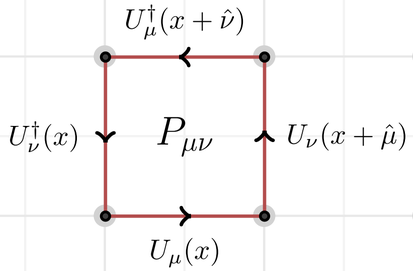
Easier parameter scans with hmcdj



Alexis Verney-Provatas, Ed Bennett and Gaurav Ray

28 July 2026

Introduction

hmcdj is a C++ library built on Grid[1,2] for automating parameter scans

- May need a large number of ensembles
- Build thin adaptation layer
  - Allow changing only required parameters
  - Ensuring remainder of interface remains constant

} **Minimise human error**

Fixed

Modifiable

Dynamic
at runtime

- Quality of life features
- Built on input files
  - Step towards full workflow reproducibility

[1] P. Boyle, A. Yamaguchi, G. Cossu, A. Portelli [1512.03487]
[2] Grid: Data parallel C++ mathematical object library [https://github.com/paboyle/Grid]

Motivation

Large parameters scans in lattice field theory are hard:

- Prone to human error
- Human input required
- Mistakes can prove costly

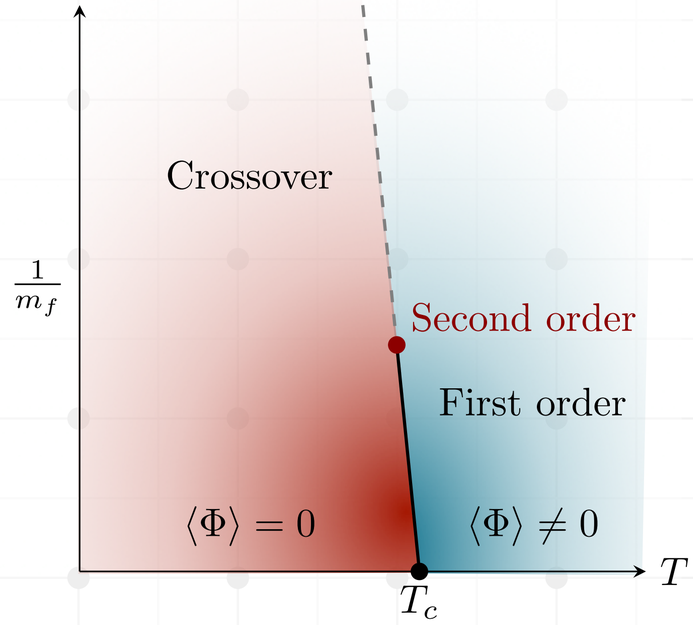
Example problems from the TELOS collaboration's research programme:

- Sp(4) at finite temperature with fermions
- Tuning the Möbius Domain Wall Fermion action parameters

Motivation

Sp(4) lattice gauge theory at finite temperature in the presence of fermions

- Aim is to explore phase space of the theory
- Two dimensional parameter space
- Require multiple spatial volumes to classify phase transition
- Require multiple temporal extents for continuum limit



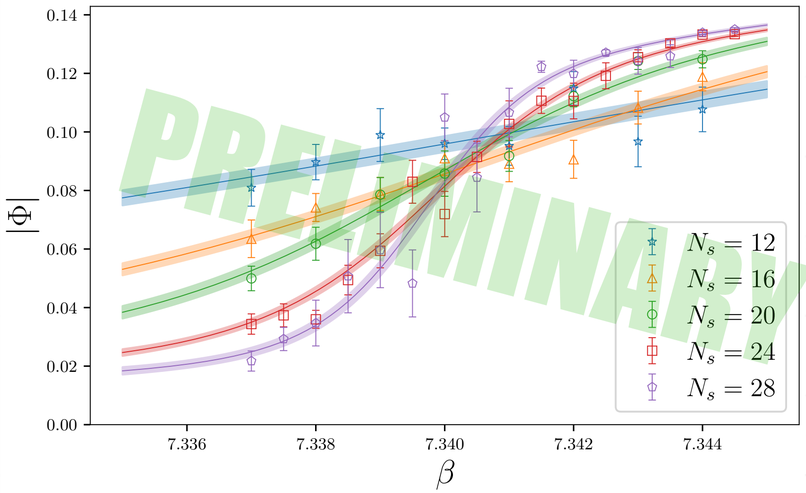
Problem parameters

- Action and most of executable remains (mostly) the same
- Need to control change of specific parameters
- Require hundreds of ensembles

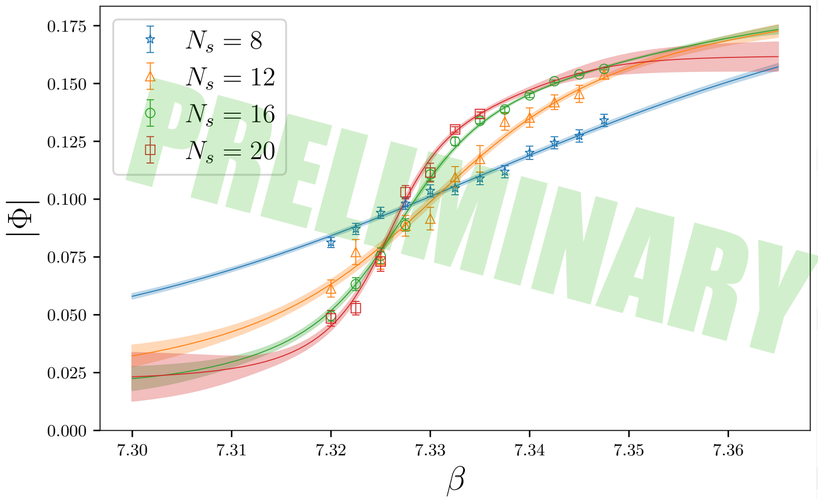
For more information see talk by AVP

Motivation Sp(4) at finite temperature in the presence of fermions

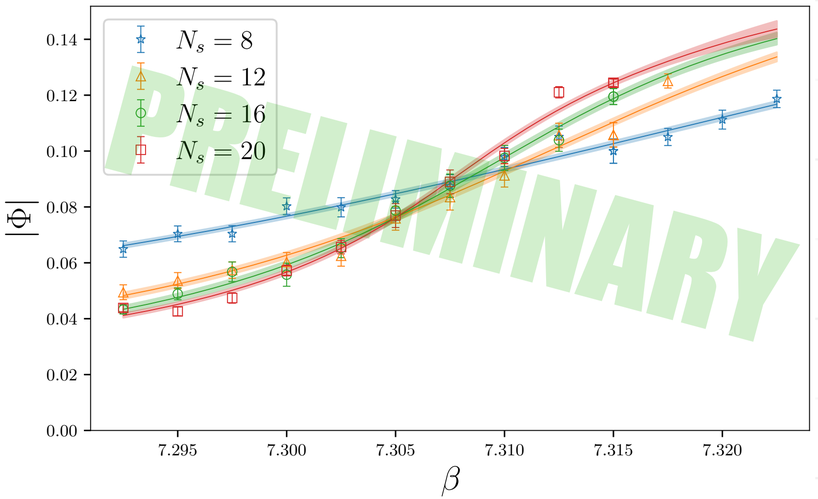
Yang-Mills



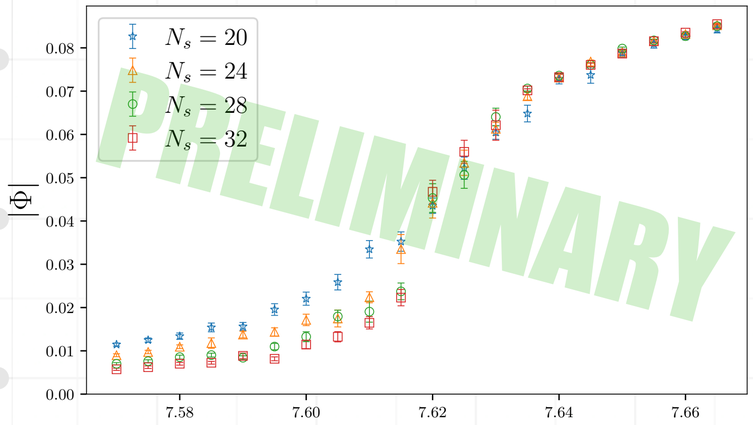
$am_0 = 4$



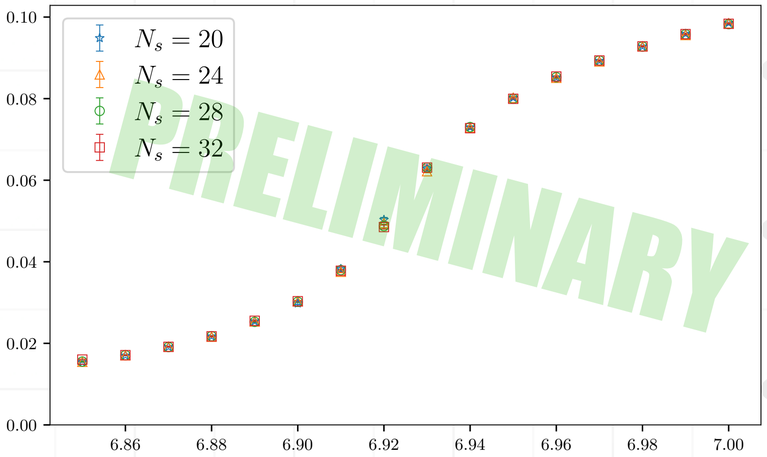
$am_0 = 2.5$



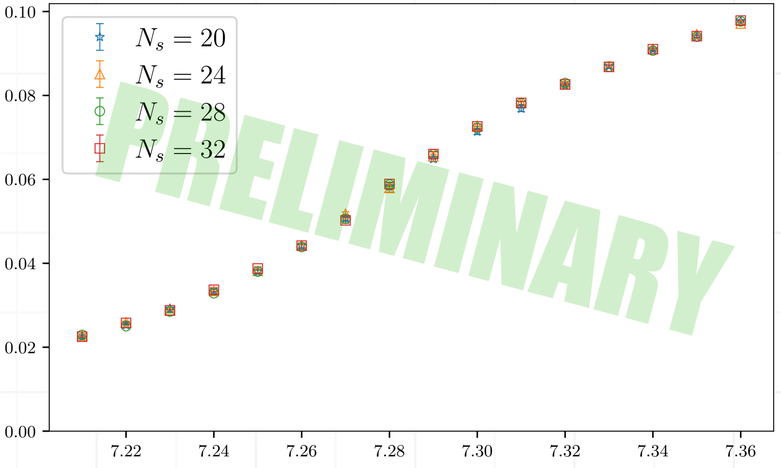
$Sp(4)$ Yang-Mills



$am_0 = -0.87$



$am_0 = -0.5$



Motivation

Tuning the Möbius Domain Wall Fermion algorithm for Sp(4)

- Aim is to tune the MDWF algorithm to Sp(4) with $N_f = 2$ fermions in the fundamental representation
- MDWF algorithm requires tuning parameters to minimise am_{res}
- α, a_5, m_5, L_s

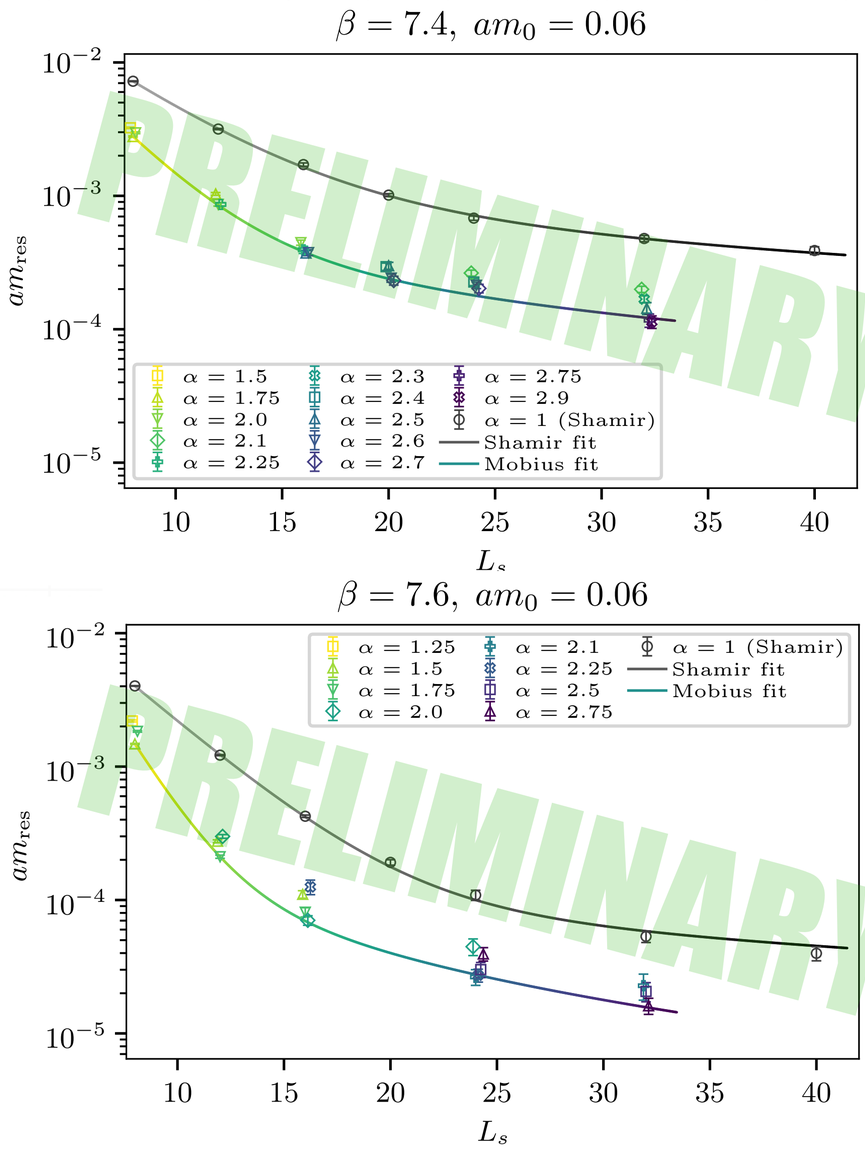
Problem parameters

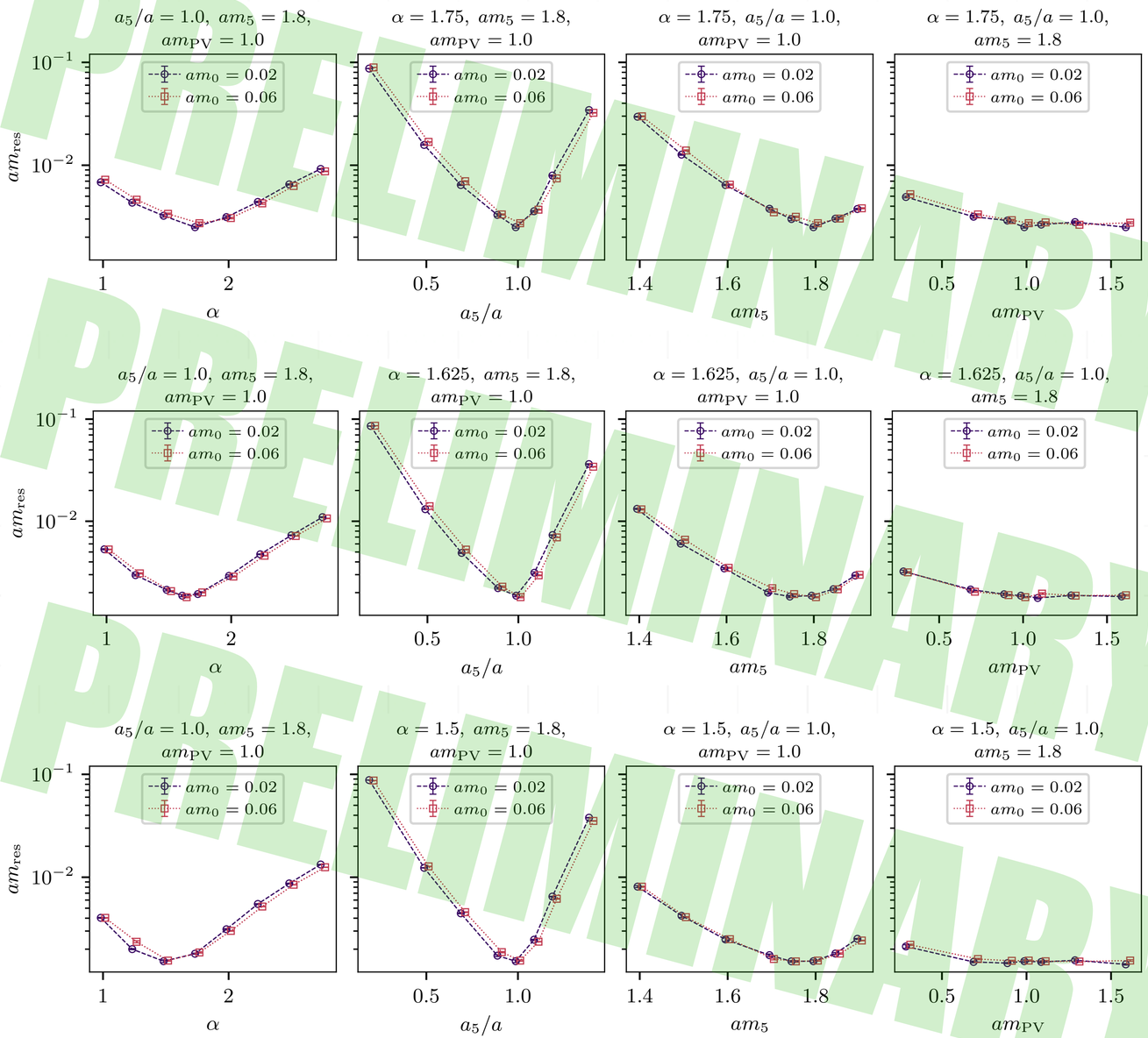
- Action and the executable remains the same
- Need to control change of specific parameters
- Require hundreds of ensembles

For more information see talk by G. Simonetti

Motivation

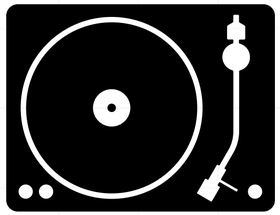
Tuning the Möbius Domain Wall Fermion algorithm for Sp(4)





Design

hmcdj is a library that allows users to design “decks" which run “tracks"



Decks:

Template executables

Tracks:

Parameter input files

Tracks

Tracks act as input files for decks

- Required Global namespace entries
- Separate namespace defined by the deck name
  - Load deck defined parameters
  - Check track belongs to correct deck

Tracks are YAML files

- Easily machine and human readable and editable

```
Global:
  name: "Ensemble"

  volume:
    nx: 4
    ny: 4
    nz: 4
    nt: 4


  checkpoint:
    saveInterval: 5
    format: "IEEE64BIG"

  HMC:
    MD:
      MDsteps: 10
      trajL: 1.0

    Thermalisations: 0

    Trajectories: 10

    StartingType: “HotStart"

    …

PureGauge:
    beta: 6.9
```

Decks

Template executables

hmcdj is a library

- Developers can easily create custom decks
- Define custom parameters
- Users can run executables

Decks:

- Remain constant
- Fail if track namespace does not match deck name

```
#include <Grid/Grid.h>
#include <hmcdj/hmcdj.h>

int main(int argc, char* argv[]) {

  typedef Grid::GenericSpHMCRunner<Grid::MinimumNorm2> HMCWrapper;

  // Define parameters
  auto beta = djParameter<double>("beta");
  djParameterList params = {beta};

  const bool reducedStorage = true;
  DJ<HMCWrapper> hmcdj("PureGauge", argc, argv, params, reducedStorage);

  /* Observables */
  // Add the temporal Polyakov Loop observable
  typedef Grid::PolyakovMod<HMCWrapper::ImplPolicy> PolyakovObs;
  hmcdj.TheHMC.Resources.AddObservable<PolyakovObs>();

  /* Action */
  Grid::SpWilsonGaugeActionR Waction(beta);

  Grid::ActionLevel<HMCWrapper::Field> Level1(1);
  Level1.push_back(&Waction);
  hmcdj.TheHMC.TheAction.push_back(Level1);

  hmcdj.Play();

  return 0;
}
```

Example

./PureGauge track.yaml

```
Global:
  name: "Ensemble"

  volume:
    nx: 4
    ny: 4
    nz: 4
    nt: 4

 checkpoint:
    saveInterval: 5
    format: "IEEE64BIG"

  HMC:
    MD:
      MDsteps: 10
      trajL: 1.0
    Thermalisations: 0
    Trajectories: 10
    StartingType: “HotStart"

      …

PureGauge:
    beta: 6.9
```

```
#include <Grid/Grid.h>
#include <hmcdj/hmcdj.h>

int main(int argc, char* argv[]) {

  typedef Grid::GenericSpHMCRunner<Grid::MinimumNorm2> HMCWrapper;

  // Define parameters
  auto beta = djParameter<double>("beta");
  djParameterList params = {beta};

  const bool reducedStorage = true;
  DJ<HMCWrapper> hmcdj("PureGauge", argc, argv, params, reducedStorage);

  /* Observables */
  // Add the temporal Polyakov Loop observable
  typedef Grid::PolyakovMod<HMCWrapper::ImplPolicy> PolyakovObs;
  hmcdj.TheHMC.Resources.AddObservable<PolyakovObs>();

  /* Action */
  Grid::SpWilsonGaugeActionR Waction(beta);

  Grid::ActionLevel<HMCWrapper::Field> Level1(1);
  Level1.push_back(&Waction);
  hmcdj.TheHMC.TheAction.push_back(Level1);

  hmcdj.Play();

  return 0;
}
```

Quality of life features

Checkpointing

- Automatic dynamic run start
- ILDG 1.2 compliant configurations
- Interfaces with Slurm timing
  - If there is not enough time left to complete another full trajectory the current configuration is saved and the run terminated.
  - Automatic resubmitting.
  - Easier monitoring of cases that fail with error/time out.

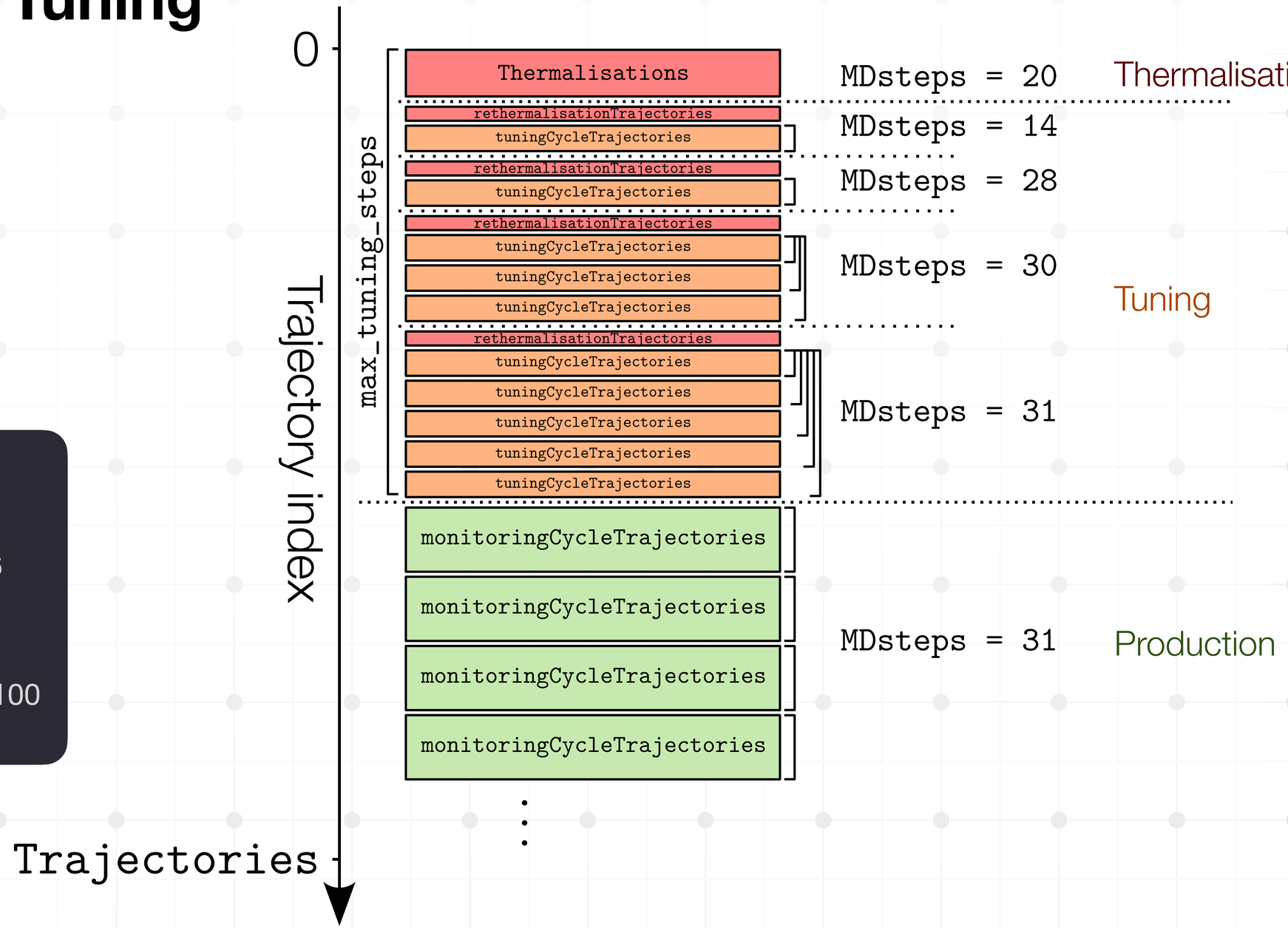
Unified directory structure for outputs

- Deterministic
- Including gauge and RNG configurations and log files
- Base path control via environment variable

Acceptance Rate Tuning

```
Global:

  AcceptanceRateTuning:
    rethermalisationTrajectories: 6
    tuningCycleTrajectories: 7
    targetAcceptance: 0.85
    deltaTargetAcceptance: 0.1
    monitoringCycleTrajectories: 100
    maxTuningTrajectories: 250
```



Using hmcdj

Available on GitHub under GPL-2.0 license:

https://github.com/telos-collaboration/hmcdj

Compiling

- Requires Grid and yaml-cpp[1]
- Compilation with autotools

Testing

- Requires GoogleTest

Documentation

https://telos-collaboration.github.io/hmcdj/

[1] yaml-cpp: A YAML parser and emitter in C++ [https://github.com/jbeder/yaml-cpp]

Future

Add further quality of life features

- Logging of on the fly observables into data files
- Add ability to run multiple chains
- More at: https://github.com/telos-collaboration/hmcdj/issues

Develop further decks

- Problem specific
- Expect users to develop their own

Contributing

- Features, decks, issues…
- Guidelines: see CONTRIBUTING.md

Summary

hmcdj is a thin C++ Grid wrapper library for automating parameter scans

Paradigm: run “**tracks**” using “**decks**”

Separates concerns into fixed, variable and dynamic

Tidy and reproducible HMC ensemble generation

Introduces quality of life features minimising human effort

Backup

Acceptance rate tuning update of MD steps[1]

$$\langle \Delta H \rangle \propto (\Delta \tau)^4$$

$$p_{\mathrm{acc}} = \mathrm{erfc}(\sqrt{\Delta \mathrm{H}}/2)$$

[1] L. Del Debbio, C. Pica, A. Patella [0805.2058]