In [2]:

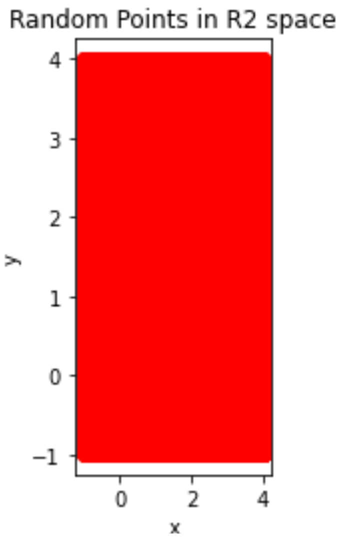
```
import numpy as np
import matplotlib.pyplot as plt
import math
from numpy import random
```

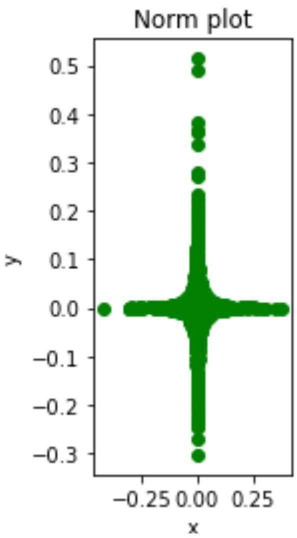
1.a.Write a function that accepts p and plots the ||.||p unit norm ball. Test with integer p ≥ 1 as well as 0 < p < 1. (3)

In [18]:

```
p="None"
print(p.isnumeric())
while p.isalpha()==True:
    p=input("Enter value of p as values in range 0-inf")
    if p=="inf":
        break
if p!="inf":
    p=float(p)
def compute_Norm(p):
    x=[]
    y=[]
    x1=[]
    y1=[]
    for i in range(1000000):
        x0=random.rand()*5-1
        y0=random.rand()*5-1
        x.append(x0)
        y.append(y0)
        if p== "inf":
            if max(abs(x0),(abs(y0)))<1:
                x1.append(x0)
                y1.append(y0)
        elif p>=1:
            if ((abs(x0)**p)+(abs(y0)**p))**(1/p)<1:
                x1.append(x0)
                y1.append(y0)
        elif p>=0:
            if ((abs(x0)**p)+(abs(y0)**p))<1:
                x1.append(x0)
                y1.append(y0)



    plt.subplot(1, 3, 1)
    plt.scatter(x,y,color='red')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title("Random Points in R2 space")

    plt.subplot(1, 3, 3)
    plt.scatter(x1,y1,color='green')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title("Norm plot")
    plt.show()

    return x1,y1,p
x1,y1,p=compute_Norm(p)
```

```
False
Enter value of p as values in range 0-inf0.2
```



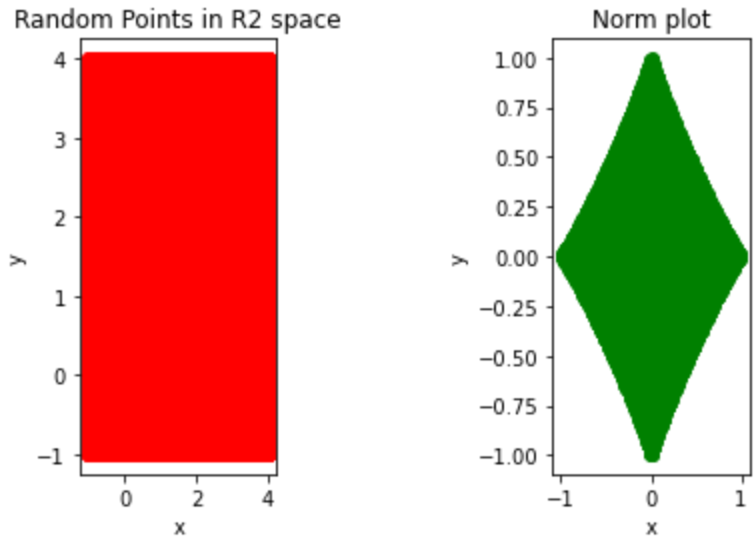


1.b.We claimed that for integer p ≥ 1 the unit norm ball is convex. Is this clear from the unit norm ball plots? What happens when 0 < p < 1? Print your observations. (2)

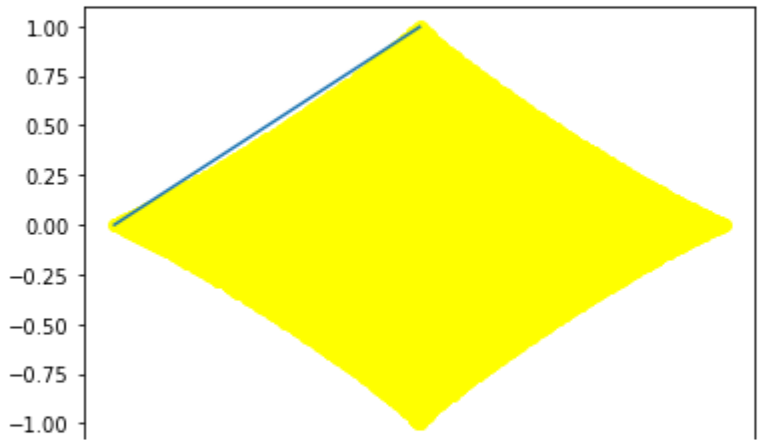
In [23]:

```
# def coord(x0,y0):
#     top=[]
#     left=[]
#     minimum=1000
#     maximum=0
#     for i in x0:
#         for j in y0:
#             if i<minimum:
#                 minimum=i
#                 left=[i,j]
#             if j>maximum:
#                 maximum=j
#                 top=[i,j]
#     return top,left


p=3.0
# print(float(p))
while float(p)>=1.0 or float(p)<=0.0:
#     print(p)
    p=float(input("Enter p value in range 0 and 1"))
# p=0.2
count=0
# p=p_list[0]
x1,y1,p=compute_Norm(p)
count+=1
#     convexity_check(x1,y1,p)
#     top,left=coord(x1,y1)

ind1=y1.index(max(y1))
ind2=x1.index(min(x1))
top=[x1[ind1],y1[ind1]]
left=[x1[ind2],y1[ind2]]
plt.figure()
#     plt.subplot(1,len(p_list),count)
plt.scatter(x1,y1,color='yellow')
plt.plot([top[0],left[0]],[top[1],left[1]])
# print(p,top,left)
```

```
Enter p value in range 0 and 10.9
```



Out[23]: `[<matplotlib.lines.Line2D at 0x7f27a7783f40>]`



As p increases from 0-1, the plots are appearing to emerge as empty plot to a twinkling star shape. Its size increases and attains a rhombus at p=1. As it further increases, shape slowly changes to circle and finally emerge as a rectangle. For shapes attained from p>=1, they have a minima where local minima and global minima are same. So in that range of p, norm is said to be convex. Also for range 0<p<1, the if we draw lie between any 2 points,say top most point and left most point, the line joining them will pass outside the set, making them concave. In range 0<p<1, the norm is not convex.

1. Completeness: Recall from class that a metric space (X, d) is said to be complete if all Cauchy se- quences in X converge to a point in X. Show with a numerical example that the space of continuous

functions defined on the closed interval [0, 1] and denoted C[0, 1] is incomplete with respect to the metric derived from the L1 norm (i.e., d(f , g) = || f − g||1 = R 1 0 | f(x) − g(x)|dx for any f , g ∈ C[0, 1]).

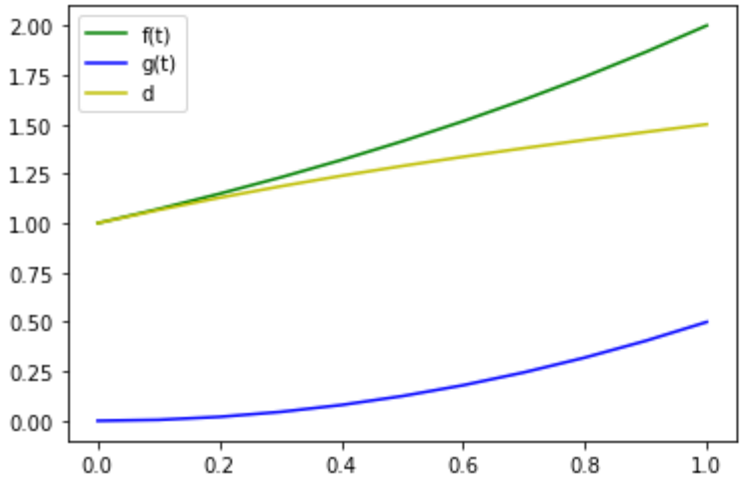
Code your example and demonstrate the result either using a plot or numerically. (5)

Come up with sequence functions that doenot converge to C[0,1] or have a discontinuity in it Show that the function is incomplete

In [11]:

```
time=np.arange(0.0, 1.1, 0.1)
print(time)
from math import log,sin,cos,exp,tan,factorial
f=[]
g=[]
for t in time:
#     if t!=0:
#         f.append(exp(t))
#         f.append(tan(t))#alternate f
        f.append(2**t)#alternate f


#     else:
#         f.append(0)
# g=[cos(t) for t in time]
g=[0.5*(t**2) for t in time]
# g=[sin(t) for t in time]#alternate g
p=1
d=[]
for i in range(len(time)):
    d.append(((abs(f[i]-g[i])**p))**(1/p))


plt.plot(time,f,'g',label ='f(t)')
plt.plot(time,g,'b',label ='g(t)')
plt.plot(time,d,'y',label ='d')
plt.legend()
plt.show()
for  i in d:
    if i>1:
        print("Metric doesn't converge in the duration [0,1] as metric valu
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```




```
Metric doesn't converge in the duration [0,1] as metric value grows to  1.0
667734625362932  > epsilon(=1)
Metric doesn't converge in the duration [0,1] as metric value grows to  1.1
28698354997035  > epsilon(=1)
Metric doesn't converge in the duration [0,1] as metric value grows to  1.1
861444133449164  > epsilon(=1)
Metric doesn't converge in the duration [0,1] as metric value grows to  1.2
395079107728941  > epsilon(=1)
Metric doesn't converge in the duration [0,1] as metric value grows to  1.2
892135623730951  > epsilon(=1)
Metric doesn't converge in the duration [0,1] as metric value grows to  1.3
357165665103983  > epsilon(=1)
Metric doesn't converge in the duration [0,1] as metric value grows to  1.3
```

```
79504792712471  > epsilon(=1)
Metric doesn't converge in the duration [0,1] as metric value grows to  1.4
21011265922482  > epsilon(=1)
Metric doesn't converge in the duration [0,1] as metric value grows to  1.4
610659830736148  > epsilon(=1)
Metric doesn't converge in the duration [0,1] as metric value grows to  1.5
> epsilon(=1)
```

1. Entropy of a discrete RV: Recall the definition of entropy of a discrete RV X from class, H(X) = − ∑ x∈X p(x) log p(x), where p(x) is the probability mass function (PMF) of X, and X is the set of possible values that the random variable X can take. (a) Write a function that accepts a PMF as input and outputs the entropy in bits. Do check for the condition of a value being assigned zero probability. (3)

In [12]:

```
from math import log2
def compute_entropy(pmf):
    entropy=[]
    for p in pmf:
        if p!=0:
            entropy .append( - p * np.log2(p))
        else:
            entropy .append(0.0)

    return entropy
pmf=[]
while np.sum(pmf)<1:
    if len(pmf)<1:
        number=float(input("Enter the first element of pmf"))
        pmf.append(number)

    else:
        pmf.append(float(input("Enter the next element of pmf")))

print(pmf)
# compute_entropy(np.array([0.1, 0.5, 0.1, 0.3,0.0]))
entropy=compute_entropy(pmf)
print("For PMF entered for the random variable, entropy = ",np.sum(entropy
```

```
Enter the first element of pmf0.3
Enter the next element of pmf0.2
Enter the next element of pmf0.1
Enter the next element of pmf0.4
[0.3, 0.2, 0.1, 0.4]
For PMF entered for the random variable, entropy =  1.8464393446710154 bit
```

In [14]:

```
plt.scatter(pmf,entropy)
print("Maximum value at which we get maximum at when PMF is ",pmf[entropy.i
```

```
Maximum value at which we get maximum at when PMF is  0.4
```

3(b) Now use the above function to plot the entropy of X ∼ Bern(p) as a function of p. Where does this plot attain its maximum? (2)

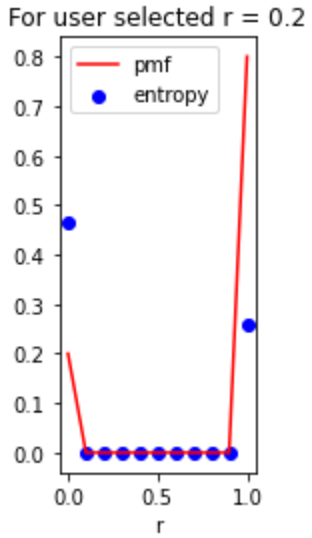
expression for entropy in function bernouli RV, bias in coin toss can be related, as you approach fairness/ move away what is the entropy variation save values of p we take in a variable detect where we get the maximum

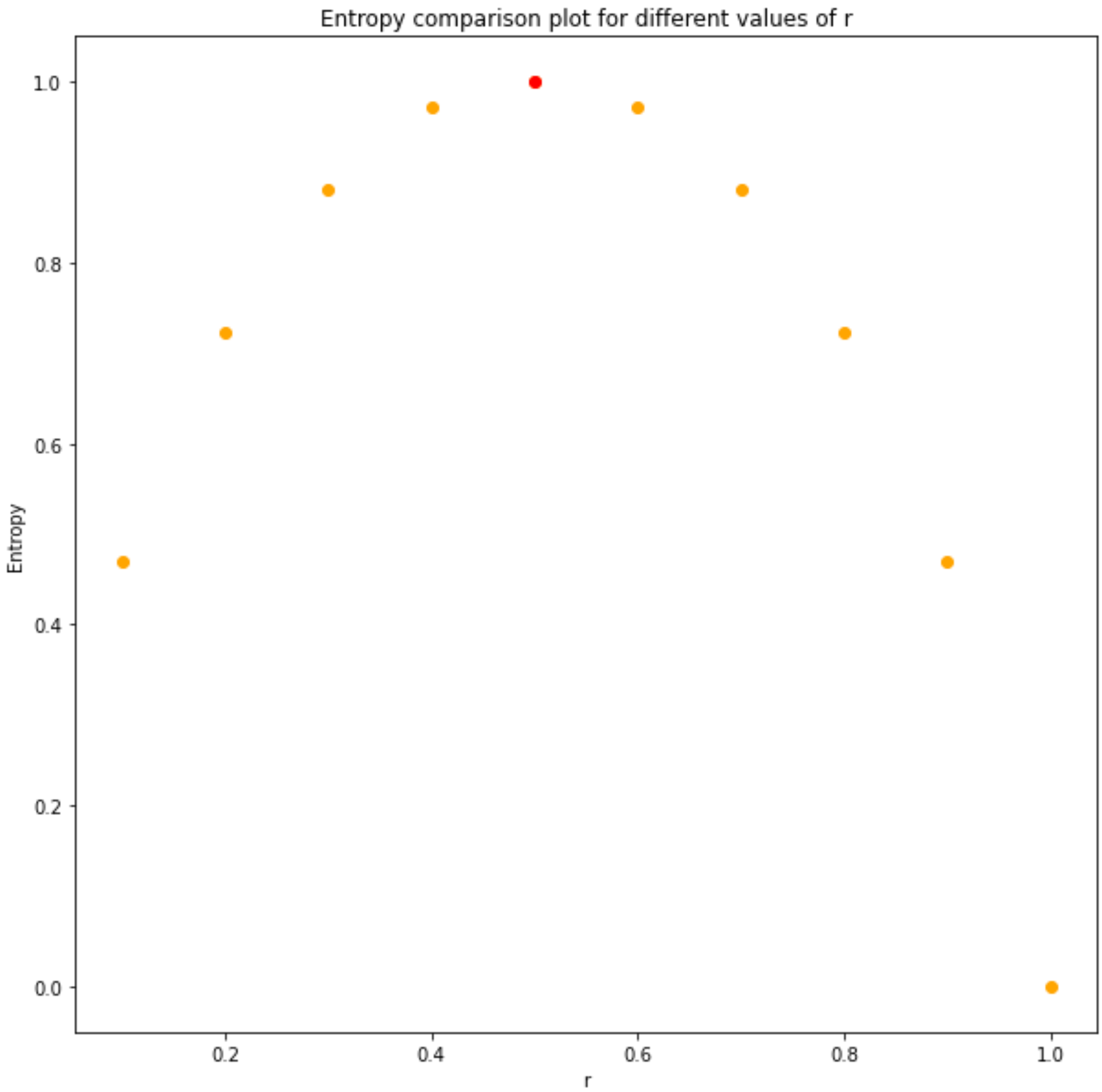
In [42]:

```
def compute_bern_pmf(r):
    time=np.arange(0.0, 1.1, 0.1)
    res=[]
    for i in time:
        if i==0:
            res.append(r)
        elif i==1:
            res.append(1-r)
        else:
            res.append(0)
    return res
time=np.array([0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0])
r=float(input("enter r value"))
bern_pmf=compute_bern_pmf(r)
entropy=compute_entropy(bern_pmf)
# print(entropy)
# print(bern_pmf)
print("For BERN PMF entered for the random variable, entropy = ",np.sum(en
plt.subplot(1,3,1)
plt.plot(time,bern_pmf,'r',label='pmf')
plt.scatter(time,entropy,color='blue',label='entropy')
plt.legend()
plt.xlabel("r")
plt.title("For user selected r = "+str(r))
entropy_compare=[]
f=plt.figure()
f.set_figwidth(10)
f.set_figheight(10)
plt.title("Entropy comparison plot for different values of r")
plt.xlabel("r")
plt.ylabel("Entropy")
maximum=0
indices=[]

count=0
for r in time:
    bern_pmf=compute_bern_pmf(r)
    entropy=compute_entropy(bern_pmf)
    entropy_compare.append([r,np.sum(entropy)])
    plt.scatter(r,np.sum(entropy),color='orange')
#     print(r,np.sum(entropy))

# #     plt.plot(r,np.sum(entropy_compare[count]),'o',label=str(r))
# #     plt.legend()
#     if maximum<=max((compute_bern_pmf(r))) and r not in indices:
#         indices.append(r)
#         maximum=np.sum((compute_bern_pmf(r)))
#     count+=1
print(entropy_compare)
maximum=max(map(lambda x: x[1], entropy_compare))
print(maximum)
for i in entropy_compare:
    if i[1]==maximum:
        plt.scatter(i[0],i[1],color='red')
        indices.append(i[0])


print("Maximum value of entropy",maximum,"we get maximum at when r is ",in
```

```
enter r value0.2
For BERN PMF entered for the random variable, entropy =  0.7219280948873623
```

```
bit
[[0.0, 0.0], [0.1, 0.4689955935892812], [0.2, 0.7219280948873623], [0.3, 0.
8812908992306927], [0.4, 0.9709505944546686], [0.5, 1.0], [0.6, 0.970950594
4546686], [0.7, 0.8812908992306927], [0.8, 0.7219280948873623], [0.9, 0.468
99559358928117], [1.0, 0.0]]
1.0
```





Entropy will be maximum at at p=1 when r<=0.5 and vice versa.

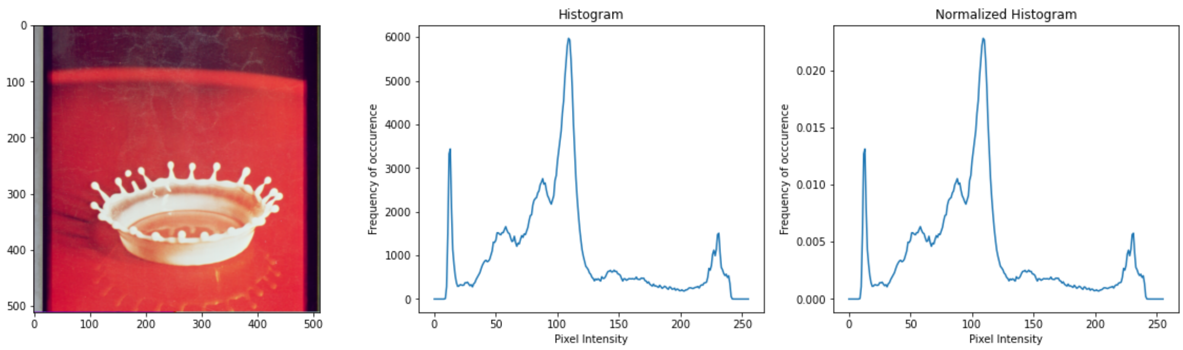
1. Image entropy: Download a gray scale image from the link provided in the instructions. By gray scale is meant that the image has one intensity channel. Further, the pixel intensities are in the range [0, 255]. (a) Write a function that accepts an image as input and returns its normalized histogram. Note that the normalized histogram is found by dividing the original histogram by the total number of pixels in the image. (3)

In [45]:

```
from PIL import Image, ImageOps
import matplotlib.image as mpimg
def compute_histogram(path):
    im = Image.open(path)
    im= ImageOps.grayscale(im)
    pixel_count=im.width*im.height
    intensity=[i for i in range(256)]
    histogram=np.zeros(256)
    for i in range(im.width):
        for j in range(im.height):
            histogram[im.getpixel((i,j))]+=1
    normalized_histogram=histogram/pixel_count
    return histogram, normalized_histogram,intensity
path='misc/4.2.01.tiff'
img = mpimg.imread(path)
histogram, normalized_histogram,intensity=compute_histogram(path)
f=plt.figure()
f.set_figwidth(20)
f.set_figheight(5)
plt.subplot(1,3,1)
plt.imshow(img)
plt.subplot(1,3,2)
plt.plot(intensity,histogram)
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency of occcurence')
plt.title("Histogram")

plt.subplot(1,3,3)
plt.plot(intensity,normalized_histogram)
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency of occcurence')
plt.title("Normalized Histogram")
```

Out[45]:

```
Text(0.5, 1.0, 'Normalized Histogram')
```



4(b) Use your entropy function from the previous problem to find the image entropy. Experiment with different gray scale images from the aforementioned link and note your observations. (2)

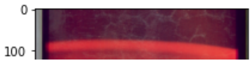
In [46]:

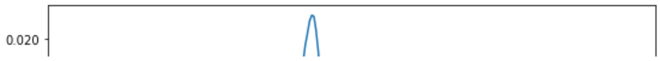
```
import matplotlib.image as mpimg
test_images=['misc/4.2.01.tiff','misc/7.1.02.tiff','misc/4.1.08.tiff','mis
count=1
f=plt.figure()
f.set_figwidth(20)
f.set_figheight(15)
for t in test_images:
#     print(t)
    img = mpimg.imread(t)
    histogram, normalized_histogram,intensity=compute_histogram(t)
    plt.subplot(len(test_images),2,count)
    plt.imshow(img)
#     plt.plot(intensity,histogram)
    plt.subplot(len(test_images),2,count+1)
    plt.plot(intensity,normalized_histogram)
    plt.xlabel(' Intensity')
    plt.ylabel('Frequency ')
    entropy=compute_entropy(normalized_histogram)
    print("For PMF entered of the image in gray scale, entropy = ",np.sum(
    count=count+2
plt.show
```

```
For PMF entered of the image in gray scale, entropy =  7.253417326381689 bi
t
For PMF entered of the image in gray scale, entropy =  4.004499444666612 bi
t
For PMF entered of the image in gray scale, entropy =  6.242832029429284 bi
t
For PMF entered of the image in gray scale, entropy =  6.702450619383894 bi
t
```

Out[46]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



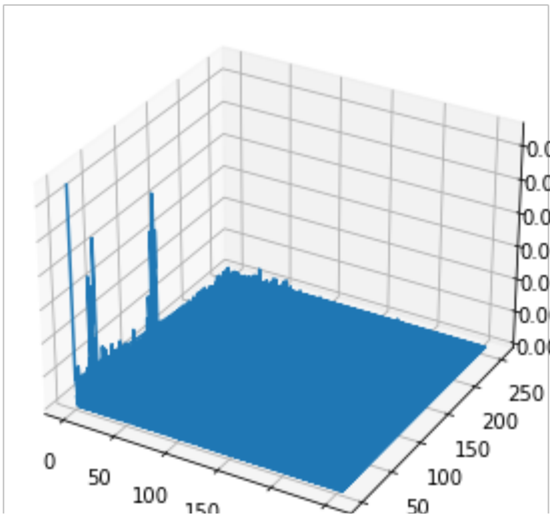


stereo pair are similar but with small displacement with which we can compute depth a. take image pair Y= left X=right x=0 and y takes 128 occur like wise for all combinations joint histogram= how many times a pixel intensity pair occurs 3d plot is expected

In [48]:

```
def joint_histogram(path1,path2):
    im1 = Image.open(path1)
    im2 = Image.open(path2)
    im1= ImageOps.grayscale(im1)
    im2= ImageOps.grayscale(im2)
    pixel_count1=im1.width*im1.height
    pixel_count2=im2.width*im2.height
    print(im1.width,im1.height)
    print(im2.width,im2.height)
    print("total_pixel_count=",pixel_count1+pixel_count2)
    intensity=[i for i in range(256)]
#     histogram=[]
#     print(im1.shape,im2.shape)
    histogram=np.zeros((256, 256))
    pixel1=[]
    pixel2=[]
    for i in range(im1.width):
        for j in range(im1.height):
#             print(im1.getpixel((i,j)),im2.getpixel((i,j)))
            histogram[im1.getpixel((i,j)),im2.getpixel((i,j))]+=1


#             for l in range(im2.width):
#                 for m in range(im2.height):
#                     histogram[im1.getpixel((i,j)),im2.getpixel((l,m))]+=
    normalized_histogram=histogram/(pixel_count1)#*pixel_count2)
    x=[]
    y=[]
    z=[]
    for i in range (256):
        for j in range(256):
            x.append(i)
            y.append(j)
            z.append(normalized_histogram[i,j])
    x=np.array(x)
    y=np.array(y)
    z=np.array(z)
#     print(np.sum(z))
    f=plt.figure()
    f.set_figwidth(20)
    f.set_figheight(5)

#     z=z/(pixel_count1+pixel_count2)
    ax = plt.axes(projection ='3d')
    ax.plot(x, y,z )
#     plt.plot(histogram)
    plt.show()
    return x,y,z,normalized_histogram,histogram
x,y,z,normalized_histogram,histogram=joint_histogram("left.png","right.png
```

```
105 70
105 70
total_pixel_count= 14700
```



(b) Write a function that accepts the joint PMF of a pair of random variables as input and outputs the joint entropy. (1)

In [49]:

```
def joint_entropy(p_xy):
    joint_entropy=[]
    for p in p_xy:
        if p!=0:
            joint_entropy.append(-p * np.log2(p))
    return joint_entropy

# <!-- entropy=joint_entropy(temp) -->
```

Test your joint entropy function using the normalized joint histogram computed in Problem 5 (a). (1)

In [50]:

```
print("Joint entropy of stereo images : ",np.sum(joint_entropy(z)),"bits")
```

```
Joint entropy of stereo images :  12.18321427295902 bits
```

Conditional PMF and conditional entropy: Continue to work with the stereo image pair. (a) Write a function that accepts as input the joint PMF of a pair of random variables, the index of the conditioning random variable, and the value of the conditioning random variable. The function must output the appropriate conditional PMF. (3)

In [51]:

```
def conditional_(temp,index,thresh):
#     for
#     pB = 0.3
# pAB=0.4

# pAifB = pAB / pB
# print(pAifB)
    x=0
    y=0
    for i in range(temp.shape[0]):
        for j in range(temp.shape[1]):
            if i==index and j==thresh:
                x+=(temp[index,j])
            if j==thresh:
                y+=temp[i,thresh]
#     x1=np.sum(x)
#     y1=np.sum(y)
#     x=np.sum(temp[index,thresh])
#     y=np.sum(temp[:,thresh])
#     print(x,y)
    if y!=0:
        return x/y
    else:
        return 0
#     for i in range()
#     c_pmf=[]
#     intensities=[]
#     for i in range(len(temp)):
#         intensities.append(i)
#         if temp1==0:#x
#             c_pmf.append(temp[thresh][i])
#             den+=temp[thresh][i]
#         if temp1==1:#y
#             c_pmf.append(temp[i][thresh])
#             den+=temp[thresh][i]
# #     p_cond=temp*v_pmf
#     plt.plot(intensities,c_pmf)
#     return x/y
#             c_pmf=temp[i]*
#         else:
#             c_pmf.append(0)
# #left.png
# histogramX, norm_histX,intensity=compute_histogram("left.png")
# p_x_100=
cond_pmf=conditional_(normalized_histogram,120,0)
print(cond_pmf)
```

```
0.008064516129032263
```

(b) Write a function that accepts as input the joint PMF as well as a conditional PMF, and outputs the conditional entropy. (1)

In [52]:

```
def conditional_entropy(joint,thresh):
    result=0
    for i in range (joint.shape[0]):
        for j in range(joint.shape[1]):
            cond=conditional_(joint,i,thresh)
            if cond!=0:
#               if conditional

#               print((-1*joint[i,j]*log2(conditional_(joint,j,i))))
                result+=(-1*joint[i,j]*log2(cond))

    return result
```

(c) Test your conditional entropy function using the normalized joint histogram computed in Prob- lem 4 (a), the conditional PMF computed in Problem 5 (a) for your choice of the conditioning

random variable and its value. (1)

In [53]:

```
print("Entropy: ",conditional_entropy(normalized_histogram,120)," bits")
```

```
Entropy:  1.7433851486519831  bits
```

1. KL divergence: We showed in class that for PMFs p and q defined on X , D(p||q) ≥ 0, D(q||p) 6= D(p||q). This problem explores these properties experimentally. (a) Write a function that accepts as input two PMFs p and q as input, and outputs D(p||q). (1)

In [54]:

```
def compute_kl_divergece(p,q):
    divergence=[]
    for i in range(len(p)):
        if q[i]!=0 or p[i]!=0:

            divergence.append(p[i]*np.log2(p[i]/q[i]))
        else:
            divergence.append(0)
#     print(divergence.shape)
    return divergence
def pmf(x,p):
    f = p**x*(1-p)**(1-x)
    return f
```

(b) As discussed in class, let p ∼ Bern(r) and q ∼ Bern(s). For a fixed value of r, vary s and do the following: (4) i. Plot D(p||q), D(q||p). ii. Verify that D(p||q), D(q||p) are indeed non-negative. iii. Verify that D(p||q) 6= D(q||p) and are both equal to zero only when r = s. iv. Finally, find D(p||q) and D(q||p) where p and q are the normalized histograms of left.png and right.png respectively. Do you think D(p||q) is a good metric for image similarity? Print your response.

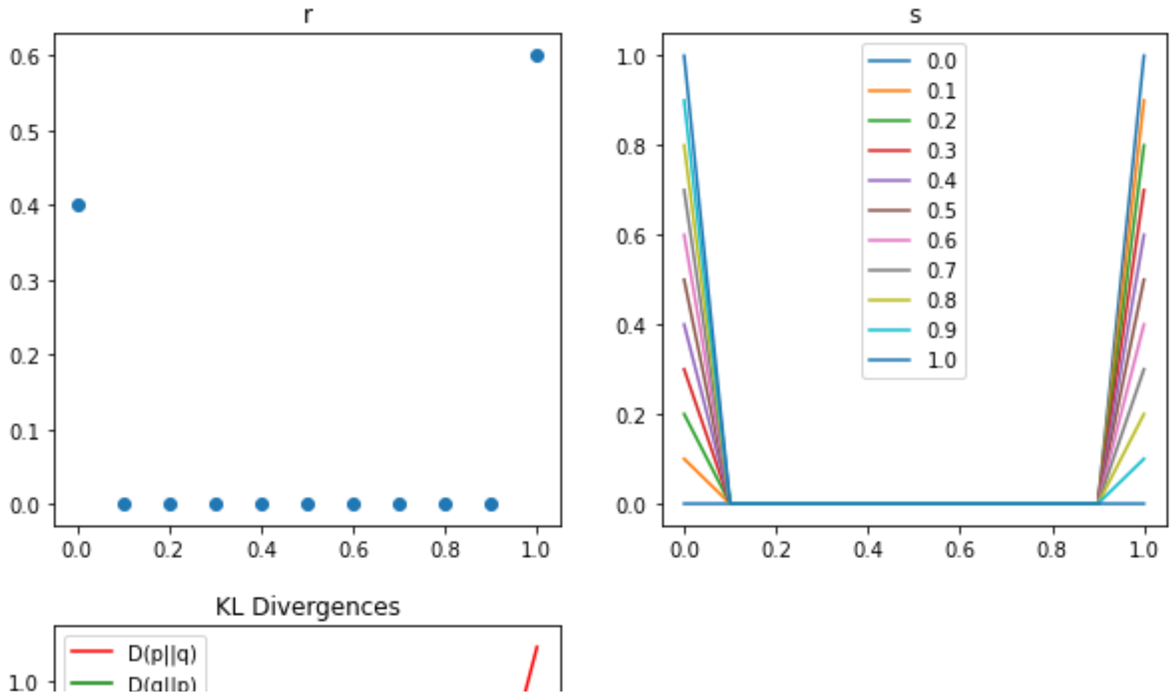
In [76]:

```
r=0.4
s_val=np.arange(0.0, 1.1, 0.1)
p=compute_bern_pmf(r)
D_p_q=[]
D_q_p=[]
q_val=[]
for s in s_val:
    q=compute_bern_pmf(s)
    q_val.append(q)
    D_p_q.append(np.sum(compute_kl_divergece(p,q)))

    D_q_p.append(np.sum(compute_kl_divergece(q,p)))
# print(s_val,p,q)
f=plt.figure()
f.set_figwidth(10)
f.set_figheight(10)

plt.subplot(2,2,1)
plt.scatter(s_val,p)
plt.title("r")
plt.subplot(2,2,2)
count=0
for q in q_val:
#     print(s_val[count])
    plt.plot(s_val,q,label=round(s_val[count],2))
    count+=1
plt.title("s")
plt.legend()

plt.subplot(2,2,3)
plt.plot(s_val,D_p_q,color='red',label="D(p||q)")
plt.title("KL Divergences")
# plt.subplot(2,2,4)
plt.plot(s_val,D_q_p,color='green',label="D(q||p)")
plt.legend()
```

```
/tmp/ipykernel_4436/3229533446.py:6: RuntimeWarning: divide by zero encount
ered in double_scalars
  divergence.append(p[i]*np.log2(p[i]/q[i]))
/tmp/ipykernel_4436/3229533446.py:6: RuntimeWarning: divide by zero encount
ered in log2
  divergence.append(p[i]*np.log2(p[i]/q[i]))
/tmp/ipykernel_4436/3229533446.py:6: RuntimeWarning: invalid value encounte
red in double_scalars
  divergence.append(p[i]*np.log2(p[i]/q[i]))
```

Out[76]:

```
<matplotlib.legend.Legend at 0x7f63d133b670>
```



In [56]:

```
def non_negative(val):
    res=True
    for i in val:
        if i<0:
            res=False
            break
    return res
print("Range of values of D(p||q): ",D_p_q)
print(non_negative(D_p_q))
print("Range of values of D(q||p): ",D_q_p)
print(non_negative(D_q_p))
print("Values are greater than zero=> Therefore the obtained values are ve
```

```
Range of values of D(p||q):  [inf, 0.06405999884615018, 0.0, 0.037123562209
68563, 0.1320299994230752, 0.27807190511263774, 0.48300749985576896, 0.7705
590150115547, 1.2000000000000006, 1.966014999711538, inf]
True
Range of values of D(q||p):  [nan, 0.052932501298081136, 0.0, 0.04063719565
666968, 0.15097750043269362, 0.3219280948873624, 0.5509775004326939, 0.8406
371956566698, 1.2000000000000002, 1.652932501298081, nan]
True
Values are greater than zero=> Therefore the obtained values are verified a
s non negative
```

In [77]:

```
detected=False
s_index=0
for i in D_p_q:

    for j in D_q_p:
        if i==j:
            print(D_p_q.index(i),D_q_p.index(j))
            s_index=D_p_q.index(i)
            detected=True
if detected==True:
    print("Value is same for index ", s_index," of D(p||q) and D(q||p)")
    print("When matching divergence term becomes: ",D_q_p[s_index])
    print("s value: ",s_val[s_index])
    print("r value",r)
```

```
4 4
Value is same for index  4  of D(p||q) and D(q||p)
When matching divergence term becomes:  0.0
s value:  0.4
r value 0.4
```

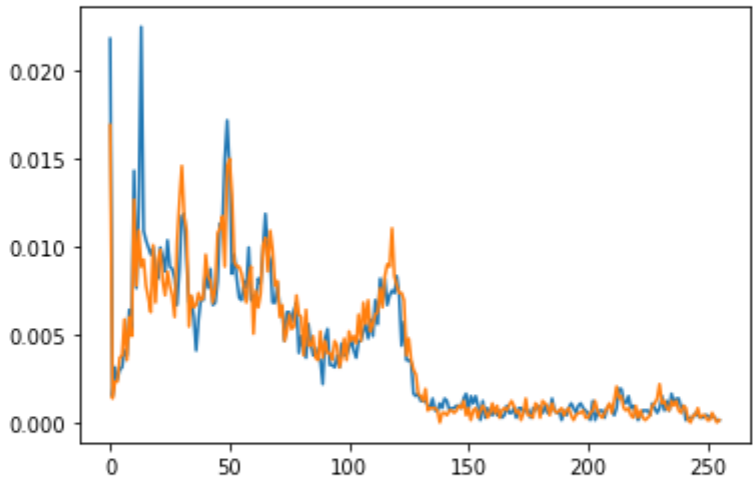
In [80]:

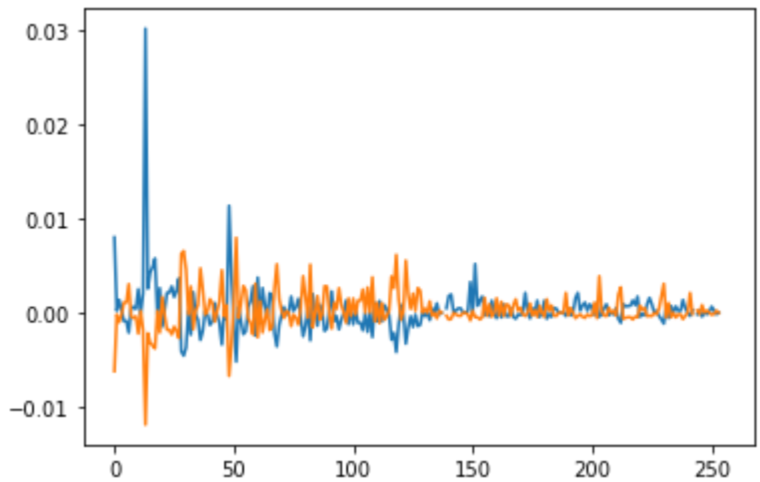
```
_, p,_=compute_histogram("left.png")
_, q,_=compute_histogram("right.png")
D_p_q=compute_kl_divergece(p,q)
D_p_p=compute_kl_divergece(p,p)
D_q_p=compute_kl_divergece(q,p)
plt.plot(p)
plt.plot(q)
plt.figure()
plt.plot(D_p_q)
plt.plot(D_q_p)
plt.figure()
plt.plot(D_p_p)
```

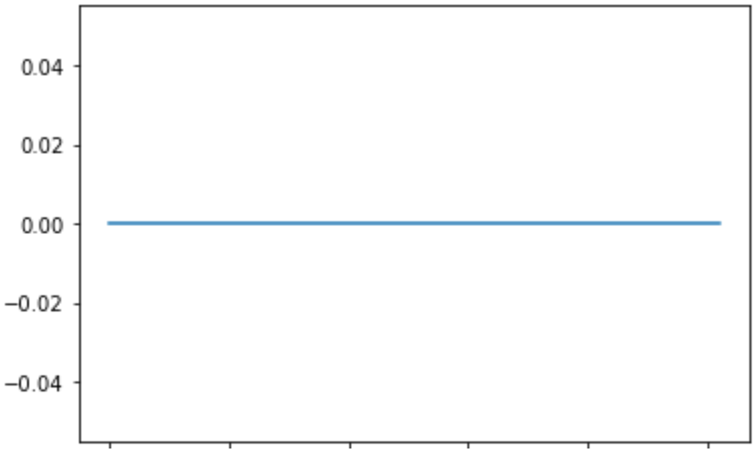
```
/tmp/ipykernel_4436/3229533446.py:6: RuntimeWarning: divide by zero encount
ered in double_scalars
  divergence.append(p[i]*np.log2(p[i]/q[i]))
/tmp/ipykernel_4436/3229533446.py:6: RuntimeWarning: divide by zero encount
ered in log2
  divergence.append(p[i]*np.log2(p[i]/q[i]))
/tmp/ipykernel_4436/3229533446.py:6: RuntimeWarning: invalid value encounte
red in double_scalars
  divergence.append(p[i]*np.log2(p[i]/q[i]))
```

Out[80]:

```
[<matplotlib.lines.Line2D at 0x7f63d106ae50>]
```







Yes Points of coincidence of D(p||q) and D(q||p) with zero means p=q at that point, then this matching points can help us in determining similarity index.

In [ ]:

In [88]:

```
count=0
sim=0
for i in range(len(D_p_q)):
    count+=1
    if D_p_q[i]==D_p_p[i]:
            sim+=1
print("Percentage similarity measure : ",(sim/count)*100)
```

```
Percentage similarity measure :  10.15625
```

In [ ]: