Sum of Gaussian random variables using python

Asked 2 months ago Modified 2 months ago Viewed 104 times

0

Given two independent Gaussian variables X and Y, with probability density functions pdf1 and pdf2, then I want to calculate Z = X + Y ~ PDF(Z).

The probability density function of Z is given by the convolution of pdf1 and pdf2. I have taken the code base (see scipy - Python: How to get the convolution of two continuous distributions? - Stack Overflow) and adapted it.

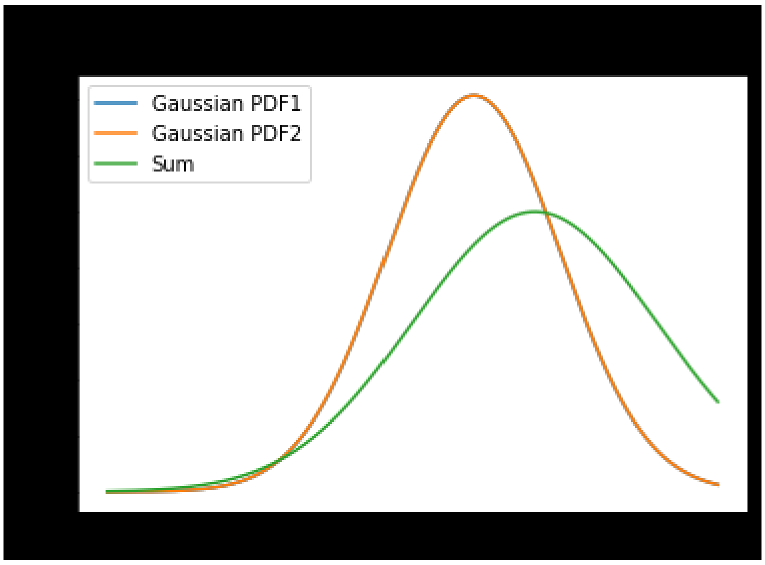
First, I tested the solution with mean=0 and sigma²=1 for both pdf1 and pdf2. I got the correct solution. E(Z)=E(X)+E(Y)=0 and Var(Z)=Var(X)+Var(Y)=2

Second, I tested the solution with mean=2 and sigma²=8 for both pdf1 and pdf2. I got an approximate solution with large errors. Result was E(Z)=E(X)+E(Y)=3.21 and Var(Z)=Var(X)+Var(Y)=12.21 but expected was E(Z)=E(X)+E(Y)=4.0 and Var(Z)=Var(X)+Var(Y)=16.0.

The critical part in the code is the convolution of pmf1 and pmf2. The sum of the convoluted PDF should be 1.0 and not 0.93.

Hint: I used a reference implementation based on the "openturns" library to verify my results.

```
#given two independent gaussian variables X,Y; calculate Z = X + Y ~ PDF(Z)

delta = 1e-4
big_grid = np.arange(-10,10,delta)

mean = 2  #E(X)=E(Y)=2
std = np.sqrt(8)  #Var(X)=Var(Y)=8

X = norm(loc=mean, scale=std)
Y = norm(loc=mean, scale=std)

pmf1 = X.pdf(big_grid)*delta
print("Sum of gaussian pmf: "+str(sum(pmf1)))

pmf2 = Y.pdf(big_grid)*delta
print("Sum of gaussian pmf: "+str(sum(pmf1)))

conv_pmf = signal.fftconvolve(pmf1,pmf2,'same')  #convolution of pmf1 and pmf2
print("Sum of convoluted pmf: "+str(sum(conv_pmf)))

pdf1 = pmf1/delta
pdf2 = pmf2/delta
conv_pdf = conv_pmf/delta
print("Integration of convoluted pdf: " + str(np.trapz(conv_pdf, big_grid)))

plt.plot(big_grid, pdf1, label='Gaussian PDF1')
plt.plot(big_grid, pdf2, label='Gaussian PDF2')
plt.plot(big_grid, conv_pdf, label='Sum')
plt.legend(loc='best'), plt.suptitle('PDFs')
plt.show()
```



```
Mean and variance of convoluted PDF

#E(Z)=E(X)+E(Y); Var(Z)=Var(X)+Var(Y); if E(X)=E(Y)=2 and Var(X)=Var(Y)=8 it follows
E(Z)=4 and Var(Z)=16
E_Z = (big_grid * conv_pmf).sum(); E_Z  #E(Z) = Σ z . P(z): sum(z[j] * p(z[j]))
expected: E(Z)=4
E_Z_squared = (big_grid**2 * conv_pmf).sum(); E_Z_squared  #E(Z²) = Σ z² . P(z):
sum(z[j]² * p(z[j]))
Var_Z =  E_Z_squared - (E_Z)**2; Var_Z  #Var(Z) = E(Z²) - E(Z)²; expected: Var(Z)=16
```

This is the output I get.

Sum of gaussian pmf1: 0.9976499589626819

Sum of gaussian pmf2: 0.9976499589626819

Sum of convoluted pmf: 0.9321607580277965

Integration of convoluted pdf: 0.9321591482687606

E_Z = 3.210819533318452

E_Z_squared = 22.52303025237063

Var_Z = 12.21366817683131

So what is going wrong here? How can I adapt the code to get correct results?

python convolution gaussian Edit tags

Share Edit Follow Close Flag

edited Jun 21 at 4:47
mkl
86.3k 14 124 246

asked Jun 20 at 17:05
josef12
43 6

Sorted by:

1 Answer

Date modified (newest first)

1

The results you have now are fine. There is no reason to believe the sums you are printing here would be equal to 1. Although it is true that the integral of the PDF over the entire support (from negative to positive infinity) would be `1`, this doesn't have to be true discretised version because it is an approximation.

Remember also that your grid is `arange(-10, 10, delta)`, and that a significant proportion of the total probability of `norm(4, 4)` lies outside of that range.

Luckily, you know the PDF for the sum of normal variables, so you can check your results yourself using the CDF of the real distribution.

```
def realcdf(x):
    return stats.norm(loc = 4, scale = 4).cdf(x)

print("Supposed to be: " + str(realcdf(max(big_grid)) - realcdf(min(big_grid))))
```

With output:

```
Supposed to be: 0.9329569316499936
```

Which is not `1`. In fact the `fftconvolve` approximation is quite close. Errors arising from floating point arithmetic and the discretisation onto the grid likely account for the relatively small difference between the two.

As for the statistics at the end, enlarging the size of the grid should help. For example, on the grid:

```
big_grid = np.arange(-20,20,delta)
```

Produces statistics closer to the truth:

```
E_Z = 3.9994379102826576
Var_Z = 15.991432657282482
```

Share Edit Follow Flag

edited Jun 21 at 11:12

answered Jun 20 at 18:57
L.Grozinger
2,065 1 8 22

Great many thanks for your insightful explanation! – josef12 Jun 21 at 17:23