A3_Question1

August 29, 2021

0.1 Question1

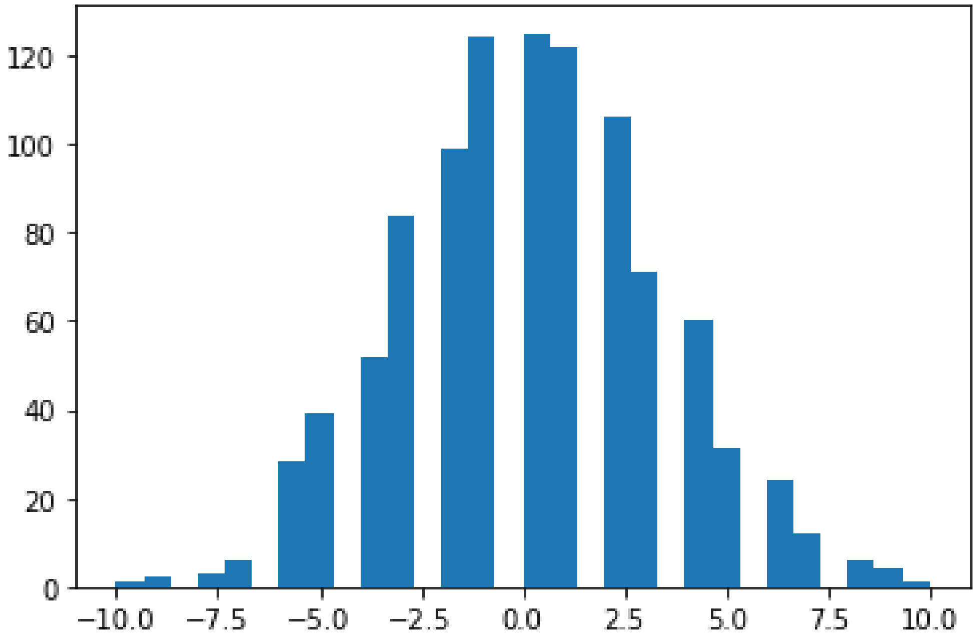
0.1.1 Required Packages:

```
[ ]: #This was a good one, task is to fill a given square with a bunch of circles,␣
      ↪whoose radii obey this normal distribution:
```

```
[98]: import matplotlib.pyplot as plt
      import math
      import scipy.stats as ss
      import numpy as np
```

```
[99]: x = np.arange(-15,15)
      xU, xL = x + 0.5, x - 0.5
      prob = ss.norm.cdf(xU, scale = 3) - ss.norm.cdf(xL, scale = 3)
      prob = prob / prob.sum() # normalize the probabilities so their sum is 1
      nums = np.random.choice(x, size = 1000, p = prob)
      plt.hist(nums, bins = len(x))
```

```
[99]: (array([  1.,   2.,   0.,   3.,   6.,   0.,  28.,  39.,   0.,  52.,  84.,
                0.,  99., 124.,   0., 125., 122.,   0., 106.,  71.,   0.,  60.,
               31.,   0.,  24.,  12.,   0.,   6.,   4.,   1.]),
       array([-10.        ,  -9.33333333,  -8.66666667,  -8.        ,
               -7.33333333,  -6.66666667,  -6.        ,  -5.33333333,
               -4.66666667,  -4.        ,  -3.33333333,  -2.66666667,
               -2.        ,  -1.33333333,  -0.66666667,   0.        ,
                0.66666667,   1.33333333,   2.        ,   2.66666667,
                3.33333333,   4.        ,   4.66666667,   5.33333333,
                6.        ,   6.66666667,   7.33333333,   8.        ,
                8.66666667,   9.33333333,  10.        ]),
       <a list of 30 Patch objects>)
```



```
[1]: #These are the radii of the circles^
     #So we are supposed to come up with an algorithm to compute some approximate minimal length square and illustrate filling that square with circles
     #It was a famous quest
```

```
[100]: #Choosing 10 values for radii of 10 circles in gaussian random distribution
```

```
[101]: x = np.arange(-15,15)
       xU, xL = x + 0.5, x - 0.5
       prob = ss.norm.cdf(xU, scale = 3) - ss.norm.cdf(xL, scale = 3)
       prob = prob / prob.sum()
       nums = np.random.choice(x, size = 10, p = prob)
```

```
[102]: nums
```

```
[102]: array([-4, -2,  3,  6,  0,  2,  3,  1, -1, -8])
```

```
[103]: l=[abs(i) for i in list(nums)]
```

```
[104]: l.sort()
       l=l[-1::-1]
```

```
[105]: l
```

```
[105]: [8, 6, 4, 3, 3, 2, 2, 1, 1, 0]
```

[106]:

```
def check_border(x,y,r,s):
    return x+r<=s and x-r>=0 and y+r<=s and y-r>=0
def check_touch(x1,y1,r1,x2,y2,r2):
    distSq = (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2);
    radSumSq = (r1 + r2) * (r1 + r2);
    if (distSq == radSumSq):
        return 1
    elif (distSq > radSumSq):
        return 1
    else:
        return 0
```

[107]:

```
def compute(l):
    sum=0
    for i in l:
        sum+=3*i*i
    return sum
```

[108]:

```
#Assuming a step length of 0.1 while traversing the coordinates to fill the␣
 ↪void:
#Note this can also be solved modern optimization-theory (difference of␣
 ↪convex-functions + Concave-convex-procedure)
#But since the question specified not to solve for minimal square cover and␣
 ↪requires illustration of any filling algorithm
#I've used a simple searching for voids and checking if they intersect the␣
 ↪border or touch previously
#placed circles after sorting
```

[109]:

```
init=[(l[0],l[0],l[0])]
pts=[(l[0],l[0])]
#rds=[l[0]]
side=math.ceil(math.sqrt(compute(l)))
print(side)
num=len(l)
l.remove(l[0])
print(l)
while(len(pts)!=num):
    z=[]
    k=0
    while(k<len(l)):
        f=0
        if(len(l)==0):break
        #Note:Change this to (1,side,1) if you want faster processing and place␣
 ↪the cirles' centers in integer coordinates only
        for i in list(np.arange(1,side,0.1)):#range(1,side,0.1):
            #Note:Change this to (1,side,1) if you want faster processing and␣
 ↪place the cirles' centers in integer coordinates only
```

```
            for j in list(np.arange(1,side,0.1)):
                #print((i,j) not in␣
↪pts,check_border(i,j,k,side),[check_touch(i,j,k,u[0],u[1],u[2]) for u in␣
↪init])

                if((i,j) not in pts and check_border(i,j,l[k],side) and (0 not␣
↪in [check_touch(i,j,l[k],u[0],u[1],u[2]) for u in init] )):
                    init.append((i,j,l[k]))
                    pts.append((i,j))
                    z.append(l[k])
                    l.remove(l[k])
                    if(len(l)==0):break
                    f=1
                    k=0
                if(len(l)==0):break
            if(len(l)==0):break
        if(f==0):side+=0.1
        if(len(l)==0):break
        k+=1
    #[l.remove(o) for o in z]
```

```
21
[6, 4, 3, 3, 2, 2, 1, 1, 0]
```

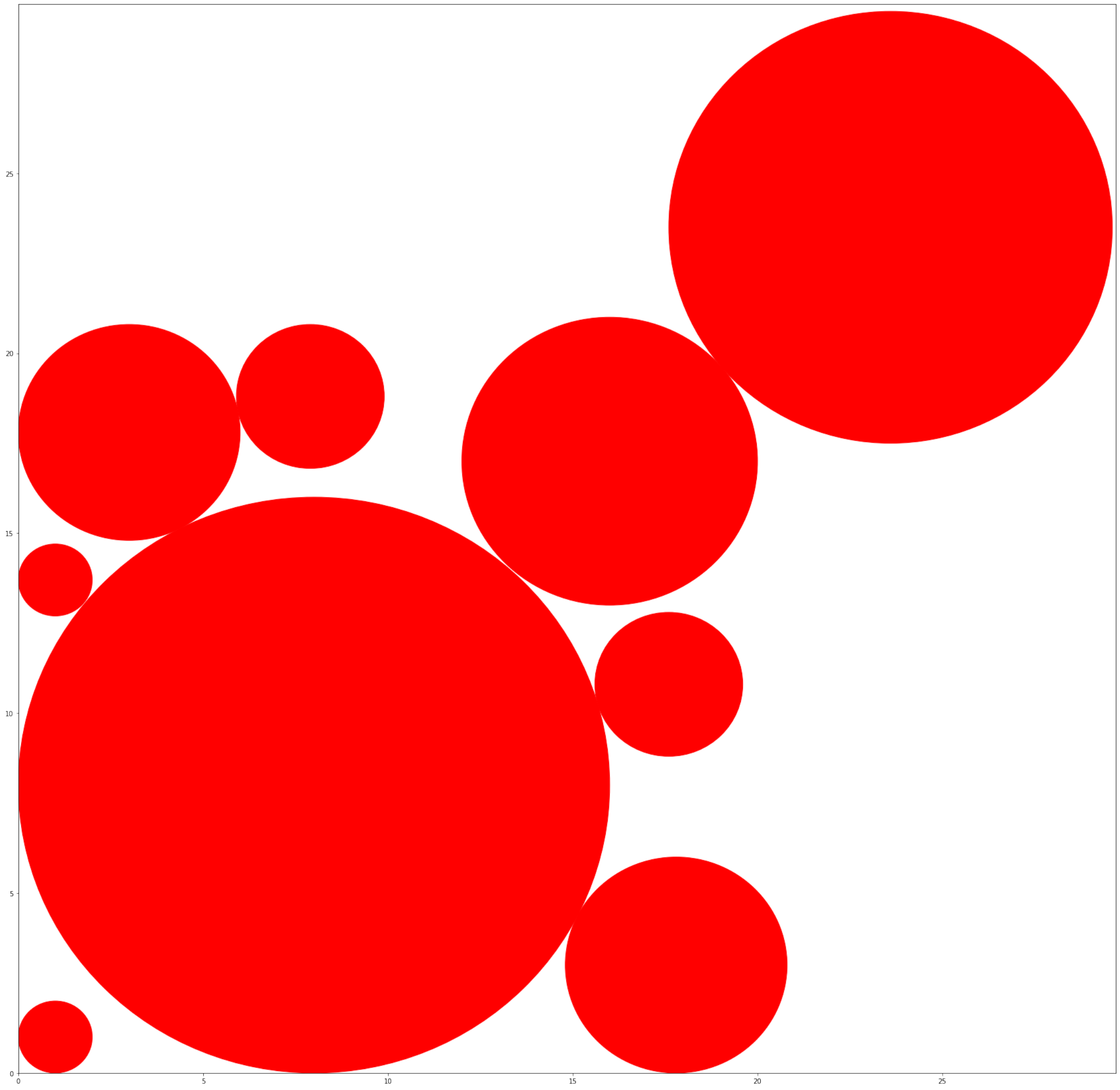
[110]:

```
print("Circles fit in square of side length:")
print(side)
```

```
Circles fit in square of side length:
29.700000000000124
```

[111]:

```
fig, ax = plt.subplots(figsize=(side, side))
plt.xlim([0, side])
plt.ylim([0, side])
for i in init:
    c1 = plt.Circle((i[0], i[1]), i[2], color='r')
    ax.add_patch(c1)
plt.show()
```



[112]:
```
print("Coordinates, radius of circles: (x,y,r):")
print(init)
```

```
Coordinates, radius of circles: (x,y,r):
[(8, 8, 8), (16.000000000000014, 17.000000000000014, 4), (3.0000000000000018,
17.800000000000015, 3), (17.800000000000015, 3.0000000000000018, 3),
(7.900000000000006, 18.800000000000015, 2), (17.600000000000016,
10.800000000000008, 2), (1.0, 1.0, 1), (1.0, 13.700000000000012, 1), (1.0,
2.000000000000001, 0), (23.60000000000002, 23.50000000000002, 6)]
```