



Lyncs-API

*A Python Interface for
Lattice QCD applications*

Dr. Simone Bacchio

Computational Scientist
CaSToRC, The Cyprus Institute

for the Lyncs-API Community
<https://github.com/Lyncs-API>



Lyncs-API in a nutshell

What?

A new community-oriented open-source software for Lattice QCD

Where?

On Github: <https://github.com/Lyncs-API>

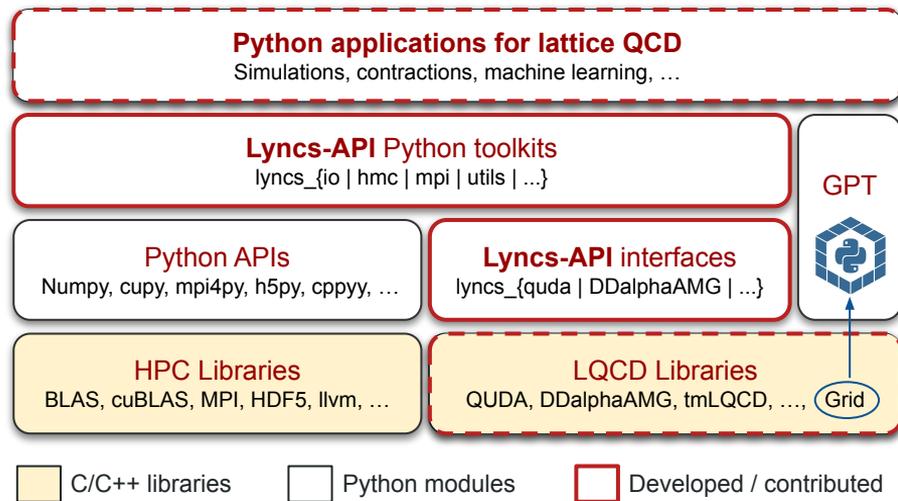
Why?

- **3Ps:** Performance, Portability, Productivity
- **In Python!**
 - Interfaces to Lattice QCD Libraries
 - Python tools for Lattice QCD
 - High-level Python applications

When?

Under development! Stay tuned or contribute! :)

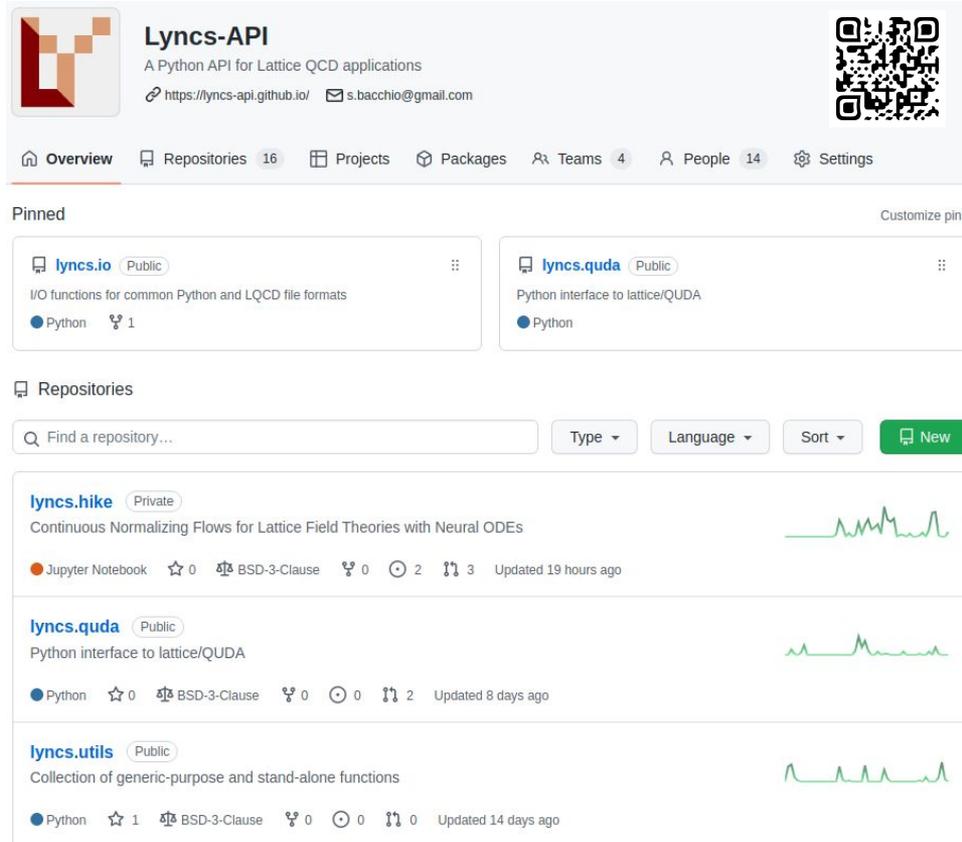
Python ecosystem for Lattice QCD



GPT - Grid Python Toolkit <https://github.com/lehner/gpt>
C. Lehner, 08/08, 14:20, Software session

Design and originality

- Interface to many LQCD libraries
 - Performance and portability
 - Involvement of the community
 - **Crosschecks** and benchmarks!
 - Second life to legacy code
- A modular ecosystem
 - Separation of concerns
 - Clean dependencies
 - Easy distribution and reuse
 - Lightweight installation
- Bazaar model
 - Many tools available for building applications
 - Easy to extend and contribute
 - Freedom on design...
 - ... but with high-quality standards!



Lyncs-API
A Python API for Lattice QCD applications
https://lyncs-api.github.io/ s.bacchio@gmail.com

Overview Repositories 16 Projects Packages Teams 4 People 14 Settings

Pinned

- lyncs.io** (Public)
I/O functions for common Python and LQCD file formats
Python 1
- lyncs.quda** (Public)
Python interface to lattice/QUDA
Python

Repositories

Find a repository... Type Language Sort New

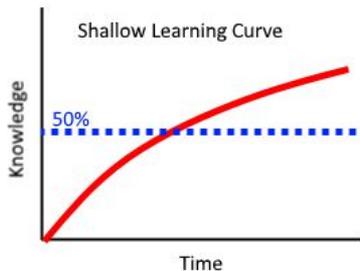
- lyncs.hike** (Private)
Continuous Normalizing Flows for Lattice Field Theories with Neural ODEs
Jupyter Notebook 0 stars BSD-3-Clause 0 forks 2 issues 3 discussions Updated 19 hours ago
- lyncs.quda** (Public)
Python interface to lattice/QUDA
Python 0 stars BSD-3-Clause 0 forks 0 issues 2 discussions Updated 8 days ago
- lyncs.utilis** (Public)
Collection of generic-purpose and stand-alone functions
Python 1 star BSD-3-Clause 0 forks 0 issues 0 discussions Updated 14 days ago

The Cathedral and the Bazaar (a Cologne's representation)



Cathedral model

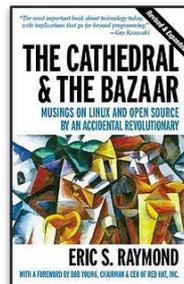
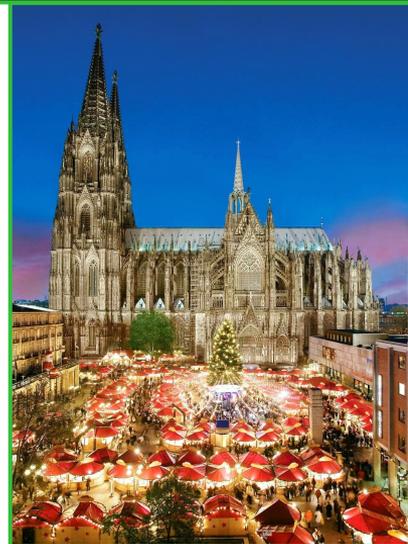
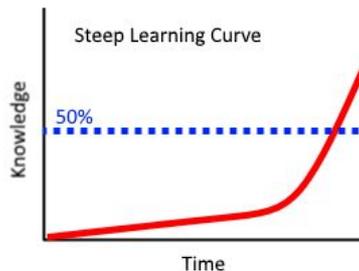
- Strict development rules
- Focus on end-user API
- Tries to cover all aspects
- Limited to implementation



VS

Bazaar model

- Open to contributions
- Focus on tools and simplicity
- User-developed APIs
- Flexible and allows hacking



Key features

Interfaces to LQCD libraries

- Automatic binding via cppy
- Libraries installed via CMake
- Loyal interfaces to libraries
- Static version and patches

```
>>> from lyncs_cppy import Lib
>>> lib = Lib(
    path="...",
    header="DDalphaAMG.h",
    library="libDDalphaAMG.so",
    c_include=True,
)
>>> lib.DDalphaAMG_init()
>>> ...
>>> rhs = np.random.rand(vec_shape)
>>> sol = np.zeros_like(rhs)
>>> lib.DDalphaAMG_solve(sol, rhs, 1e-9)
```

Unit testing and coverage

- Test parameter space
- Monitored coverage
- Continuous crosschecks
- Github actions and CI/CD

```
>>> @mark_mpi
>>> @dtype_loop # loop on dtype
>>> @device_loop # loop on device
>>> @parallel_loop # loop on procs
>>> @lattice_loop # loop on lattice
>>> def test_gauge(lib, lattice, procs,
    device, dtype):
>>>     comm = get_cart(procs)
>>>     gf = gauge(lattice,
    dtype=dtype,
    device=device,
    comm=comm)
>>>     gf.unify()
>>>     assert gf.plaquette() == 1
```

User-friendly and easy to use

- Documented source code
- Distributed via pip
- Version control and releases
- Full compatibility with Numpy

```
$$$ pip install lyncs_io
>>> import lyncs_io
>>> help(lyncs_io)
Help on package lyncs_io:

DESCRIPTION
    I/O functionalities for lyncs

FUNCTIONS
    load(filename, format=None,
    **kwargs)
        Loads data from a file.

    Parameters
    -----
    filename: ...
```

Python scripting

- Access to Python ecosystem
- Interactive shell and notebooks
- Advanced visualization tools
- Full open-source and editable



Highlights: Simplified IO

In collaboration with C. Stylianou and A. Angeli

- **Seamlessly IO**, reading and writing made simple.
- **Many formats** supported, including LQCD specific.
- **Parallel IO** supported via MPI or Dask
- **Archive formats** supported, such as HDF5, tar

Format	Extensions	Binary	Archive	Parallel MPI	Parallel Dask
pickle	pkl	yes	no	no	no
dill	dill	yes	no	no	no
JSON	json	no	no	no	no
ASCII	txt	no	no	no	no
Numpy	npz	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

- ✓ Flexible
- ✓ Extendible
- ✓ User-friendly

TODO list:

- Non-blocking IO
- More formats
- New ILDG standard

I/O

```
>>> from lyncs_io import load, save

>>> # Load Numpy array
>>> arr = load("array.npy",
              format="numpy")

>>> # Format deduced from the extension
>>> arr = load("array.npy")

>>> # Load in parallel with MPI
>>> arr = load("array.npy", comm=cart)

>>> # Load in parallel with Dask
>>> arr = load("array.npy",
              chunks=(4,4,4))

>>> # Support for HDF5
>>> arr = load("data.h5/array")

>>> # And lattice formats
>>> arr = load("conf.lime")

>>> # And the same for saving
>>> save(arr, "array.npy")
>>> save(arr, "data.h5/array")
>>> save(arr, "conf.lime", comm=cart)
```

```
$ pip install lyncs_io
```

Highlights: Running on GPUs via QUDA

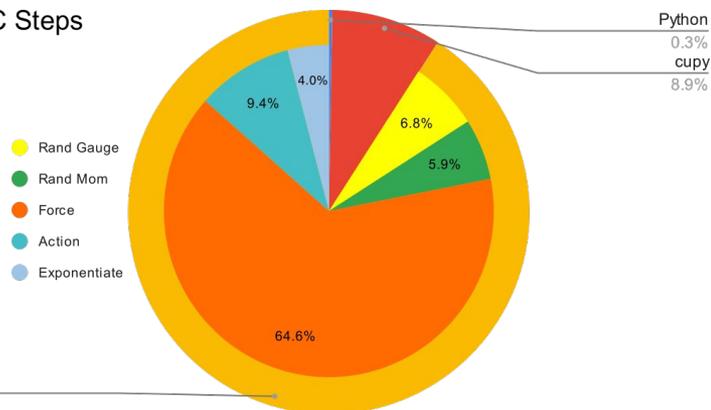
In collaboration with S. Yamamoto and D. Nole

- **One of largest communities** in LQCD
- **Many operators** supported and lattice variants
- **Solvers for Dirac operator**, including state-of-the-art multigrid (todo)
- **HMC on GPUs**, pure-gauge and with fermionic action (todo)

[S. Yamamoto, 11/08, 09:40, Software session](#)

- ✓ Performance
- ✓ Of high interest
- ✓ Portability (todo)

HMC Steps



TODO list:

- Recent version of QUDA
- Multigrid solver
- HMC with fermions

QUDA

```
>>> import lynx_quda as quda
>>> from mpi4py import MPI

>>> lattice = [4, 4, 4, 4]
>>> procs = [2, 2, 1, 1]
>>> comm = MPI.COMM_WORLD
>>> comm = comm.Create_cart(procs)

>>> gauge = quda.gauge(lattice=lattice,
                      comm=comm, device="GPU")
>>> gauge.random()

>>> plaq = gauge.plaquette
>>> print("Plaquette:", plaq)

>>> vec = quda.spinor(lattice=lattice,
                     comm=comm, device="GPU")
>>> vec.random()

>>> dirac = gauge.Dirac(kappa=0.125)
>>> mat = dirac.MMdag
>>> sol = mat.solve(vec, inv_type="CG")
>>> res = mat(sol) - vec
```

Install via source

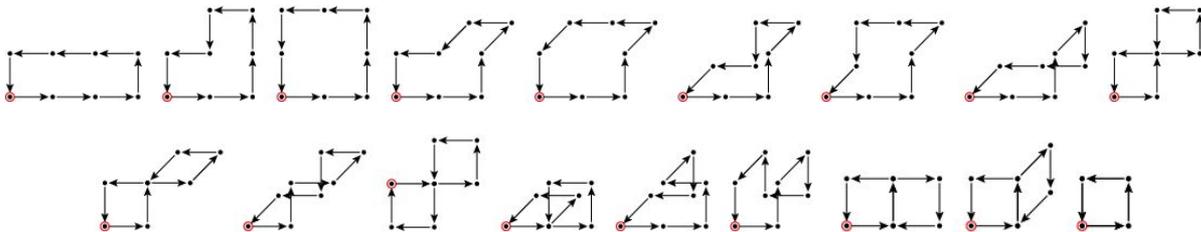
Highlights: Machine Learning

In collaboration with P. Kessel, L. Vaitl and S. Schaefer

- **Access to machine learning frameworks**, e.g. torch, tensorflow
- **Eased collaboration** with machine learning experts
- **QUDA enabled**, for SU(3) in 4D or 2D/3D with batch size
- **Deep testing** and easy cross checks with Python implementations
- **Visualization tools** and Jupyter notebooks

✓ *Stimulating*
✓ *Challenging*
✓ *... and fun!*

Fig: all unique geometries of length 8 in 3D



???

```
>>> from lyncs_??? import NumpyField,
                             QudaField,
                             ClosedPaths

>>> ndims = 3
>>> ncol = 3
>>> lattice = [4,]*ndims
>>> batch_size = 16

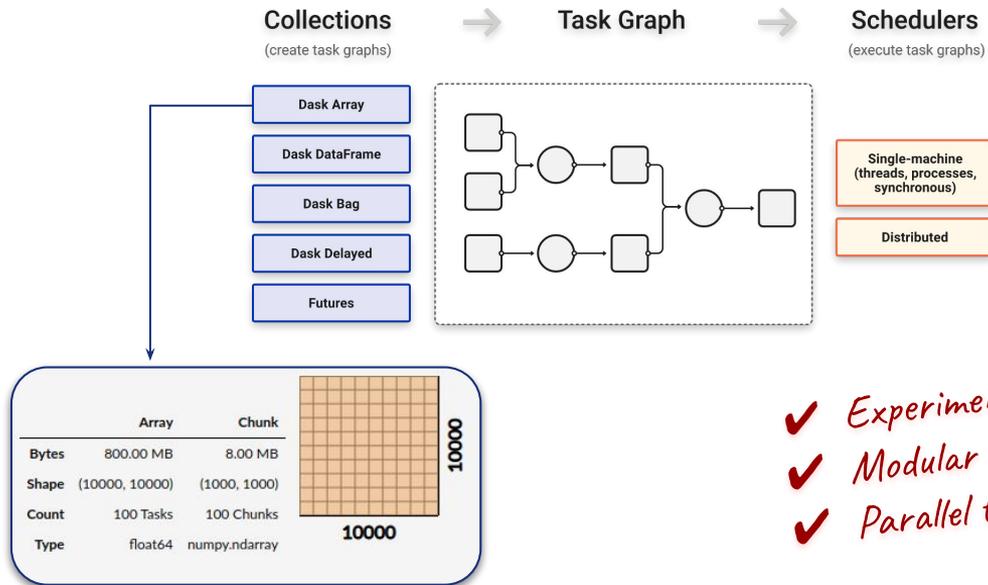
>>> # QudaField: compatible interface
>>> field = NumpyField.random_gauge(
            lattice, ncol, batch_size)

>>> for path in ClosedPaths(ndims,
                            length=8):
>>>     res = field.path_product(path)
>>>     action = res.trace().sum()
>>>     force = res.project_algebra()
```

Not public yet

Highlights: Interoperability with Dask

- **Distributed arrays API** for numpy-like N-dimensional arrays
- **Task scheduler** and resolution of computational graphs
- **Drop-in replacement** to MPI, via Lyncs functionalities



- ✓ *Experimental*
- ✓ *Modular computing*
- ✓ *Parallel tasking*

DDalphaAMG

```
>>> from lyncs_DDalphaAMG import Solver
>>> from lyncs_mpi import Client

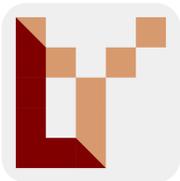
>>> # Creating a client with 4 workers
>>> client = Client(num_workers = 4)
>>> comm = client.create_comm()
>>> procs = [2, 2, 1, 1]
>>> comm = comms.create_cart(procs)

>>> solver = Solver(
    global_lattice=[4, 4, 4, 4],
    comm=comm, kappa=0.125)

>>> # Reading the configurations
>>> conf = solver.read_configuration(
    "test/conf.random")
>>> plaq = solver.set_configuration(
    conf)
>>> print("Plaquette:", plaq)

>>> # Solution for a random vector
>>> vector = solver.random()
>>> result = solver.solve(vector)
```

Conclusions



Lyncs-API

A Python API for Lattice QCD applications

<https://lyncs-api.github.io/> [✉ s.bacchio@gmail.com](mailto:s.bacchio@gmail.com)

- The Lyncs-API is / wants to be ...
 - Fresh, modern, ambitious
 - A community-wise effort
 - Flexible, Portable, Modular
 - **Pythonic** and user-friendly
- Do you want to be part of the effort?
 - Contact me! s.bacchio@gmail.com
- More details on lyncs_quda on Thu
 - S. Yamamoto, 11/08, 09:40, Software session

*Thank you for
your attention!*

*Available for
discussions! :)*

Interfacing to Libraries

1. Installation:

a. CMakeLists.txt

- i. Clone package
- ii. Apply patches
- iii. Compile and install

b. setup.py (lyncs_setuptools)

- i. Run CMake
- ii. Install dependencies
- iii. Distribute on pip

lyncs.DDalphaAMG / CMakeLists.txt

```
ExternalProject_Add(DDalphaAMG
  GIT_REPOSITORY https://github.com/sbacchio/DDalphaAMG
  GIT_TAG master
  PATCH_COMMAND git apply ${PATCHES} || git apply ${PATCHES}
  CONFIGURE_COMMAND ""
  BUILD_COMMAND make library MPI_C_COMPILER=${MPI_C_COMPILER}
  BUILD_IN_SOURCE 1
  INSTALL_COMMAND ""
)
```

lyncs.DDalphaAMG / setup.py

```
setup(
  "lyncs_DDalphaAMG",
  exclude=["*.config"],
  ext_modules=[CMakeExtension("lyncs_DDalphaAMG.lib", ".", flags)],
  data_files=[(".", ["config.py.in"])],
  install_requires=[
    "lyncs-mpi",
    "lyncs-cppyy",
    "lyncs-utils",
    "lyncs-clime",
  ],
  extras_require={
    "test": ["pytest", "pytest-cov", "pytest-benchmark"],
  },
)
```

lyncs-API / lyncs.DDalphaAMG

- └─ .github/workflows
- └─ lyncs_DDalphaAMG
- └─ patches
- └─ test
- └─ .gitignore
- └─ .pylintrc
- └─ CMakeLists.txt
- └─ LICENSE
- └─ README.md
- └─ config.py.in
- └─ pyproject.toml
- └─ setup.cfg
- └─ setup.py

Interfacing to Libraries

1. Installation:

a. CMakeLists.txt

- i. Clone package
- ii. Apply patches
- iii. Compile and install

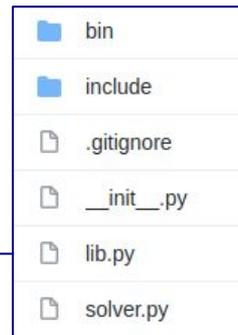
b. setup.py (lyncs_setuptools)

- i. Run CMake
- ii. Install dependencies
- iii. Distribute on pip

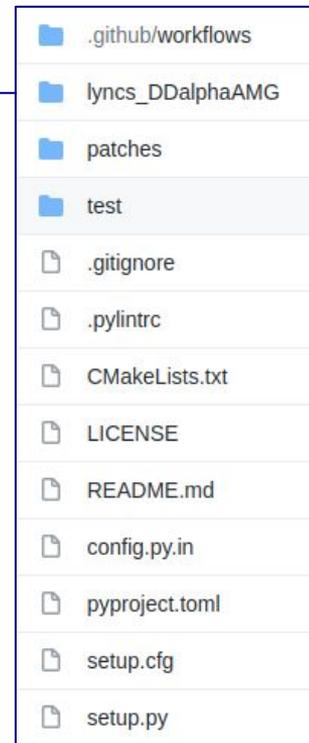
```
from lyncs_mpi import lib as libmpi
from lyncs_cppyy import Lib

libraries = [
    "libDDalphaAMG.so",
    libmpi,
]

lib = Lib(
    path=PATHS,
    header="DDalphaAMG.h",
    library=libraries,
    c_include=True,
    check="DDalphaAMG_init",
)
```



Lyncs-API / lyncs.DDalphaAMG



2. Interface:

a. lib.py

- i. Load headers and library
- ii. Manage initialization and global parameters

b. solver.py

- i. High-level python interface
- ii. Follow library structure

```
>>> from lyncs_DDalphaAMG import Solver
```

```
>>> # Creating the solver
>>> solver = Solver(global_lattice=[4, 4, 4, 4],
                   kappa=0.125)
```

```
>>> # Reading the configurations
>>> conf = solver.read_configuration("...")
>>> plaq = solver.set_configuration(conf)
>>> print("Plaquette:", plaq)
```

```
>>> # Computing the solution of a random vector
>>> vector = solver.random()
>>> result = solver.solve(vector)
```

Interfacing to Libraries

3. Testing:

a. Python tests (pytest)

- i. Unit testing
- ii. Loops over all cases
- iii. High **coverage**

b. CI/CD (Github Actions)

- i. Run tests for every PR ...
- ii. and when deps are updated
- iii. Upload new release on pip

```
>>> @mark_mpi
>>> @dtype_loop # enables dtype
>>> @device_loop # enables device
>>> @parallel_loop # enables procs
>>> @lattice_loop # enables lattice
>>> def test_gauge(lib, lattice, procs, device, dtype):
>>>     comm = get_cart(procs)
>>>     gf = gauge(lattice, dtype=dtype,
>>>               device=device, comm=comm)
>>>
>>>     gf.unity()
>>>     assert gf.plaquette() == 1
```

```
build passing coverage 90% pylint score 9.6/10 code style black
```

Lyncs-API / lyncs.DDalphaAMG

- └─ .github/workflows
- └─ lyncs_DDalphaAMG
- └─ patches
- └─ test
- └─ .gitignore
- └─ .pylintrc
- └─ CMakeLists.txt
- └─ LICENSE
- └─ README.md
- └─ config.py.in
- └─ pyproject.toml
- └─ setup.cfg
- └─ setup.py

4. Documentation:

a. README.md

- i. Installations instructions
- ii. Short documentation and examples

b. Readthedocs (Planned)

- i. Standard for Python packages
- ii. Collective documentation for Lyncs-API

A Python interface to the DDalphaAMG multigrid solver library

Python 3 | PyPI v0.1.2 | License GPL-3.0 | build passing | coverage 90% | pylint score 9.6/10 | code style black

This package provides a Python interface to DDalphaAMG. DDalphaAMG is a solver library for inverting Wilson Clover and Twisted Mass fermions from lattice QCD. It provides an implementation of an adaptive aggregation-based algebraic multigrid (SAlphaSAMG) method.

Installation

NOTE: lyncs_DDalphaAMG requires a working MPI installation. This can be installed via apt-get :

```
sudo apt-get install libopenmpi-dev openmpi-bin
```

OR using conda :

```
conda install -c anaconda mpi4py
```