Prifysgol Abertawe
Swansea University

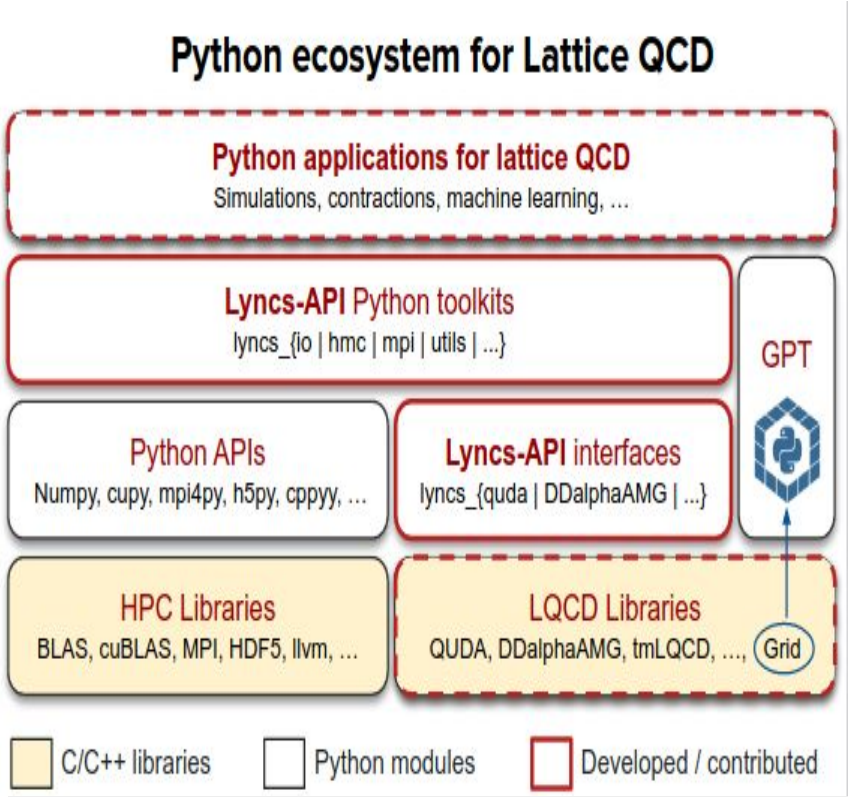
An introduction to lyncs (io)

Ben Page & Ryan Bignell
https://github.com/RJaBi/learnLyncs

FASTSUM Meeting
2023

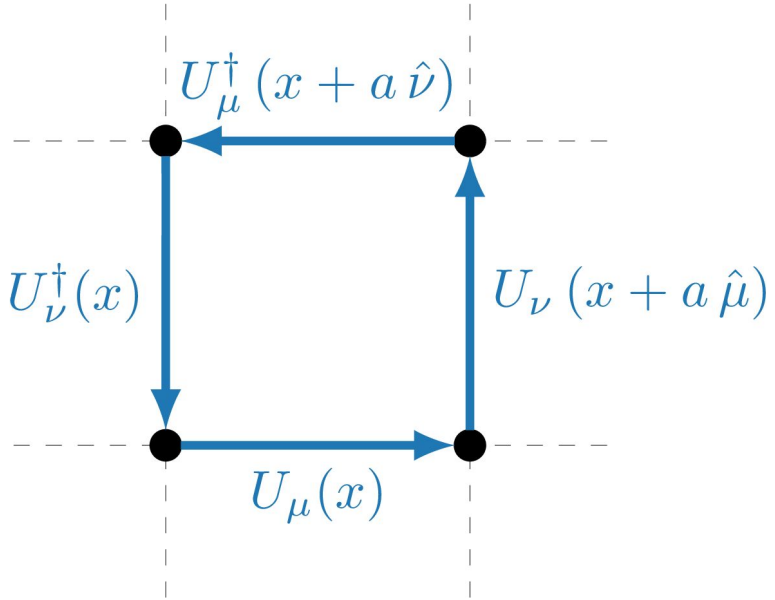
A Python API for Lattice QCD

"Lyncs is a Python API for Lattice QCD, currently under development. Lyncs aims to bring several popular libraries for Lattice QCD under a common framework."



READ GAUGEFIELD

- We would like to read our gaugefield into a rank 7 array
- NT, NX, NY, NZ, MU, NC, NC
  - Space-time dimensions, direction, colour, colour
- This will make understanding and interoperability easier
- Example calculation will be the plaquette
  - Used in gluonic portion of QCD action



$$P_{\mu\nu}(x) = U_\mu(x)\, U_\nu(x + a\,\hat{\mu})\, U_\mu^\dagger(x + a\,\hat{\nu})\, U_\nu^\dagger(x)$$

IO functions (load and save) that supports various formats and also parallel IO via MPI and Dask.

Available via pip (see Installation)

```
➤  $ pip install lyncs_io
```

Supported file formats

Format	Extensions	Binary	Archive	Parallel MPI	Parallel Dask
pickle	pkl	yes	no	no	no
dill	dll	yes	no	no	no
JSON	json	no	no	no	no
ASCII	txt	no	no	no	no
Numpy	npy	yes	no	yes	yes
Numpyz	npz	yes	yes	TODO	TODO
HDF5	hdf5,h5	yes	yes	yes	TODO
lime	lime	yes	TODO	yes	yes
Tar	tar, tar.*	-	yes	yes	no
openqcd	oqcd	yes	no	TODO	TODO

pip / conda

Lyncs can be installed via the pip or conda package installers using:

```
$ pip install lyncs
```

or

```
$ conda install lyncs
```

Individual lyncs packages may be installed by explicitly specifying the subpackage:

```
$ conda install lyncs_io
```

Environment.yml

- We supply conda environment files for linux, max and windows (untested)
- These set up the packages needed
- After installing conda, simply run
- `$ conda env create -f environment.yml`

- With the appropriate environment file
- Activate environment with
- `$ conda activate lyncs`

Installation

Jupyter may be installed in a similar manner to Lyncs via pip:

```
$ pip install jupyterlab
```

conda requires the forge channel to be set first:

```
$ conda config --add channels conda-forge
```

```
$ conda config --set channel_priority strict
```

```
$ conda install jupyterlab
```

Executing learnLyncs.ipynb

The demonstration notebook may be found at github.com/RJaBi/learnLyncs

Follow the installation section to configure the notebook environment

Start jupyter via

```
(linux) $ jupyter notebook
```

```
(mac) $ jupyter-notebook
```

Plaquette Calculation

Algorithm

1. Loop over each direction combination
2. Loop over each lattice site
3. Calculate plaquette on each site/direction combination
    a. Respecting periodic boundary conditions
4. Return sum, average and time taken

Here data has shape

➤ NT, NX, NY, NZ, MU, NC, NC

```python
def plaquette(data: np.ndarray, muStart: int = 0, muEnd: int = 4, nuEnd: int = 4) -> Tuple[float, int, float, float]:
    """
    Calculates the plaquette over muStart to muEnd
    data is [nt, nx, ny, nz, mu, colour, colour] complex
    the plaquette over all lattice is muStart=0, muEnd=4
    the spatial plaquette is muStart=1, muEnd=4, nuEnd=4
    the temporal plaquette is muStart=0, muEnd=1, nuEnd=4
    returns the sum of plaquettes, number of plaquettes measured,
    the average plaquette and the time taken to calculate it
    """
    start = time()
    shape = np.shape(data)
    # hold the sum
    sumTrP = 0.0
    # hold the number measured
    nP = 0
    for mu in range(muStart, muEnd):
        muCoord = [0] * 4
        # This is the shift in mu
        muCoord[mu] = 1
        for nu in range(mu + 1, nuEnd):
            nuCoord = [0] * 4
            # This is the shift in nu
            nuCoord[nu] = 1
            # loop over all sites
            for nx in range(0, shape[1]):
                for ny in range(0, shape[2]):
                    for nz in range(0, shape[3]):
                        for nt in range(0, shape[0]):
                            # U_mu(x)
                            coordBase = np.asarray([nt, nx, ny, nz])
                            coord = coordBase
                            Umu_x = data[coord[0], coord[1], coord[2], coord[3], mu, :, :]
                            # U_nu(x + amu)
                            coord = coordBase + muCoord
                            # respect periodic boundary conditions
                            for cc, co in enumerate(coord):
                                if co >= shape[cc]:
                                    coord[cc] = 0
                            Unu_xmu = data[coord[0], coord[1], coord[2], coord[3], nu, :, :]
                            # U_mu(x + anu)
                            coord = coordBase + nuCoord
                            for cc, co in enumerate(coord):
                                if co >= shape[cc]:
                                    coord[cc] = 0
                            Umu_xnu = data[coord[0], coord[1], coord[2], coord[3], mu, :, :]
                            # U_nu(x)
                            coord = coordBase
                            Unu_x = data[coord[0], coord[1], coord[2], coord[3], nu, :, :]
                            # Multiply bottom, right together
                            UmuUnu = MultiplyMatMat(Umu_x, Unu_xmu)
                            # Multiply left, top together, take dagger
                            UmudagUnudag = MultiplyMatdagMatdag(Umu_xnu, Unu_x)
                            # multiply two halves together, take trace
                            P = RealTraceMultMatMat(UmuUnu, UmudagUnudag)
                            sumTrP = sumTrP + P
                            nP = nP + 1
    end = time()
    return sumTrP, nP, sumTrP / float(nP), end - start
```

Reading gaugefield data

```
import lyncs_io as lio  # type: ignore
import sys
import numpy as np  # type: ignore
gfFile = 'confs/Gen2_8x24n7'
data = lio.load(gfFile, format='openqcd')
print(data.shape)
```

```
(8, 24, 24, 24, 4, 3, 3)
```

Reading header data

```
# We can probe the header data of the gaugefield files
# Loop over each ID
for iid in gfIDs:
    gfFile = os.path.join(gfDir,f'{gfName}{iid}')
    # Read and print header
    print(f"{gfFile}:", lio.head(gfFile, format='openqcd'))
```

```
confs/Gen2_8x24n7: {'shape': (8, 24, 24, 24, 4, 3, 3), 'dtype':
'<c16', '_offset': 24, 'plaq': 1.6265985010264397}
confs/Gen2_8x24n8: {'shape': (8, 24, 24, 24, 4, 3, 3), 'dtype':
'<c16', '_offset': 24, 'plaq': 1.6235420123884416}
confs/Gen2_8x24n9: {'shape': (8, 24, 24, 24, 4, 3, 3), 'dtype':
'<c16', '_offset': 24, 'plaq': 1.6244340856720185}
```

```
# Read the header of our new lime file
lio.head('Gen2_8x24_gfAr.lime')
```

```
{'_lyncs_io': '0.2.3',
 'created': '2023-03-15 10:24:15',
 'type': "<class 'numpy.ndarray'>",
 'shape': (3, 8, 24, 24, 24, 4, 3, 3),
 'dtype': dtype('>c16'),
 'fortran_order': False,
 'descr': '<c16',
 'nbytes': 191102976,
 '_offset': 848}
```

C

- Lyncs offers Numpy as a data container
- Numpy has the same (row-major) memory ordering as C

```
data = lyncs_io.load(<infile>)

data.to_file(<outfile>)
```

See ‘readC.c’ for reading back into C

FORTRAN

- Fortran uses column-major memory ordering
- Need to recast data before saving

```
data = lyncs_io.load(<infile>)

data.reshape(data.shape,  order=`F`)

data.to_file(<outfile>)
```

See ‘readFortran.f90’ for reading back into Fortran

https://github.com/RJaBi/learnLyncs/blob/main/learnLyncs.ipynb

This is a link to the demonstration notebook