1.1 Plotting the Distribution of Discretized Continuous Random Variables

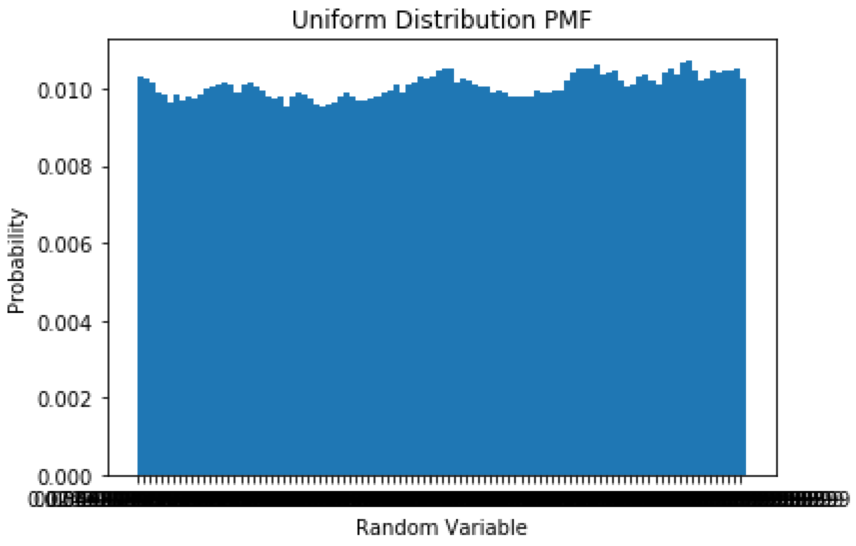
In [70]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from random import uniform
import random
import math
import csv
```

A) Uniform Distribution

In [3]:
```
#Part A - Uniform Distribution
def generator(a, b):
    value = random.uniform(a,b)
    return value
```

In [139]:
```
#a
values = []
#samples 100000
for i in range(100000):
    value = generator(0, 1)
    values.append(value)

#b
counts = []
for j in range(0, 100):
    count = 0
    for k in (values):
        if (k >= (j/1000) and k <= ((j/1000) + 0.01)):
            count += 1
    trueCount = count/100000
    counts.append(trueCount)
xvar = np.arange(0.005, 1, 0.01)

#c
plt.bar(xvar, counts, tick_label=xvar, align='edge', width=.01)
plt.title('Uniform Distribution PMF')
plt.ylabel('Probability')
plt.xlabel('Random Variable')
```

Out[139]:
```
Text(0.5, 0, 'Random Variable')
```

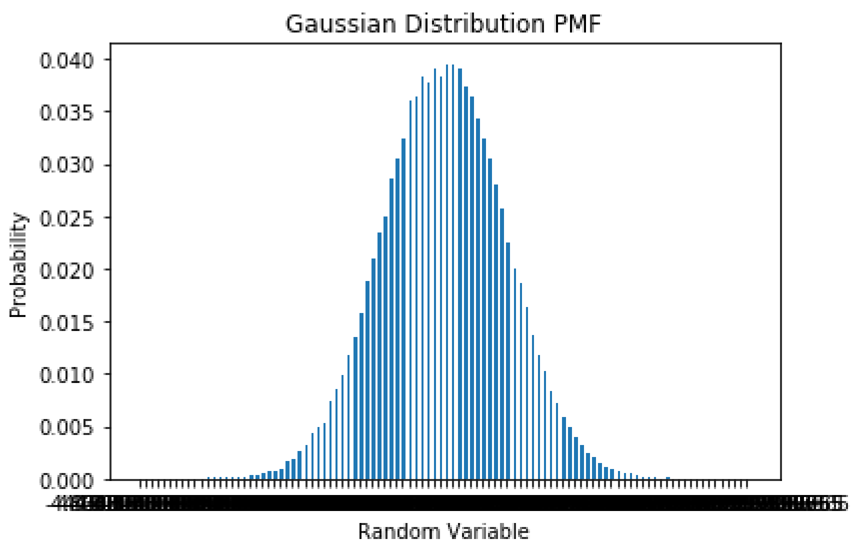


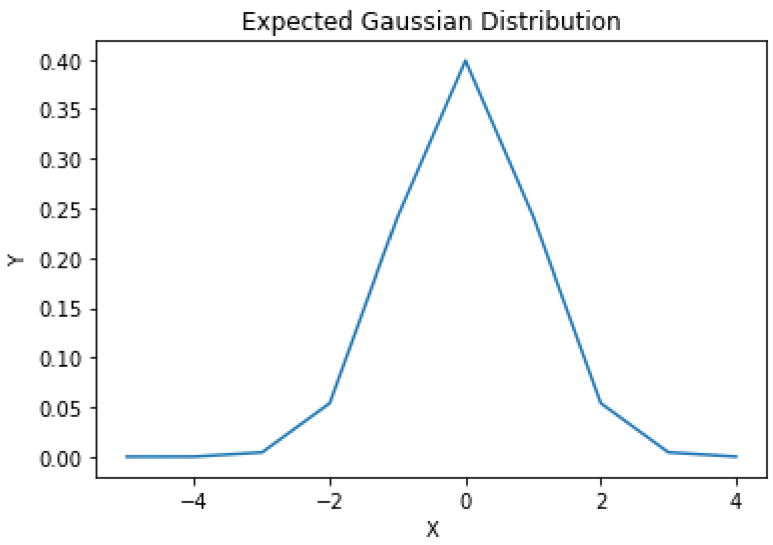
For each random variable, the probablity is roughly around 0.01. This makes sense because all the of the random variables should have equal probablity, and therefore "uniform".

B) Gaussian Distribution

In [140]:
```python
#a
values = []
counts = []
for i in range(100000):
    value = random.gauss(0,1)
    values.append(value)
#b
xvar = np.arange(-5, 5, 0.1)
for j in range(-50, 50):
    count = 0
    for k in (values):
        if (k >= (j/10) and k <= ((j/10) + 0.1)):
            count += 1
    trueCount = count/100000
    counts.append(trueCount)

#c
plt.bar(xvar, counts, tick_label=xvar, align='edge', width=.05)
plt.title('Gaussian Distribution PMF')
plt.ylabel('Probability')
plt.xlabel('Random Variable')

expected = []
xvalues = np.arange(-5, 5)
for xvalue in range(-5, 5):
    answer = (1 / (math.sqrt(2 * math.pi))) * math.exp(- (xvalue**2) / 2)
    expected.append(answer)
plt.figure()
plt.plot(xvalues, expected)
plt.title('Expected Gaussian Distribution')
plt.ylabel('Y')
plt.xlabel('X')
```

```
Out[140]: Text(0.5, 0, 'X')
```



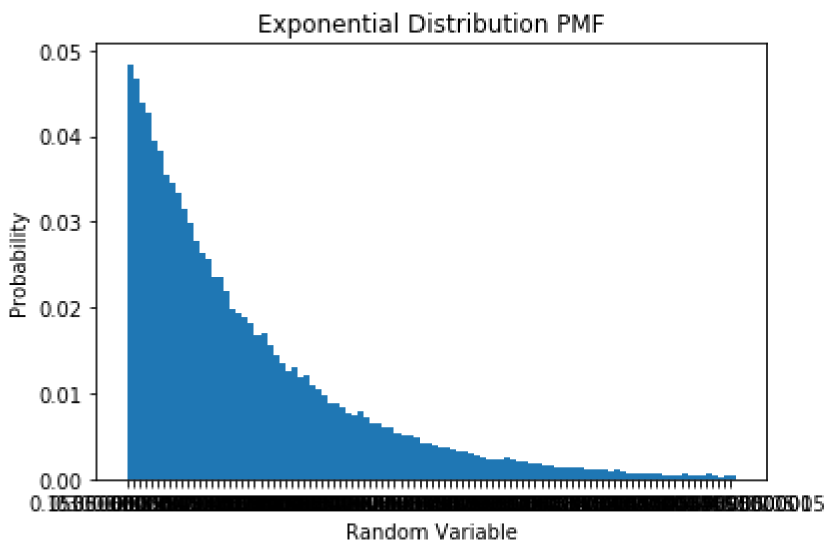


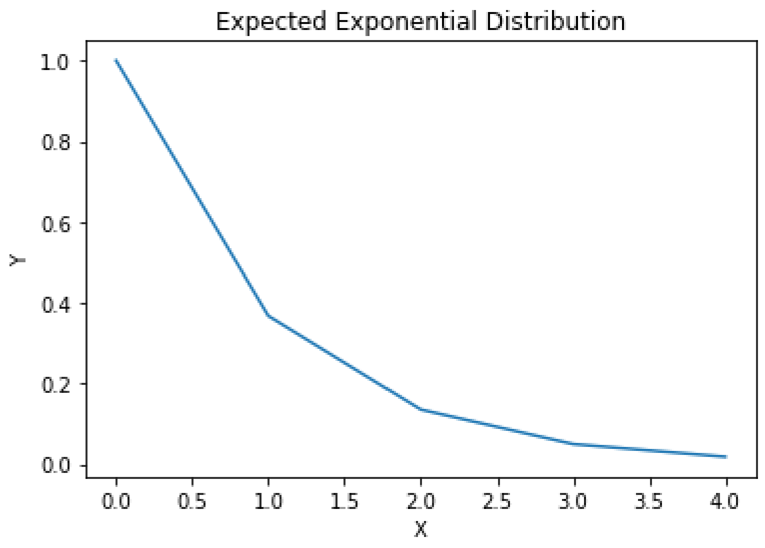
It looks roughly the same.

C) Exponential Distribution

In [141]:
```
values = []
counts = []

#a
for i in range(100000):
    value = random.expovariate(1)
    values.append(value)
#b
xvar = np.arange(0, 5, 0.05)
for j in range(0, 100):
    count = 0
    for k in (values):
        if (k >= (j/20) and k <= ((j/20) + 0.05)):
            count += 1
    trueCount = count/100000
    counts.append(trueCount)

#c
plt.bar(xvar, counts, tick_label=xvar, align='edge', width=.05)
plt.title('Exponential Distribution PMF')
plt.ylabel('Probability')
plt.xlabel('Random Variable')

expected = []
xvalues = np.arange(0, 5, 1)
for xvalue in range(0, 5):
    answer = 1 * math.exp(-1 * 1 * xvalue)
    expected.append(answer)
plt.figure()
plt.plot(xvalues, expected)
plt.title('Expected Exponential Distribution')
plt.ylabel('Y')
plt.xlabel('X')
```

```
Out[141]: Text(0.5, 0, 'X')
```





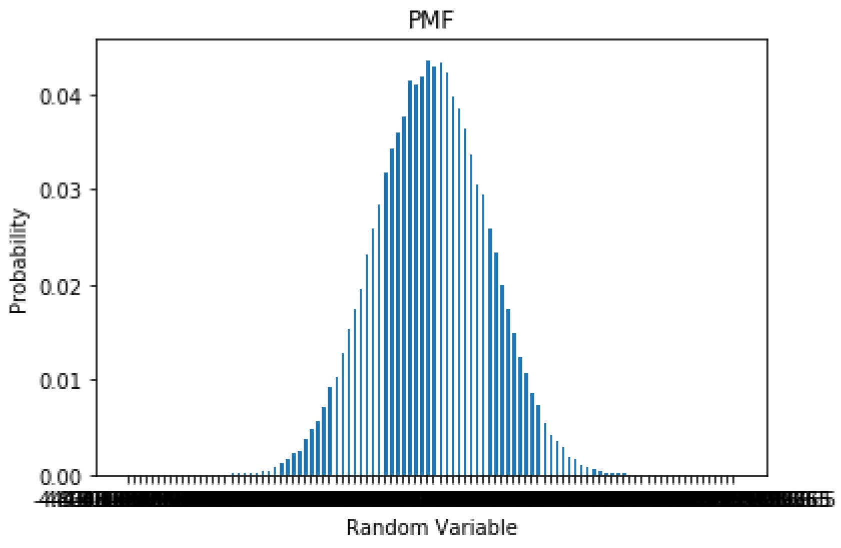
The plot of the expected exponential distribution and the Gaussian Distribution PMF look the same.

1.2 Sum of the Independent Random Variables

A)

In [142]:
```
def generator(a, b):
    value = random.uniform(a,b)
    return value
Y = []

for i in range(100000):
    sum = 0
    for j in range(10):
        answer = generator(-0.5, 0.5)
        sum += answer
    Y.append(sum)

counts = []
for k in range(-50, 50):
    count = 0
    for l in (Y):
        if (l >= (k/10) and l <= ((k/10) + 0.1)):
            count += 1
    trueCount = count/100000
    counts.append(trueCount)

xvar = np.arange(-5, 5, 0.1)
plt.bar(xvar, counts, tick_label=xvar, align='edge', width=.05)
plt.title('PMF')
plt.ylabel('Probability')
plt.xlabel('Random Variable')
```

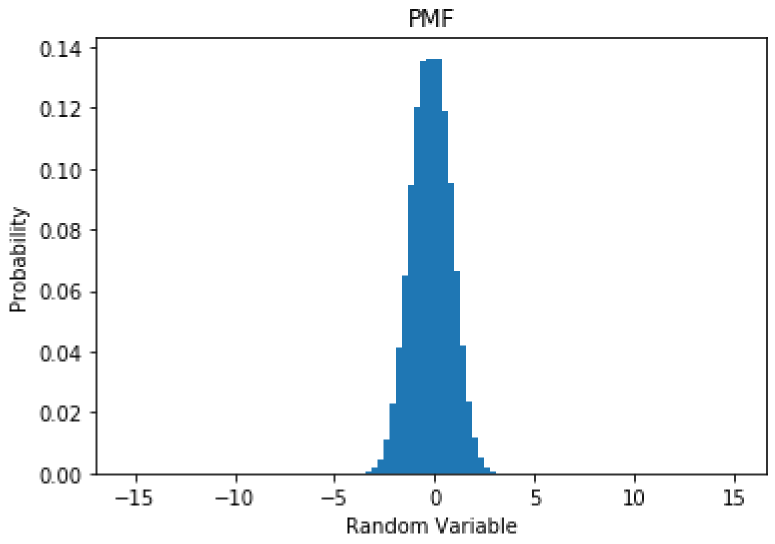
Out[142]:
```
Text(0.5, 0, 'Random Variable')
```



B)

```
In [145]: def generator(a, b):
              value = random.uniform(a,b)
              return value
          Y = []

          for i in range(100000):
              sum = 0
              for j in range(100):
                  answer = generator(-0.15, 0.15)
                  sum += answer
              Y.append(sum)

          counts = []
          for k in np.arange(-15, 15, 0.3):
              count = 0
              for l in (Y):
                  if (l >= k and l <= k + 0.3):
                      count += 1
              trueCount = count/100000
              counts.append(trueCount)

          xvar = np.arange(-15, 15, 0.3)
          plt.bar(xvar, counts)
          plt.title('PMF')
          plt.ylabel('Probability')
          plt.xlabel('Random Variable')
```

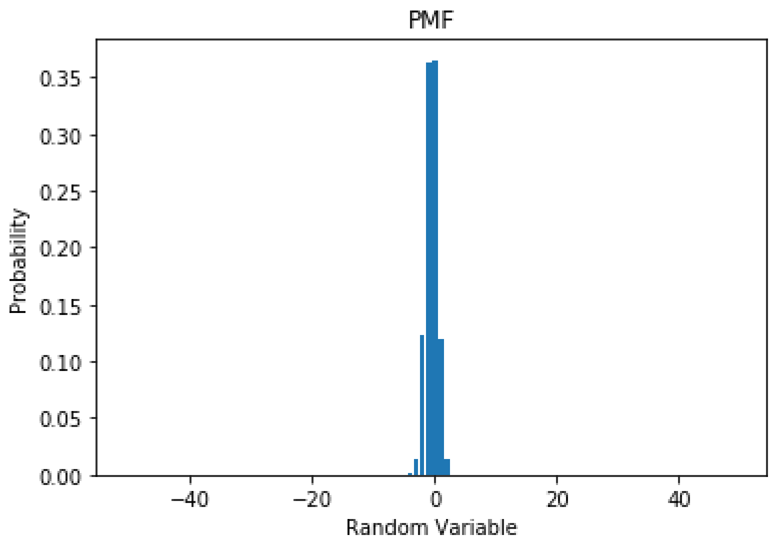
```
Out[145]: Text(0.5, 0, 'Random Variable')
```



C)

In [146]:
```
def generator(a, b):
    value = random.uniform(a,b)
    return value
Y = []

for i in range(100000):
    sum = 0
    for j in range(1000):
        answer = generator(-0.05, 0.05)
        sum += answer
    Y.append(sum)

counts = []
for k in range(-50, 50):
    count = 0
    for l in (Y):
        if (l >= k and l <= k + 1):
            count += 1
    trueCount = count/100000
    counts.append(trueCount)

xvar = np.arange(-50, 50, 1)
plt.bar(xvar, counts)
plt.title('PMF')
plt.ylabel('Probability')
plt.xlabel('Random Variable')
```

Out[146]:
```
Text(0.5, 0, 'Random Variable')
```



As you increase m, it looks like the Gaussian distribution gets narrower and narrower.

D)

2.1 Height-Weight-Temperature Dataset

A) I think X and Y are not independent, because taller people are more likely to weigh more it seems like. Height and Temperature of the room should be independent because I don't think your height changes with the temperature of the room. Same with weight and temperature of the room.

B)

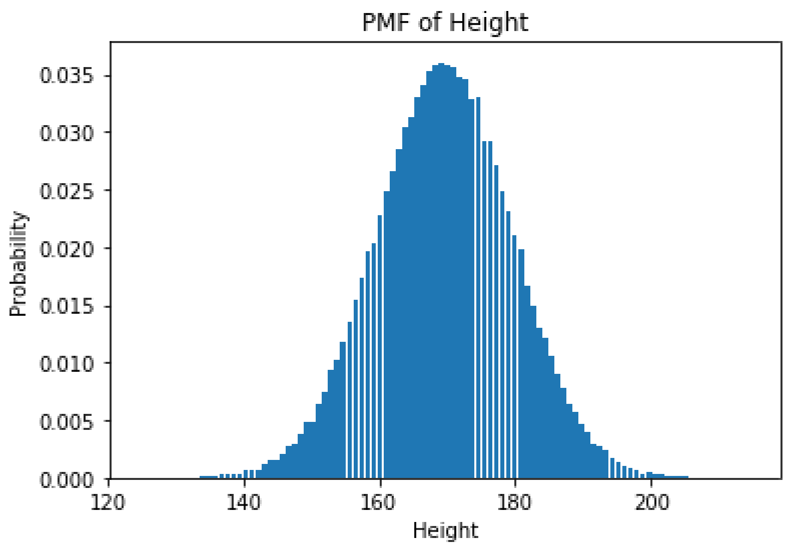
In [76]:
```
X = []
Y = []
Z = []

with open ("HeightWeightTemperatureData.csv") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        X.append(float(row[1]))
        Y.append(float(row[2]))
        Z.append(float(row[3]))
print("X ranges from: ", min(X), max(X))
print("Y ranges from: ", min(Y), max(Y))
print("Z ranges from: ", min(Z), max(Z))
```

```
X ranges from:  126.938138408 213.699805507
Y ranges from:  111.692000442 226.359271665
Z ranges from:  63.8887826264 80.9998353571
```

C)

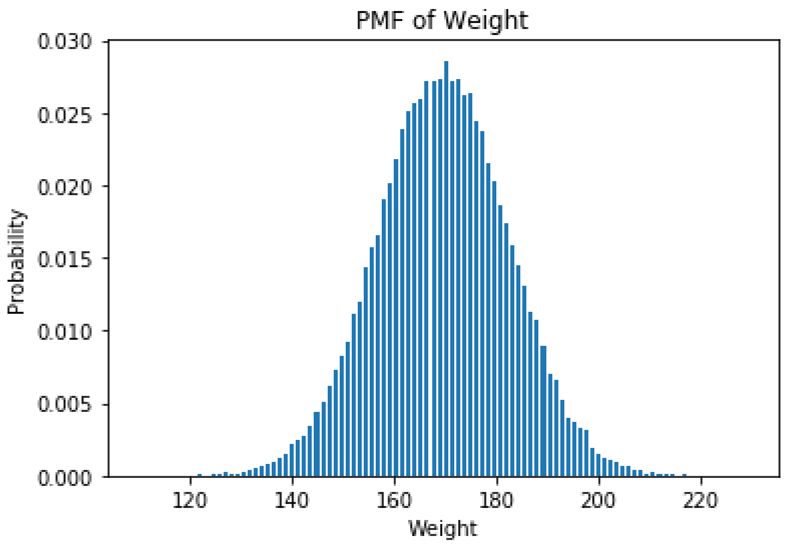
In [84]:
```
Xcount = []

for i in np.arange(125, 215, 0.9):
    count = 0
    for j in (X):
        if (j >= i and j <= i + 0.9):
            count += 1
    Xcount.append(count / 100000)
    count = 0
xvar = np.arange(125, 215, 0.9)

plt.bar(xvar, Xcount)
plt.title('PMF of Height')
plt.ylabel('Probability')
plt.xlabel('Height')
```

Out[84]:
```
Text(0.5, 0, 'Height')
```



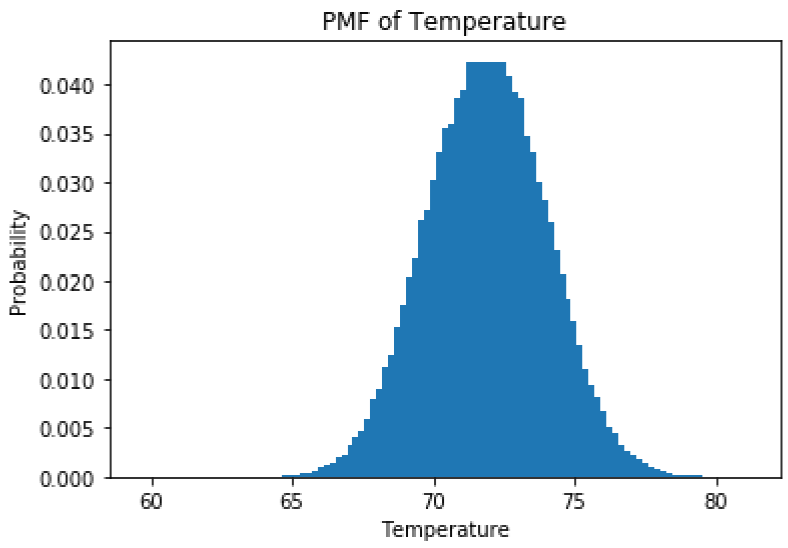
In [85]:
```
Ycount = []
for i in np.arange(110, 230, 1.2):
    count = 0
    for j in (Y):
        if (j >= i and j <= i + 0.9):
            count += 1
    Ycount.append(count / 100000)
    count = 0

xvar = np.arange(110, 230, 1.2)
plt.bar(xvar, Ycount)
plt.title('PMF of Weight')
plt.ylabel('Probability')
plt.xlabel('Weight')
```

Out[85]:
```
Text(0.5, 0, 'Weight')
```



In [86]:
```
Zcount = []
for i in np.arange(60, 81, .21):
    count = 0
    for j in (Z):
        if (j >= i and j <= i + 0.21):
            count += 1
    Zcount.append(count / 100000)
    count = 0

xvar = np.arange(60, 81, .21)
plt.bar(xvar, Zcount)
plt.title('PMF of Temperature')
plt.ylabel('Probability')
plt.xlabel('Temperature')
```

Out[86]:
```
Text(0.5, 0, 'Temperature')
```



They all look like Gaussian Distributions.

D)

In [101]:
```python
expectedHeight = np.mean(X)
expectedWeight = np.mean(Y)
expectedTemp = np.mean(Z)

print('The expected height is ', expectedHeight)
print('The expected weight is ', expectedWeight)
print('The expected temperature is ', expectedTemp)

varHeight = np.var(X)
varWeight = np.var(Y)
varTemp = np.var(Z)

print('The variance of height is ', varHeight)
print('The variance of weight is ', varWeight)
print('The variance of temperature is ', varTemp)
```

```
The expected height is  170.0306814543779
The expected weight is  169.9990103829906
The expected temperature is  71.99673178197486
The variance of height is  100.20179063301768
The variance of weight is  164.5538672757864
The variance of temperature is  4.0083627244625015
```

E)

In [98]:
```
XY = []
XZ = []
YZ = []
for i in range(100000):
    XY.append(X[i] * Y[i])
    XZ.append(X[i] * Z[i])
    YZ.append(Y[i] * Z[i])

expXY = np.mean(XY)
expXZ = np.mean(XZ)
expYZ = np.mean(YZ)

print('The expected XY (height and weight) is ', expXY)
print('The expected XZ (height and temperature) is ', expXZ)
print('The expected YZ (weight and temperature) is ', expYZ)

xy = expectedHeight * expectedWeight
xz = expectedHeight * expectedTemp
yz = expectedWeight * expectedTemp

print('Expected X (height) * expected Y (weight) is ', xy)
print('Expected X (height) * expected Z (temperature) is ', xz)
print('Expected Y (weight) * expected Z (temperature) is ', yz)
```

```
The expected XY (height and weight) is  29005.377257579126
The expected XZ (height and temperature) is  12241.644335104887
The expected YZ (weight and temperature) is  12239.226258864492
Expected X (height) * expected Y (weight) is  28905.04758198976
Expected X (height) * expected Z (temperature) is  12241.653367377254
Expected Y (weight) * expected Z (temperature) is  12239.373153745333
```

The expected value of XY is not equal to the expected value of X times the expected value of Y, meaning that the two are not independent which resonates with my prediction in part A. However, the expected value of XZ matches the expected value of X times the expected value of Z, and the expected value of YZ matches the expected value of Y times the expected value of Z, meaning that X and Z are independent and Y and Z are independent. I predicted these in part A as well.

F)

In [107]:
```
XplusY = []
XplusZ = []
YplusZ = []
for i in range(100000):
    XplusY.append(X[i] + Y[i])
    XplusZ.append(X[i] + Z[i])
    YplusZ.append(Y[i] + Z[i])

varXplusY = np.var(XplusY)
varXplusZ = np.var(XplusZ)
varYplusZ = np.var(YplusZ)

print('Var[X+Y] = ', varXplusY)
print('Var[X+Z] = ', varXplusZ)
print('Var[Y+Z] = ', varYplusZ)

print('Var[X] + Var[Y] = ', varHeight + varWeight)
print('Var[X] + Var[Z] = ', varHeight + varTemp)
print('Var[Y] + Var[Z] = ', varWeight + varTemp)
```

```
Var[X+Y] =  465.4150090875374
Var[X+Z] =  104.19208881275344
Var[Y+Z] =  168.2684402385671
Var[X] + Var[Y] =  264.7556579088041
Var[X] + Var[Z] =  104.21015335748018
Var[Y] + Var[Z] =  168.5622300002489
```

Yes, these answers conform to my prediction in A because it shows that X and Y are dependent, X and Z are independent, and Y and Z are independent.

2.2 Communication Channel

A)

In [115]:
```
pulses = []
for i in range(100000):
    prob = uniform(0, 1)
    if prob < 0.5:
        pulses.append(-1)
    else:
        pulses.append(1)
```

X should have a uniform distribution of -1 and 1 being roughly equal

```
In [116]: mean = np.mean(pulses)
          variance = np.var(pulses)

          print('mean of X is ', mean)
          print('variance of X is ', variance)
```

```
mean of X is  0.00144
variance of X is  0.9999979264000004
```

These match the theorical values because the average should be 0 because the distribution is split between -1 and 1. The variance also is 1 which makes sense because that is how far values are going to vary from the mean.

C)

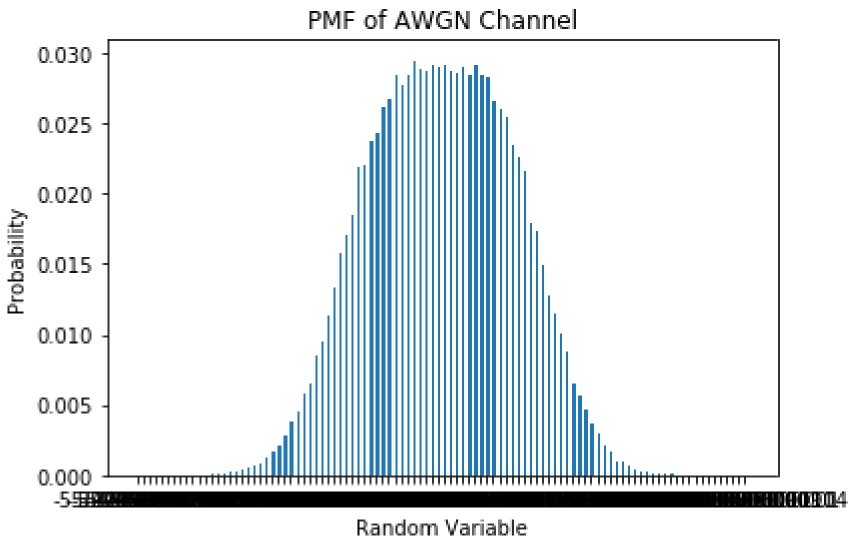
```
In [118]: awgn = []
          for i in pulses:
              awgn.append(pulses[i] + random.gauss(0,1))
```

D)

In [130]:
```
print(min(awgn), max(awgn))
dist = []
for i in np.arange(-6, 6, .12):
    count = 0
    for j in (awgn):
        if (j >= i  and j <= i + .12):
            count += 1
    dist.append(count / 100000)
xvar = np.arange(-6, 6, 0.12)
plt.bar(xvar, dist, tick_label=xvar, align='edge', width=.05)
plt.title('PMF of AWGN Channel')
plt.ylabel('Probability')
plt.xlabel('Random Variable')
```

```
-5.300614548757546 5.180764517696076
```

Out[130]:
```
Text(0.5, 0, 'Random Variable')
```



It looks like a Gaussian distribution

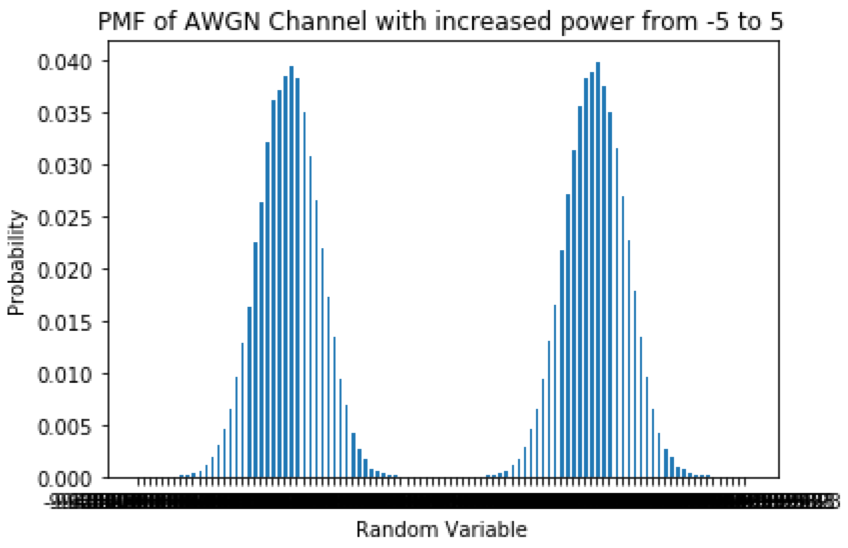
E)

In [153]:
```
correct = 0
incorrect = 0
receiver = []
for i in (awgn):
    if i > 0:
        correct += 1
        receiver.append(1)
    else:
        incorrect += 1
        receiver.append(-1)
errors = 0
for x in range(len(awgn)):
    if (pulses[x] != receiver[x]):
        errors += 1
print('errors = ', errors)
print('probability of error = ', errors/100000)
```

```
errors =  15894
probability of error =  0.15894
```

In [166]:
```
pulses = []
positive = 0
negative = 0
for i in range(100000):
    prob = uniform(0, 1)
    if prob < 0.5:
        pulses.append(-5)
        negative += 1
    else:
        pulses.append(5)
        positive += 1

mean = np.mean(pulses)
variance = np.var(pulses)

print('mean of X is ', mean)
print('variance of X is ', variance)

awgn = []
for i in pulses:
    awgn.append(pulses[i] + random.gauss(0,1))

print(min(awgn), max(awgn))

print(min(awgn), max(awgn))
dist = []
for i in np.arange(-10, 10, .2):
    count = 0
    for j in (awgn):
        if (j >= i  and j <= i + .2):
            count += 1
    dist.append(count / 100000)
xvar = np.arange(-10, 10, 0.2)
plt.bar(xvar, dist, tick_label=xvar, align='edge', width=.1)
plt.title('PMF of AWGN Channel with increased power from -5 to 5')
plt.ylabel('Probability')
plt.xlabel('Random Variable')

correct = 0
incorrect = 0
receiver = []
for i in (awgn):
    if i > 0:
        correct += 1
        receiver.append(5)
    else:
        incorrect += 1
        receiver.append(-5)
errors = 0
for x in range(len(awgn)):
    if (pulses[x] != receiver[x]):
        errors += 1
print('increased power errors = ', errors)
print('probability of error with increased power = ', errors/100000)
```

```
mean of X is  0.01
variance of X is  24.9999
-8.930245624511791 9.067932724404143
-8.930245624511791 9.067932724404143
increased power errors =  0
probability of error with increased power =  0.0
```



The probability of error decreased to 0 when the transmitter increased its power to -5 and 5.