `pip install lyncs_io`
- Lyncs-cppyy `pip install lyncs_cppyy`
- GVAR `pip install gvar`
- C-LIME

1.1 Lyncs-io and Lyncs-cppyy

The analysis notebooks make use of the Lyncs-API python package[1] which allows for simple I/O functionality for a range of standard lattice data formats such as `.lime` and `.openqcd`. Note: as of 05/04/2024 the `.openqcd` file read code created by Ben Page and Ryan Bignell has not been implemented in the version made available to the `pip` package manager by the Lyncs collaboration. If this is still not the case, the package must be built directly from the GitHub repository if the OpenQCD write functionality is needed (i.e. for gauge field scrambling).

Lyncs-cppyy is needed for the LIME file conversions, but is not needed to run the propagator analysis notebook.

1.2 C-LIME

In order to use the gauge fixing algorithm, some form of LIME (Lattice [QCD] Interchange Message Encapsulation) I/O software is required for the gauge fixing pre- and post-processing (see below). One such example is the C-LIME package maintained by the USQCD collaboration[2].

1.3 GVAR

Error calculation and propagation is handled using the GVAR Python package, which may be found at `https://github.com/gplepage/gvar`.

1.4 Memory requirements

The gauge field files are large and consume an equal amount of space in memory during manipulation. Because the gluon propagator code is accelerated in C, two copies of the gauge-field file must be created (one addressable by Python and the other by C) and so at least twice the largest gauge-field file size is needed in memory. For 128×32^3 this is about 5GB. The gluon propagator code can also easily exceed 8GB of memory when averaging the gluon propagator over a large set of configurations (i.e many cached propagator files).

2 Gauge Fixing

The Coulomb and anisotropic Landau gauge fixing is performed using a successive over-relaxation algorithm[3]. The anisotropic gauge fixing code was developed by Jesuel Marques with support from Jonivar Skullerud.

The gauge-fixing workflow is shown in Fig. 1.

[1]`https://github.com/Lyncs-API/lyncs.io`
[2]`https://github.com/usqcd-software/c-lime`
[3]`https://github.com/jesuel-marques/GaugeFix-Coulomb/tree/coulomblandau`

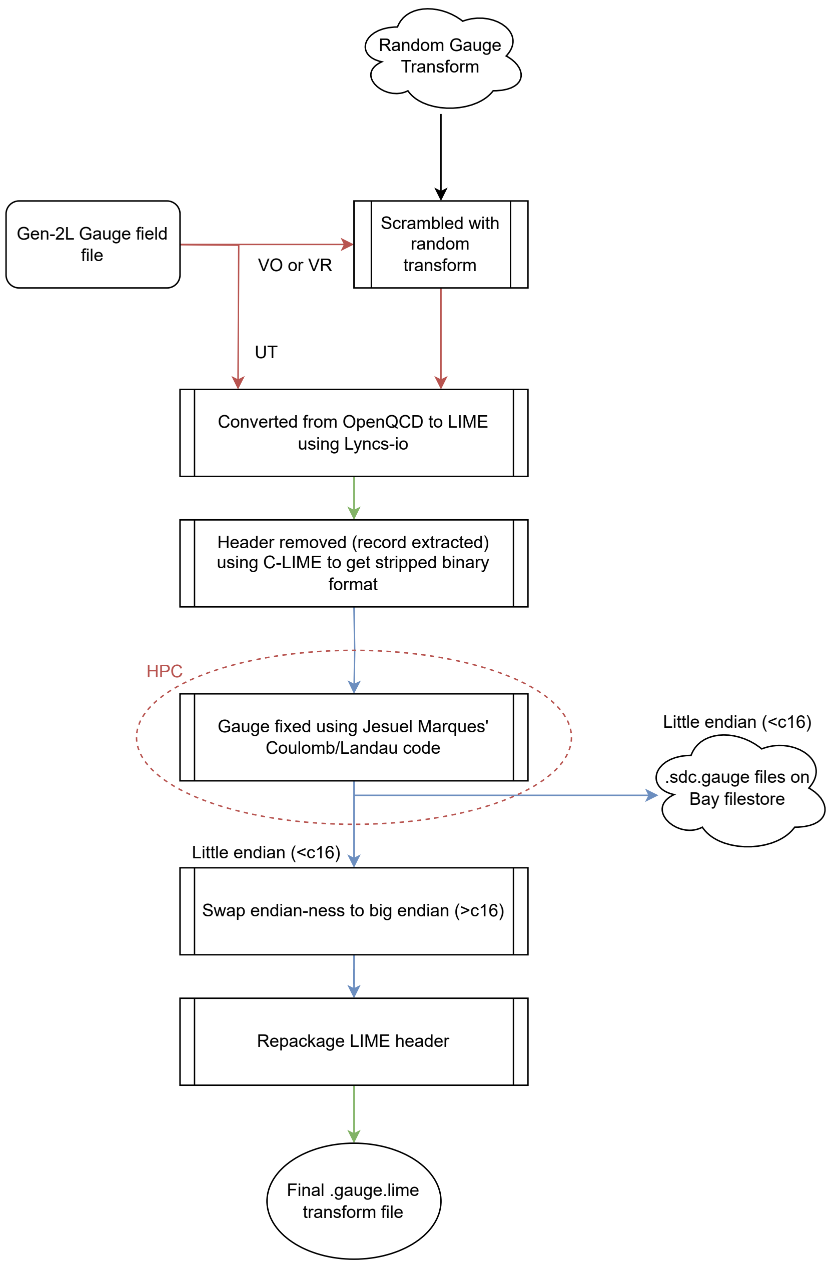
Random Gauge Transform
Gen-2L Gauge field file
Scrambled with random transform
VO or VR
UT
Converted from OpenQCD to LIME using Lyncs-io
Header removed (record extracted) using C-LIME to get stripped binary format
HPC
Gauge fixed using Jesuel Marques' Coulomb/Landau code
Little endian (<c16)
.sdc.gauge files on Bay filestore
Little endian (<c16)
Swap endian-ness to big endian (>c16)
Repackage LIME header
Final .gauge.lime transform file

Figure 1: Gauge fixing procedure. Red arrows denote `.openqcd` format files, green arrows denote `.lime` files and blue arrows denote stripped binary files of variable endian-ness.

2.1 Scrambling (for vortex-decomposed configurations)

If using the vortex-decomposed configurations, the gauge field files may need to be scrambled by a random SU(3) transform to prevent trivial fixing of the gauge. This is especially needed for the vortex-only configurations, which (by definition) have no A_μ field and so naturally satisfy both Coulomb and Landau gauge.

In the case of vortex-removed configurations, scrambling is required in the case of Coulomb gauge fixing due to the residual gauge freedom in the temporal direction (unless using an additional fixing procedure such as integrated Polyakov gauge– but this is not available in Jesuel Marques' gauge fixing code). If the configurations are not scrambled, "banding" may occur in the propagator where there is a scale separation for each temporal momentum slice. I suspect this is because there is some residual Maximum Centre Gauge fixing artefact (which is a 4D fixing) that survives.

Note: Because of this, I also scramble the vortex-removed configurations for Landau gauge fixing. Although this should technically be valid, it would be interesting to see how the unscrambled vortex-removed gluon propagator compares.

2.2 Pre-processing

The gauge fixing code requires the gauge configurations to be in a header-stripped SciDAC/LIME format. This can be achieved in multiple ways, but a straightforward approach would be to convert the OpenQCD configurations to LIME using the Lyncs-API and then using the `lime_extract_record` function of C-LIME to create the stripped configuration.

Some utility scripts designed for the pre- and post-processing of gauge field files and gauge transform files can be found at `https://github.com/jadot-bp/vortex-utils`.

2.3 Gauge fixing

Operation of the gauge fixing code requires changing the settings of two files: `parameters.par` and `settings.h`.

The parameter file `parameters.par` controls the parameters for the gauge-fixing algorithm, namely the gauge fixing type (Coulomb or Landau) and other parameters associated with the successive over-relaxation algorithm which are detailed in the code repository. The parameters can be left as their default values, or fine-tuned as necessary.

The settings file `settings.h` MUST BE MODIFIED BEFORE USE. For use with the Gen-2L ensembles, the following settings must be set:

```
#define NEED_BYTE_SWAP_IN
#define NEED_BYTE_SWAP_OUT

#define T_INDX 0
#define Z_INDX 1
#define Y_INDX 2
#define X_INDX 3
```

Since the Gen-2L ensembles are in double precision, the setting `#define CONV_CFG_TO_WORKING_PRECISION` should be commented out.

The gauge fixing algorithm must be run on a HPC in order to obtain results in a reasonable timeframe, and can run in parallel.

2.4 Post-processing

In order to correctly read the gauge-fixed transforms using Lyncs-API, the transform files must be manually swapped from little-endian (`<c16`) to big-endian (`>c16`)[4] and re-packaged with a LIME header. If using `numpy` with Lyncs-API, a valid LIME header would be given by the pickled (i.e. written byte-wise) dictionary:

[4]Auto-swapping is implemented for the input configurations and the gauge-fixed output configurations only.

```
info_dict = {‘dtype’: ‘>c16’,
             ‘shape’: SHAPE,
             ‘nbytes’: prod(SHAPE)*dtype(‘>c16’).itemsize,
             ‘fortran_order’: False,
             ‘misc’: ‘generated at 00:00:00 on 01/01/1970’)
```

where SHAPE is a rank-7 array of the form $N_\tau \times N_s^3 \times N_d \times N_c^2$ for the gauge configurations and a rank-6 array of the form $N_\tau \times N_s^3 \times N_c^2$ for the gauge transformations. Note the ordering of the space-time indices in the shape array, which is C-ordered.

3 Gluon Propagator Calculation

3.1 Code preparation

The gluon propagator calculation is performed by a C-accelerated python script and requires the compilation of two library files: `libgprop.so` and `libgutils.so`. The utility library `libgutils.so` enables accelerated calculation of the lattice divergence, $\nabla \cdot A$ or $\nabla_\mu A_\mu$, whilst the propagator library `libgprop.so` contains the propagator calculation code. Both files may be simply compiled using the shell script `compile_gfiles.sh`.

The gauge type is chosen by the parameter `MU_START`, which is 1 for Coulomb gauge and 0 for Landau gauge. If `MU_START=0`, the propagator code also returns the (unmodified) temporal component of the correlator, $D_4(q)$, which is useful for diagnosing anisotropy issues.

3.2 Stored Transforms

The gauge-fixing transforms are only required if regenerating the propagator files (see Stored Propagators) from scratch. The transforms files are stored on the Bay filestore (`sa2c-backup2.swan.ac.uk`) in stripped format. In order to read them in, the LIME header needs to be attached as in §2.4[5].

The latest, most-relevant gauge transformation files can be found at

```
sa2c-backup2:/cdt_storage/scw1383/users/b.page/vortex
```

The Landau gauge transforms for vortex-only and vortex-removed have been de-scrambled and repackaged and are ready to be directly used with the Gen-2L configurations. The untouched transforms are unscrambled but require repackaging. The Coulomb gauge transforms are still in a scrambled and unpackaged form– contact me if you wish to use these configurations and I will explain the repackaging/unscrambling process.

3.3 Stored propagators

The stored propagator files are in a simple CSV format and stored for each temperature, vorticity and gauge type at

```
sa2c-backup2:/cdt_storage/scw1383/users/b.page/vortex/props
```

The Coulomb gauge propagators have the suffix “`.prop`” whilst the Landau gauge propagators have the suffix “`.prop.landau`”. The header for the propagator file is

```
qt,qx,qy,qz,D(q)_s,[D4(q)_s]
```

where `D4(q)_s` is omitted if the propagator is in Coulomb gauge. Note that the suffix “`_s`” does not denote anything special here[6]

[5]This is because the configurations were moved to the filestore directly from the HPC which had better upload speeds, but did not have the repackaging code available due to conflicts with the Python setup.

[6]The suffix used to denote “spatial” before the `MU_START` variable was introduced, but is now redundant.

3.4 Utility scripts

There are several utility scripts which facilitate the operation of the gluon propagator calculator code and the code which constructs and processes the raw gluon propagator data for analysis.

3.4.1 `compile_gfiles.sh,compile_gluon_utils.sh`

These are the compiler files for the utility scripts and construct the library objects necessary to facilitate the C-acceleration with `ctypes`. These compiler files must be run once upon cloning the repo, otherwise the subsequent Python scripts will not work.

3.4.2 `gluon_utils.py,gluon_utils.c,gluon_utils.h`

These scripts provide a front-end for the Lyncs-io code and define a class and method structure for the OpenQCD Gen-2L correlators. The operation of this class is outlined in the Lattice library MWE.

3.4.3 `gluon.py,gluon.c,gluon.h`

These scripts perform the gluon propagator calculation itself and are accelerated using `ctypes`.

3.4.4 `view_gluon.py`

This script contains a series of utility functions and contains the functionality for reading the raw propagator files from storage ("`*.prop`" and "`*.prop.landau`") and performing the necessary post-processing and data cutting for the analysis.

The first few functions correspond to Landau-gauge fit forms, described in `https://arxiv.org/pdf/hep-lat/9809031.pdf`.

One should note that this script is designed to work with both Coulomb and Landau gauge propagator files, so some functions are Coulomb-gauge specific (such as `calculate_gz`, `calculate_f`, `renormalize` – see G. Burgio's Coulomb renormalization scheme for these: `https://arxiv.org/abs/0812.3786`) while others have clauses for handling both gauge fixing types. The gauge fixing type is specified using either the `gtype` or the `pattern` variables.

3.5 Lattice library MWE

Explain how to load gauge field with lyncs, manipulate with the ctypes library, calculate divA, etc.

Manipulations of the gauge field is achieved using the class and methods in the `gluon_utils.py` file. A minimum working example for the lattice class is available in the analysis repository and is called `Lattice library MWE.ipynb`.

3.6 Analysis scripts

All of the analysis code is located in the `https://gitlab.com/fastsum/vortex` repository. Note: these notebooks may raise an error as the default kernel, 'gluprop' , is not found. This is a custom kernel created using the listed dependencies, but if the dependencies have been installed into the root conda environment then the notebook should run using the base Python 3 (ipykernel) kernel.

3.6.1 Linking Anaconda environments with Jupyter

If using Anaconda, a custom environment can be made using

```
conda create --name <env_name>
conda activate <env_name>
```

This environment can then be used to install the necessary dependencies such as GVAR and Lyncs. The environment is then linked to Jupyter using

```
        pip install ipykernel
        python -m ipykernel install --user --name <env_name> --display-name <display_name>
```

This will make the environment accessible under the 'Change Kernel' tab in the Jupyter notebook, under the display name given.

3.6.2 Gluon propagator generator code

The notebook which generates the gluon propagator cache files is called `prop_generator.ipynb`. This notebook is straightforward to use and requires the setting of some path variables before running. These are labelled at the top of the notebook, but correspond to the gaugefield file location, the gauge transform file location and the output path for the calculated propagators.

The range of the for-loop should be adjusted to suit the number of propagators, or individual gauge field paths should be passed instead.

3.6.3 Gluon propagator analysis code

The notebook which conducts the gluon propagator analysis is called `decomp-vorticity-landau.ipynb`.
The notebook reads the stored propagator files using `view_gluon` and then calles the necessary functions to perform the Z_3 averaging, the data cutting and the momentum correction (for the lattice action).
This procedure is repeated for each vorticity at fixed (specified) temperature, and then for each temperature at fixed (specified) vorticity. This notebook saves some parameters to a cached file (the renormalization factors) because they are parameter dependent (i.e. depend on renormalization point, type of data cut, averaging window, etc.). The notebook should be run once through for every temperature to generate this cached file.

4 Closing remarks and 'gotcha's'

.

This section lists some **important** 'gotcha's', as an when I find them:

- Lyncs-io has a catch-all wildcard for reading file extensions, i.e. passing the file path `Gen2l_128x32n1` corresponds to a search for files corresponding to `Gen2l_128x32n1*`. If multiple results are returned, an exception is raised. HOWEVER, if you pass (and expect) `Gen2l_128x32n1` but only `Gen2l_128x32n1-VOS` exists, the gauge field corresponding to `Gen2l_128x32n1-VOS` will instead be returned **silently**. This is a pitfall of the Lyncs library that one should be careful of.