8,5/10

Sheet05

May 24, 2022

Exercise 11

a) Signal MC

[1]:

```
import numpy as np
import pandas as pd

seed = 1

rng = np.random.default_rng(seed)

def event_neutrino(size):
    gamma = 2.7
    u = rng.uniform(size=size)

    # unit of result is TeV

    return np.power(-u*1**(-gamma+1)+1**(-gamma+1),1/(-gamma+1))

energies = event_neutrino(100000)

data_frame = pd.DataFrame(data = energies, columns=['Energy'])

data_frame
```

[1]:

```
         Energy
0      1.524716
1      5.857226
2      1.095895
3      5.734597
4      1.245873
…           …
99995  2.106965
99996  1.212095
99997  1.899604
```

```
99998  1.416921
99999  1.339617

[100000 rows x 1 columns]
```

b) Neumann Rejection

[2]:

```
def det_prop(E):
    return np.power((1-np.exp(-E/2)), 3)

u_1 = energies

u_2 = rng.uniform(0,1, len(u_1))

acceptance_mask = [det_prop(u_1[i]) > u_2[i] for i in range(len(u_1))]

data_frame['AcceptanceMask']= acceptance_mask
data_frame
```

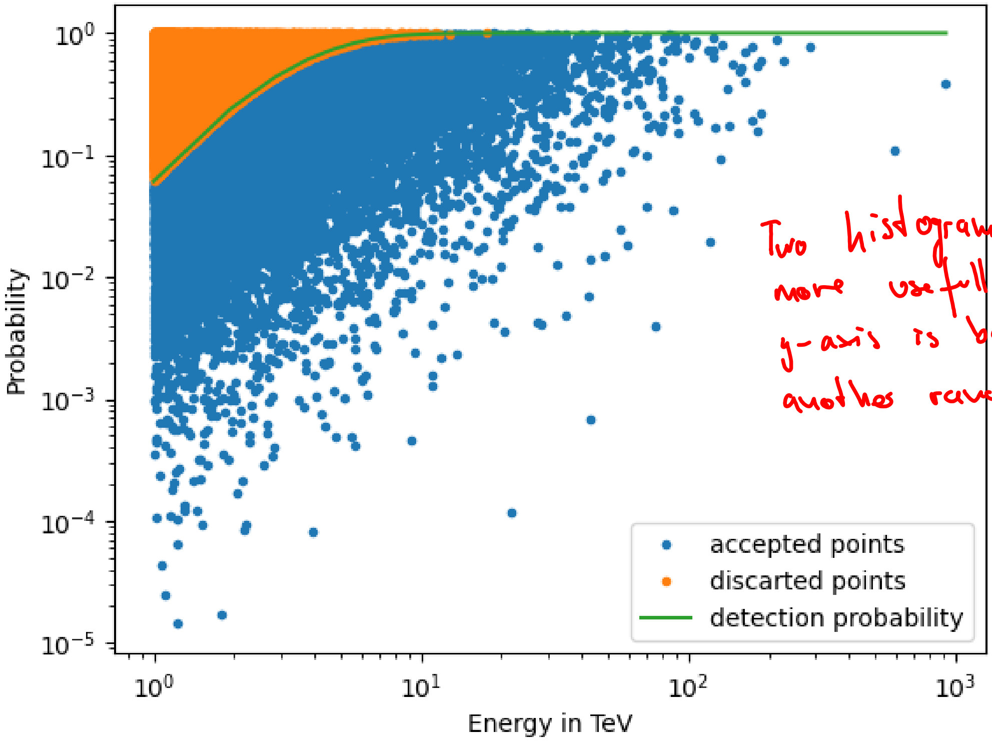
[2]:

```
          Energy  AcceptanceMask
0      1.524716           False
1      5.857226            True
2      1.095895           False
3      5.734597            True
4      1.245873           False
…           …              …
99995  2.106965           False
99996  1.212095           False
99997  1.899604           False
99998  1.416921           False
99999  1.339617           False

[100000 rows x 2 columns]
```

[3]:

```
import matplotlib.pyplot as plt

%matplotlib widget


plt.figure()

plt.plot(u_1[acceptance_mask], u_2[acceptance_mask], linestyle='none', marker='.
 ↪', label = 'accepted points')
plt.plot(u_1[np.invert(acceptance_mask)], u_2[np.invert(acceptance_mask)],␣
 ↪linestyle='none', marker='.', label = 'discarted points')
```

```
x = np.linspace(min(u_1[acceptance_mask]), max(u_1[acceptance_mask]), 1000)
plt.plot(x, det_prop(x), label='detection probability')
plt.xlabel('Energy in TeV')
plt.ylabel('Probability')
plt.yscale('log')
plt.xscale('log')
plt.legend()


None

#Note: thickness of markers make discarted points appear below the distribution
#But they are exactly on top of it!
```



Two histograms would be more usefull as your y-axis is basically just another random number.

(✓)

c)

```
[ ]: def standard_normal(self, size=None, seed = 1):
             '''
             Override standard normal with using the Marsaglia polar method

             Blatt 5, Aufgabe 11b)
```

```
        '''
        # Fügen Sie hier den Code ein um Zufallszahlen aus der
        # angegebenen Verteilung zu erzeugen

        # dummy, so the code works. Can be removed / replaced

        rng_1 = np.random.default_rng(seed=seed)
        rng_2 = np.random.default_rng(seed=seed+1)

        u_1 = rng_1.uniform(size=size)
        u_2 = rng_2.uniform(size=size)

        values = np.sqrt(-2*np.log(u_1))*np.cos(2*np.pi*u_2)
```

you could just draw 2n numbers from one rng

This is the Box-Müller-Method, not the Polar-method.

Not using the other half of the possible random numbers is highly inefficient. You could just draw half the rngs above and concatenate two arrays here.

```
        return values

    def normal(self, loc=0, scale=1, size=None, seed = 1):
        '''
        Scale and shift standard normal values

        Blatt 5, Aufgabe 11b)
        '''
        # Fügen Sie hier den Code ein um Zufallszahlen aus der
        # angegebenen Verteilung zu erzeugen

        # dummy, so the code works. Can be removed / replaced

        values = self.standard_normal(size=size, seed = seed)*scale + loc

        return values
```

✓

The results are attached at the end of the document. It can be seen that the polar method works and the results are satisfactory. the generated data fit the normalized and the standard normal distribution

d) Energy measurement

[4]:
```
from project_c3.random import Generator

gen = Generator(seed = 10)

i = -1
seed = 10983
hits = np.zeros(len(energies))

while i < len(energies)-1:
    i += 1
    seed += 1
    hits[i] = int(gen.normal(loc=10*energies[i], scale=2*energies[i], seed =␣
  ↪seed))

    if hits[i] <= 0:
        i -= 0
        print(here)
    else:
        None

print(np.where(hits == 0))

data_frame['NumberOfHits'] = hits
data_frame
```

```
(array([], dtype=int64),)
```

[4]:

```
         Energy  AcceptanceMask  NumberOfHits
0      1.524716           False          18.0
1      5.857226            True          43.0
2      1.095895           False          11.0
3      5.734597            True          60.0
4      1.245873           False           9.0
…           …               …            …
99995  2.106965           False          21.0
99996  1.212095           False          13.0
99997  1.899604           False          18.0
99998  1.416921           False          12.0
99999  1.339617           False           9.0

[100000 rows x 3 columns]
```

e) Spatial measurement

[5]:

```
def sigma(N):
    return 1/np.log10(N+1)

gen_y = Generator(seed = 1)
gen_x = Generator(seed = 1)

x_location = np.zeros(len(hits))
y_location = np.zeros(len(hits))

i = 0

#x_location = gen_x.normal(loc=7, scale=sigma(hits[i]), seed=seed, size =␣
  ↪len(x_location))
```

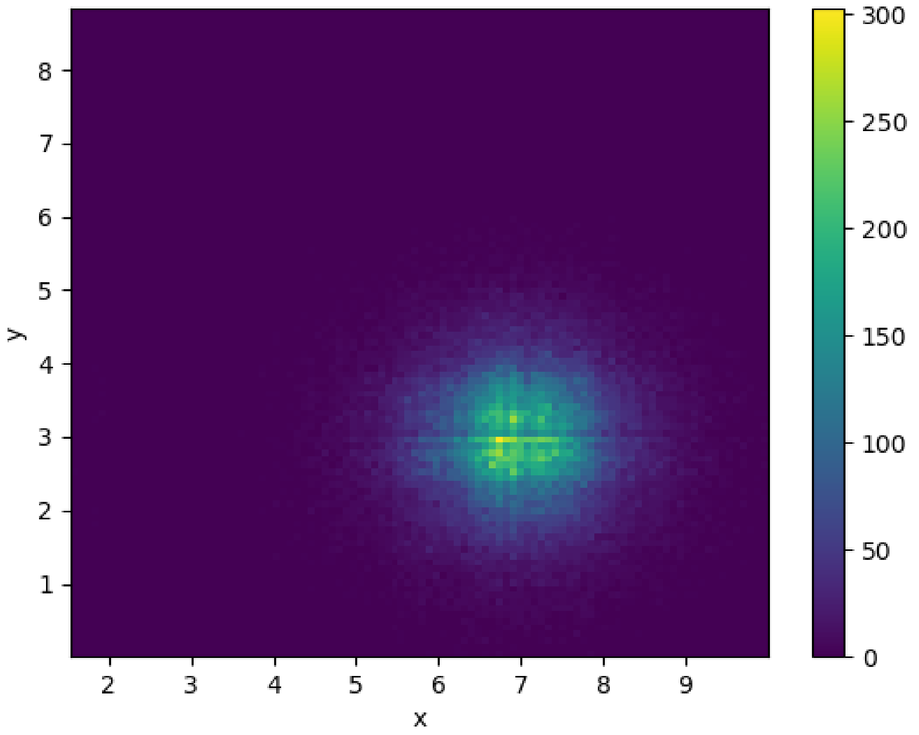
```
#y_location = gen_x.normal(loc=7, scale=sigma(hits[i]), seed=seed, size =␣
 ↪len(x_location))
seed = 125
rng_seed = np.random.default_rng(seed=seed)

seed_array_x = np.array(rng_seed.uniform(0,1000,size=len(hits)), dtype = int)

seed_array_y = np.array(rng_seed.uniform(0,1000,size=len(hits)), dtype = int)



while i < len(hits):

    x_location[i] = gen_x.normal(loc=7, scale=sigma(hits[i]),␣
 ↪seed=seed_array_x[i])
    y_location[i] = gen_y.normal(loc=3, scale=sigma(hits[i]),␣
 ↪seed=seed_array_y[i])

    i += 1

    '''
    if (x_location[i] <0 or x_location[i] >10) or (y_location[i] < 0 or␣
 ↪y_location[i] > 10):
        rng_seed = np.random.default_rng(seed=seed +10)
        seed_array_x = np.array(rng_seed.uniform(0,1000,size=len(hits)), dtype␣
 ↪= int)
        seed_array_y = np.array(rng_seed.uniform(0,1000,size=len(hits)), dtype␣
 ↪= int)
        continue
    else:
        print(i)
        i+=1
    '''


while True:
    seed = int(rng.uniform(0, 1000))
    mask = np.logical_or(y_location > 10, y_location < 0)
    if np.invert(mask.all()) == True :
        y_location[mask] = gen_y.normal(loc=3, scale=sigma(hits[mask]),␣
 ↪seed=seed)
    if np.count_nonzero(mask) == 0:
        break

while True:
    seed = int(rng.uniform(0, 1000))
    mask = np.logical_or(x_location > 10, x_location < 0)
    if np.invert(mask.all()) == True :
        x_location[mask] = gen_x.normal(loc=3, scale=sigma(hits[mask]),␣
 ↪seed=seed)
    if np.count_nonzero(mask) == 0:
        break

data_frame['x'] = x_location
data_frame['y'] = y_location

data_frame
```

[5]:

```
         Energy  AcceptanceMask  NumberOfHits         x         y
0      1.524716           False          18.0  6.688328  3.347590
1      5.857226            True          43.0  7.259255  2.514566
2      1.095895           False          11.0  7.883510  1.836809
3      5.734597            True          60.0  6.794347  3.025733
4      1.245873           False           9.0  7.563170  3.420931
…          …               …              …         …       …
99995  2.106965           False          21.0  8.665172  3.199063
99996  1.212095           False          13.0  5.860965  1.789484
99997  1.899604           False          18.0  6.240113  3.779963
99998  1.416921           False          12.0  5.810179  2.724178
99999  1.339617           False           9.0  6.525750  4.525097

[100000 rows x 5 columns]
```

[6]:

```
plt.figure()

plt.hist2d(x_location, y_location, bins= 100)
plt.ylabel('y')
plt.xlabel('x')
plt.colorbar()

None
```



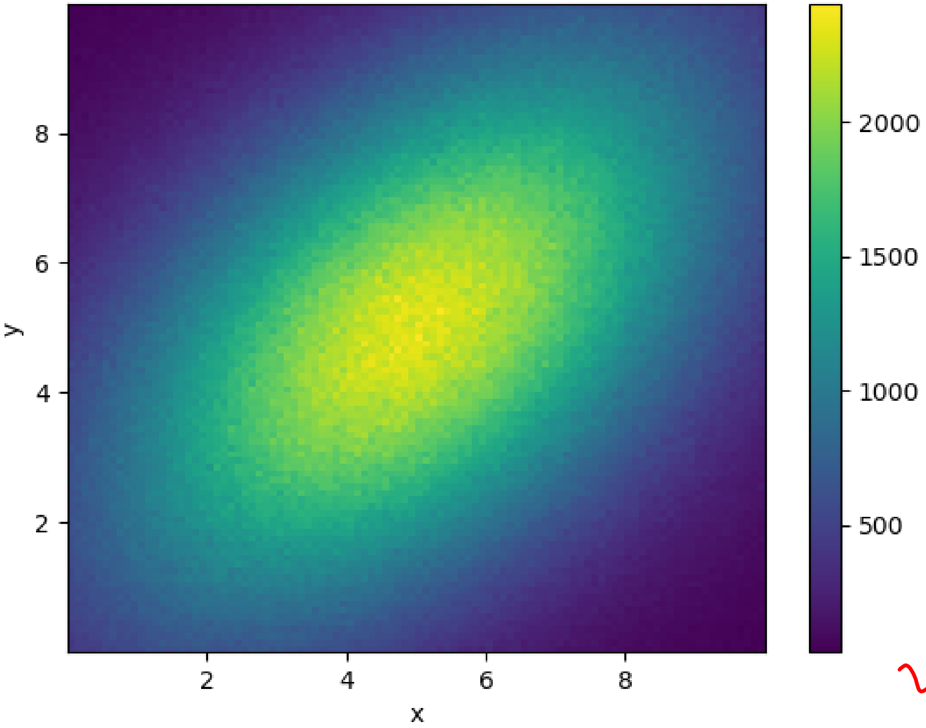
✓

f) Underground MC

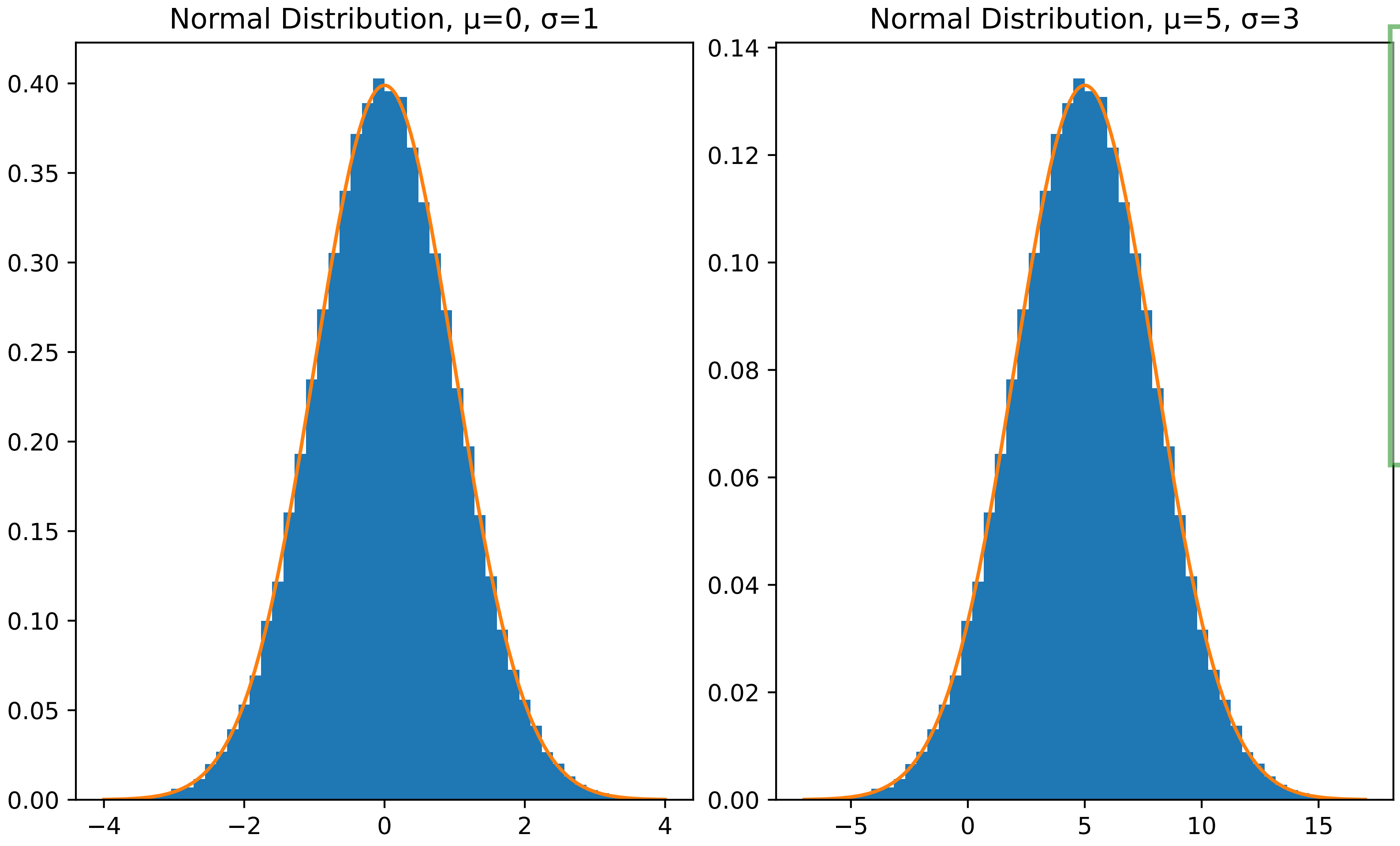
[7]:

```
number_events = 10000000

seed =+1

hits = gen.normal(loc = 2, scale = 1, size = number_events, seed = seed)

hits = np.array(np.power(10, hits), dtype = int)

data_frame = pd.DataFrame(data = hits, columns=['NumberOfHits'])


sigma = 3

mu = 5

roh = 0.5

x_location = np.zeros(number_events)
y_location = np.zeros(number_events)
```

```
i = 0


y_norm = gen_y.standard_normal(seed=20, size = number_events)
y = sigma * y_norm + mu
x_norm = gen_x.standard_normal(seed=4000, size = number_events)

x = np.sqrt(1-roh**2)*sigma*x_norm + roh * sigma * y_norm +mu


while True:
    seed = int(rng.uniform(0, 1000))
    mask = np.logical_or(y > 10, y < 0)
    if mask.all()== False :
        y_norm[mask] = gen_y.standard_normal(seed=seed, size = np.count_nonzero(mask))
        y[mask] = sigma* y_norm[mask] +mu
    if np.count_nonzero(mask) == 0:
        break

while True:
    seed = int(rng.uniform(0, 1000))
    mask = np.logical_or(x > 10, x < 0)
    if mask.all() == False :
        x_norm[mask] = gen_x.standard_normal(seed=seed, size = np.count_nonzero(mask))
        x[mask] = np.sqrt(1-roh**2)*sigma*x_norm[mask] + roh * sigma * y_norm[mask] +mu
    if np.count_nonzero(mask) == 0:
        break



data_frame['x'] = x
data_frame['y'] = y




plt.figure()

plt.hist2d(x, y, bins= 100)
plt.ylabel('y')
plt.xlabel('x')
plt.colorbar()

None
```


8
6
y
4
2
2
4
6
8
x
2000
1500
1000
500

✓



```
Information:
  Exercise: Polarmethode
  Group name: project_c3
Runtime:
  6.9 ms (ref: 12.6 ms)
Tests:
  size correct: True
  mean correct: True
  std correct: True
```