Lab1_Natalia_Tabja

December 6, 2024

0.1 1. Numerical Error

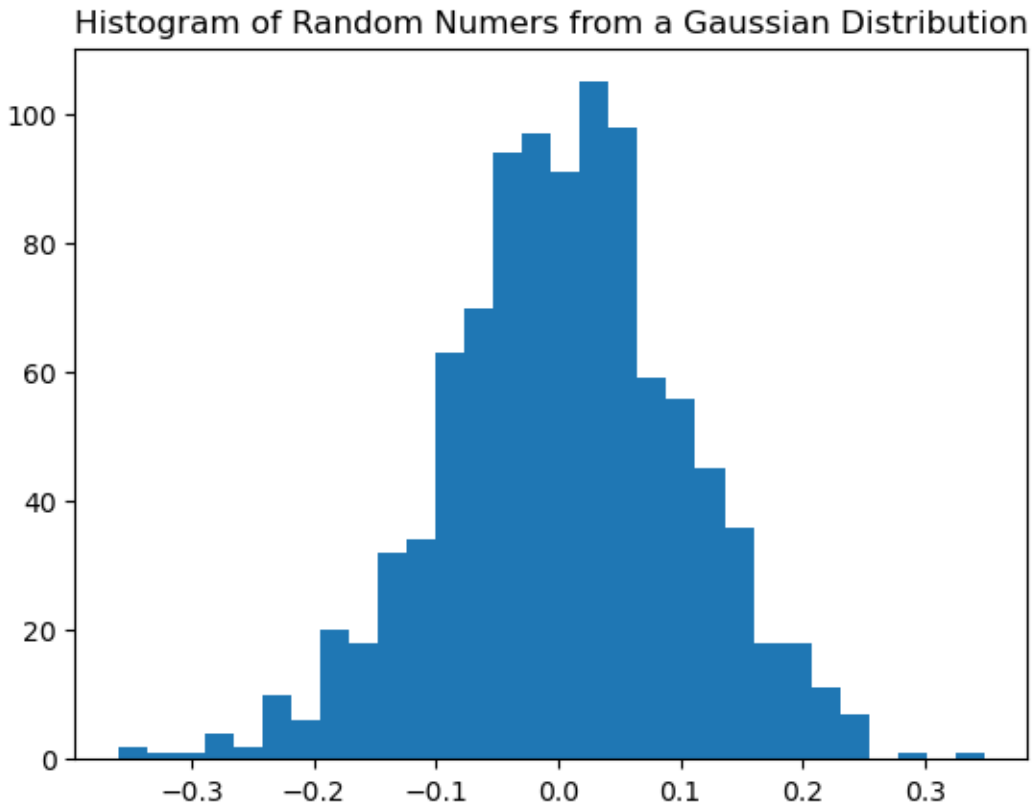
[40]:
```
import numpy as np
import matplotlib.pyplot as plt
```

0.2 Part (a)

Write code to generate an array of length N, filled with random numbers from a Gaussian distribution (with mean x and standard deviation σ), and make a histogram (with m bins) of the array values. (You are encouraged to import functions from numpy and matplotlib.pyplot) Test this code, performing at least two sanity checks to make sure it's working properly.

[41]:
```
N = 1000
bin_n = 50
mu, sigma = 0, 0.1 # mean and standard deviation
A = np.random.normal(mu, sigma, N)

plt.title("Histogram of Random Numers from a Gaussian Distribution")
plt.hist(A, 30)
```

[41]:
```
(array([  2.,   1.,   1.,   4.,   2.,  10.,   6.,  20.,  18.,  32.,  34.,
         63.,  70.,  94.,  97.,  91., 105.,  98.,  59.,  56.,  45.,  36.,
         18.,  18.,  11.,   7.,   0.,   1.,   0.,   1.]),
 array([-0.35982816, -0.3362162 , -0.31260424, -0.28899228, -0.26538032,
        -0.24176836, -0.2181564 , -0.19454443, -0.17093247, -0.14732051,
        -0.12370855, -0.10009659, -0.07648463, -0.05287266, -0.0292607 ,
        -0.00564874,  0.01796322,  0.04157518,  0.06518714,  0.08879911,
         0.11241107,  0.13602303,  0.15963499,  0.18324695,  0.20685891,
         0.23047087,  0.25408284,  0.2776948 ,  0.30130676,  0.32491872,
         0.34853068]),
 <BarContainer object of 30 artists>)
```



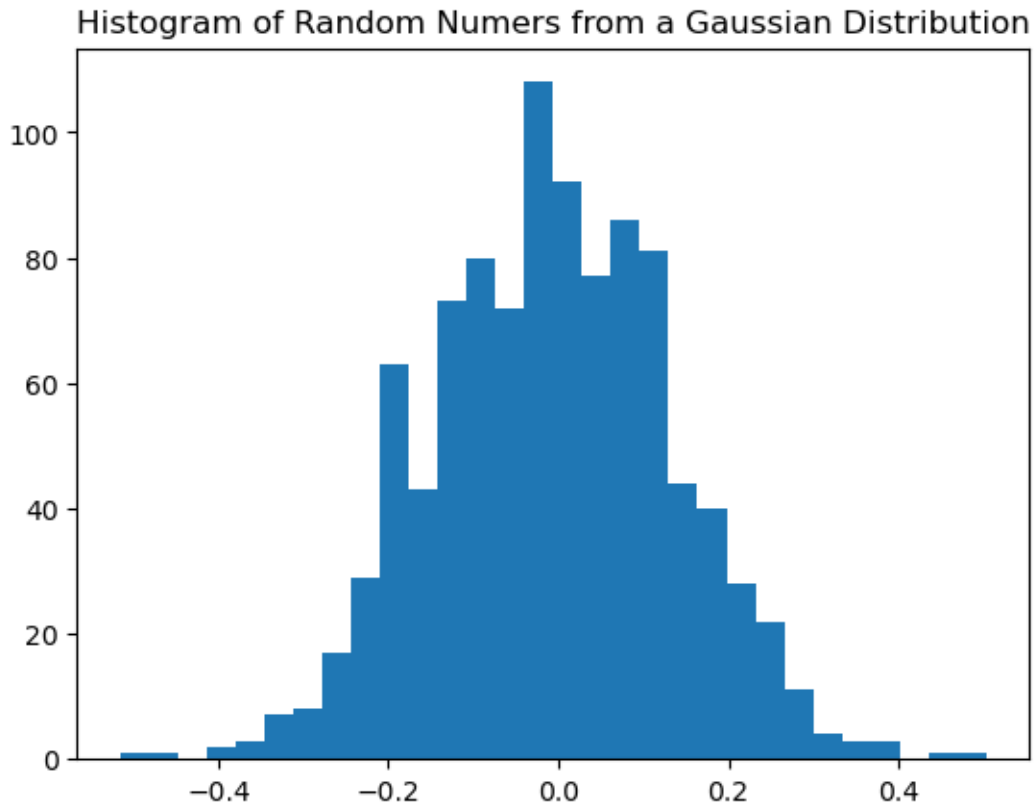
0.3 Part (b)

Now write a function to: generate two arrays as in the previous part (where the first Gaussian has x_1, σ_1 and the second Gaussian has x_2, σ_2), make a third array that is the element-wise sum of these two arrays, and make a histogram of the 'array values. The function should take $x_1, \sigma_1, x_2, \sigma_2$ as arguments.

[42]:
```
def gaussian_generator(mu1, sigma1, mu2, sigma2):
    A1 = np.random.normal(mu1, sigma1, N)
    A2 = np.random.normal(mu2, sigma2, N)
    A3 = A1 + A2
    return A3


N = 1000
mu1, sigma1 = 0, 0.1 # mean and standard deviation for first array
mu2, sigma2 = 0, 0.1 # mean and standard deviation for second array

A = gaussian_generator(mu1, sigma1, mu2, sigma2)

plt.title("Histogram of Random Numers from a Gaussian Distribution")
```

```
plt.hist(A, 30)
```

```
[42]: (array([  1.,   1.,   0.,   2.,   3.,   7.,   8.,  17.,  29.,  63.,  43.,
         73.,  80.,  72., 108.,  92.,  77.,  86.,  81.,  44.,  40.,  28.,
         22.,  11.,   4.,   3.,   3.,   0.,   1.,   1.]),
  array([-0.5152821 , -0.48133741, -0.44739273, -0.41344804, -0.37950335,
         -0.34555867, -0.31161398, -0.27766929, -0.24372461, -0.20977992,
         -0.17583523, -0.14189055, -0.10794586, -0.07400117, -0.04005649,
         -0.0061118 ,  0.02783289,  0.06177757,  0.09572226,  0.12966695,
          0.16361163,  0.19755632,  0.23150101,  0.26544569,  0.29939038,
          0.33333507,  0.36727975,  0.40122444,  0.43516913,  0.46911381,
          0.5030585 ]),
  <BarContainer object of 30 artists>)
```



0.4 Part (c)

Recall from your physics labs, when you make repeated measurements of a quantity, the measurements follow a Gaussian distribution, where the mean (hopefully) represents the 'true' value of the quantity and the standard deviation represents the statistical uncertainty (error) on the measurement. If you make repeated measurements of two different quantities, and use these two quantities to calculate a third quantity, the error propagates, so you have to use error propagation formulas

to figure out the uncertainty on the third quantity.

Numerical errors in calculations propagate in a similar manner. We can represent the numerical error on stored value x as σ, and define our fractional error constant as C such that $\sigma = C|x|$.

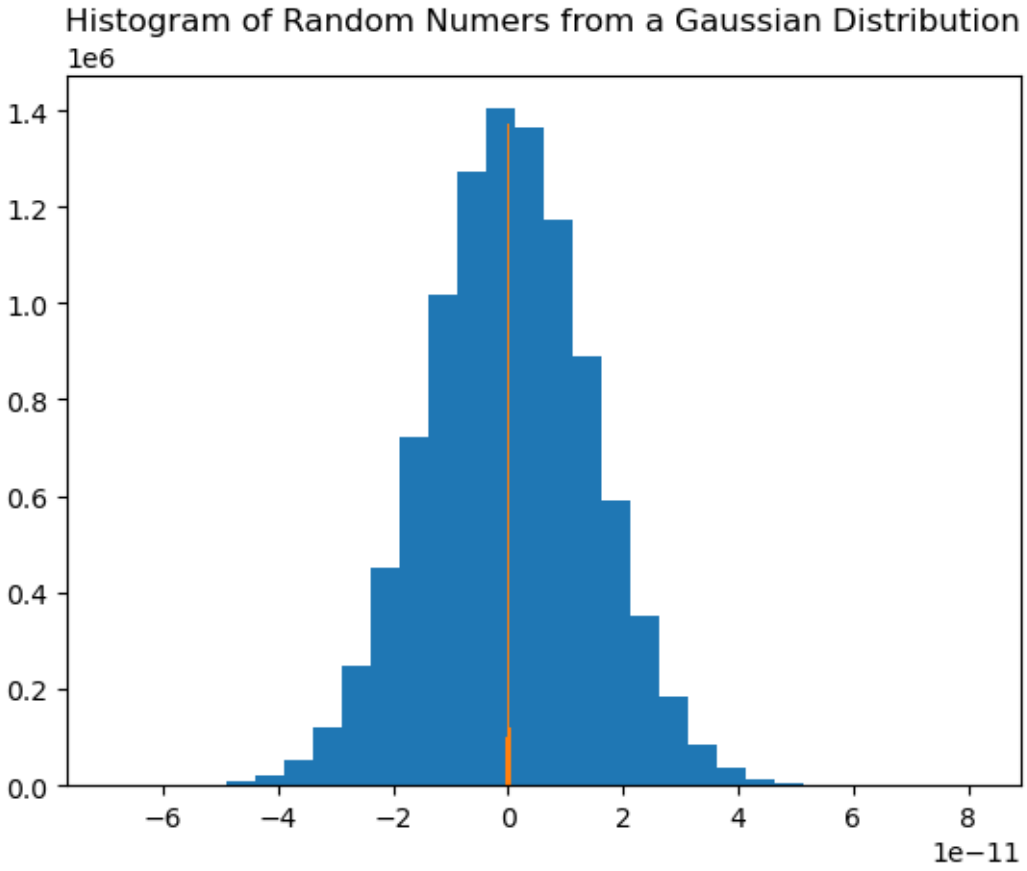
Plugging this into the previous part, we get $\sigma_1 = C|x_1|, \sigma_2 = C|x_2|$. Try this with: $N = 10^6, m = 100, C = 10^{-14}, x_1 = 100, x_2 = -100$. (For the purposes of this exercise, we're using a much larger value of C than the machine precision, so that we can see its effect.) Now try again with different mean values: $x_1 = 1.0, x_2 = -1.0$

Hopefully, you now see visually why adding a positive number to a negative number, where both have large absolutely values, can produce large errors.

[48]:
```
N = int(10e6)
m = 100
C = 10e-14

x1 = 100
x2 = -100
sig1 = C * x1
sig2 = C * np.abs(x2)

x_1 = 1.0
x_2 = -1.0
sig_1 = C * x_1
sig_2 = C * np.abs(x_2)


arr1 = gaussian_generator(x1, sig1, x2, sig2)
arr2 = gaussian_generator(x_1, sig_1, x_2, sig_2)

plt.title("Histogram of Random Numers from a Gaussian Distribution")
plt.hist(arr1, 30)
plt.hist(arr2, 30)
```

[48]:
```
(array([6.000000e+00, 2.300000e+01, 8.700000e+01, 4.090000e+02,
        1.472000e+03, 5.172000e+03, 1.561400e+04, 4.219200e+04,
        9.853700e+04, 2.060960e+05, 3.803540e+05, 6.295020e+05,
        9.132380e+05, 1.180917e+06, 1.352462e+06, 1.373653e+06,
        1.235346e+06, 9.843950e+05, 6.991750e+05, 4.368670e+05,
        2.433810e+05, 1.189430e+05, 5.233800e+04, 2.010400e+04,
        6.962000e+03, 2.039000e+03, 5.500000e+02, 1.410000e+02,
        2.000000e+01, 5.000000e+00]),
 array([-7.45958850e-13, -6.96635342e-13, -6.47311834e-13, -5.97988326e-13,
        -5.48664817e-13, -4.99341309e-13, -4.50017801e-13, -4.00694293e-13,
        -3.51370784e-13, -3.02047276e-13, -2.52723768e-13, -2.03400260e-13,
        -1.54076751e-13, -1.04753243e-13, -5.54297349e-14, -6.10622664e-15,
         4.32172816e-14,  9.25407898e-14,  1.41864298e-13,  1.91187806e-13,
         2.40511315e-13,  2.89834823e-13,  3.39158331e-13,  3.88481839e-13,
```

```
        4.37805348e-13,  4.87128856e-13,  5.36452364e-13,  5.85775872e-13,
        6.35099380e-13,  6.84422889e-13,  7.33746397e-13]),
<BarContainer object of 30 artists>)
```



1 2. Approximation Error

Consider this system representing phasor rotation in the complex plane:

$$\dot{Z} = i\omega Z, \quad \text{given} \quad Z_0 = Z(t=0).$$

The analytical solution is: $Z(t) = Z_0 \exp(i\omega t)$.

How can we solve it numerically? We could try using its Taylor expansion:

$$\dot{Z}(t) = \frac{Z(t+\Delta t) - Z(t)}{\Delta t} + H.O.T. = i\omega Z(t).$$

And use a simple algorithm such as: * Start with $Z_0 = Z(t=0) = Z_{old}$, * $Z_{new} = (1 + i\omega\Delta t)Z_{old}$, * repeat for a large number n of timesteps until we complete a full rotation

It turns out this simple algorithm is unstable, because of the accumulation of error.

1.1 Part (a)

Write the code to implement the above algorithm, with ω, Z_0, n as parameters that can be set in the code. You should end up with an array of t values and an array of Z values covering a full rotation.

You may want to make use of numpy.pi

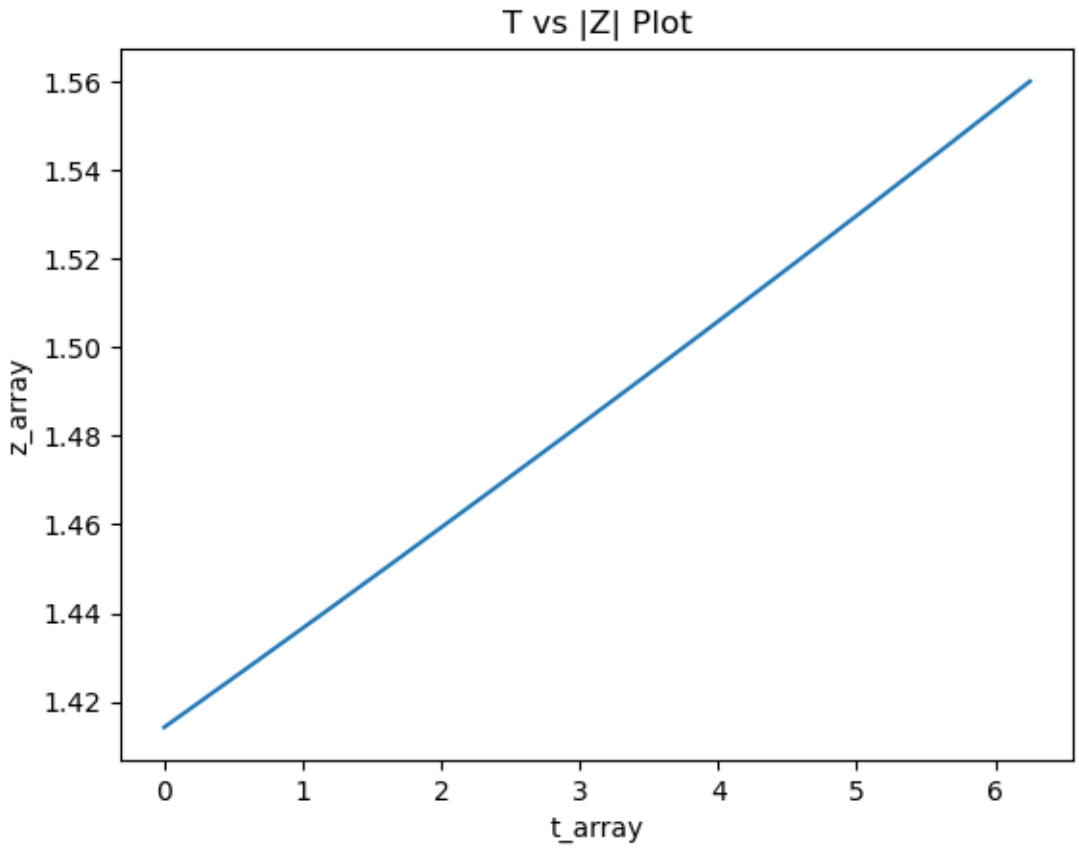
[44]:
```
omega = 1.0 # Hz
n = 200
Z0 = complex(1.0, 1.0)
t_array = np.empty(n, float)
z_array= np.empty(n, complex)

dt = 2 * np.pi/(n*omega)
z_array[0] = Z0
t_array[0] = 0
for k in range(n - 1):
    z_array[k + 1] = z_array[k] * (1 + 1j * omega * dt)
    t_array[k + 1] = t_array[k] + dt
```

1.2 Part (b)

Using the code in the previous part, plot $|Z|(t)$ (absolute value of Z, as a function of t) for $n = 200, Z_0 = 1, \omega = 1\text{Hz}$. Is the result what you expected? Why?

[47]:
```
plt.title("T vs |Z| Plot")
plt.xlabel("t_array")
plt.ylabel("z_array")
plt.plot(t_array, np.abs(z_array))
```

[47]:
```
[<matplotlib.lines.Line2D at 0x7f96eacbee90>]
```



This makes sense because removing the imaginary component (i.e, taking the absolute value) of the Z array would only leave the real component, leading to a linear plot.