Wang-Landau Simulation Readme

James Bounds

May 3, 2015

1 Introduction

Conventional monte carlo methods, like the Metropolis algorithm, directly sample a canonical distribution at a given temperature $P(E,T) = g(E)e^{E/K_BT}$. The Wang-Landau algorithm takes the approach of generating the density of states $g(E)$ directly. With the density of states, canonical distributions can be computed for any temperature. This program computes the density of states for the Ising, Potts, and spin glass models using the Wang-Landau procedure.

2 Version Notes

There are two versions of the main simulation. There is a version specialized for the Ising model in zero magnetic field. It is capable of running faster since it uses the fact that all energy states can be easily enumerated. The other version is more general and is able to compute the density of states for the Ising, Potts, and spin glass models. The main executable scripts included in this package are given in the table. We have also included a Metropolis-Hastings simulation that will work with the same three models.

`wangLandau.py`	Specialized version for zero field Ising model.
`wangLandauGeneral.py`	General version that works with any model
`metropolis.py`	Metropolis simulation for comparison of observables.
`spinGlassCompare.py`	Script to run `WangLandauGeneral.py` and then the metropolis algorithm on the same model. This is used because the spin glass model uses random interaction constants and is therefore different for each initialization.

3 Command line options for `wangLandau.py`

This version of the Wang-Landau simulation is specialized for the zero field Ising model. The command line parameters for `wangLandau.py` are given in the table.

Command Line Argument	Parameter	Description
–help		Display this table
-g		Graphically display $\log g(E)$ and the histogram.
-o	[filename]	Write output into [filename]
-d	[dimension]	Set the dimension of the model.
-n	[Length L]	Set the size of the model.
-a	[accuracy a]	Accuracy parameter. Iterate until $\log f < 10^{-a}$

As an example, if we wanted to compute the density of states for a 3 dimensional Ising grid of size 10x10x10, we would use the command

```
python wangLandau.py -d 3 -n 10 -o OUTPUTFILE
```

The default accuracy is to iterate until $\log f < 10^{-8}$. This is the same used in the Wang-Landau publication. The graphical display of the algorithm progress is by default not shown.

4 Command line options for `wangLandauGeneral.py`

This is the most general version of the Wang-Landau simulation. It computes the density of states for either the Ising, Potts, or spin glass models. For the zero field Ising model, it is considerably slower than `wangLandau.py` on large grids. This is because it functions using a dynamic, rather than pre-allocated histogram. The command line parameters for `wangLandauGeneral.py` are given in the table.

Command Line Argument	Parameter	Description
–help		Display this table
-g		Graphically display $\log g(E)$ and the histogram.
-m	[Model number]	Set the model which is to be simulated.
-o	[filename]	Write output into [filename]
-d	[dimension]	Set the dimension of the model.
-n	[Length L]	Set the size of the model.
-a	[accuracy a]	Accuracy parameter. Iterate until $\log f < 10^{-a}$
-f	[Magnetic Field B]	Set the magnitude of the magnetic field in the model
-t	[Test Period t]	How often the program checks the histogram 'flatness'
-v		Turn on verbose mode

The program is capable of simulating three built in models. These are specified by the model number

Mode Number	Description
0	Ising Model
1	Potts Model
2	Spin Glass

As an example, if we wanted to compute the density of states for a 3 dimensional Potts model of size 8x8x8 with zero field, we would use the command

```
python wangLandauGeneral.py -d 3 -n 8 -m 1 -o OUTPUTFILE
```

The default accuracy is to iterate until $\log f < 10^{-8}$. This is the same used in the Wang-Landau publication. The graphical display of the algorithm progress is by default not shown. The default output file name is `output`.

4.1 Output File Format

The format of the output files are given as the following example for a 10x10 Ising model

E/L^d	$\log g(E)$
-2.0	0.693
-1.92	5.298l
....	......
1.92	5.298l
2.0	0.693

The format of the data is in two tab-separated columns. There is also a header which is lead by a pound sign. The first column contains the state in terms of energy per lattice site. The second column is the logarithm of the density of states $g(E)$.

4.2 Program Structure

The program is structured so that each model that is built into a separate class. In order for a model to work with the main execution of the algorithm, it must contain the functions

flipIndex
calculateEnergy
deltaEnergy

In principle any model can be used with the alogorithm. In order to be fully specified as a working class, it must be able to return its energy, change its configuration and report the energy change of the new configuration. The program relies on our ability to compute the change in system energy for a new configuration to speed up performance. It is not always possible to compute ΔE for an arbitrary system, however.

4.3 Extending the Program for other models

Since the program is structured modularly, a new model can simply be inserted into the `WLObjects.py` script as a new class. The main script `wangLandauGeneral.py` must include the new class in the first `import` lines and implement it in the last lines where the `grd` object is assigned to the model. As long as the class has definitions for the functions specified in the previous section, it will be compatible with the program.

As mentioned in the last section, it is not always possible to compute ΔE for an arbitrary system. If this is the case, we simply just implement the function using the `calcEnergy`. This is in general slower since it must explore the entire system configuration.

4.4 To be done.....

In order for the program to be easily implemented with an arbitrary model, we should write an abstract class that defines exactly how a model should define its functions for implementing the algorithm. Using this scheme, if a class can correctly implement the prototyped class then it will automatically be compatible with the main script.