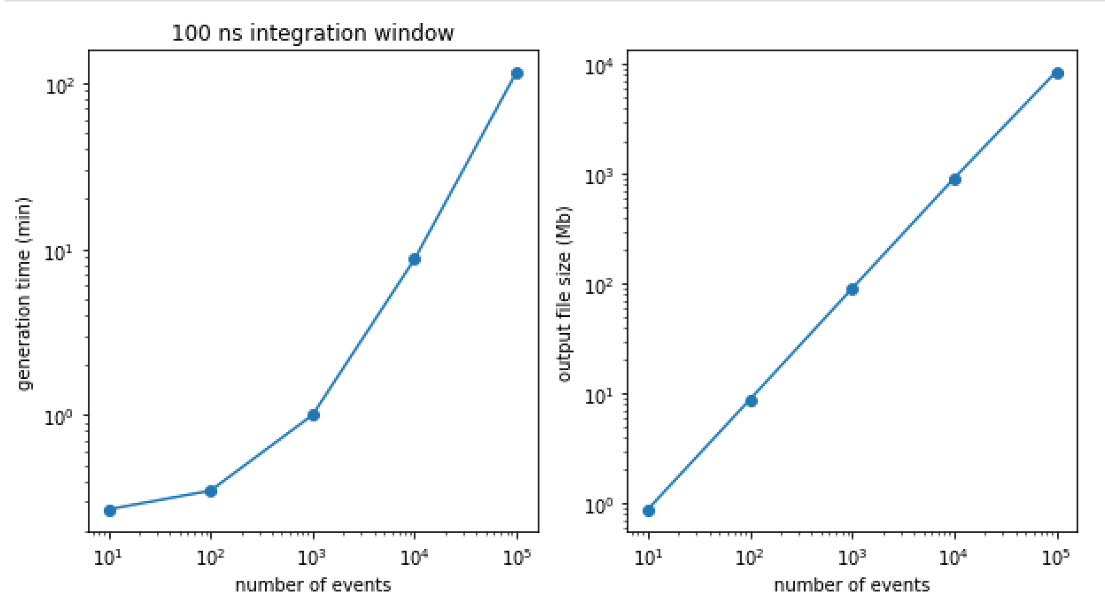
Synchrotron-radiation event generator from Synrad+ table

authors: Rey Cruz-Torres, Benjamen Sterwerf

Begin by choosing the number of events and time integration window wanted. See the next cell to get an idea of the cost of running certain number of events.

In [1]:
```
# number of synchrotron-radiation photon events to generate and integration window
n_events = 10000
integration_window = 100.e-9 # seconds
```

In [2]:
```
import matplotlib.pyplot as plt
x_nevt = [10,100,1e+03,1e+04,1e+05]
t_min = [0.27,0.35,1,8.7,117]
size_MB = [0.878,8.8,90,909,8500]
fig = plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.plot(x_nevt,t_min,marker='o')
plt.xlabel('number of events'); plt.ylabel('generation time (min)'); plt.title('100 ns integration window')
plt.xscale('log'); plt.yscale('log')
plt.subplot(1,2,2)
plt.plot(x_nevt,size_MB,marker='o')
plt.xlabel('number of events'); plt.ylabel('output file size (Mb)')
plt.xscale('log'); plt.yscale('log')
```



In [3]:
```
import numpy as np
import pandas as pd
import math
import pyhepmc_ng as hep
import ROOT
ROOT.gStyle.SetOptStat(0)
import time
t0 = time.time()
```

```
Welcome to JupyROOT 6.26/02
```

Set some important variables

In [4]:
```
pid = 22 # 22 = photon, 211 = charged pion (for testing)

outfname = 'out_int_window_{}ns_nevents_{}.hepmc'.format(integration_window*1e+09,n_events)
ROOT.gRandom.SetSeed(42)

testParticleKin = {'px':0.,'py':3.0,'pz':0.,'pid':-11,'m':5.11e-4}
```

Load data

In [5]:
```
df = pd.read_csv('combined_data.csv')
```

Explore structure of dataframe and drop some redundant columns. Save a version of this dataframe

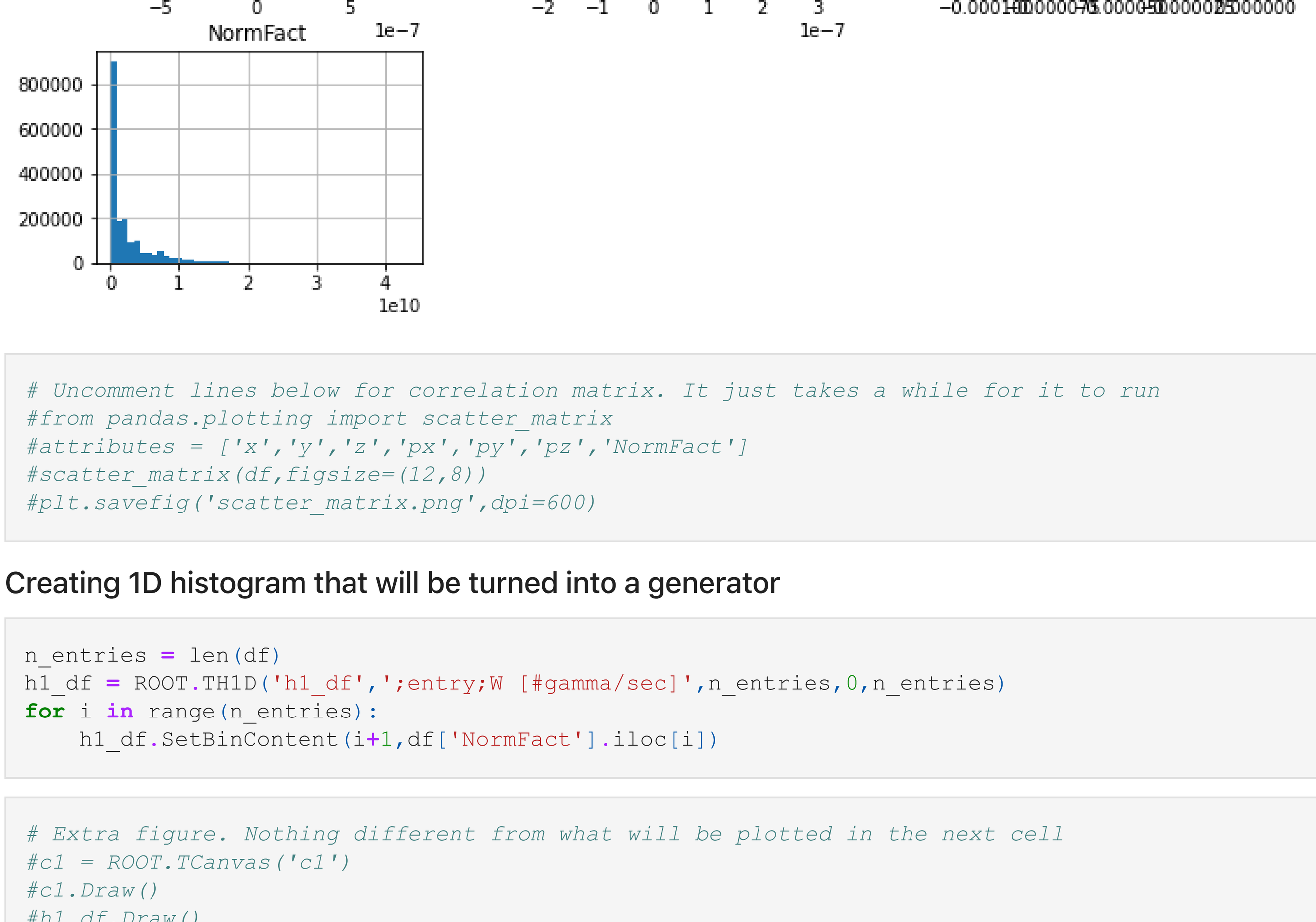
In [6]:
```
df = df.drop('E'    ,axis=1)
df = df.drop('P'    ,axis=1)
df = df.drop('Fs'   ,axis=1)
df = df.drop('facet',axis=1)
df = df.drop('rho'  ,axis=1)
df = df.drop('theta',axis=1)
df = df.drop('phi'  ,axis=1)

df = df.sort_values('NormFact')

# x, y, z are in mm, and px, py, pz in GeV

df.head()
```

Out[6]:

	x	y	z	px	py	pz	NormFact
1620504	28.2006	8.29940	-3078.390	0.0	0.0	-0.0	0.0
1420970	-27.0238	-11.58470	-3383.230	-0.0	-0.0	-0.0	0.0
1323913	-25.2181	-15.22080	-706.387	-0.0	-0.0	-0.0	0.0
1035621	25.6664	-14.49560	-3924.800	0.0	-0.0	-0.0	0.0
845663	29.0895	-4.69247	-3911.200	0.0	-0.0	-0.0	0.0

In [7]:
```
df.hist(figsize=(12,8),bins=50)
```

Out[7]:
```
array([[<AxesSubplot:title={'center':'x'}>,
        <AxesSubplot:title={'center':'y'}>,
        <AxesSubplot:title={'center':'z'}>],
       [<AxesSubplot:title={'center':'px'}>,
        <AxesSubplot:title={'center':'py'}>,
        <AxesSubplot:title={'center':'pz'}>],
       [<AxesSubplot:title={'center':'NormFact'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



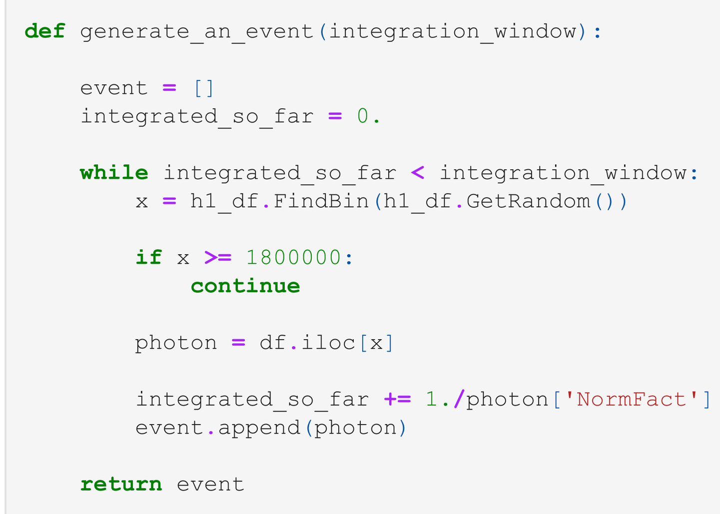
In [8]:
```
# Uncomment lines below for correlation matrix. It just takes a while for it to run
#from pandas.plotting import scatter_matrix
#attributes = ['x','y','z','px','py','pz','NormFact']
#scatter_matrix(df,figsize=(12,8))
#plt.savefig('scatter_matrix.png',dpi=600)
```

Creating 1D histogram that will be turned into a generator

In [9]:
```
n_entries = len(df)
h1_df = ROOT.TH1D('h1_df',';entry;W [#gamma/sec]',n_entries,0,n_entries)
for i in range(n_entries):
    h1_df.SetBinContent(i+1,df['NormFact'].iloc[i])
```

In [10]:
```
# Extra figure. Nothing different from what will be plotted in the next cell
#c1 = ROOT.TCanvas('c1')
#c1.Draw()
#h1_df.Draw()
#ROOT.gPad.SetLogy()
```

In [11]:
```
plt.figure()
x = np.linspace(0,1800000,1800000)
plt.hist(x,weights=df['NormFact'],bins=1800)
plt.yscale('log')
plt.xlabel('entry')
plt.ylabel(r'flux [$\gamma$/sec]')
plt.savefig('generated_events/generator.png',dpi=400)
plt.show()
```



Implementing the event generator

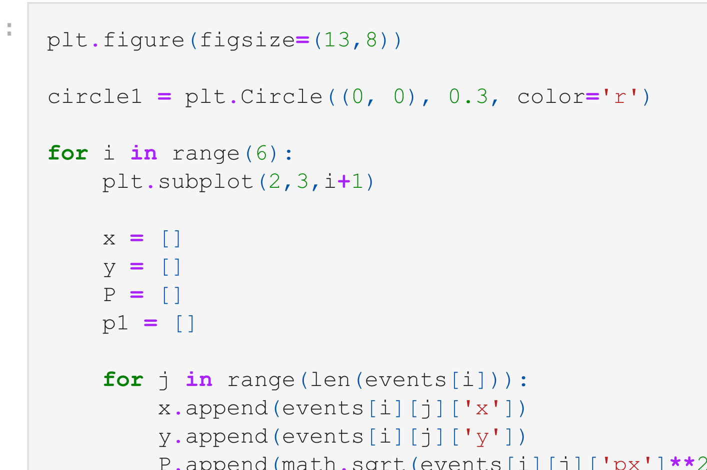
In [12]:
```
def generate_an_event(integration_window):

    event = []
    integrated_so_far = 0.

    while integrated_so_far < integration_window:
        x = h1_df.FindBin(h1_df.GetRandom())

        if x >= 1800000:
            continue

        photon = df.iloc[x]

        integrated_so_far += 1./photon['NormFact']
        event.append(photon)

    return event
```

Generating events and saving them to hepmc files

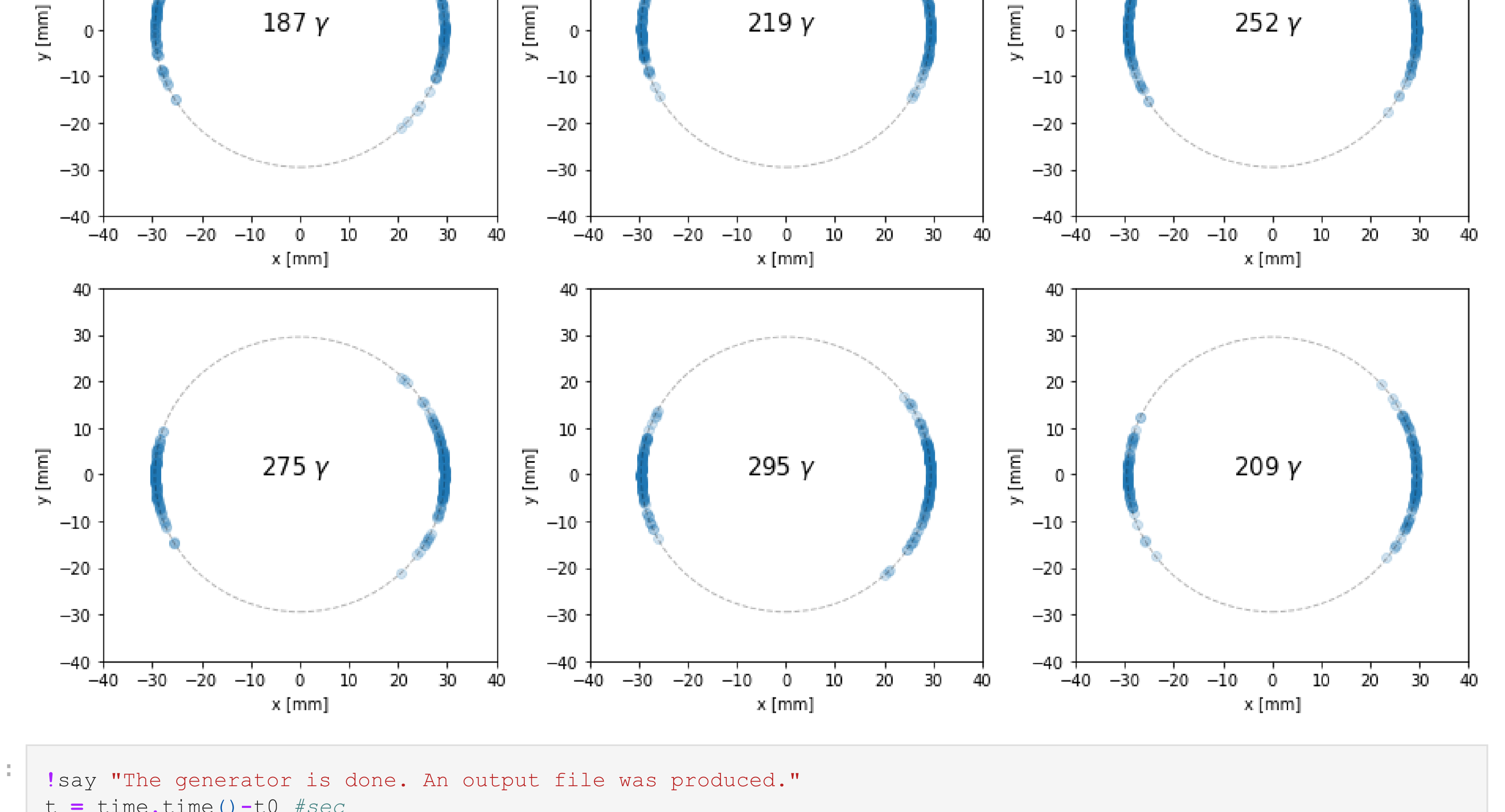
In [13]:
```
h_n_photons_per_event = ROOT.TH1D('h_n_photons_per_event','',50,100,400)

events = []
hep_events = []
photons_per_event = []

f = hep.WriterAscii(outfname)

for i in range(n_events):
    event = generate_an_event(integration_window)

    # ----------------------------------------------------
    # Save to hepmc format
    # implemented following the example from:
    # https://github.com/scikit-hep/pyhepmc/blob/master/tests/test_basic.py
    evt = hep.GenEvent(hep.Units.GEV, hep.Units.MM)
    evt.event_number = i+1
    particles_out = []
    particles_in = []
    vertices = []

    # loop over each photon in the event
    for g in range(len(event)):

        x = event[g]['x']
        y = event[g]['y']
        z = event[g]['z']

        px = event[g]['px']
        py = event[g]['py']
        pz = event[g]['pz']
        E = math.sqrt(px**2 + py**2 + pz**2)

        pinx = E*x/math.sqrt(x*x+y*y+z*z)
        piny = E*y/math.sqrt(x*x+y*y+z*z)
        pinz = E*z/math.sqrt(x*x+y*y+z*z)

        # Particles going into the vertex
        pin = hep.GenParticle((pinx,piny,pinz,E),pid,1)
        pin.generated_mass = 0.
        evt.add_particle(pin)
        particles_in.append(pin)

        # Particles coming out of the vertex
        pout = hep.GenParticle((px,py,pz,E),pid,1)
        pout.generated_mass = 0.
        evt.add_particle(pout)
        particles_out.append(pout)

        # make sure vertex is not optimized away by WriterAscii
        v1 = hep.GenVertex((x,y,z,0.))
        v1.add_particle_in(pin)
        v1.add_particle_out(pout)
        evt.add_vertex(v1)
        vertices.append(v1)

    # -------------------
    #evt.weights = [1.0]
    if i == 0:
        evt.run_info = hep.GenRunInfo()
        #evt.run_info.weight_names = ["0"]
    hep_events.append(evt)
    # --------------------------------------------------

    photons_per_event.append(len(event))

    if i < 15:
        events.append(event)

with hep.WriterAscii(outfname) as f:
    for e in hep_events:
        f.write_event(e)
```

Making some plots with the generated data

In [14]:
```
plt.figure()
plt.hist(photons_per_event)
plt.xlabel('# photons per event')
plt.savefig('generated_events/Nphotons_per_event.png',dpi=400)
plt.show()
```



In [15]:
```
plt.figure(figsize=(13,8))

circle1 = plt.Circle((0, 0), 0.3, color='r')

for i in range(6):
    plt.subplot(2,3,i+1)

    x = []
    y = []
    P = []
    p1 = []

    for j in range(len(events[i])):
        x.append(events[i][j]['x'])
        y.append(events[i][j]['y'])
        P.append(math.sqrt(events[i][j]['px']**2+events[i][j]['py']**2+events[i][j]['pz']**2))

    min_p = min(P)
    max_p = max(P)
    for p in P:
        p1.append((p-min_p)/(max_p-min_p))

    plt.scatter(x,y,marker='o',alpha=0.2)

    plt.xlim(-40,40)
    plt.ylim(-40,40)
    plt.xlabel('x [mm]')
    plt.ylabel('y [mm]')

    plt.text(-8,0,r'{} $\gamma$'.format(len(events[i])),fontsize=15)

    circle1 = plt.Circle((0,0), radius=29.5,color='black',fill=False,linestyle='--',alpha=0.3)
    plt.gca().add_patch(circle1)

plt.tight_layout()
plt.savefig('generated_events/xy.png',dpi=400)
plt.show()
```



In [16]:
```
!say "The generator is done. An output file was produced."
t = time.time()-t0 #sec
print('time that took to finish: ',t/60.,'min')
```

```
time that took to finish:  8.882911348342896 min
```