version 1: Prato 2018
version 2: Copenhagen 2019

Understanding the Glauber Model

Christian Bierlich
Email: bierlich@thep.lu.se

Contents

1 Summary

You will be familiarizing yourself with the framework for doing Glauber Model calculations, using as a baseline the PISTA implementation [1] of nuclear geometry sampling. Download the virtual machine from `home.thep.lu.se/~bierlich/pp/leightweight.vdi`. The user name is mcnet and the password is jetset. Once you are logged in, open a terminal, create a working directory and clone the code from the git repository using:
`git clone http://bierlich.net/git/glauber-tutorial.git`

2 Collisions of protons and Lead (pPb)

The most basic type of heavy ion collision features only a heavy ion object on one side, and it is thus instructive to start from these simpler systems to understand the principles behind the implementation.

2.1 Understanding the Glauber calculation

Glauber calculations are based on semi-classical, geometric approach to collisions of nuclei, where the individual *sub–collisions*, and which nucleons are *participating* in them, are determined. We will not go through the theoretical basis here, but refer the interested reader to the reviews in refs. [2, 3].

In these exercises you will use some of the underlying features in PISTA, in order to make your own Glauber calculation. In the next part of the exercise you will use the Glauber analysis to better understand the concept of "centrality" in heavy ion collisions, and create a simple heavy ion analysis using PISTA simulation as well as your own calculation.

In the folder you will find a file called `understandGlauber.py`. This serves as a skeleton for the exercises you will be doing. The first exercise (posed below) is already solved in the file, such that you can use the initialization, plotting features *etc.* as a skeleton for solving the rest.

Exercise **1** Displaying the sampled nucleus

Set up an instance of the `GlauberBase` base class from PISTA, colliding pPb. Generate a proton and a Lead ion with default parameters, extract the transverse coordinates of each nucleon, and make a figure showing the nucleons as "disks". The radius of the disks should correspond to a disk-disk cross section of 49.23 mb (reasonable value for pp inelastic, non–diffractive cross section at $\sqrt{s} = 2.76$ TeV). Both proton and Lead should be centered at transverse coordinates (0,0).

Run the script:

```
python understandGlauber.py
```

to solve exercise 1. This produces the cartoon plot shown in figure 1.

Feel free to run the script a few times and see that the lead distribution changes (random numbers in Python does not have a fixed seed per default). In the next exercise you will use the extracted coordinates to figure out which of the Lead nucleons are "hit" by the proton.

Exercise **2** Wounded nucleons

Write first a function called `dTrans` which given two particles `p1` and `p2` as input, returns the transverse distance between them, like this:

```
def dTrans(p1, p2):
   #calculate the distance
   return distance
```

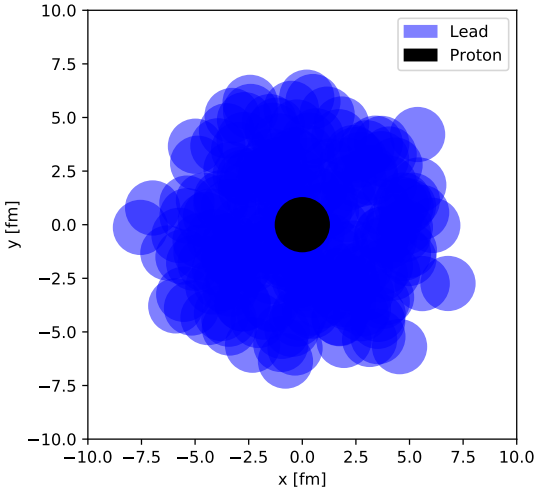

Figure 1: Cartoon figure showing a sampled Lead ion and a proton with radii corresponding to a disk–disk cross section of 49 mb, both centered at (0,0).

Once you have done so, loop over all nucleons in the lead ion, and use your `dTrans` function to compare the nucleon–proton distance with the given black disk radius `r`. Mark the nucleons which overlaps with the proton as "hit", like this:

```
for p in lead:
   if dTrans(p,proton[0]) < r:
      p["hit"] = 1
   else:
      p["hit"] = 0
```

Extract coordinates of nucleons again, this time making a distinction between "hit" and "non-hit", and produce a figure where you show which nucleons have been hit, by giving them a different colour.

When solving exercise 2, feel free to re-use the pre-written stuff from exercise 1 to make a nice figure. Your result should look similar to figure 2.

2.2 Impact parameter

We will now start to make this into a more realistic calculation. In the above cartoon examples, both proton and Lead have been centered at coordinate (0,0), giving an impact parameter of $b = 0$ and an arbitrary angle ϕ. Since it it not experimentally possible to

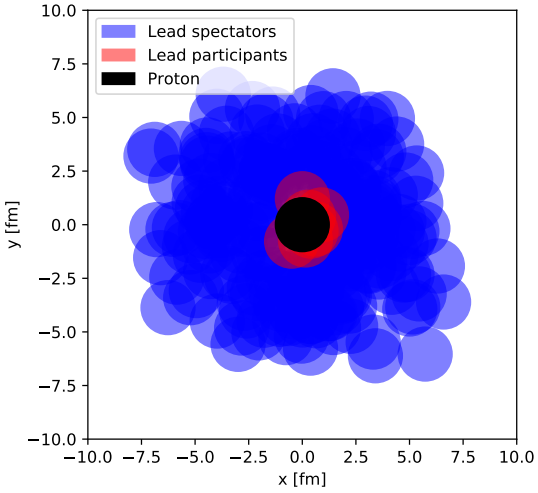

Figure 2: Cartoon figure showing a sampled Lead ion and a proton with radii corresponding to a disk–disk cross section of 49 mb, both centered at (0,0). Participating nucleons are marked red, while spectators are marked blue.

control the impact parameter of a collision, you should now implement an impact parameter sampler. Consider first the physical situation. The impact parameter should be a random point in the (x, y) plane. Because of the (near) symmetry of the collision, it is conventional to use polar coordinates (b, ϕ) instead of (x, y). This also makes it easy to introduce a sampling cut directly on b.

Exercise **3** Impact parameter, unweighted

Construct two sampler functions. One which samples impact parameter in cartesian coordinates, and returns (b, ϕ), and one which samples directly in (b, ϕ). Make a histogram showing the sampled b-values for both, you can use `pyplot.hist` directly in the file. The two histograms should agree, and you can not sample b from a flat distribution – how should you sample it?.

The first part of the exercise is already partly solved (ϕ is not calculated, only b) in the code to provide a skeleton which you can work on (resulting figure shown in figure 3). The sampler and the production of the histogram looks like:

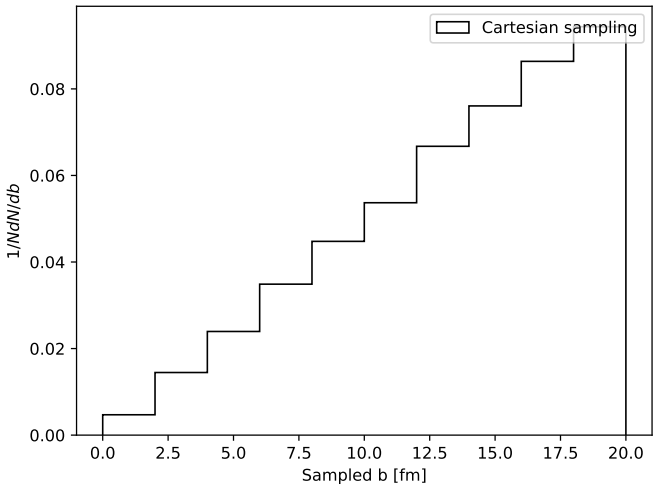

Figure 3: Cartesian sampled impact parameter, shown in a histogram.

```
# Here we sample a point in cartesian coordinates.
def impactC(sMax):
    x = 2. * sMax * (random() - 1.)
    y = 2. * sMax * (random() - 1.)
    r = math.sqrt(x*x + y*y)
    return r

# We make a list of impact parameters
bC = [impactC(20) for i in range(100000)]
# And the histogram
fig3 = plt.figure()
ax3 = fig3.add_subplot(1,1,1)
ax3.hist(bC,range=[0,20],normed=True,
        histtype="step",color="black",label="Cartesian sampling")
ax3.legend()
ax3.set_xlabel("Sampled b [fm]")
ax3.set_ylabel(r"$1/N dN/db$")
```

Sampling like this does, however, have an immediate drawback: We don't sample enough of the interesting, central events! We will therefore implement importance samling of impact parameters. This has the benefit that you can sample however you like, but comes with the price of having event weights.

Exercise **4** Impact parameter, weighted

Implement two new impact parameter samplers, and show the sampled b-values in a histogram like before. The first one should look like this, sampling from a Gaussian distribution in polar coordinates:

```
def impactGauss(width):
   r = random()
   b = math.sqrt( -math.log(r) * 2.  * width * width)
   phi = 2.0 * math.pi * random()
   w = 1./r
   return b, phi, w
```

Plot the histograms both with and without the event weights. The second should sample from an exponential distribution, like this:

```
.       def impactExp(width):
         r = random()
         b = -2.0 * math.log(r) * width
         phi = 2.0 * math.pi * random()
         w = width * math.exp(-width * b) / b
         return b, phi, w
```

For exercise 4, note that the `pyplot.hist` used above allows you to simply add an array of weights, by putting `weights=weightArray` (where `weightArray` is your array with weights) into the call to `ax.hist`.

You will now use one of your created impact parameter samplers to shift the position of the proton.

Exercise **5** Shift position

Sample an impact parameter (b, ϕ) using your favorite sampler. Shift the proton to the new position. Make a cartoon figure like in exercises 1 and 2, showing that the position did in fact change.

2.3 First Glauber calculation

You have now made the ingredients to make your first real Glauber calculation for pPb! You will need everything you have done up until this point, you may want to change to a new file for your own clarity's sake.

Exercise **6** First Glauber calculation

Make an "event loop" where you generate 1000 pPb events, shift the proton position according to a generated impact parameter, determine which nucleons are hit or not, and finally fill histograms of "Number of participants" (how many nucleons were hit) and "Number of subcollisions" (how many individual sub–collision did you have). How do you expect these numbers to be correlated? Will it be correlated in the same way in $PbPb$?

You can use the following as a skeleton for the event loop, make sure to fill in the missing parts yourself.

```
# Initialize lists counting number of subcollisions...
ncList = []
# ... number of participants...
npList = []
# ... and saving event weights.
wList = []
# Generate 1000 events.
for i in range(1000):
    if i%100 == 0:
        print "pPb generating "+str(i)+"/1000"
    # Setup both sides.
    proton = g.setupProton()
    lead = g.setupNucleus(g.At,g.rt)
    # Sample an impact parameter.
    b, phi, w = your impact parameter generator here.
    # Shift the proton.
    # proton[0]["bx"]+=
    # proton[0]["by"]+=
    # Count each sub-collision.
    nColl = 0
    for p in lead:
        if dTrans(p,proton[0]) < r:
            nColl+=1
            p["hit"] = 1
        else:
            p["hit"] = 0
    # Count the participating nucleons
    nPart = 0
```

```
for p in lead:
    if p["hit"]:
        nPart+=1
if nColl > 0:
    ncList.append(nColl)
    npList.append(nPart)
    wList.append(w)
```

You can then generate histograms from the arrays like you did before.

3 Collisions of Lead on Lead ($PbPb$)

We are now ready to simulate PbPb collisions. Since by now you are familiar with the framework, the instructions will not be quite as verbose.

Exercise 7 *PbPb* cartoon plot

Setup a cartoon plot for a *PbPb* collision, marking projectile and target nucleons in different colors. Mark also those which have participated in a subcollision.

For exercise 7, make sure that you generate one impact parameter, and shift *all* nucleons on one side by this. You set up target and projectile Lead like this:

```
g = GlauberBase("1000822080","1000822080")
leadP = g.setupNucleus(g.Ap,g.rp)
leadT = g.setupNucleus(g.At,g.rt)
```

Check if your cartoon plot makes sense. It will reveal to you if you replaced correctly loops with double loops etc. If it does, you can now do a Glauber calculation for *PbPb*.

Exercise 7 *PbPb* Glauber calculation

Make an event loop generating 1000 *PbPb* events, and generate histograms for number of subcollisions and number of participants.

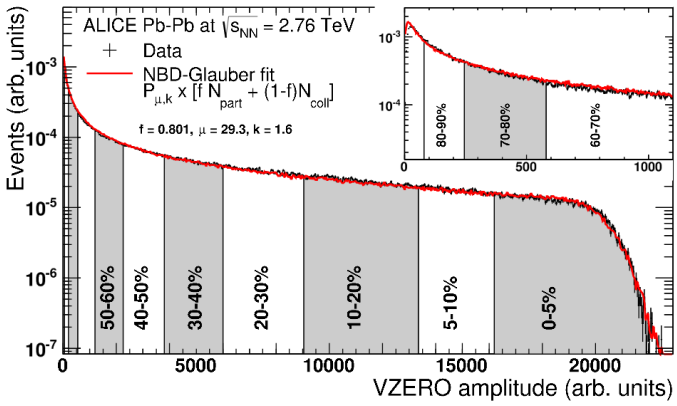

Figure 4: Distribution of the forward energy flow in the ALICE forward detectors, called VZERO. The distribution is fitted with the Glauber-based fit shown as a line. The extracted centrality classes are indicated. Figure taken from ref. [4], which also contains more information about the concrete analysis.

4 Understanding centrality

A key concept of heavy ion collision is centrality". What theorists often wants to use as an input for models, is the impact parameter of a collision – or maybe the number of participants or sub–collisions. Unfortunately none of those things are experimentally accessible quantities. What experimentalists measure, is a "forward event activity" – $E_\perp$ or number of particles in the forward and backward directions – and then correlate that to centrality. Events with more activity are more central. An example is shown in figure 4 by ALICE [4].

In the `.yoda` (histogram) file present in the folder, there are histograms of such a "centrality estimator" generated with the Pythia Monte Carlo as well. In this case it is $\sum E_\perp$ in the ALICE VZERO region (look in the RIVET analysis for the precise definition). This is used in the RIVET analysis to put the events in the correct centrality bin. In this section you will correlate the generated centrality measure with your Glauber calculation to figure out how many "participants" a given centrality bin corresponds to.

4.1 Correlating Glauber and centrality

In the folder there is a file called `centrality.py`, which you can use as a skeleton for this part. It has been prepared with several of the things you implemented in the previous exercises; a distance function, nucleus setup, impact parameter sampler and an event loop – feel free to replace this with your own implementations!

If you take a closer look at figure 4, you will notice that they don't use N_{part} or N_{coll} directly as a proxy for the centrality measure, but rather a linear combination called number of ancestors:

$$N_a = \alpha N_{part} + (1 - \alpha) N_{coll}. \tag{1}$$

Exercise 8 Number of ancestors

Take as an ansatz $\alpha = 0.8$, and construct histograms of all three quantities. Divide the histograms into deciles (10 % percentiles) and find the average number of ancestors, participants and subcollisions in for each centrality. Construct scatter plots of your finding with centrality on the abscissa and the calculated average on the ordinate.

It is a good idea to implement a general function to calculate the deciles once and for all, as you will be using it again. Take the following as a skeleton implementation, where you can fill in the blanks.

```
# Skeleton implementation which assumes that the input distribution
# is already normalized.
def percentiles(xArray, yArray):
    # Check that integral is 1
    if abs(sum(yArray) - 1.) > 1e-4:
        print "Percentiles error! sum(y) is not 1, but "+str(sum(yArray))
        return
    sumY = 0.
    xVals = []
    # Put here a loop over the input arrays, which fills
    # xVals with a value every time you get to a new decile,
    # for ...
    #
    #
    return xVals
```

In the next exercise you should do the same thing with the centrality measure $\sum E_{\perp}$, calculated with the Monte Carlo. The events are already run, and the relevant histogram is called `etaCent1` in one of the two ALICE analyses in the `.yoda` file. The file is called `pbpbPythia.yoda`. The `.yoda` histogram can be imported directly into your Python script using the function below:

```
import yoda

def readHisto1D(histname, filename):
    histos = yoda.core.read(filename+".yoda")
    x = []
    y = []
    h = histos[histname]
    for b in h:
        x.append(b.xMid)
        y.append(b.sumW)
    return x,y

# Use the function like this:
etx, ety = readHisto1D("/ALICE_2013_I1225979/etaCent1","pbpbPythia")
```

Exercise **9** Ancestors versus measured centrality

Import the centrality measure histogram and divide it into deciles. Construct first a scatter plot with the average $\sum E_\perp$ as function of centrality. Construct then three scatter plots where you for each of the average quantities calculated in the previous exercise, plot $\sum E_\perp$ as function of the average quantities. They should all be approximately linear. Which one has the best correlation? Is it sensible to use number of ancestors as a proxy for the centrality measure instead of the others?

If you feel like playing around at this point, you can try to optimize the ansatz of $\alpha = 0.8$ to get a better correlation between ancestors and $\sum E_\perp$.

4.2 Scaling behaviours

Particle production at mid-rapidity scales with the number of participants across different colliding systems (PbPb, AuAu, XeXe *etc.*). In this final part you will produce a scaling behaviour plot for PbPb similar to figure 2 in ref. [5], from which you can get the similar thing for AuAu. The multiplicities are also generated in the output `.yoda` file where you got the centrality measure from. It is, however, not at mid-rapidity, but the full η-spectrum. You can either read if off by hand, or use the following function to extract multiplicity at mid-η (again, use the pre-generated `.yoda` files).

```
# Extract the generated multiplicites at mid-rapidity from the
# generated yoda-files.

def multAtMid(histname,filename):
    xArray, yArray = readHisto1D(histname,filename)
    integral = 0.
    for x,y in zip(xArray,yArray):
        if x <= 0.5 and x >= -0.5:
            integral += y
    return integral
```

Exercise **10** Scaling plot

Extract the average multiplicity at mid-rapidity in each of the centrality bins (the produced `.yoda` files has a different binning for the two most central bins, just take the average of 0-5% and 5-10% and count that as 0-10%.). Use your own Glauber calculation to estimate the number of participants in each centrality bin *where centrality should be defined in terms of ancestors!*. Plot $\langle dN/d\eta \rangle / (\langle N_{part} \rangle / 2)$ as a function of $\langle N_{part} \rangle$ and compare to the paper.

References

[1] J. Bellm and C. Bierlich, "PISTA: Posterior Ion STAcking," 2018.

[2] M. L. Miller, K. Reygers, S. J. Sanders, and P. Steinberg, "Glauber modeling in high energy nuclear collisions," *Ann. Rev. Nucl. Part. Sci.*, vol. 57, pp. 205–243, 2007.

[3] C. Bierlich, G. Gustafson, and L. Lnnblad, "Diffractive and non-diffractive wounded nucleons and final states in pA collisions," *JHEP*, vol. 10, p. 139, 2016.

[4] B. Abelev *et al.*, "Centrality determination of Pb-Pb collisions at $\sqrt{s_{NN}} = 2.76$ TeV with ALICE," *Phys. Rev.*, vol. C88, no. 4, p. 044909, 2013.

[5] K. Aamodt *et al.*, "Centrality dependence of the charged-particle multiplicity density at mid-rapidity in Pb-Pb collisions at $\sqrt{s_{NN}} = 2.76$ TeV," *Phys. Rev. Lett.*, vol. 106, p. 032301, 2011.