LatticeToolbox: Streamlined data analysis in Python

L. Altenkort[1], D. A. Clarke[2], J. Goswami[3], H. Sandmeyer
[1]Bielefeld University, [2]University of Utah, [3]RIKEN
HotQCD, MILC

Introduction

Python is a particularly appealing language to carry out data analysis, owing in part to its user-friendly character as well as its access to well maintained and powerful libraries like NumPy and SciPy. Still, for the purpose of analyzing data in a lattice QCD context, some desirable functionality is missing from these libraries. Moreover, scripting languages tend to be slower than compiled ones. To help address these points we present the LatticeToolbox [1], a collection of Python modules to facilitate lattice QCD data analysis. Modules are sped up behind the scenes using Numba and parallelizers.

Motivation and strategy

The LatticeToolbox was originally developed by H. Sandmeyer in the context of HotQCD projects. Taking a cue from other open data movements like the ILDG, we have refactored the code, improved its performance, then made it publicly available on GitHub [1]. As part of the refactoring, we try to modularize as shown below. We have a large set of unit tests which we regularly run to ensure the code is robust against changes. As a modest step toward interoperability, we dedicate a section of the code to interfacing with configuration binaries and other lattice software.

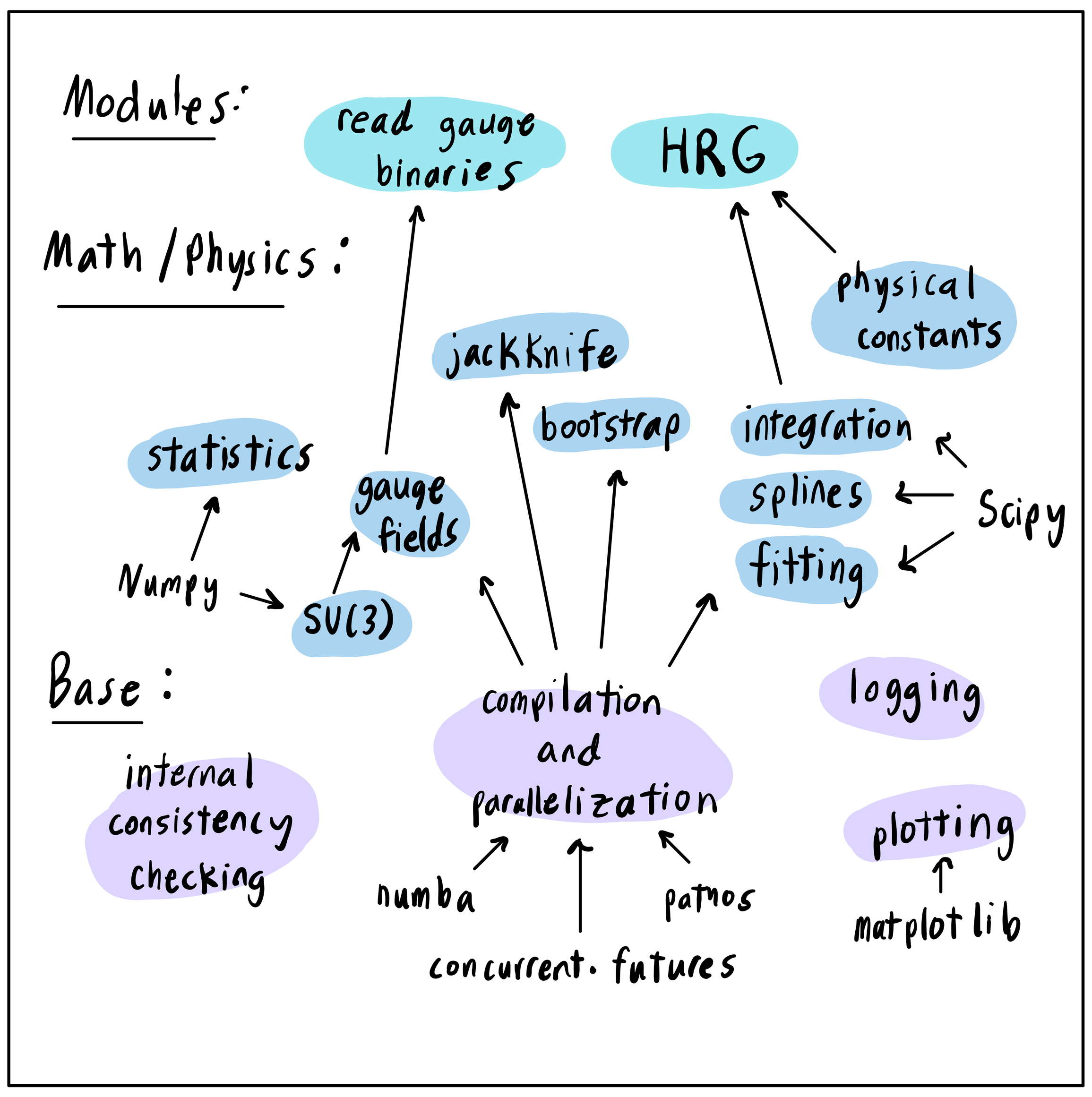

Figure 1: Rough sketch of organizational hierarchy. Base modules encapsulate combinations of well maintained Python modules. These are used to construct and enhance math and physics objects, which in turn build up modules. Here the configuration reader and HRG are given as two examples.

Interfacing

Having evolved in the context of HotQCD and MILC projects, the code interfaces with some software and conventions of these groups. We also try to make the code flexible to conventions in the broader lattice community. For instance:

- Reading in gauge configurations (NERSC, eventually ILDG)
- Jackknifing of C. Schmidt's `DenseCode` output
- Reading `.gpl` files from P. LePage's tools

References

[1] LatticeToolbox public code repository, https://github.com/LatticeQCD/LatticeToolbox.

[2] HotQCD Collaboration, Phys. Rev. D 104, 074512 (2021).

[3] J. Goswami, PoS(Lattice2022)149.

Examples of streamlined coding

```python
import numpy as np
import latqcdtools.base.logger as logger
from latqcdtools.physics.HRG import HRG
from latqcdtools.base.readWrite import readTable, writeTable
from latqcdtools.base.initialize import initialize, finalize

# Write terminal output to log file. Includes git commit hash.
initialize('HRG.log')

T = np.arange(4, 166, 0.5)

# Read in hadron names, masses, charges, baryon number, strangeness,
# charm, and degeneracy factor. This table is provided with LatticeToolbox.
hadrons, M, Q, B, S, C, g = readTable('QM_hadron_list_ext_strange_2020.txt',
                                      usecols=(0,1,2,3,4,5,6),
                                      dtype="U11,f8,i8,i8,i8,i8,i8")
w = np.array([1 if ba==0 else -1 for ba in B])

# Instantiate HRG object.
QMhrg = HRG(M,g,w,B,S,Q,C)

# This computation is vectorized since T is a numpy array.
logger.info('Computing chi2B.')
chi = QMhrg.gen_chi(T, B_order=2, Q_order=0, S_order=0, C_order=0,
                    muB_div_T=0.3, muQ_div_T=0, muS_div_T=0, muC_div_T=0)

# Output T and chi2B as columns in this table.
writeTable("chi2B.txt", T, chi, header=['T [MeV]','QM-HRG'])

finalize()
```

Listing 1: An example of how the LatticeToolbox can be used to carry out a simple hadron resonance gas computation of χ_2^B. As one can see from the `gen_chi` call, arbitrary conserved-charge cumulants are supported.

```python
import numpy as np
from latqcdtools.base.readWrite import readTable
from latqcdtools.base.printErrorBars import get_err_str
from latqcdtools.math.num_deriv import diff_deriv
from latqcdtools.math.spline import getSpline
from latqcdtools.statistics.statistics import gaudif
from latqcdtools.statistics.bootstr import bootstr_from_gauss
from latqcdtools.physics.continuumExtrap import continuumExtrapolate
from latqcdtools.physics.referenceScales import r0_phys, CY_A_DIV_R0
from latqcdtools.physics.lattice_params import latticeParams

Nts    = [6,8,10,12,14,16,18, 20]
Tlist  = []
Telist = []

for Nt in Nts:

    T  = []
    Ns = Nt*3

    # Read in Polyakov loop measurements,
    data = readTable('Nt'+str(Nt)+'.txt')
    beta, PM, PE = data[0], data[1], data[2]

    # Create array of temperatures in physical units
    for b in beta:
        # Use the most recent parameterization of r0 to set the scale
        lp = latticeParams(Ns, Nt, b, scaleType='r0', paramYear=CY_A_DIV_R0)
        T.append( lp.getT() )
    t = np.linspace(T[0],T[-1],1001)

    # Extract Tc from inflection point of <|P|>, using natural spline
    def getTc(pm):
        spl  = getSpline(T, pm, natural=True)
        dPdT = diff_deriv(t, spl)
        maxIndex = np.argmax(dPdT)
        return t[maxIndex]

    # Error in Tc estimate comes from 1000 Gaussian bootstrap samples
    Tc, Tce = bootstr_from_gauss(getTc, PM, PE, 1000)
    Tlist.append(Tc)
    Telist.append(Tce)


# Perform O(a^4) continuum-limit extrapolation
result, result_err, chidof = continuumExtrapolate( Nts, Tlist, Telist ,show_results=True,
                                                   plot_results=True, xtype = "Nt",
                                                   order = 2,
                                                   xlabel = "$1/N_\\tau^2$",
                                                   ylabel = "$T$ [MeV]" )

# Do a Z-test against literature result,
Tcr0      = r0_phys(year=2014, units="MeVinv") * result[0]
Tcr0e     = r0_phys(year=2014, units="MeVinv") * result_err[0]
Tcr0_lit  = 0.7457
Tcr0_lite = 0.0045
q         = gaudif(Tcr0,Tcr0e,Tcr0_lit,Tcr0_lite)
```

Listing 2: Given are results for $\langle|P|\rangle$ at various N_τ from pure $\mathrm{SU}(3)$ lattice calculations. This code (1) estimates the inflection point of $\langle|P|\rangle$ as a function of T to get T_c; (2) performs a parallelized bootstrap of (1) to get σ_{T_c}; (3) repeats for all N_τ and performs a continuum-limit extrapolation; and (4) compares $T_c r_0$ against the literature result using a Z-test. The bootstrapping method is agnostic to the to-be-bootstrapped function.

Physics modules

We conclude with some physics modules that might be of interest both to lattice practitioners and those studying QCD phenomenology:

- HotQCD parameterizations of, e.g. $af_K(\beta)$, $r_1m_s(\beta)$
- Physical parameters and their errors, e.g. m_π, m_ρ
- Hadron Resonance Gas model [2]
- QCD Equation of State [3]
- Static quark potential and Polyakov loop observables
- We will continue adding more!

THE UNIVERSITY OF UTAH
UNIVERSITÄT BIELEFELD
Faculty of Physics
PUNCH 4NFDI