In [ ]:
```
from sympy import binomial
```

Chem 30324, Spring 2025, Homework 1

Due on January 24, 2025

Problem 1: Discrete, probably

In five card study, a poker player is dealt five cards from a standard deck of 52 cards.

1. How many different 5-card hands are there? (Remember, in poker the order in which the cards are received does *not* matter.)

In poker, the order in which cards are received does not matter. To calculate the total number of 5-card hands, we use the formula for combinations:

$$\binom{n}{r} = \frac{n!}{r!(n-r)!} \tag{1}$$

the total number of card (n) is 52, and the number of card in hand is 5

The total number of hands is:

$$\binom{52}{5} = 2598960 \tag{2}$$

In [ ]:
```
# Total number of cards and cards dealt
n = 52
r = 5

# Number of different 5-card hands
num_hands = binomial(n, r)
print("There are", num_hands, "different 5-card hands")
```

```
There are 2598960 different 5-card hands
```

2. What is the probability of being dealt four of a kind (a card of the same rank from each suit in a five card hand)?

A four of a kind means that the hand includes 4 cards of the same rank (e.g., four Aces, four Kings, etc.) and one additional card of a different rank.

To calculate this probability, we start by noting that there are 13 possible ranks to choose from. For each rank, there is only one way to select all 4 cards (one from each suit). After choosing the four of a kind, there are (52 - 4 = 48) cards remaining in the deck to choose the fifth card. Thus, the total number of four-of-a-kind hands is:

$$\text{Number of Four of a Kind Hands} = 13 \times 48 \tag{3}$$

The total number of 5-card hands is:

$$\binom{52}{5} \tag{4}$$

The probability of being dealt four of a kind is given by:

$$P(\text{Four of a Kind}) = \frac{\text{Number of Four of a Kind Hands}}{\text{Total Number of Hands}} \tag{5}$$

Substituting the values:

$$P(\text{Four of a Kind}) = \frac{13 \times 48}{\binom{52}{5}} \approx 0.00024 \, (\text{or } 0.024\backslash\%). \tag{6}$$

In [ ]:
```
# Calculate the total number of hands
total_hands = binomial(52, 5)

# Calculate the number of four-of-a-kind hands
four_of_a_kind_hands = 13 * 48

# Calculate the probability
probability_four_of_a_kind = four_of_a_kind_hands / total_hands

print(f"The probability of being dealt four of a kind is: {probability_four_
```

```
The probability of being dealt four of a kind is: 0.00024010
```

3. What is the probability of being dealt a flush (five cards of the same suit)?

A flush means that all 5 cards in the hand belong to the same suit but are not in sequence (if they are in sequence, it's a straight flush).

To calculate this probability, we first note that there are 4 suits to choose from (Hearts, Diamonds, Clubs, Spades). For each suit, we can choose 5 cards out of 13 cards, which can be calculated using binomial.

Thus, the total number of flush hands is:

$$\text{Number of Flush Hands} = 4 \times \binom{13}{5} \tag{7}$$

The total number of 5-card hands is:

$$\binom{52}{5} \tag{8}$$

The probability of being dealt a flush is given by:

$$P(\text{Flush}) = \frac{\text{Number of Flush Hands}}{\text{Total Number of Hands}} \tag{9}$$

Substituting the values:

$$P(\text{Flush}) = \frac{4 \times \binom{13}{5}}{\binom{52}{5}} \approx 0.00198\,(\text{or } 0.198\backslash\%). \tag{10}$$

In [ ]:
```
# Calculate the total number of hands
total_hands = binomial(52, 5)

# Calculate the number of flush hands
flush_hands = 4 * binomial(13, 5) - 40

# Calculate the probability
probability_flush = flush_hands / total_hands

# Print the result
print(f"The probability of being dealt a flush is: {probability_flush:.8f}")
```

```
The probability of being dealt a flush is: 0.00196540
```

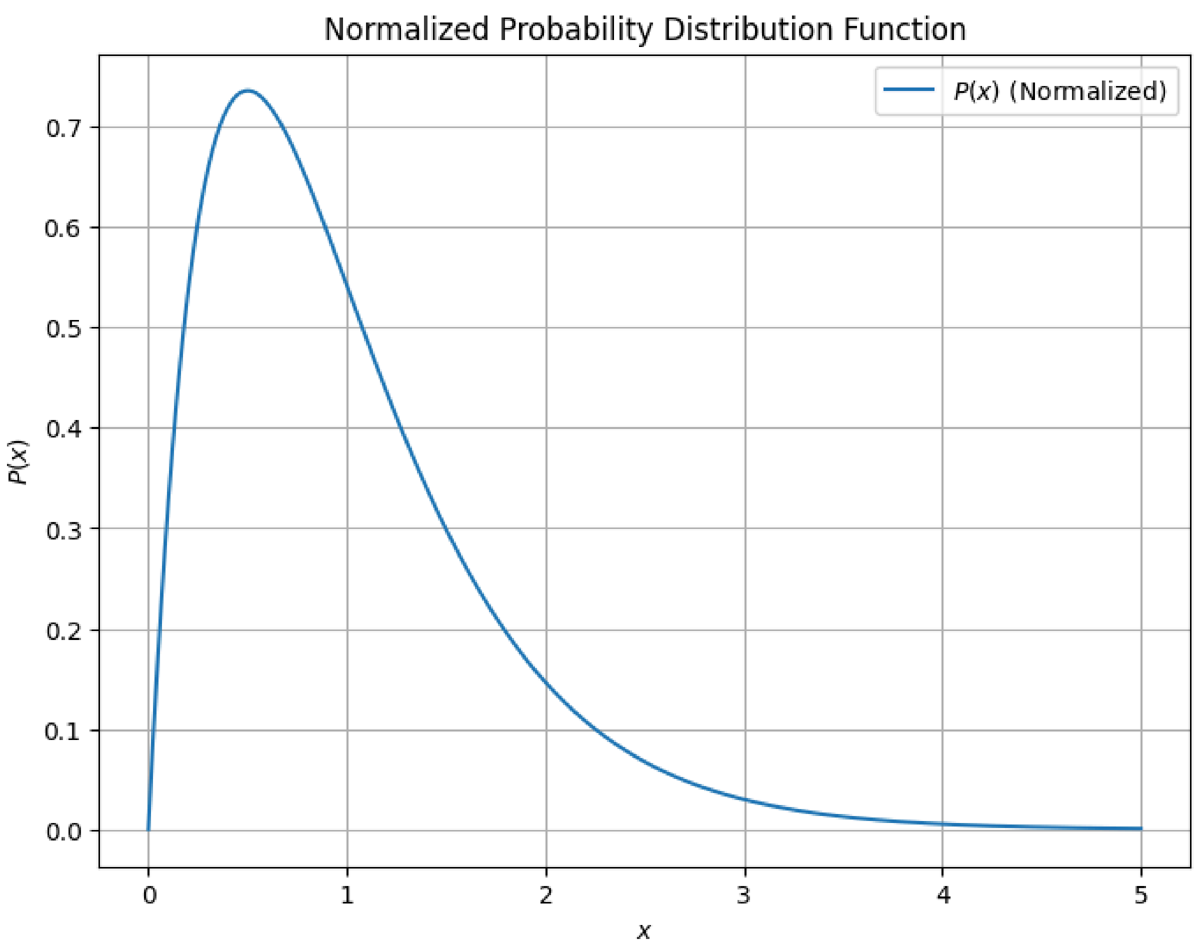
Problem 2: Continuous, probably

The probability distribution function for a random variable x is given by $P(x) = xe^{-2x}, 0 \leq x < \infty$.

1. Is $P(x)$ normalized? If not, normalize it. Plot the normalized $P(x)$.

In [ ]:
```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# Define the variables and function
x = sp.symbols('x', positive=True)
```

```
P = x * sp.exp(-2 * x)

# 1. Check normalization
normalization_integral = sp.integrate(P, (x, 0, sp.oo))

# If not normalized, calculate normalization constant
normalization_constant = 1 / normalization_integral
P_normalized = normalization_constant * P

x_vals = np.linspace(0, 5, 500)
P_normalized_func = sp.lambdify(x, P_normalized, 'numpy')
P_vals = P_normalized_func(x_vals)

plt.figure(figsize=(8, 6))
plt.plot(x_vals, P_vals, label=r"$P(x)$ (Normalized)")
plt.title("Normalized Probability Distribution Function")
plt.xlabel("$x$")
plt.ylabel("$P(x)$")
plt.grid(True)
plt.legend()
plt.show()
```



2. What is the most probable value of x?

In [ ]:
```
#
P_derivative = sp.diff(P_normalized, x)
```

```
most_probable_value = sp.solve(P_derivative, x)
[float(v) for v in most_probable_value]
```

Out[ ]:
```
[0.5]
```

3. What is the expectation value of x?

In [ ]:
```
expectation_value = sp.integrate(x * P_normalized, (x, 0, sp.oo))
float(expectation_value)
```

Out[ ]:
```
1.0
```

4. What is the variance of x?

In [ ]:
```
x_squared_expectation = sp.integrate(x**2 * P_normalized, (x, 0, sp.oo))
variance = x_squared_expectation - expectation_value**2
float(variance)
```

Out[ ]:
```
0.5
```

Problem 3: One rough night

It's late on a Friday night and people are stumbling up Notre Dame Ave. to their dorms. You observe one particularly impaired individual who is taking steps of equal length 1m to the north or south (i.e., in one dimension), with equal probability.

In [ ]:
```
from math import comb
import matplotlib.pyplot as plt

# Parameters for the random walk
n_steps = 20
step_size = 1
p = 0.5  # Probability for each direction

# Calculate distances and probabilities
distances = np.arange(-n_steps, n_steps + 1, 2)
probabilities = [comb(n_steps, (n_steps + d) // 2) * (p**n_steps) for d in c
```

In [ ]:
```
distances
```

Out[ ]:
```
array([-20, -18, -16, -14, -12, -10,  -8,  -6,  -4,  -2,   0,   2,   4,
         6,   8,  10,  12,  14,  16,  18,  20])
```

In [ ]:
```
probabilities
```

Out[ ]:
```
[9.5367431640625e-07,
 1.9073486328125e-05,
 0.000181198120117187,
 0.0010871887207031255,
 0.004620552062988281,
 0.0147857666015625,
 0.03696441650390625,
 0.0739288330078125,
 0.12013435363769531,
 0.16017913818359375,
 0.17619705200195312,
 0.16017913818359375,
 0.12013435363769531,
 0.0739288330078125,
 0.03696441650390625,
 0.0147857666015625,
 0.004620552062988281,
 0.0010871887207031255,
 0.000181198120117187,
 1.9073486328125e-05,
 9.5367431640625e-07]
```

1. What is the furthest distance the person could travel after 20 steps?

In [ ]:
```
n_steps
```

Out[ ]:
```
20
```

2. What is the probability that the person won't have traveled any net distance at all after 20 steps?

In [ ]:
```
prob_zero_distance = probabilities[np.where(distances==0)[0][0]]
prob_zero_distance
```

Out[ ]:
```
0.17619705200195312
```

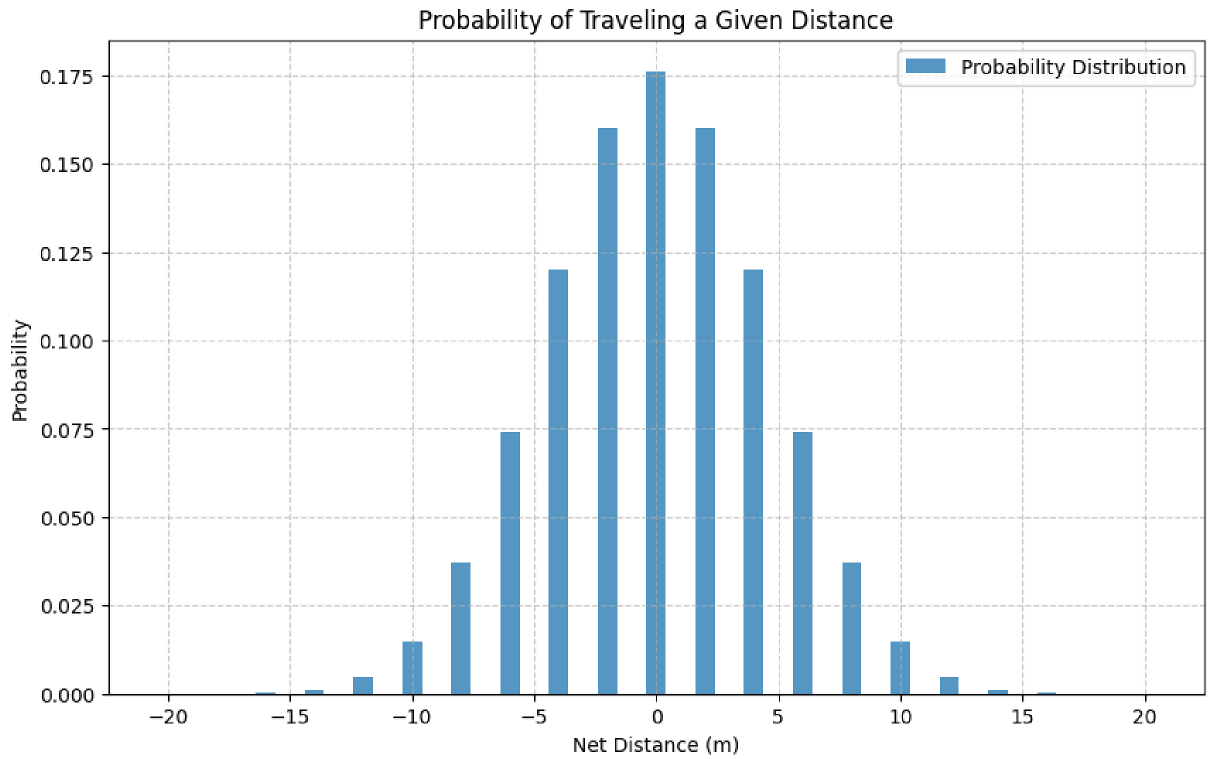
3. What is the probability that the person has traveled half the maximum distance after 20 steps?

In [ ]:
```
prob_half_max_distance = probabilities[np.where(distances==10)[0][0]]+probab
prob_half_max_distance
```

Out[ ]:
```
0.029571533203125
```

4. Plot the probability of traveling a given distance vs distance. Does the probability distribution look familiar? You'll see it again when we talk about diffusion.

In [ ]:
```
plt.figure(figsize=(10, 6))
plt.bar(distances, probabilities, width=0.8, alpha=0.75, label="Probability
plt.title("Probability of Traveling a Given Distance")
plt.xlabel("Net Distance (m)")
plt.ylabel("Probability")
plt.grid(True, linestyle="--", alpha=0.6)
plt.legend()
plt.show()
```



Problem 4: Now this is what I call equilibrium

The Boltzmann distribution tells us that, at thermal equilibrium, the probability of a particle having an energy $E(q, p)$ is proportional to $\exp(-E(q, p)/k_\mathrm{B}T)$, where k_B is the Boltzmann constant and q and p are position and momentum ($p = mv$), respectively. Suppose a bunch of Ar molecules are in thermal equilibrium at temperature T and are traveling back and forth in one dimension with various momenta and kinetic energies $K = p^2/2$.

1. What is the expectation value of the momentum p of an Ar molecule?

following along with Ex. kinetic energy in Lecture 1-Probability pdf and using information from the class outline and the Temperature example:

$$-inf < p_x < inf \tag{11}$$

$$KE(p_x) = \frac{(p_x)^2}{(2m)} \tag{12}$$

$$\Phi(p_x) \propto e^{-\frac{p_x^2}{2mk_bT}} \tag{13}$$

Calculating the Normalization factor:

$$\int_{-inf}^{inf} \Phi(p_x) = \sqrt{2\pi m k_b T} \tag{14}$$

gives normalized probability distribution

$$\Phi(p_x) = \sqrt{\frac{1}{(2\pi m k_b T}} e^{-\frac{p^2}{2m*kb*T}} \tag{15}$$

Comparing this distribution to the gaussian curve equation gives the following:

$$< p_x >= \mu = 0 \tag{16}$$

This makes intuitive sense as the momentum is the velocity scaled by the mass and the expectation value of the velocity is also 0.

2. What is the expectation value of the kinetic energy K of an Ar molecule?

By comparing the normalized equation to the gaussian distribution

$$< p_x^2 >= \mu = mk_bT \tag{17}$$

plugging this into the equation for KE gives:

$$< KE(p_x) >= \frac{< p_x^2 >}{2m} = \frac{mk_bT}{2m} = \frac{k_bT}{2} \tag{18}$$

This also matches the value when calculating $< KE >$ using velocity

3. How would your answers change if the molecules were Xe?

As neither answer is dependent on mass, neither answer would change

Solving 2.4 via code

In [ ]:
```
#The following is just graphing the calculated momentums and probabilities/p
#while checking we match the above results
#
```

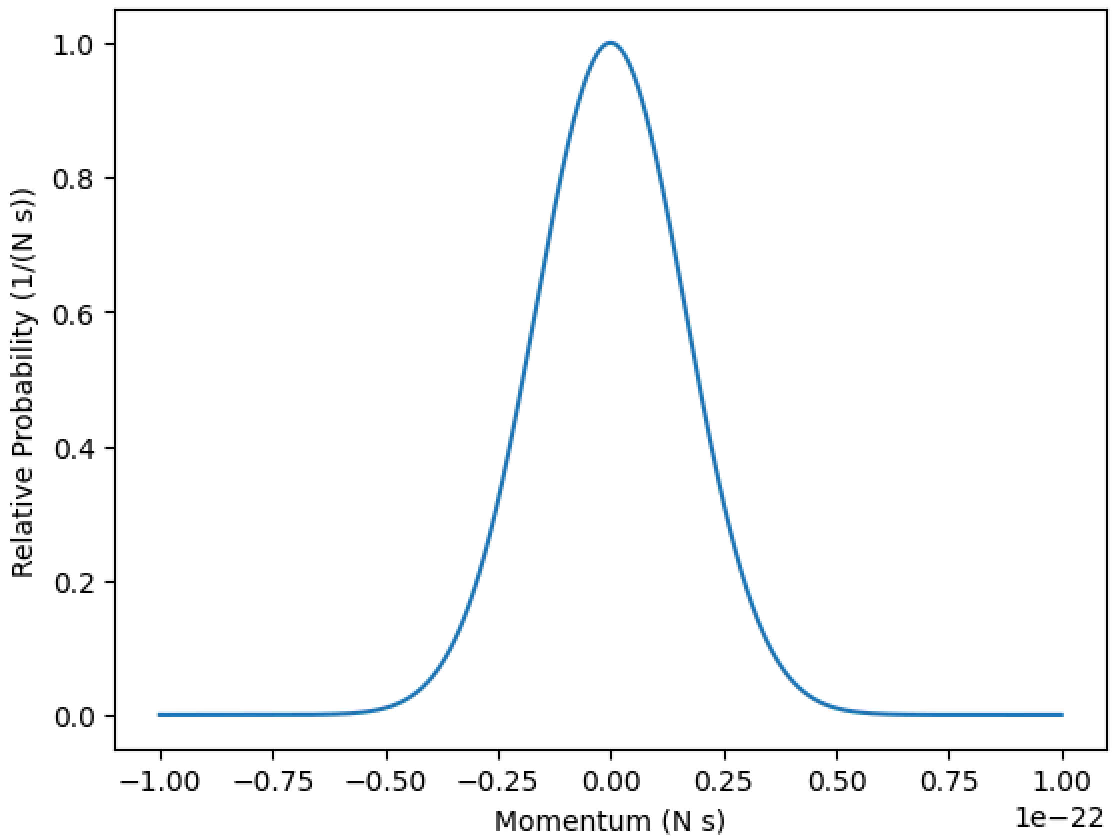
```
#There are some unintuitive results on the y axis due to the small scale
#additionally both using the explicit normalization factor and numeric integ
#and propagated to give the expectation value.
#
#Libraries and code altered from code given by Temperature example
#in the chem30324 outline

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
from scipy.optimize import minimize,fsolve
from scipy.misc import derivative

#Defining Variables and probability function, can play around and see how ma
#mass of 1 Argon atom in kg
m = 6.634e-26
#room temperature in kelvin
T = 293
#boltzman constant in Joules/Kelvin
k = 1.4e-23
#pi
pi = 3.14

#Defining our initial probability equation
def phif(p):
  return np.exp(-(p**2)/(2*m*k*T))
```

In [ ]:

```
#defines p as a 1000 digits spaced between -1e-25 and 1e-25, change these fi
#leaves the bounds of the area
p = np.linspace(-1e-22,1e-22, 1000)
#calculates probabilities of each point p given our previously defined funct
phi = phif(p)
#plots function, axes are being multiplied by the values near the corners, h
#if its normalized or not, use integration to check
plt.plot(p,phi)
plt.xlabel('Momentum (N s)')
plt.ylabel('Relative Probability (1/(N s))')
```

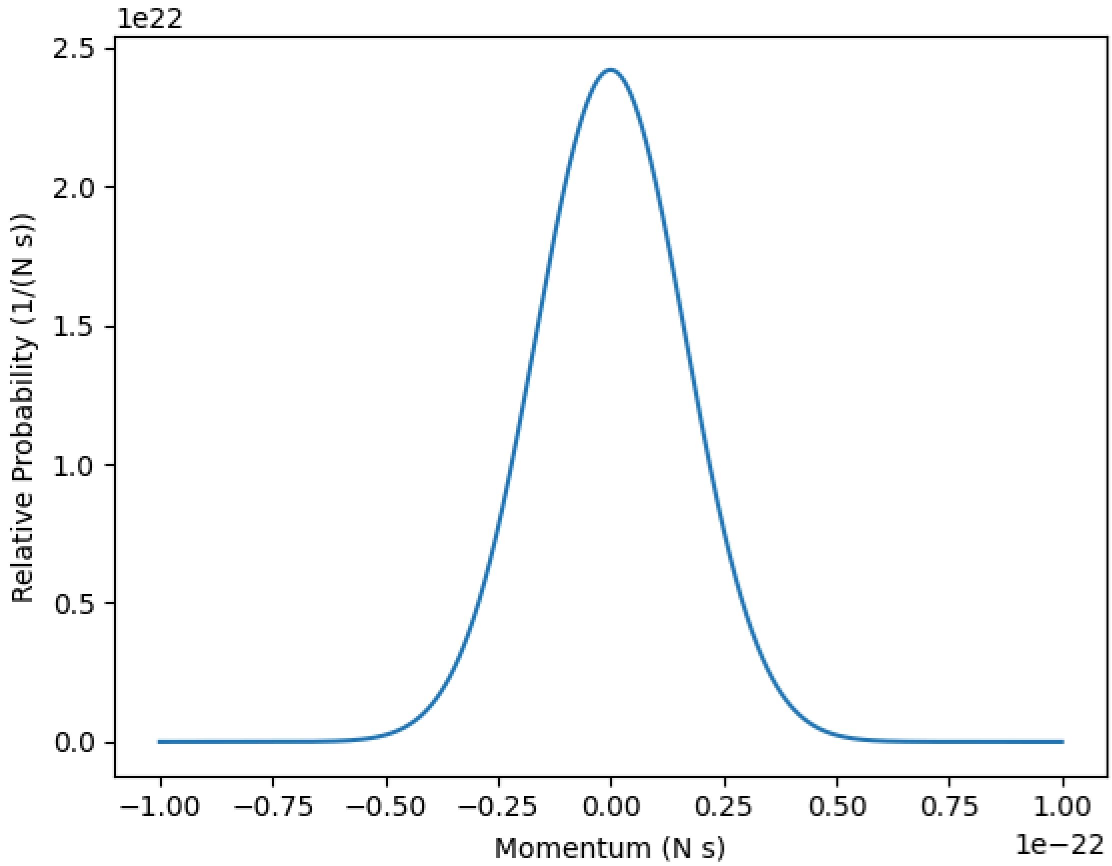
Out[ ]:

```
Text(0, 0.5, 'Relative Probability (1/(N s))')
```



In [ ]:
```
#using the calculated normalization from the first section:

def phif_norm(p):
  return (1/(2*pi*m*k*T))**(1/2)*np.exp(-(p**2)/(2*m*k*T))

phi2 = phif_norm(p)
#plots function, axes are being multiplied by the values near the corners, h
#if its normalized or not, use integration to check
plt.plot(p,phi2)
plt.xlabel('Momentum (N s)')
plt.ylabel('Relative Probability (1/(N s))')

N,err = quad(phif_norm, -1e-22,1e-22)
print("normalization check: ", N)

#The y axis is unintuitive because of how small the x axis is
```

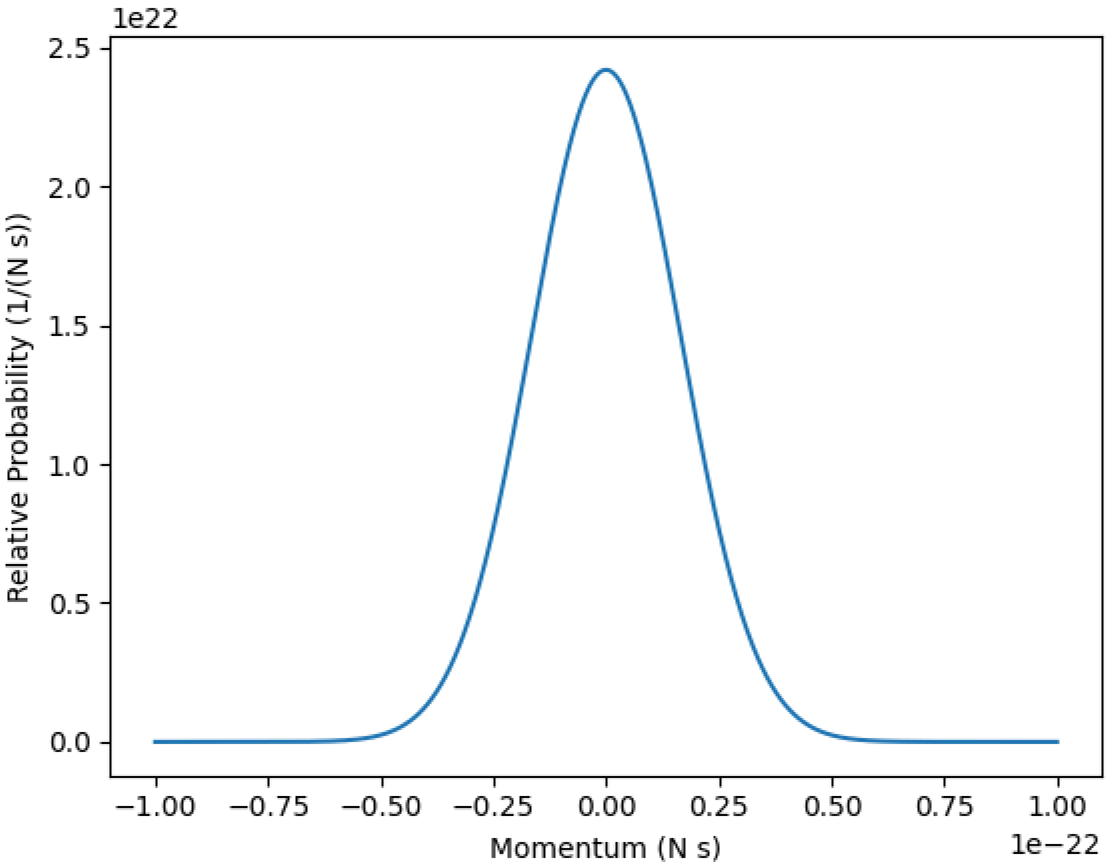
```
normalization check:  1.0002535737644458
```



In [ ]:

```
#Normalization using the same numerical technique in the Temperature example
#quad function takes the function, then bounds as parameters and returns the

N,err = quad(phif, -1e-22,1e-22)
print('Normalization factor: ', N)

def phif_norm2(p):
  return phif(p)/N

#We'll graph the updated probability density
#plots function, axes are being multiplied by the values near the corners, h
#if its normalized or not, use integration to check

plt.plot(p,phif_norm2(p))
plt.xlabel('Momentum (N s)')
plt.ylabel('Relative Probability (1/(N s))')

#Check that this returns 1 to see if its normalized
N2, err2 = quad(phif_norm2, -1e-22,1e-22)
print('Normalization check: ', N2)
```

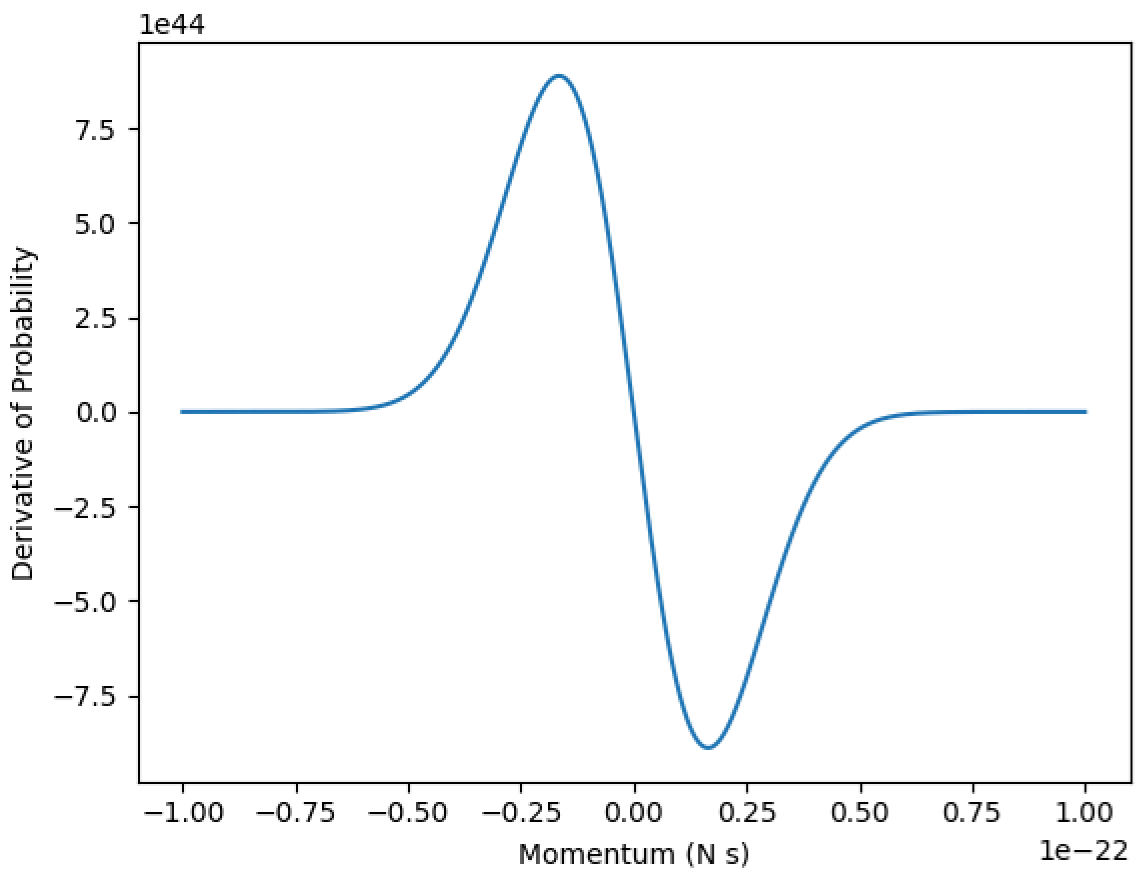
```
Normalization factor:  4.134999822091436e-23
Normalization check:  1.0000000000000002
```



In [ ]:

```
#using derivative of our calculated normalized phi to find the maximum of th
#Looking for the roots/inflection point
def phi_prime(p):
  return derivative(phif_norm, p,dx=1e-24)

plt.plot(p,phi_prime(p))
plt.xlabel('Momentum (N s)')
plt.ylabel('Derivative of Probability')

pguess = 0
root=fsolve(phi_prime, pguess)
print('The likeliest root is: ', root)

#using derivative of our numerically integrated normalized phi to find the m
def phi_prime2(p):
  return derivative(phif_norm2, p,dx=1e-24)

root=fsolve(phi_prime2, pguess)
print('The likeliest root from is: ', root)
```

```
<ipython-input-21-06d444a9f237>:4: DeprecationWarning: scipy.misc.derivative
is deprecated in SciPy v1.10.0; and will be completely removed in SciPy v1.1
2.0. You may consider using findiff: https://github.com/maroba/findiff or nu
mdifftools: https://github.com/pbrod/numdifftools
  return derivative(phif_norm, p,dx=1e-24)
<ipython-input-21-06d444a9f237>:16: DeprecationWarning: scipy.misc.derivativ
e is deprecated in SciPy v1.10.0; and will be completely removed in SciPy v
1.12.0. You may consider using findiff: https://github.com/maroba/findiff or
numdifftools: https://github.com/pbrod/numdifftools
  return derivative(phif_norm2, p,dx=1e-24)
The likeliest root is:  [0.]
The likeliest root from is:  [0.]
```



In [ ]:

```python
#Integrating p*phi(p) to get <p>

def pphif(p):
  return p*phif_norm(p)
pbar,err = quad(pphif,-1e-22,1e-22)
print('The expectation value of momentum is: ', pbar)

def pphif2(p):
  return p*phif_norm2(p)
pbar2,err = quad(pphif2,-1e-22,1e-22)
print('The expectation value of momentum is: ', pbar2)

#integrating p^2*phi(p) to get <p^2>
def p2phif(p):
  return p**2*phif_norm(p)
p2bar,err = quad(p2phif,-1e-22,1e-22)
```

```python
print('The expectation value of momentum squared is: ', p2bar)

#integrating p^2*phi(p) to get <p^2>
def p2phif2(p):
  return p**2*phif_norm2(p)
p2bar2,err = quad(p2phif2,-1e-22,1e-22)
print('The expectation value of momentum squared is: ', p2bar2)

print('Using our calculated expectation for the momentum squared:', m*k*T)

#plugging values into kinetic energy, keeping in mind we've already squared
print('kinetic energy: ', p2bar/(2*m))
print('kinetic energy: ', p2bar2/(2*m))
print('kinetic energy; ', k*T/(2))
```

```
The expectation value of momentum is:  0.0
The expectation value of momentum is:  0.0
The expectation value of momentum squared is:  2.721956703892415e-46
The expectation value of momentum squared is:  2.7212666620608564e-46
Using our calculated expectation for the momentum squared: 2.7212668e-46
kinetic energy:  2.051519975800735e-21
kinetic energy:  2.0509998960362196e-21
kinetic energy;  2.0509999999999997e-21
```