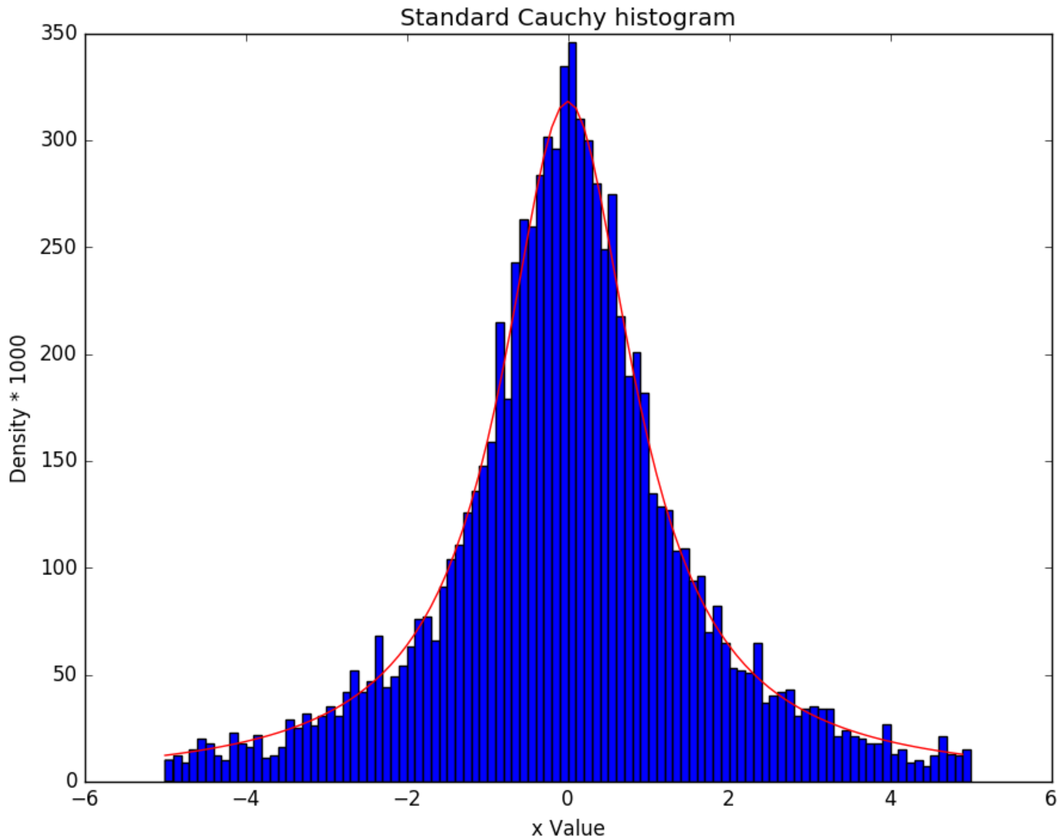
MA 573
HW 2
Alex Shoop

Question 1)b)

```
# importing the necessary packages
import numpy as np
from numpy import pi, tan
import matplotlib.pyplot as plt

###############################################################################
# CAUCHY

# density function
def f(x):
    return (1/pi) * 1/(1 + x**2)

# standard cauchy distribution
X = np.random.standard_cauchy(10000)

# generalized inverse
# there is no built in function for cotangent.
# so I used the identity: cot(x) = 1/tan(x)
Y = -(1./tan(pi*X))

# setting up variables to plot original density
Z = np.arange(-5, 5, 0.1)
W = f(Z)*1000

# standard cauchy distribution plot
plt.plot(Z, W, color = 'red')
# histogram sample plot
plt.hist(Y, range = (-5, 5), bins = 100, color = 'blue')
plt.title ("Standard Cauchy histogram")
plt.xlabel ("x Value")
plt.ylabel ("Density * 1000")

plt.show()
```

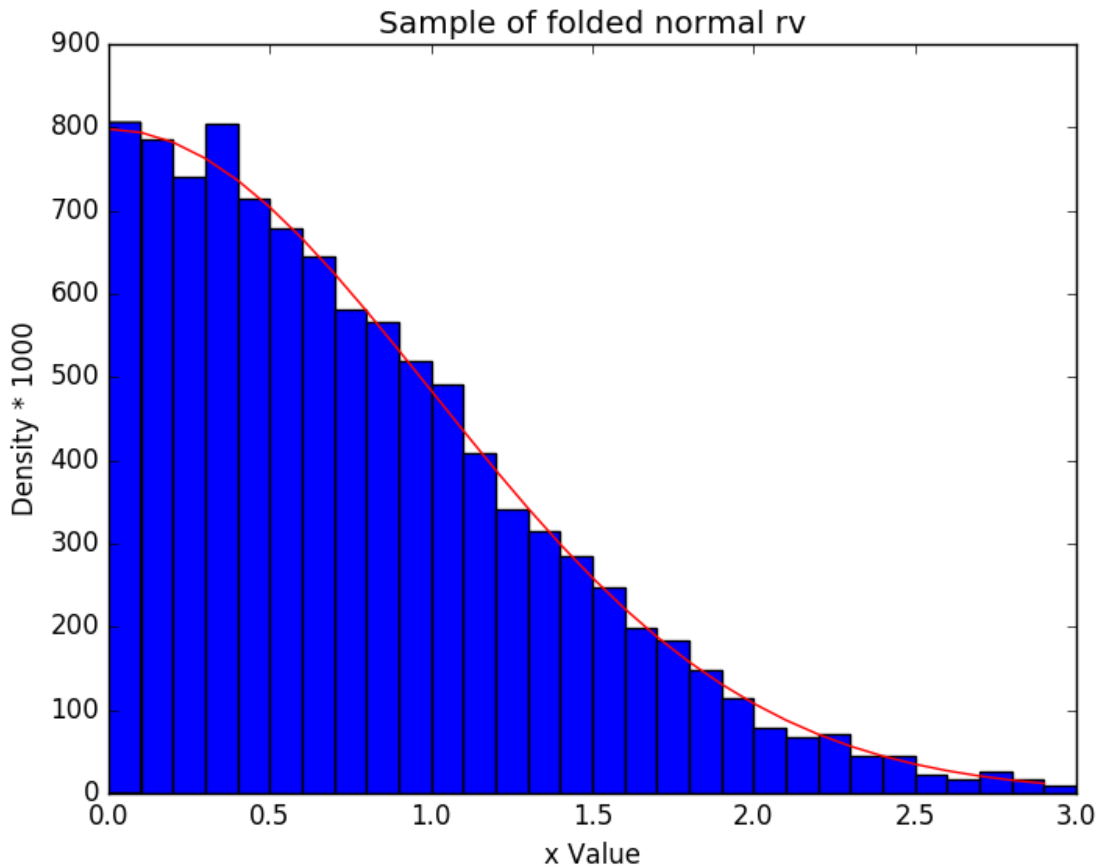
Figure:


Standard Cauchy histogram
Density * 1000
x Value
350
300
250
200
150
100
50
0
−6
−4
−2
0
2
4
6

Question 3)b)

```
# importing the necessary packages
import numpy as np
from numpy import pi, sqrt, exp
import matplotlib.pyplot as plt

################################################################################
# acceptance rejection method

# original density function
def f(x):
    return 2./sqrt(2.*pi) * exp(-(x**2) / 2.)
# density function of dominant exponential distribution
def g(x):
    return exp(-x)

# array that will be filled with accepted variables
a = np.zeros(0)

# constant c for method
c = 2./sqrt(2.*pi) * exp(0.5)

n = 0

plt.clf()

# want 10000 samples
while n < 10000:
    U1 = np.random.rand()
    U2 = np.random.rand()
    Y = -np.log(1 - U1)

    if c * g(Y) * U2 <= f(Y):
        n += 1
        # accept
        a = np.append(a, Y)



plt.hist(a, range = (0, 3), bins = 30, color = 'blue')

# setting up variables to plot original density
Z = np.arange(0, 3, 0.1)
W = f(Z)*1000

# original distribution plot
plt.plot(Z, W, color = 'red')
plt.title ("Sample of folded normal rv")
plt.xlabel ("x Value")
plt.ylabel ("Density * 1000")

plt.show()
```

Figure:


Sample of folded normal rv
Density * 1000
x Value
900
800
700
600
500
400
300
200
100
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Question 5)
Box Muller .py

```python
# importing the necessary packages
import numpy as np
from numpy import pi, sin, cos, sqrt, exp, log
import matplotlib.pyplot as plt
import timeit

###############################################################################
# Box Muller method

# starting timer
start = timeit.default_timer()

plt.clf()

# exact normal density function
def f(x):
    return 1/(sqrt(2 * pi)) * exp(-(x)**2 / 2 )

# initialize array of X rv's
X1 = np.zeros(0)
X2 = np.zeros(0)

# defining Box Muller function producting std normal variables
def BoxMull(X1, X2):
    n = 0
    while n < 50000:

        U1 = np.random.rand()
        U2 = np.random.rand()

        X1 = np.append(X1, cos(2*pi*U1) * sqrt(-2*log(U2)))
        X2 = np.append(X2, sin(2*pi*U1) * sqrt(-2*log(U2)))

        n += 1

    return X1, X2

# get function values
X1, X2 = BoxMull(X1, X2)

# setting up variables to plot original density
Z = np.arange(-3, 3, 0.1)
W = f(Z)*5000

# standard normal distribution plot
plt.plot(Z, W, color = 'red')

# histogram sample plot
plt.hist([X1, X2], range = (-3, 3), bins = 60, alpha = 0.5)
plt.title ("Box-Muller method sampling")
plt.xlabel ("x Value")
plt.ylabel ("Density * 5000")

plt.show()

```

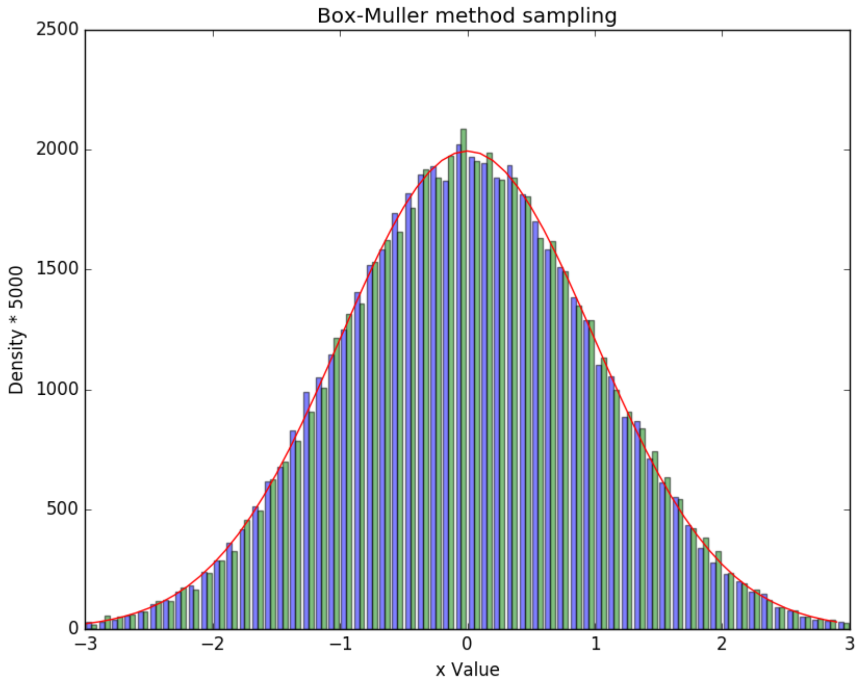
```
# stopping timer
stop = timeit.default_timer()

print "Box Muller method runtime = " , stop - start, " seconds"
```

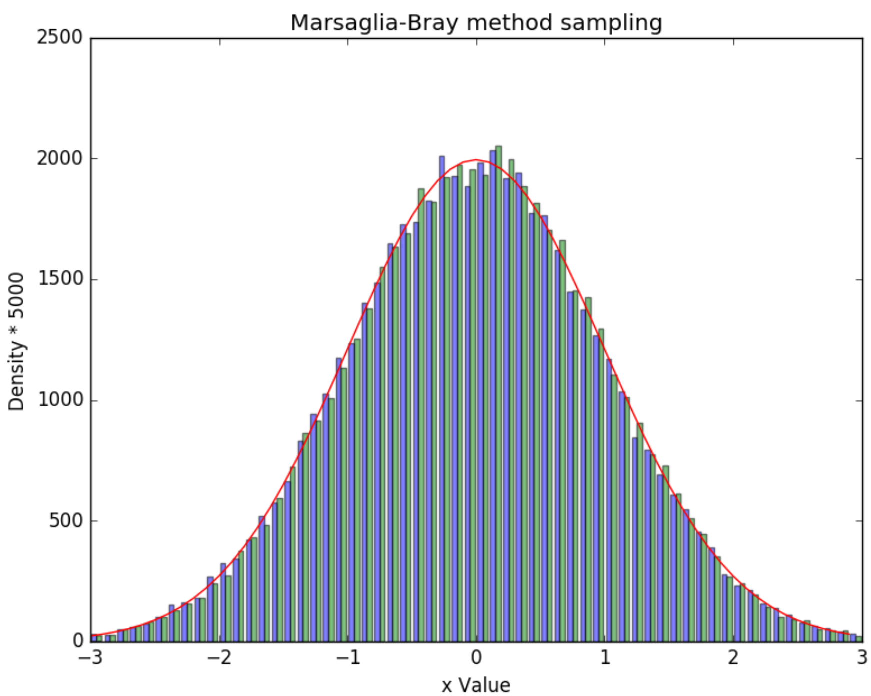
Marsaglia Bray .py

```
# importing the necessary packages
import numpy as np
from numpy import pi, sqrt, exp, log
import matplotlib.pyplot as plt
import timeit

##############################################################################
# Marsaglia-Bray method

# starting timer
start = timeit.default_timer()

plt.clf()

# exact normal density function
def f(x):
    return 1/(sqrt(2 * pi)) * exp(-(x)**2 / 2 )

# initialize array of accepted rv's
X1 = np.zeros(0)
X2 = np.zeros(0)

# defining Marsaglia Bray function producing std normal variables
def MarsBray(X1, X2):
    n = 0
    while n < 50000:
        # setting dummy initial value
        S = 2

        # this while loop makes sure that we are using an accepted S
        while S > 1:
            V1 = 2 * np.random.rand() - 1
            V2 = 2 * np.random.rand() - 1
            S = (V1 ** 2) + (V2 ** 2)

        X1 = np.append(X1, V1 * sqrt(-2 * log(S) / S))
        X2 = np.append(X2, V2 * sqrt(-2 * log(S) / S))

        n += 1

    return X1, X2

# get function values
X1, X2 = MarsBray(X1, X2)

# setting up variables to plot original density
Z = np.arange(-3, 3, 0.1)
W = f(Z)*5000

# standard normal distribution plot
plt.plot(Z, W, color = 'red')

```

```
# histogram sample plot
plt.hist([X1, X2], range = (-3, 3), bins = 60, alpha = 0.5)
plt.title ("Marsaglia-Bray method sampling")
plt.xlabel ("x Value")
plt.ylabel ("Density * 5000")

plt.show()

# stopping timer
stop = timeit.default_timer()

print "Marsaglia-Bray method runtime = " , stop - start, " seconds"
```

Figures:





Output of code:
Box Muller method runtime = 2.83222103119 seconds
Marsaglia-Bray method runtime = 2.75477385521 seconds

Question 6)

```
# importing the necessary packages
import numpy as np
from numpy import sqrt, exp, log
from scipy.stats import norm
import matplotlib.pyplot as plt

###############################################################################
# question 6)a)

# variables given
S_0 = 100.
B0 = 1
mu = 0.05
sigma = 0.2
r = 0.03

# European Put, using Black Scholes formula
K = 110.
T = 0.5

d1 = (log(S_0/K) + (r + (sigma**2)/2)*T) / (sigma * sqrt(T))
d2 = d1 - sigma*sqrt(T)

Price1 = K*exp(-r*T)*norm.cdf(-d2) - S_0*norm.cdf(-d1)

print("European Put option price (using Black Scholes formula) is:")
print(round(Price1,2))

###############################################################################
# question 6)b)

# defining the stock generation function
def generateStock(S_0, r, sigma, T):
    return S_0 * exp((r - (sigma**2)/2)*T + sigma * sqrt(T) * np.random.standard_normal
())

# defining payoff function, for this case it's regular European put
def payoff1(S, K):
    return max(0, K - S)

# setting number of simulation runs
runs = 10000
# initialize array that will have payoffs of option
payoffs = np.zeros(0)

# looping through
for i in xrange(runs):
    # generate future stock
    S_T = generateStock(S_0, r, sigma, T)

    # append to the payoffs list whatever the payoff is
    payoffs = np.append(payoffs,
                        payoff1(S_T, K))

Price2 = exp(-r*T) * sum(payoffs)/runs

print("European Put option price (using Monte Carlo integration) is: ")
print(round(Price2,2))
```

```

###############################################################################
# question 6)c)

# defining monte carlo integration formula, to make graphing easier
def MonteCarlo1(runs):
    # initialize array that will have payoffs of option
    payoffs = np.zeros(0)

    # looping through
    for i in xrange(runs):
        # generate future stock
        S_T = generateStock(S_0, r, sigma, T)

        # append to the payoffs list whatever the payoff is
        payoffs = np.append(payoffs,
                            payoff1(S_T, K))

    return exp(-r*T) * sum(payoffs)/runs

################################
plt.subplot(2, 2, 1)

n = 0

while n < 50:
    y = MonteCarlo1(n)

    plt.plot(n, y, color = 'red', marker = '.')

    n += 1

plt.plot(n, y, color = 'red', marker = '.',
         label = "samples n = 50");

plt.title ("Approximation of European Put")
plt.xlabel ("Trials")
plt.ylabel ("Value")
plt.legend()

################################
plt.subplot(2, 2, 2)

       n = 0

       while n < 100:
           y = MonteCarlo1(n)

           plt.plot(n, y, color = 'red', marker = '.')

           n += 1

       plt.plot(n, y, color = 'red', marker = '.',
                label = "samples n = 100");

       plt.title ("Approximation of European Put")
       plt.xlabel ("Trials")
       plt.ylabel ("Value")
       plt.legend()

```

```
        ################################
        plt.subplot(2, 2, 3)

        n = 0

        while n < 500:
            y = MonteCarlo1(n)

            plt.plot(n, y, color = 'red', marker = '.')

            n += 1

        plt.plot(n, y, color = 'red', marker = '.',
                 label = "samples n = 500");

        plt.title ("Approximation of European Put")
        plt.xlabel ("Trials")
        plt.ylabel ("Value")
        plt.legend()

        ################################
        plt.subplot(2, 2, 4)

        n = 0

        while n < 1000:
            y = MonteCarlo1(n)

            plt.plot(n, y, color = 'red', marker = '.')

            n += 1

        plt.plot(n, y, color = 'red', marker = '.',
                 label = "samples n = 1000");

        plt.title ("Approximation of European Put")
        plt.xlabel ("Trials")
        plt.ylabel ("Value")

        plt.legend()

        plt.show()

        ##############################################################################
        # question 6)d)

        # European asset-or-nothing digital call option
        # same maturity T = 0.5, same strike K = 110

        # defining payoff function
        def payoff2(S, K):
            if S > K:
                return S
            else:
                return 0

        # setting number of simulation runs
        runs = 10000
        # initialize array that will have payoffs of option
        payoffs = np.zeros(0)
```

```python

    # looping through
    for i in xrange(runs):
        # generate future stock
        S_T = generateStock(S_0, r, sigma, T)

        # append to the payoffs list whatever the payoff is
        payoffs = np.append(payoffs,
                            payoff2(S_T, K))

    Price3 = exp(-r*T) * sum(payoffs)/runs

    print("European asset-or-nothing digital call option price (using Monte Carlo integration) is: ")
    print(round(Price3,2))

    ##############################################################################
    # question 6)e)

    # European cubic put option
    # same maturity T = 0.5, same strike K = 110

    # defining payoff function
    def payoff3(S, K):
        return (max(0, K - S))**3

    # setting number of simulation runs
    runs = 10000
    # initialize array that will have payoffs of option
    payoffs = np.zeros(0)

    # looping through
    for i in xrange(runs):
        # generate future stock
        S_T = generateStock(S_0, r, sigma, T)

        # append to the payoffs list whatever the payoff is
        payoffs = np.append(payoffs,
                            payoff3(S_T, K))

    Price4 = exp(-r*T) * sum(payoffs)/runs

    print("European cubic put option price (using Monte Carlo integration) is: ")
    print(round(Price4,2))

    ##############################################################################
    # question 6)f)

    # European gap call option
    # same maturity T = 0.5, same strike K = 110

    # L exercise level
    L = 105

    # defining payoff function
    def payoff4(S, K):
        if S > K:
            return max(0, S - L)
        else:
            return 0
```

```
    # setting number of simulation runs
    runs = 10000
    # initialize array that will have payoffs of option
    payoffs = np.zeros(0)

    # looping through
    for i in xrange(runs):
        # generate future stock
        S_T = generateStock(S_0, r, sigma, T)

        # append to the payoffs list whatever the payoff is
        payoffs = np.append(payoffs,
                            payoff4(S_T, K))

    Price5 = exp(-r*T) * sum(payoffs)/runs

    print("European gap call option price (using Monte Carlo integration) is: ")
    print(round(Price5,2))

    ##############################################################################
    # question 6)g)

    # European exponential put option
    # same maturity T = 0.5, same strike K = 110

    # defining payoff function
    def payoff5(S, K):
        return exp(max(0, K - S))

    # setting number of simulation runs
    runs = 10000
    # initialize array that will have payoffs of option
    payoffs = np.zeros(0)

    # looping through
    for i in xrange(runs):
        # generate future stock
        S_T = generateStock(S_0, r, sigma, T)

        # append to the payoffs list whatever the payoff is
        payoffs = np.append(payoffs,
                            payoff5(S_T, K))

    Price6 = exp(-r*T) * sum(payoffs)/runs

    print("European exponential put option price (using Monte Carlo integration) is: ")
    print(round(Price6,2))
```

Output of code:

European Put option price (using Black Scholes formula) is:
10.97
European Put option price (using Monte Carlo integration) is:
11.03
European asset-or-nothing digital call option price (using Monte Carlo integration) is:

31.27
European cubic put option price (using Monte Carlo integration) is:
5809.78
European gap call option price (using Monte Carlo integration) is:
3.92
European exponential put option price (using Monte Carlo integration) is:
4.1053809922e+19

Figure: