Computer Assignment (CA) No. 8: Central Limit Theorem

Tyler Berezowsky

March 15, 2015

1 Problem Statement

The goal of this assignment is to demonstrate an application of the Central Limit Theorem. The tasks to be accomplished are:

1. Generate a sum of uniformly distributed mutually independent random variables:

$$S_n = X_1 + X_2 + \cdots + X_n \tag{1.1}$$

   Write this as a function in MATLAB with arguments that include the number of random variables (n), the number of total samples generated (N), and the range of the uniform random number generator (e.g., min=-1, max=1).

2. In the main part of your program, write a loop for n=1,100, and call this function for N=10,000 with a range of [-1,1]. For each iteration, compute the mean and variance of the output sequence, Sn, and plot the RMS error between a Gaussian fit of this distribution and the actual distribution (I hope you are using your code from a previous homework assignment in a function!).

3. Plot the RMS error as a function of the value of n. Also display the actual distribution and overlay its Gaussian fit for n=1, n=10 and n=100.

4. Consider the following technique for generating a Gaussian distribution from a uniform random number generator:
   `http://en.wikipedia.org/wiki/Box\E2\80\93Muller\_transform`.
   Generate N=10,000 random numbers using this technique, estimate a pdf, and compare the result using the RMS error to (2) for n=10, N=10,000, range=[0,1]. Time the code in both cases using MATLABs built-in timing tools. Which technique gives the better fit? Which technique is faster? How low can you set n to get comparable performance in both time and RMS error?

Discuss how (1)-(3) demonstrates the Central Limit Theorem.

2 Approach and Results

2.1 Tasks 1 to 3

A function `uniformsum(n, N, a, b)` was constructed which generates n random variables with a length of N and limits equal to $[b, a]$. This function was called with $n = [1, 100]$ and $N = 10000$, and for each value of n the PDF calculated from the resulting vector S_n The normal distribution was also fitted per n. The RMS error, defined below, was calculated for each value of n and plotted as a function of n where f_1 is the fitted normal distribution and f_2 is the PDF of S_n. This can be seen in figure 2.1.

$$RMS_{error} = \sqrt{MSE} = \sqrt{E[(f_1 - f_2)^2]} \tag{2.1}$$

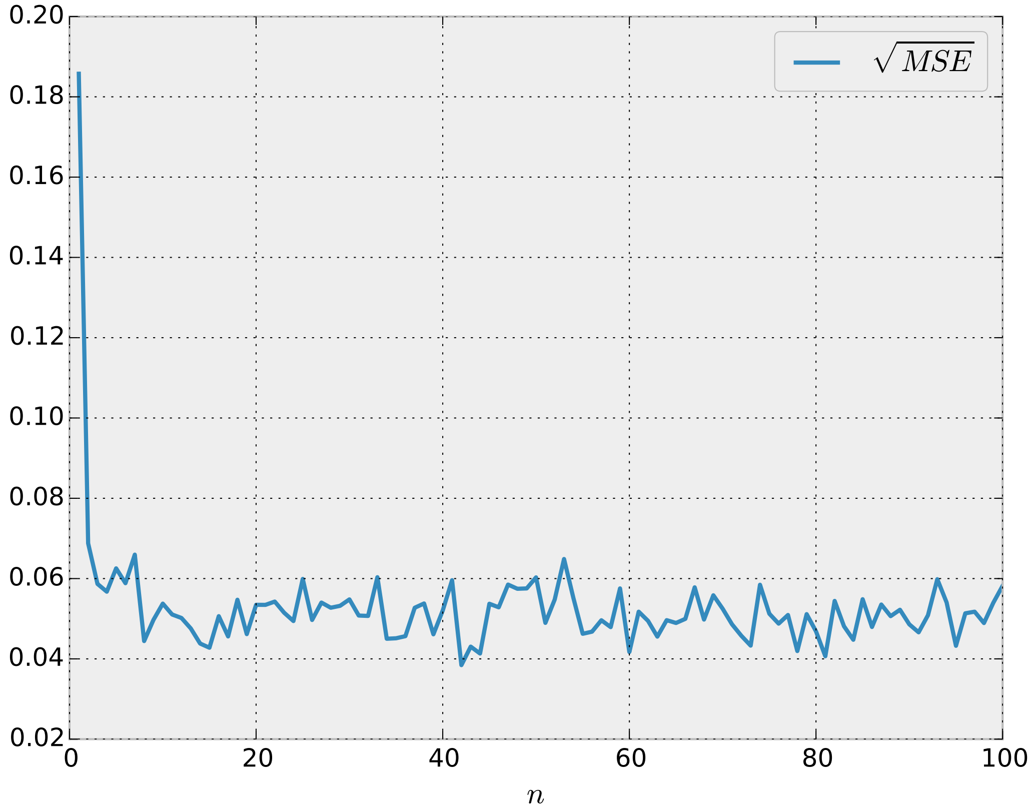

Figure 2.1: $\sqrt{MSE})$ as a function of n

Interestingly, the $\sqrt{MSE}$ appears to converge between 0.06 and 0.04 after only ten iterations which infers diminishing returns after $n = 10$. Plots of the PDT_{S_n} are displayed in figure 2.2 with $n = \{1, 10, 100\}$. As previously stated, n is equal to number of uniformly distributed random variables of length N being summed to generate S_n.

When $n = 1$, S_n represents a single random variable, and is thus a uniform distribution. This is illustrated in the top subplot. When $n = 10$, S_n represents the sum of 10 random variables. As the middle subplot illustrates, the distribution of S_n now approaches a normal distribution. When $n = 100$, S_n represents the sum of 100 random variables. The PDF of S_n is displayed in the bottom subplot, and as predicted by error plot in figure 2.1, the distribution appears identical in approximation to the fitted normal distribution of S_n.

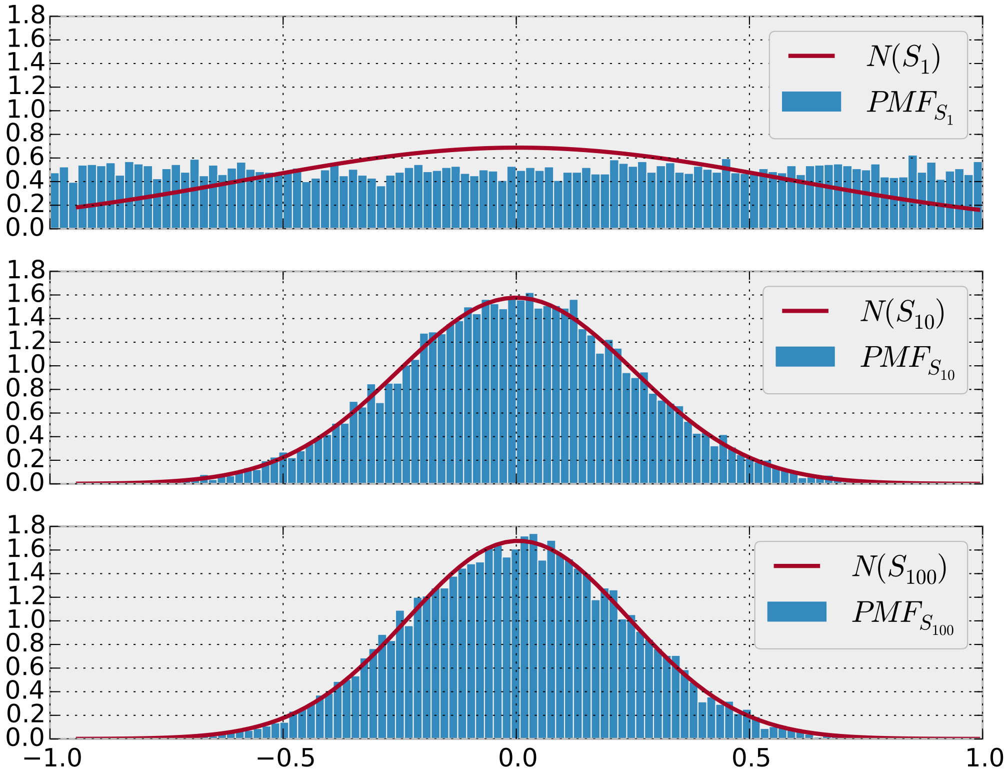

Figure 2.2: Histogram of S_n and fitted normal distribution

2.2 Task 4

The Box-Muller method was utilized to generate a normally distributed random variable and timed against the calculation for S_n with $n = [1, 10]$. The Box-Muller method is described below where Z_0 and Z_1 are normally distributed random variables, and U_1 and U_2 are independent uniformly distributed random variables:

$$Z_0 = R\cos(\theta) = \sqrt{-2\ln U_1}\cos(2\pi U_2) \tag{2.2}$$

$$Z_1 = R\sin(\theta) = \sqrt{-2\ln U_1}\sin(2\pi U_2) \tag{2.3}$$

The Box-Muller method and calculation for S_n was trialed 10 times for each value of n. The result was averaged per n. The RMS error between the fitted normal distribution, S_n, and the Box-Muller method was also determined. The results are illustrated in figure 2.3.

The top subplot illustrates the time for each function given n. As expected the time to complete the calculation for S_n increases with n. The Box-Muller method is constant as n is irrelevant for the calculation. The functions intersect when $n \approx = 2$. The bottom subplot illustrates the error between the normal distribution and the PDF of the Box-Muller method and S_n. As previously described, the RMS between S_n and the normal decreases with n. The Box-Muller method is constant because n is irrelevant in the calculation. The error of the Box-Muller method is consistently lower than that of S_n as seen for $n = [1, 10]$. Figure 2.1 displays that the error converges to 0.06 and 0.04 as n increases beyond 10, therefore it can be expected that the Box-Muller will be more accurate despite increasing values of n. Further trials would have to be attempted to confirm the previous assumption, but if error for S_n does decrease with n, the time required would be magnitudes larger than that of the Box-Muller method.

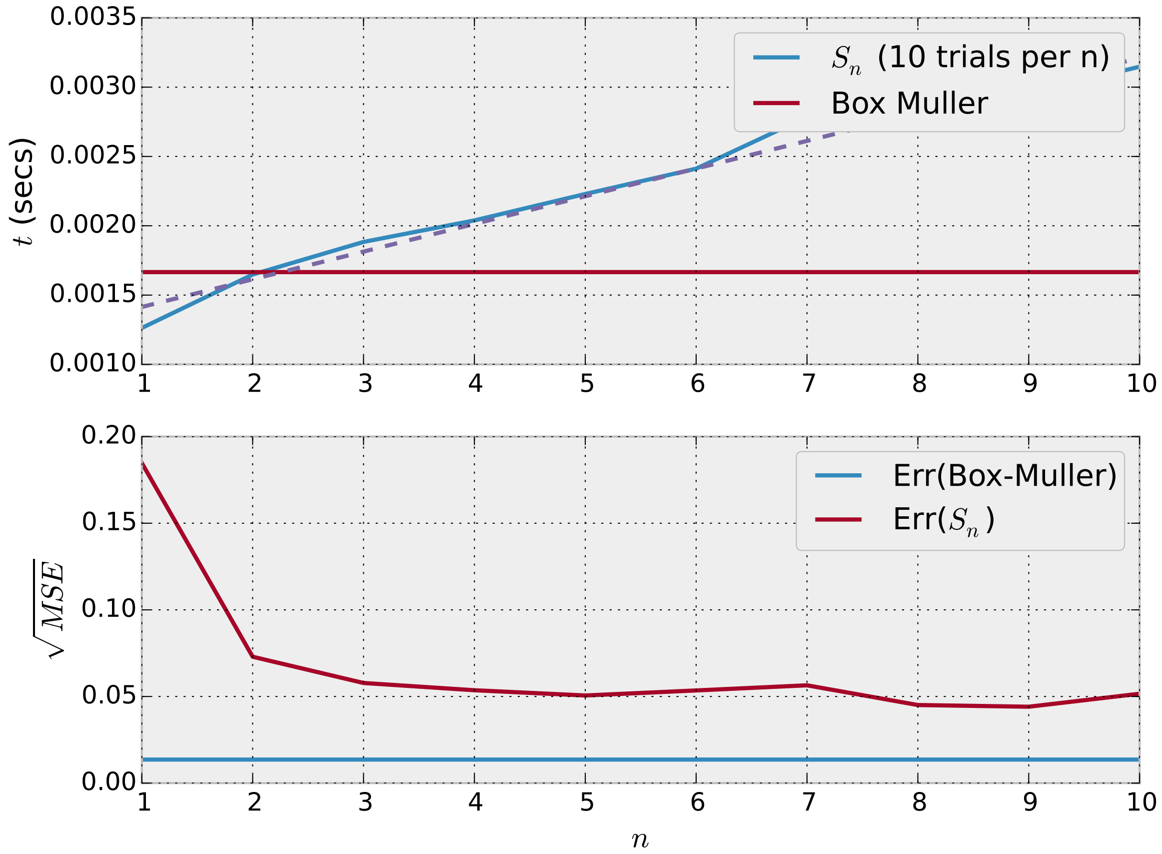

Figure 2.3: Comparison of calculation for S_n and Box-Muller method for the generation of a random normally distributed vector.

3 Source Code

```
import numpy as np
import scipy.stats as stats
from numpy.random import rand
import matplotlib.pyplot as plt
import time
plt.style.use('bmh')


def uniformSum(n, N, a, b):
    X_n = (b - a)*rand(N, n) + a  # generate uniform RVs between b and a
    S_n = np.sum(X_n, axis=1)  # sum
    S_n = S_n/max(abs(S_n))
    return S_n


def normal(x, mu, variance):
    pdf = (1./np.sqrt(2*np.pi*variance))*np.exp((-(x-mu)**2)/(2*variance))
    return pdf


```

```
def findcenterbins(bins):
    centerBins = np.zeros(len(bins) - 1)
    for index in range(0, len(bins) - 1):
        centerBins[index] = np.mean([bins[index + 1], bins[index]])
    return centerBins


def MSE(f1, f2):
    mse = np.mean((f1-f2)**2)
    return mse


# PARAMETERS
N = 10000
a = -1
b = 1
BINS = 100
NSUMS = 100

# MEMORY ALLOCATION
S_n = np.zeros(N)
error = np.zeros(NSUMS)
for n in range(1, NSUMS+1):
    S_n = uniformSum(n, N, a, b)                          # generate sum
    sVar = np.var(S_n)                                    # compute variance
    sMu = np.mean(S_n)                                    # compute mean
    sPDF, bins = np.histogram(S_n, bins=BINS, normed=True)  # compute PDF
    centerbins = findcenterbins(bins)           # compute the center of bins
    sNormal = normal(centerbins, sMu, sVar)     # compute norm dis via metrics

    error[n-1] = np.sqrt(MSE(sPDF, sNormal))

# PLOTS
n = range(1, NSUMS+1)
eplt = plt.figure(1)
ax = eplt.add_subplot(111)
ax.plot(n, error, label=r'$\sqrt{MSE}$')
ax.legend()
ax.set_xlabel(r'$n$')
plt.show()

# plot n = {1, 10, 100}
ns = [1, 10, 100]
hplt, axs = plt.subplots(3, 1, sharex=True, sharey=True)
for n, ax in zip(ns, axs):
    S_n = uniformSum(n, N, a, b)
    sVar = np.var(S_n)                                    # compute variance
    sMu = np.mean(S_n)                                    # compute mean
    centerbins = findcenterbins(bins)           # compute the center of bins
    sNormal = normal(centerbins, sMu, sVar)     # compute norm dis via metrics

```

```
    ax.hist(S_n, bins=BINS, normed=True, alpha=0.5, label=r'$PMF_{S_{%d}}$' % n)
    ax.plot(centerbins, sNormal, label=r'$N(S_{%d})$' % n)
    ax.legend()


# BOX-Muller Transform
def boxmuller(N):
    a = 0; b = 1; n = 1
    U1 = (b - a)*rand(N, n) + a
    U2 = (b - a)*rand(N, n) + a  # generate uniform RVs between b and a
    R = np.sqrt(-2*np.log(U1))
    theta = 2*np.pi*U2
    Z1 = R*np.cos(theta)
    return Z1

# TIME TEST

N = 10000
ns = range(1, 11)
trials = range(0, 10)
start = zeros(10)
end = zeros(10)
tstart = zeros(10)
tend = zeros(10)

for n in ns:
    for trial in trials:
        tstart[trial] = time.time()
        uniformSum(n, N, 0, 1)
        tend[trial] = time.time()
    start[n-1] = np.mean(tstart)
    end[n-1] = np.mean(tend)
utime = end - start

S_n = []
error2 = []
for trial in trials:
        start[trial] = time.time()
        S_n = boxmuller(N)
        end[trial] = time.time()
print S_n
sVar = np.var(S_n)                                    # compute variance
sMu = np.mean(S_n)                                    # compute mean
sPDF, bins = np.histogram(S_n, bins=BINS, normed=True)  # compute PDF
centerbins = findcenterbins(bins)           # compute the center of bins
sNormal = normal(centerbins, sMu, sVar)     # compute norm dis via metrics
error2 = np.sqrt(MSE(sPDF, sNormal))

btime = np.mean(end - start)

tplot = plt.figure(3)
```

```
ax1 = tplot.add_subplot(211)
ax2 = tplot.add_subplot(212)
ax1.plot(ns, utime, label='$S_n$ (10 trials per n)')
ax1.plot(ns, [btime]*10, label='Box Muller')
ax2.plot(ns, [error2]*len(ns), label=r'Err(Box-Muller)')
n = np.arange(1, 11)
ax2.plot(n, error[0:len(n)], label='Err($S_n$)')
z = np.polyfit(ns, utime, 1)
p = np.poly1d(z)
ax1.plot(ns, p(ns), '--')
ax1.legend()
ax2.set_xlabel(r'$n$')
ax1.set_ylabel(r'$t$ (secs)')
ax2.set_ylabel(r'$\sqrt{MSE}$')

ax2.legend()
tplot.tight_layout()
```

4 CONCLUSIONS

Due to the Central Limit Theorem, as n is increased the sum of n uniformly distributed random variables approaches a vector which is normally distributed. Central Limit Theorem: The arithmetic mean of a sufficient number of independent random variables will be *approximately* normally distributed given the random variables are identically distributed.[1]

This theorem can be utilized to generate a random normally distributed variable, but as displayed takes considerable effort in comparison to other methods such as the Box-Muller method. Further exploration is required to understand why the RMS error appears to converge as n increases. The proof of the Central Limit Theorem states given any random variable Y with $E(Y) = 0$ and $var(Y) = 1$ summed n times as $\lim_{n\to\infty}$ the approximation approaches $e^{-t/2}$ which would be equal to $N(0,1)$. Therefore, error should decrease as n increases.

[1] http://en.wikipedia.org/wiki/Central_limit_theorem