Glauber

May 18, 2023

1 Glauber Monte Carlo calculation

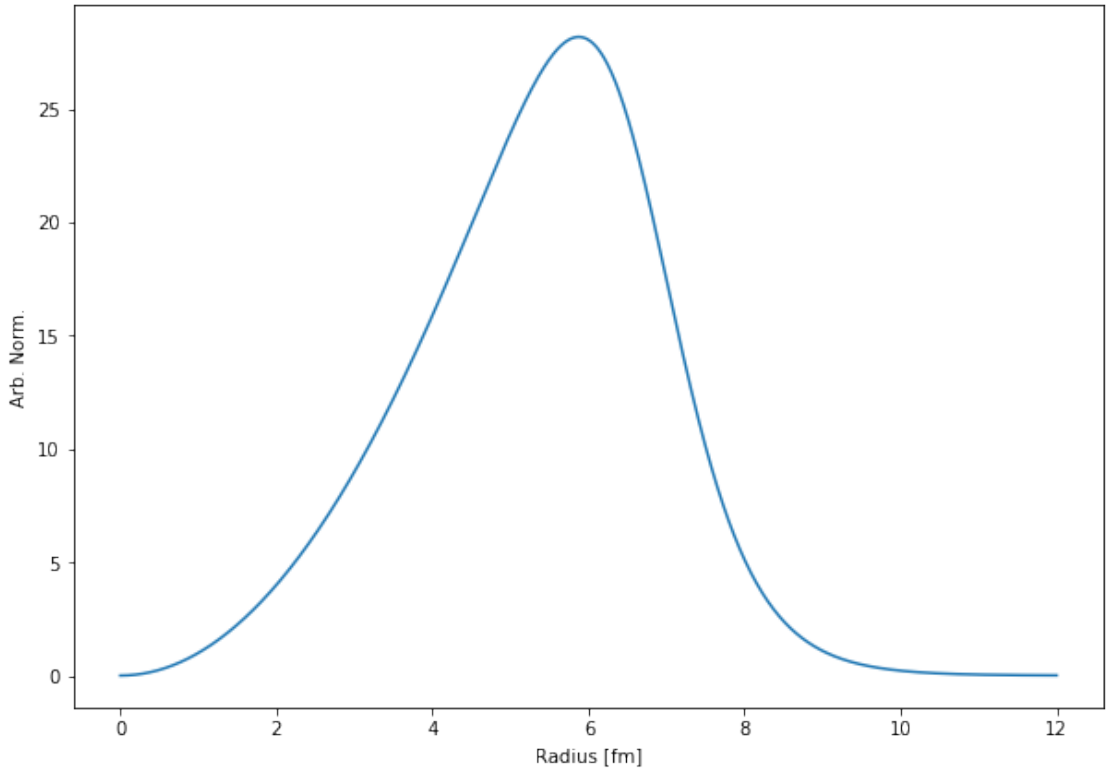
[1]:
```
# clear all variables
%reset -f

import numpy as np
import matplotlib.pyplot as plt
import itertools
import scipy as sp
import pandas as pd

##########################################

from scipy import integrate, interpolate
```

1.1 A helper function to get random numbers from a user-defined distribution:

[2]:
```
def get_random(f, xmin, xmax, n_samples):
    """Generate n_samples random numbers within range [xmin, xmax]
    from arbitrary continuous function f
    using inverse transform sampling
    """

    # number of points for which we evaluate F(x)
    nbins = 10000

    # indefinite integral F(x), normalize to unity at xmax
    x = np.linspace(xmin, xmax, nbins+1)
    F = sp.integrate.cumtrapz(f(x), x, initial=0)
    F = F / F[-1]

    # interpolate F^{-1} and evaluate it for
    # uniformly distributed rv's in [0,1[
    inv_F = sp.interpolate.interp1d(F, x, kind="quadratic")
    r = np.random.rand(n_samples)
    return inv_F(r)
```

1.2 Define nuclear density distributions

[3]:
```
# dN/dr distribution for Pb
def dndr_lead(r):
    '''nucleon density distribution of Pb (arbitrary normalization)'''
    R = 6.68 # fm
    a = 0.54 # fm
    return r**2/(1 + np.exp((r-R)/a))

##########################################

# Visualise the distribution

plt.figure(figsize=(10,7))
plt.plot(np.linspace(0, 12, 1000), dndr_lead(np.linspace(0, 12, 1000)))
plt.xlabel('Radius [fm]')
plt.ylabel('Arb. Norm.');
```



1.3 Parameters: #events, nuclear mass numbers, nucleon-nucleon cross section

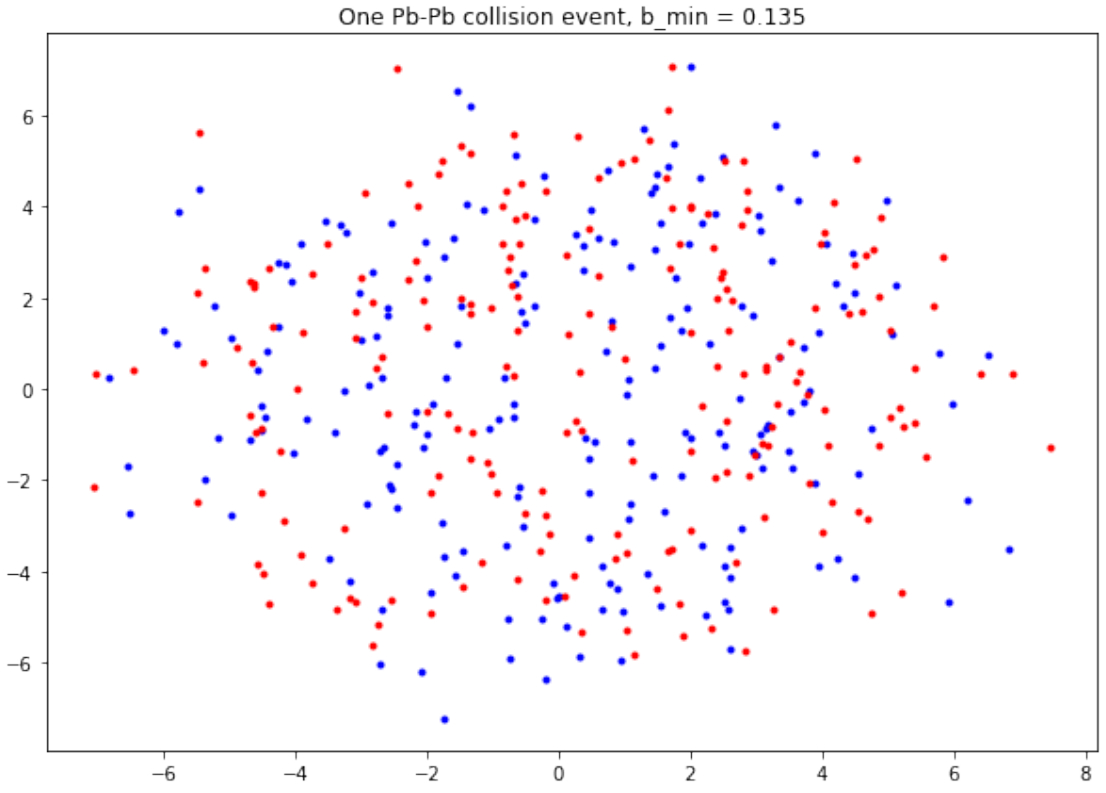
[4]:
```
n_events = 10000
A1 = 208 # lead
A2 = 208 # lead
sigma_nn_inel_fm2 = 64. / 10;  # 1 mb = 0.1 fm^2
```

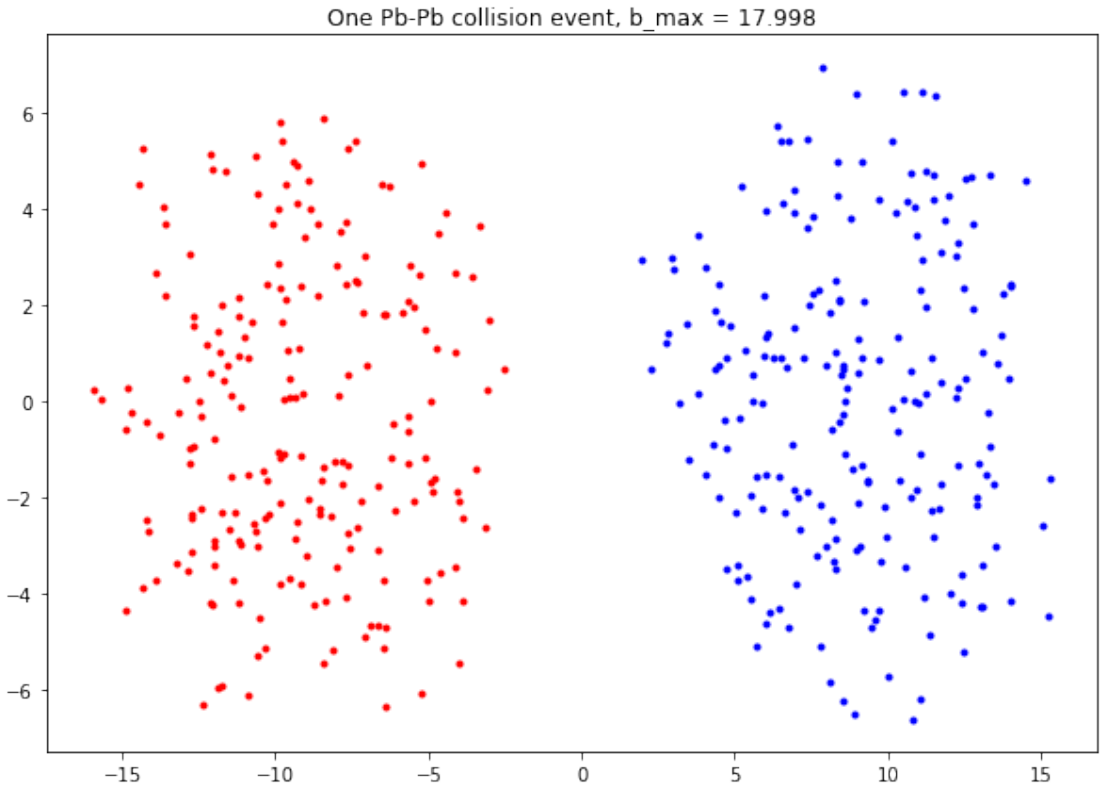
2 Define spatial positions of all nucleons in nucleus 1 and nucleus 2 for all events

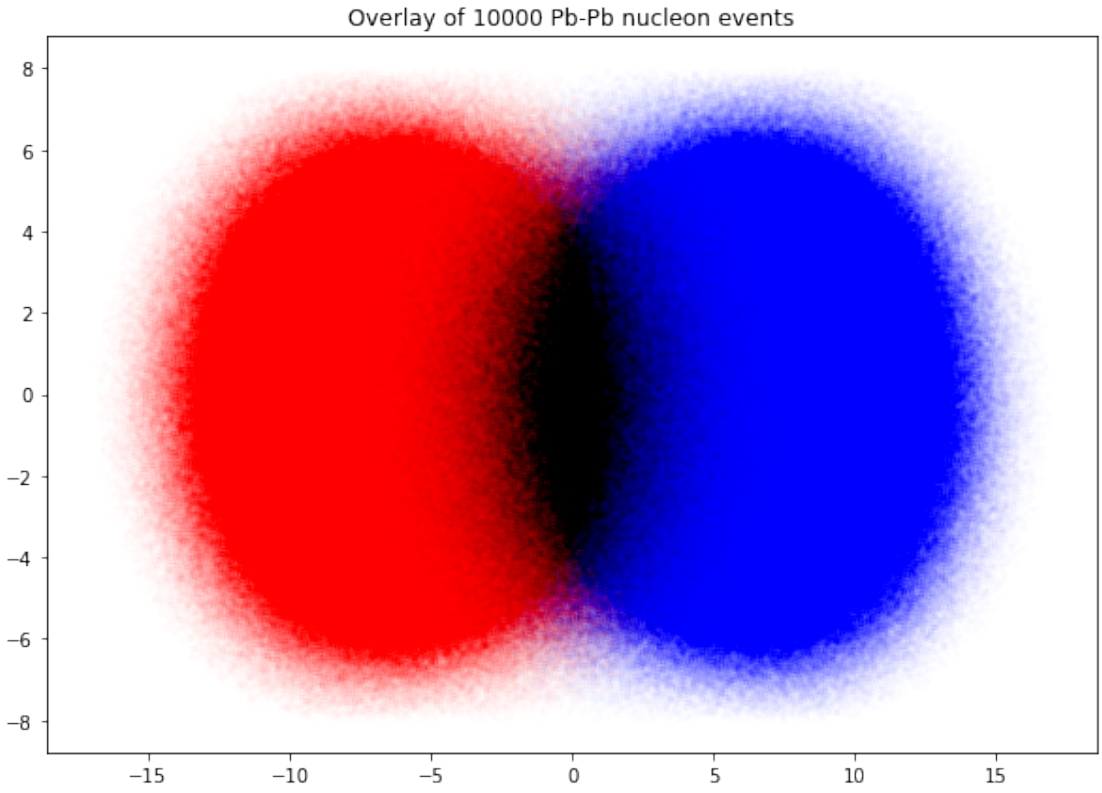
Doing this beforehand requires more memory but is faster.

[5]:
```
bmax = 18. # fm

# draw values from impact parameter distribution dN/db = 2 pi b using the␣
 ↪inverse transform method
z = np.random.uniform(0., 1., n_events)
b = bmax * np.sqrt(z)
impact_pars = b.reshape(-1, 1) # this shape (impact parameters as column␣
 ↪vector) is needed below

rmax = 8 # maximum radius in fm fm
R1 = get_random(dndr_lead, 0., rmax, A1 * n_events)
R2 = get_random(dndr_lead, 0., rmax, A2 * n_events)

R1 = R1.reshape((n_events, A1))
R2 = R2.reshape((n_events, A2))

phi1 = np.random.uniform(0., 2*np.pi, (n_events, A1))
theta1 = np.arccos(np.random.uniform(-1., 1., (n_events, A1)))
x1 = R1 * np.sin(theta1) * np.cos(phi1) + impact_pars/2.
y1 = R1 * np.sin(theta1) * np.sin(phi1)
z1 = R1 * np.cos(theta1)

phi2 = np.random.uniform(0., 2*np.pi, (n_events, A2))
theta2 = np.arccos(np.random.uniform(-1., 1., (n_events, A2)))
x2 = R2 * np.sin(theta2) * np.cos(phi2) - impact_pars/2.
y2 = R2 * np.sin(theta2) * np.sin(phi2)
z2 = R2 * np.cos(theta2)

##########################################

bmax = b.argmax()
bmin = b.argmin()

# Plot one event at minimal impact parameter
```

```
plt.figure(figsize=(10,7))
plt.plot(x1[bmin],y1[bmin],'.',color='blue',label='nucleon 1')
plt.plot(x2[bmin],y2[bmin],'.',color='red', label= 'nucleon 2')
plt.title("One Pb-Pb collision event, b_min = {}".format(b[bmin].round(3)))
plt.show()

# Plot one event at maximum impact parameter

plt.figure(figsize=(10,7))
plt.plot(x1[bmax],y1[bmax],'.',color='blue',label='nucleon 1')
plt.plot(x2[bmax],y2[bmax],'.',color='red', label= 'nucleon 2')
plt.title("One Pb-Pb collision event, b_max = {}".format(b[bmax].round(3)))
plt.show()

# Plot 10000 events at different impact parameters

plt.figure(figsize=(10,7))
for i in range(len(x1)):
    plt.plot(x1[i],y1[i],'.',color='blue', alpha = 0.005)
    plt.plot(x2[i],y2[i],'.',color='red', alpha = 0.005)
plt.title("Overlay of 10000 Pb-Pb nucleon events")
plt.show()
```




One Pb-Pb collision event, b_max = 17.998
6
4
2
0
−2
−4
−6
−15
−10
−5
0
5
10
15



2.1 The main event loop

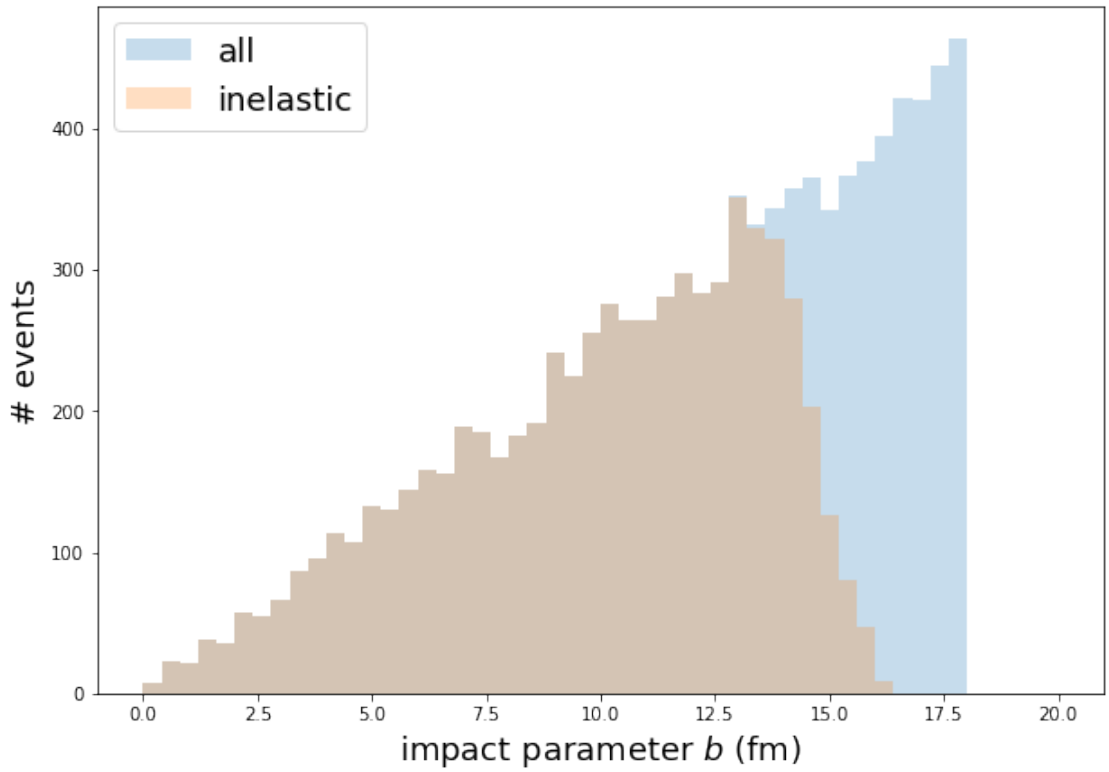
[6]:
```
from ipywidgets import IntProgress
from IPython.display import display

progress_bar = IntProgress(min=0, max=n_events, description='Events:') #␣
 ↪instantiate the bar
display(progress_bar) # display the bar

# array to store Npart and Ncoll values for each event
npart = np.zeros(n_events)  # array of Npart values (initialized with zeros)
ncoll = np.zeros(n_events)  # array of Ncoll values (initialized with zeros)

for i_event in range(n_events):

    if i_event % 10 == 0: progress_bar.value += 10

    # array to store number of collisions for each nucleon in this event (for␣
 ↪nucleus 1 and nucleus 2)
    nc1 = np.zeros(A1)
    nc2 = np.zeros(A2)
```

```
    # list with nucleon indicees
    in1 = range(A1)
    in2 = range(A2)

    # consider all pairs (nucleon 1 from nucleus 1, nucleon 2 from nucleus 2)
    for (i1, i2) in itertools.product(in1, in2):

        # calculate (squared) distance of the two nucleons in the transverse␣
↪plane

        # add code here
        d_squared =␣
↪(x1[i_event,i1]-x2[i_event,i2])**2+(y1[i_event,i1]-y2[i_event,i2])**2

        # check if two nucleon are close enough

        # add code here
        if d_squared < (sigma_nn_inel_fm2/np.pi):
            ncoll[i_event] += 1
            if (nc1[i1] == 0): npart[i_event] += 1
            if (nc2[i2] == 0): npart[i_event] += 1

            nc1[i1] += 1
            nc2[i2] += 1
```

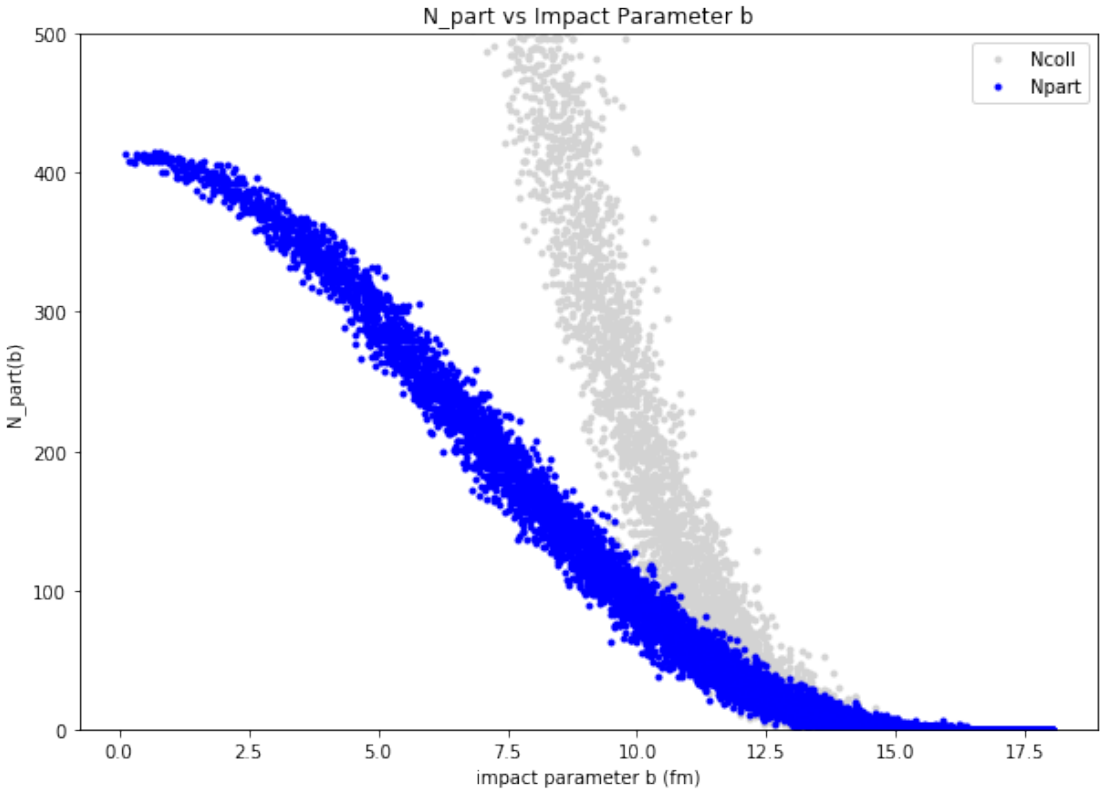
```
IntProgress(value=0, description='Events:', max=10000)
```

2.2 Plot impact parameter distribution

[17]:
```
plt.figure(figsize=(10,7))

plt.hist(b, bins=50, range=(0., 20), alpha=0.25, label='all')
plt.hist(b[ncoll > 0], bins=50, range=(0., 20), alpha=0.25, label='inelastic')
plt.xlabel('impact parameter $b$ (fm)', size=18)
plt.ylabel('# events', size=18)
plt.legend(fontsize=18, loc='upper left')
plt.show()
```



[8]:
```
# add code here

plt.figure(figsize=(10,7))

plt.plot(b,ncoll,'.',label='Ncoll',color='lightgray')
plt.plot(b,npart,'.',label='Npart',color='blue')
plt.xlabel("impact parameter b (fm)")
plt.ylim(0,500)
plt.ylabel("N_part(b)")
plt.title("N_part vs Impact Parameter b")
plt.legend()
plt.show()
```



2.3 Mean N_{part} and N_{coll} for the 10% most central inelastic collisions

[9]:
```
bmax_0_10 = np.quantile(b[ncoll > 0], 0.1)
mask = (ncoll > 0) & (b < bmax_0_10)

# add code here

npart_mean = npart[mask].mean()
ncoll_mean = ncoll[mask].mean()
b_mean = b[mask].mean()
print(f"N_part mean: ",npart_mean)
print(f"N_coll mean: ",ncoll_mean)
print(f"b mean: ",b_mean)
```

```
N_part mean:  358.3338257016248
N_coll mean:  1564.8522895125554
b mean:  3.1256126146780585
```

2.4 Total inelastic cross section

The total inelastic cross section is given by

$$\sigma_{\text{inel}} = \int_0^{b_{\text{max}}} \frac{d\sigma_{\text{inel}}}{db}\, db.$$

From the Glauber Monte Carlo calculation we have dn_{inel}/db and so we just need to determine the proper normalization N:

$$\sigma_{\text{inel}} = N \int_0^{b_{\text{max}}} \frac{dn_{\text{inel}}}{db}\, db = N \cdot n_{\text{inel}}.$$

For the impact parameter distribution of the generated events in the interval 0-b_{max} with $b_{\text{max}} = 10\,\text{fm}$ we know the geometric cross section

$$\sigma_{\text{geo}} = \pi b_{\text{max}}^2 = N \int_0^{b_{\text{max}}} \frac{dn_{\text{gen}}}{db}\, db = N \cdot n_{\text{gen}} \quad \Rightarrow \quad N = \frac{\pi b_{\text{max}}^2}{n_{\text{gen}}}.$$

This then gives

$$\sigma_{\text{inel}} = \frac{n_{\text{inel}}}{n_{\text{gen}}} \pi b_{\text{max}}^2$$

[16]:
```
# add code here

b_max = 18
n_inel = len(npart[(ncoll >= 1) & (b <= b_max)])
n_gen = len(npart[b <= b_max])
sigma_inel = (n_inel/n_gen)*np.pi*b_max**2
print("Total inelastic Pb-Pb cross section in fm²: ", sigma_inel)
```

```
Total inelastic Pb-Pb cross section in fm²:  688.7967025736849
```