toolslab5

last edited Mar 17, 2016, 4:35:32 AM by admin

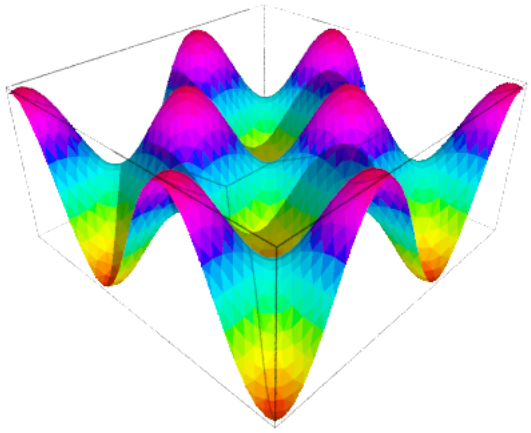
Save | Save & quit | Discard & quit

File... | Action... | Data... | sage | Typeset

Print | Worksheet | Edit | Text | Revisions | Share | Publish

```
import numpy as np
#var('x,y')
def P(x,y):
    return sin(x)*sin(y)
plot3d(P,(-5,5),(-5,5),1)
```

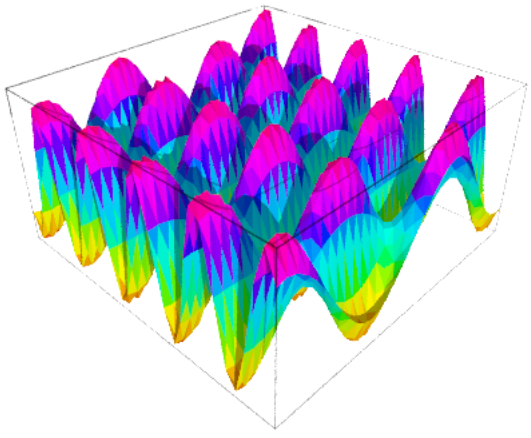
Sleeping... Make Interactive



```
def P2(a,b):
    def func(x,y):
        return sin(a*x)*sin(b*y)
    return func(x,y)
```

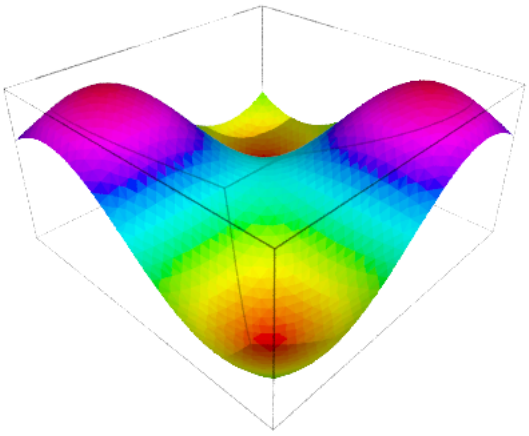
```
n=P2(3,1)
plot3d(n,(-5,5),(-5,5),1.25)
```

Sleeping... Make Interactive



```
m=P2(.5,.5)
plot3d(m,(-5,5),(-5,5),1.25)
```

Sleeping... Make Interactive



```
print integrate(sin(x))
print integrate(sin(x),(0,pi))
print integrate(sin(sin(x)))
```

```
-cos(x)
2
integrate(sin(sin(x)), x)
```

```
data=RealDistribution('gaussian',3) #creates a continuos Gaussian distribution with sd 3
my_data1=[data.get_random_element() for _ in range(10)]
print mean(my_data1)
print std(my_data1)
#lets see if we can get more precise
my_data2=[data.get_random_element() for _ in range(1000000)]
print mean(my_data2)
print std(my_data2)
#when I ran my trial the 10 sample set had a mean of -2.082 and a standard deviation of 2.9943
#It should have had a standard deviation of 3, which honestly is pretty close, but the mean should be 0
#when I ran this with a million samples the mean was -0.0006 which is much more accurate, although the standard
```

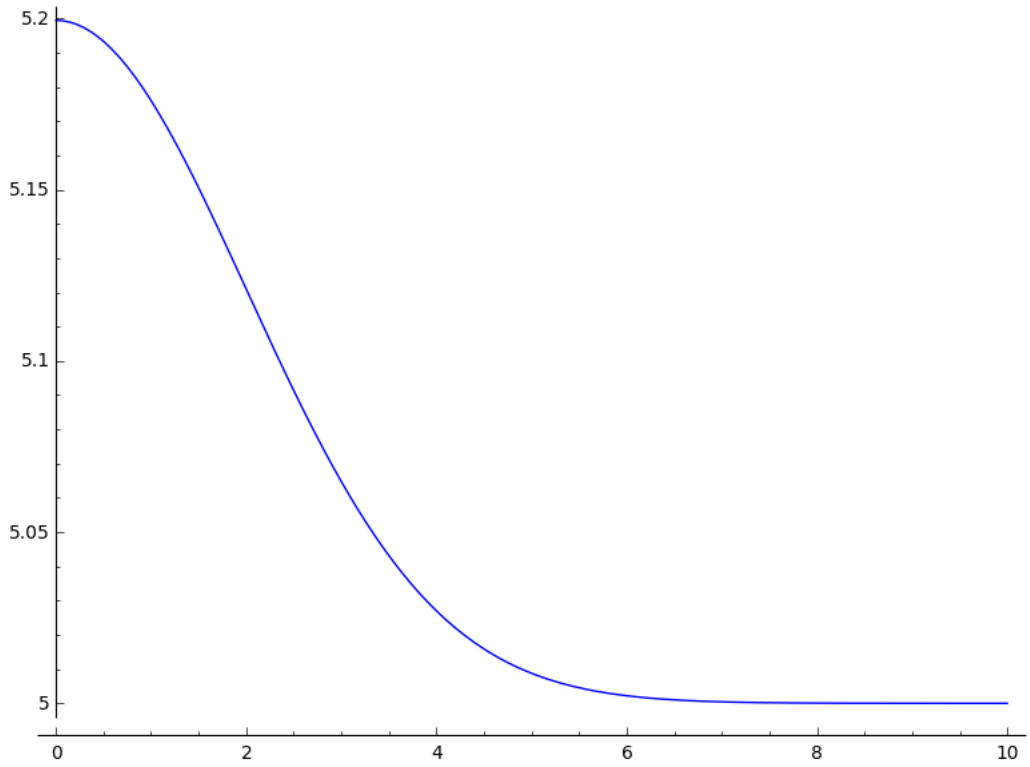
```
-2.08202061151
2.99430899344
-0.000614993772723
3.00060451959
```

```
dist = RealDistribution('gaussian',1) #this creates a normal distribution with mean 0 and standard deviation 1
print dist.distribution_function(1)
print dist.cum_distribution_function(1)-dist.cum_distribution_function(0)
```

```
0.241970724519
0.341344746069
```

```
plotdata = RealDistribution('gaussian',2)
def pdf(k):
    return plotdata.distribution_function(k)+5
```

```
plot(pdf,(0,10))
```




```
poss=[1/6,1/6,1/6,1/6,1/6,1/6]
bidist=GeneralDiscreteDistribution(poss)
bidist.get_random_element()
```

```
5
```

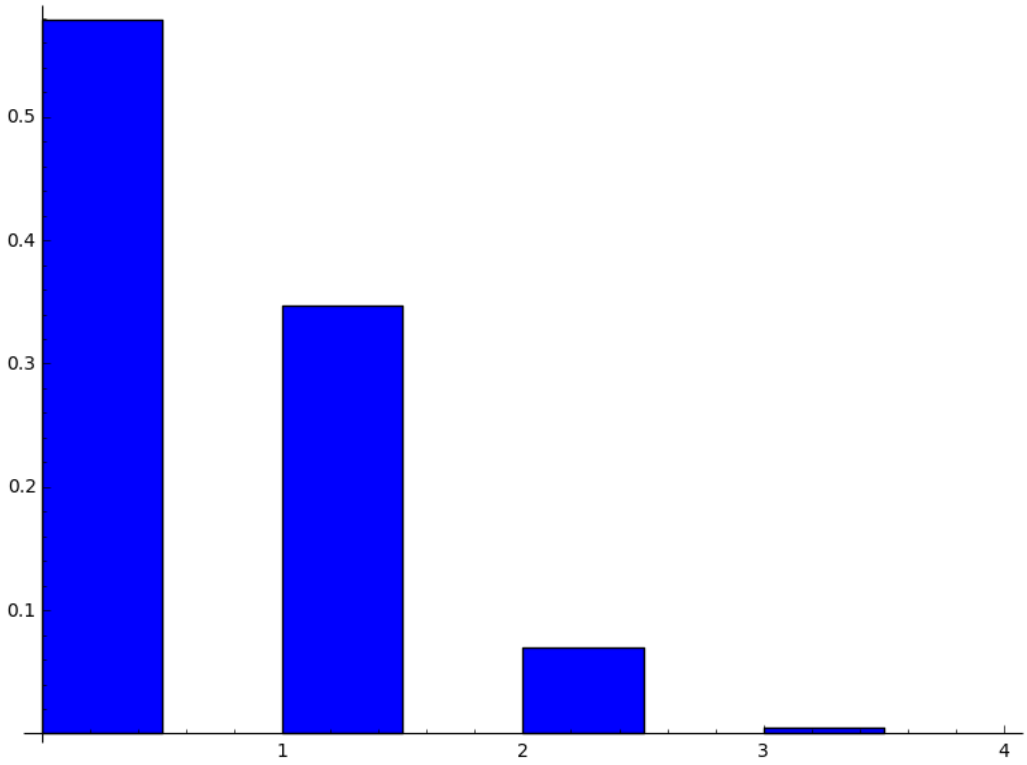
```
def bicounts(n):
    sum=0
    for i in range(n):
        if (bidist.get_random_element()==1):
            sum+=1
    return sum
```

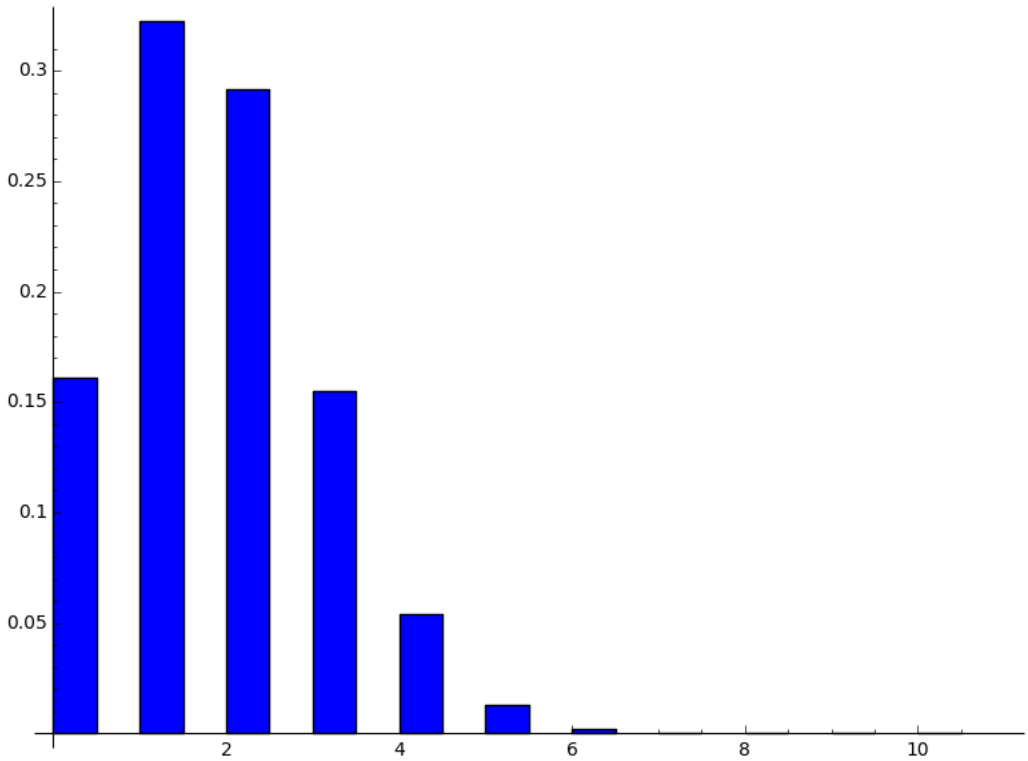
```
print bicounts(10)
```

```
0
```

```
def distFromBinomial(n):
    counts=np.zeros(n+1)
    for i in range(1000000):
        counts[bicounts(n)]+=1
    counts=counts/1000000
    return(counts)
```

```
bar_chart(distFromBinomial(3))
```

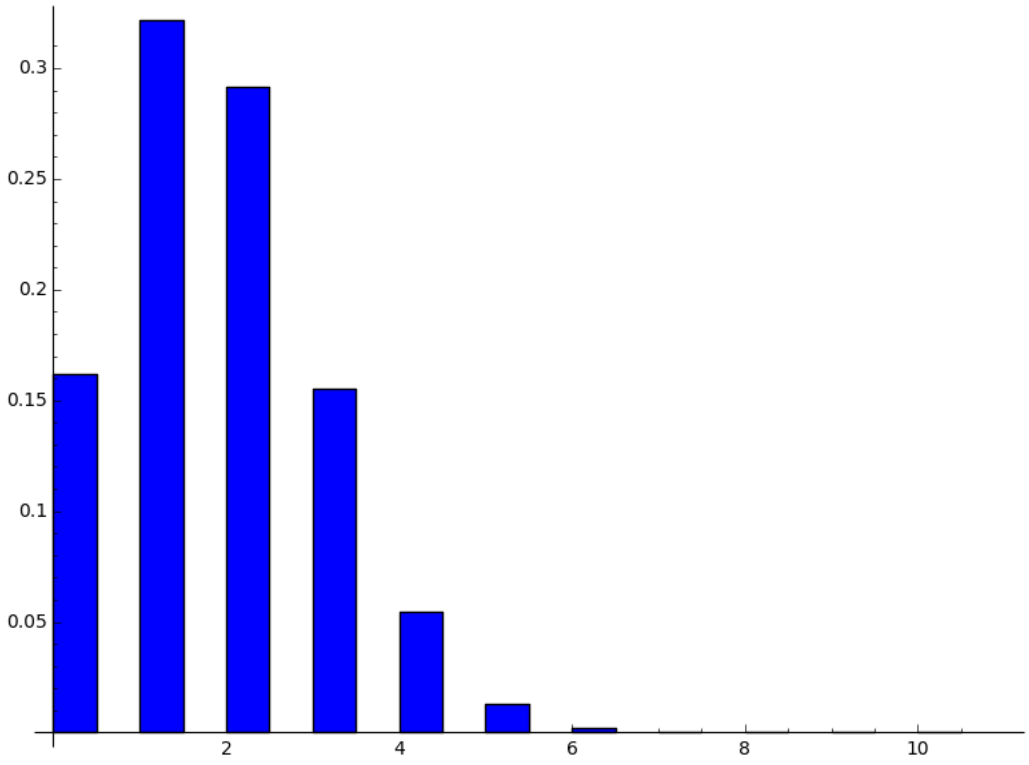

0.5
0.4
0.3
0.2
0.1
1
2
3
4


0.3
0.25
0.2
0.15
0.1
0.05
2
4
6
8
10

```
#Hey look the mode isn't 0 anymore
```

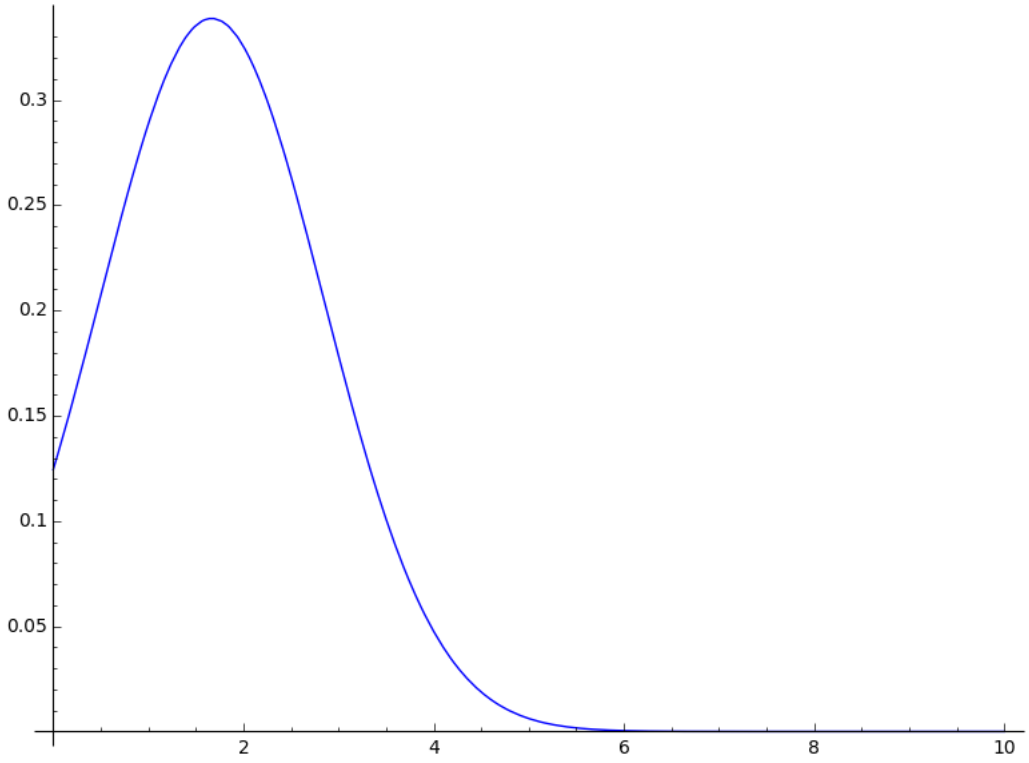
```
def gauss(m,s):
   normalization_factor=1/(s*sqrt(2*pi))
   def func(x):
       return normalization_factor*e^((-(x-m)**2)/(2*s**2))
   return func(x)
```

```
datas=distFromBinomial(10)
bar_chart(datas)
```




```
dist=GeneralDiscreteDistribution(datas)
list = [dist.get_random_element() for i in range(1000000)]
m=mean(list)
s=std(list)
F=gauss(m,s)
print F(3)
plot(F,(0,10)).show()
#Compare that Gaussian to the bar plot
```

```
83333250/346649595319*sqrt(346649595319/111111)*sqrt(2)*e^(-444276460404\
095319/693299190638000000)/sqrt(pi)
```




```
#Here's something you may or may not know about mathematica
#It's a wast of money
#This was done sorta in python
#but actually sagemath
#You should look into sagemath
#There's a lot of author's of source code, so be a little careful about data-types
#I encountered a few of those errors
#One really fun thing I learned to do was return new function from functions
#I know that you can do this in regular python, but I've never really messed with it before
#it's an interesting way to change parameters on a function while reserving the actual parameter space for graph
axes
# Also you can 3d graph in python with this also in the cell below I have one more cool thing to show you
```

```
var('z')
print integrate(z*2+5*z**4)
```

```
z^5 + z^2
```

```
#Look at that symbolic integration
```

evaluate