1 Installing, Compiling and Running Sampler

Clone the repository and move to its root directory *..../best_sampler*.

Using make First, go to the root directory and find the file *makefile_defs.mk*. Edit the file, changing the various locations and compiler options to your liking. Note that C++17 is required. The editing should be self-explanatory.

```
% cd testrun
% make sampler
% ./sampler
```

Note that the makefile inserts the contents of *../makefile_defs.mk*, so if the testrun directory is not located in the directory immediately below the location of *makefile_defs.mk*, you need to edit the one line in *makefile*,
`include ../makefile_defs.mk`.

Using cmake

2 Work Flow

The sampler fills a particle object with particles according to the information in a list of hyper-elements. This is done by writing a main program that links to the sampler library. The main program will look something like:

```
#include "pratt_sampler/master.h"
int main(int argc, char *argv[]){
  long long int nparts=0;
  CparameterMap parmap;
  parmap.ReadParsFromFile("parameters.txt");
  CmasterSampler ms(\&parmap);
  ms.partlist=new CpartList(\&parmap);
  ms.randy->reset(time(NULL));
  ms.ReadHyper2D();
  for(int ievent=0;ievent<ms.NEVENTS_TOT;ievent++)
    nparts+=ms.MakeEvent();
  return 0;
}
```

This will result in an object, ms.partlist, from which one can print out, analyze, or export the resulting particles. The parameter file, *parameters.txt* will contain the parameters required for generating events, e.g the file name for reading the hyper-elements. For typical RHIC or LHC events, the program generates roughly 100 events per second.

3 Parameter File

```
SAMPLER_TFMIN 0.130
SAMPLER_TFMAX 0.160
SAMPLER_NTF 16
SAMPLER_SIGMAFMIN 0.093
SAMPLER_SIGMAFMAX 0.093
SAMPLER_NSIGMAF 0
SAMPLER_MEANFIELD simple
SAMPLER_SETMU0 false
RESONANCES_INFO_FILE /Users/scottpratt/git/best_sampler/local/resinfo/pdg-SMASH.dat
SAMPLER_FINDT false
SAMPLER_CALCMU false
SAMPLER_BOSE_CORR false
SAMPLER_N_BOSE_CORR 6
SAMPLER_NPARTS_BLOCKSIZE 2000
SAMPLER_SFDIRNAME ../local/resinfo/spectralfunctions
HYPER_INFO_FILE ../hydrodata/surface_2D.dat
SAMPLER_NEVENTS_TOT 1000000
```

- The *CmasterSampler* object contains an array of *Csampler* objects. Each *Csampler* object handles the sampling duties for a specific temperature. In the example above, 26 Csampler objects will be created for $\boldsymbol{T = 130, 132 \cdots 160}$ MeV.
- Ultimately, the project will incorporate scalar fields, but currently the program only works with $\boldsymbol{\Omega = 93}$ MeV, i.e. it uses vacuum masses. Thus, one can ignore `SAMPLER_SIGMAFMIN`, `SAMPLER_SIGMAFMAX` and `SAMPLER_NSIGMAF`. Also leave `SAMPLER_MEANFIELD` set to "simple".
- The parameter `SAMPLER_CALCMU` can be set to false if you know that all chemical potentials are zero, or if the chemical potentials are provided when reading in the hyper-element information. In this instance the program is spared the computations of calculating the chemical potentials.
- The list of information about the resonances in included in the file `RESONANCES_INFO_FILE`.
- If the hyper-elements do not have the temperatures provided, i.e. they give the energy density and charge densities, one must set `SAMPLER_FINDT` to true. This slows down the program as it performs a Newton's method calculation to ascertain the temperature and chemical potential.
- Similarly, if the chemical potentials need to be calculated for each hyper-element, set `SAMPLER_CALCMU` to true.
- If you want pions created with a Bose distribution, set `SAMPLER_BOSE_CORR` to true. This is done to a given order $\boldsymbol{N}$ =`SAMPLER_N_BOSE_CORR` in the expansion

$$\frac{e^{-\beta(E-\mu)/T}}{1 - e^{-\beta(E-\mu)/T}} = \sum_{n=1}^{N} e^{-n\beta(E-\mu)/T}.$$

- The size of the particle list is increased as needed during the running of the program, but in blocks of `SAMPLER_NPARTS_BLOCKSIZE`. The CpartList object has a member *nparts* to describe exactly how many particles are made. Results do not depend on `SAMPLER_NPARTS_BLOCKSIZE`
- Spectral functions are described by files in the directory `SAMPLER_SFDIRNAME`. Separate programs are provided to generate the spectral functions.
- The hyper-surface from the hydrodynamics calculation is `HYPER_INFO_FILE`.
- The number of events to generate is *SAMPLER_NEVENTS_TOT*. This is not used by any files in the sampler library, but you want to set this for use in the main program.

4 Physics and Algorithms of Sampler

The sampler generates particles from a small hyper-volume, $d\Omega^\mu$, consistent with the grand-canonical ensemble for a non-interacting hadron gas. If one adds vector potentials that couple to the charges, such potentials are equivalent to changes in the chemical potential, and require no changes to the formalism aside from changing $\mu \to \mu + V$. The architecture should accommodate scalar fields (which change the masses), but that extension is on hold until there is a strategy for adjusting the spectral functions. This section covers the basics of the algorithms and physics for generating particles consistent with a thermal distribution, the treatment of "negative" phases-space contributions in the Cooper-Frye, corrections for Bose distributions and viscosity, and spectral functions. Rather than citing the papers on which the methods are based, or citing different methods, the following aims at being self contained. This comes at the expense of not attempting to compare various physics choices to others in the literature.

4.1 Generating a Thermal Distribution

Here, we describe how to generate the momenta of particles in their rest frame with a Boltzmann distribution,

$$\frac{dN}{d^3p} \propto e^{-\sqrt{p^2+m^2}/T}. \tag{4.1}$$

The algorithm can be seen in the file *randy.cc*, which is a collection of methods using random number generation. For light particles, $T/m > 0.6$, one uses the fact that

$$p = T(a+b+c),$$
$$a = -\ln(r_1),\ b = -\ln(r_2),\ c = -\ln(r_3).$$

where r_i are random numbers uniformly chosen in the interval $0 < r_i 1$. This generates random numbers with probability $\propto p^2 e^{-p/T}$. To account for the mass, one uses a fourth random number, r_4, and rejects/keeps the choice of r_{1-3} depending on whether r_4 is less/greater than

$e^{-(E-p)/T}$. To generate the angles, in polar coordinates,

$$\cos(\theta) = (a-b)/(a+b), \tag{4.2}$$
$$\phi = 2\pi \frac{(a+b)^2}{(a+b+c)^2}.$$

One can check that this choice works by generating Jacobian between r_{1-3} and p, θ, ϕ. This can be found in `void Crandy::generate_boltzmann(double mass,double T,FourVector &p)` within *randy.cc*.

For heavier particles, $T/m < 0.6$, the previous algorithm becomes inefficient because the acceptance rate (which approaches $e^{-m/T}$) becomes high. The algorithm is then based on the following,

$$p^2 dp e^{-E/T} \propto \frac{p}{E}(u+m)^2 du e^{-u/T}, \tag{4.3}$$
$$u = E - m.$$

First, the variable u is generated according to the above weight, ignoring the factor p/E. This choice is then accepted/rejected with the weight p/E. To generate with the weight $(u+m)^2 e^{-u/T}$, one uses the fact that the integrals $u^2 e^{-E/T}$, $2um e^{-E/T}$ and $m^2 e^{-E/T}$ are in weights of $w_1 = 2T^2$, $w_2 = 2mT$ and $w_3 m^2$ respectively. One throws a random number, $0 < r_0 < 1$, then chooses between the three terms with probability $w_i(w_1 + w_2 + w_3)$. Depending on whether $i = 1, 2$ or 3,

$$u = \begin{cases} -T\ln(r_5 r_6 r_7), & i = 2 \\ -T\ln(r_5 r_6), & i = 2 \\ -T\ln(r_5), & i = 1 \end{cases} \tag{4.4}$$

Then, using $E = u + m$, $p = sqrt E^2 - m^2$, one keeps or rejects the choice to account for the factor p/E. This routine is in `void Crandy::generate_boltzmann(double mass, double T,FourVector &p)` within *randy.cc*.

4.2 Collective Flow and Negative Contributions in Cooper-Frye

The Cooper-Frye formula,

$$dN = \frac{d^3p}{E} p \cdot \Omega e^{-u\cdot p/T}, \tag{4.5}$$

is invariant and can be expressed in the fluid rest frame, $u = (1,0,0,0)$, as

$$dN = d^3p' \, \frac{1}{E'} p' \cdot \Omega' e^{-E'/T}. \tag{4.6}$$

Writing $p' \cdot \Omega' = E'\Omega_0 - \vec{p}' \cdot \vec{\Omega}'$, one can ignore the second term if one only wants to generate the correct distribution of $|\vec{p}'|$ and ignores direction, because the second term is odd in $\vec{p}'$. Thus,

one first generates a distribution using only the first term, which can be accomplished using the algorithm of the previous subsection. The missing additional,

$$w = 1 - \frac{\vec{p}'}{E'} \cdot \frac{\Omega'}{\Omega_0}, \tag{4.7}$$

is then included by throwing a random number $0 < r < 1$, then with if $r > w$, one reflects $\vec{p}'$ about the $\vec{\Omega}'$ plane,

$$\vec{p}'' \ = \vec{p}' - \vec{\Omega}' \frac{\vec{\Omega}' \cdot \vec{p}'}{\vec{\Omega}' \cdot \vec{\Omega}'}. \tag{4.8}$$

This procedure reproduces the desired phase space density perfectly, unless one encounters a momentum for which $w' < 0$. This happens when the hyper surface moves sufficiently slowly into the hydrodynamic fluid that particles from the hadron gas might enter the fluid. Ignoring this piece results in both leaving off the negative contribution, where $w < 0$, and underestimates the contribution for the reflected region of phase space where $w > 2$. Thus, despite ignoring the negative contribution, this procedure correctly reproduces both the net yield and the net energy. However, the net momentum flow through the surface is not reproduced. Of course, when viewed in another reference frame the momentum discrepancy translates into an energy discrepancy. A solution to this shortcoming would to reflect all particles in the cascade that re-enter the hydrodynamic fluid, by reversing the momentum into the surface vector, as defined in the fluid frame. If the phase space distribution of the hadronic gas immediately outside the hyper-surface were indeed consistent with the hydrodynamic fluid at the boundary, this procedure would then be exact. Unfortunately, that is probably not the case. However, given that the number of particles that would be affected by such a correction might be less than one percent, this is probably negligible.

Once the momentum is generated in the fluid frame, p', one boosts it back to the frame in which $u \neq (1, 0, 0, 0)$. A computationally efficient manner in which to boost particles is to use the following representation of the Lorentz boose matrices,

$$L^{\alpha\beta} = g^{\alpha\beta} + 2u^\alpha n^\beta - \frac{(u^\alpha + n^\alpha)(u^\beta + n^\beta)}{1 + u \cdot n}. \tag{4.9}$$

Applied to a vector p' in the frame u, this gives the vector p in the frame n. Routines for such boosts can be found in *misc.cc*. The routines inside *randy.cc*,
`void Misc::Boost(FourVector &u,FourVector &p,FourVector &pprime)` and
`void Misc::BoostToCM(FourVector &u,FourVector &p,FourVector &ptilde)`, either boost the vector or express the vector in the boosted frame, the same as boosting with $-\vec{u}$.

4.3 Bose Corrections

The Bose distribution can be expanded as a power series,

$$\frac{e^{-\beta(E-\mu)}}{1 - e^{-\beta(E-\mu)}} = \sum_{n=1}^{N\to\infty} e^{-n\beta(E-\mu)}. \tag{4.10}$$

Each term in the sum behaves as a Boltzmann distribution with a temperature T/n. The distribution is treated as if it were a contribution from N resonances, each generated with a temperature T/n. The choice of including Bose effects and the order N are set by parameters. Only pions are corrected for Bose effects because they are the only species with sufficiently high phase space densities such that the $N > 1$ corrections are not negligible.

4.4 Viscous Corrections

A variety of formalisms exist in the literature for viscous corrections. They all attempt to adjust the phase space density in such a way that the stress energy density is reproduced, including the viscous corrections. Of course, there are infinite ways in which to adjust the phase space density, a continuous function of $\vec{p}$, and to spread the adjustments over numerous resonances. The method here is to first create a thermal momentum $\vec{p}$, and then to apply the following matrix to $\vec{p}$,

$$p'_i = \left(\delta_{ij} + \lambda \frac{\pi_{ij}}{\epsilon + P}\right) p_j. \tag{4.11}$$

The constant λ is a function of density and temperature, but not of $\vec{p}$. This is consistent with all particles having constant relaxation time, independent of $|\vec{p}|$ or species. The constant λ is calculated during the initialization of the sampler object for a given T and μ. This method begins to break down (energy conservation worse than 1%) when the viscous corrections exceed $\sim 50\%$ of the total pressure. As of this writing, the viscous correction was being re-analyzed and tested. The mechanism for implementing such corrections will be altered to make it straight-forward to offer competing alterations of the phase space density.

4.5 Spectral Functions

Unstable resonances, i.e. those with finite widths, are treated as a distribution of resonances with mass m. The distribution is known as the spectral function, $A(m)$, which is normalized,

$$\int dm\, A(m) = 1. \tag{4.12}$$

The spectral functions are read in from files. The files are stored in a directory, whose name is a parameter, `SAMPLER_SFDIRNAME`. The files are in the form *pid.txt*, where *pid* refers to to the particle-data book ID, e.g. *1114.txt* lists the spectral function of the Δ resonance. The files should be in the same format as one obtains by running `% smash -r pid`.

After reading in the spectral functions, each sampler stores a version of $A(m)$ weighted by the density of a species of mass m. If at temperature T the density of a spin-less species is $n_0(T, m)$, the weight spectral density is,

$$P(m) = n_0(T, m) A(m)/Z, \tag{4.13}$$

with Z chosen to ensure that $P(m)$ integrates to unity. The weight $P(m)$ is stored as a map, where

$$Q(m) = \int_{-\infty}^{m} dm' P(m'), \tag{4.14}$$

is the key and m is the value. To generate a mass with the desired weight, a random number $0 < r < 1$ is chosen. Using the `lower_bound` functionality of a C++ map, r is associated with the mass entry whose Q is the first one above r. Using the surrounding entries in the array, a value of m is chosen by linear interpolation.

4.6 Summing Across Resonances and Matching Yields

Each hyper-element stores the information $d\Omega^\mu$, u^μ, T and three chemical potentials $\vec{\mu}$, corresponding to baryon number, isospin and strangeness. The main object, `CmasterSampler` stores an array of `Csampler` objects, each with its own temperature. The specific sampler object is chosen randomly from the two objects whose temperatures bracket that of the hyper-element. The choice of which of the two sampler objects to employ is based on a linear weighting determined the three temperatures. For example, if the hyper-element temperature very nearly matches one of the two sampler objects, that one is almost always taken, and if the temperature is half-way in between that of two objects, each object has equal chance of being selected.

Each sampler object stores the densities calculated for zero chemical potential, but it is straightforward to calculate the density of hadrons. The first time the object is called, the hadron density, $n_{\mathbf{hadrons}}$ is calculated and stored for the `Chyper` ojbect. The average number of hadrons in the hyper-element is then

$$N_{\mathbf{hadrons}} = n_{\mathbf{hadrons}} u \cdot \Omega. \tag{4.15}$$

Before summing over the density of individual species, a test is performed to see whether any particles are created. Because each hyper-element typically has a very small N_h, it would be inefficient to generate new random numbers for each element. Instead. the `Crandy` objects stores two values, a running sum of probabilities, `netprob`, and a list of thresholds t_i. Each threshold is separated by an exponential distribution, i.e.

$$t_{i+1} = t_i - \ln(r_i), \tag{4.16}$$

where $0 < r_i < 1$ is a random number. Thus, the chance of encountering a threshold in some differential range between t and $t + dt$ is dt, indepenedent of where the previous thresholds are located. In this case, one first tests whether adding $N_{\mathbf{hadrons}}$ to `netprob` will pass a threshold. If not, nothing is done. Otherwise, one begins to sum over each hadron species h. The number for a specific species if $N_h = n_h u \cdot \Omega$. If by incrementing `netprob` by N_h, `netprob` exceeds a threshold, one creates species h. If `netprob` also exceeds the next threshold, a second h particle is created. This procedure is consistent with creating hadrons according to a Poissonian distribution, and works whether the differential volume is small or large. Because the net number of hadrons created equals the number of thresholds crossed, the number of random numbers called is the same as the number of particles created, which suggests that this algorthim is perfectly efficient with respect to the number of times a random number generator is called.

4.7 Finding μ and T in Terms of ϵ and ρ

In some cases the temperature and chemical potentials might be provided while reading in the hyper-element information. However, if that is not the case, the temperature and/or chemical

potentials can be calculated from the energy and charge densities. This is performed via a four-dimensional (if both the temerature and chemical potentials are required) Newton's method or by a three-dimensional method if only the chemical potentials are required. The boolean parameter `SAMPLER_CALCMU` determines whether $\boldsymbol{\mu}$ need to be calculated. If the chemical potentials are all zero, one can simply set $\boldsymbol{\mu} = \mathbf{0}$ by setting the parameter `SAMPLER_SETMU0=true`.