Homework #6 Solutions

Please drop the assignment in the HOMEWORK In box in the EE 278 drawer of the class file cabinet on the 2nd floor of the Packard Building.
Please work on problems 1-2 with your group, and submit problems 3-6 individually.

1. Neural 4-ary QAM. Quadrature Amplitude Modulation (QAM) is a modulation scheme that conveys two analog message signals, or two digital bit streams, by changing the amplitudes of two carrier waves. QAM is used extensively as a modulation scheme for digital telecommunication systems, such as in 802.11 Wi-Fi standards. In this problem, we consider a simplified case where we want to transmit a signal $H \in \{0, 1, 2, 3\}$, $p_H(i) = \frac{1}{4}$, $i = 0, 1, 2, 3$. The communication scheme is illustrated in figure 1.

When $H = i$, the transmitter transmits $s_i \in \mathbb{R}^2$. The values of $s_i$’s are shown in figure 2. The noise is $Z \sim \mathcal{N}(0, \sigma^2 I_n)$ and the observation, when $H = i$, is $Y = s_i + Z$.

a. Find the maximum likelihood decision rule and calculate its probability of error.
You can express your answer in terms of the $Q$ function.

We want to find an estimate $\hat{H}$ using a neural network and compare its performance with the maximum likelihood decoder. The neural network takes input $Y$ and outputs a probability vector $p \in \mathbb{R}^4$, where $p_i$ represents the probability that $\hat{H} = i$. This is similar to the MNIST classification problem. For more information, please review the TensorFlow tutorial slides and code on the course website. The starter code for this problem is given in neural_qam.py.
b. Write the function `generate_data` to generate the training data. For \( p \), use a one-hot representation where

\[
p_i = \begin{cases} 
1 & \text{if } H = i \\
0 & \text{otherwise.}
\end{cases}
\]

Assume \( d = 2 \) and \( \sigma^2 = 1 \).

c. Train a neural network decoder with 10,000 training samples. Use the softmax function at the output layer and the cross-entropy loss as the loss function. Tune the parameters, including but not limited to depth, number of neurons, initialization values, etc. What is the best set of parameters that you get?

d. Compare the empirical probability of error of the neural network with the probability of error of the maximum likelihood decoder computed in part a. on a separate test dataset of 1000 samples. How well does your neural network perform?

e. Now assume \( d = 3, \sigma^2 = 1 \). Train the neural network with 10,000 training samples and compare its empirical probability of error with the probability of error of the maximum likelihood decoder on a separate test dataset of 1000 samples.

**Solution** (30 points)
a. The ML decision rule is

\[
\hat{H}(y) = \arg \max_i f_{Y|H}(y | i)
\]

\[
= \arg \max_i \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left\{-\frac{||y - s_i||^2}{2\sigma^2}\right\}
\]

\[
= \arg \min_i ||y - s_i||^2
\]

\[
= \begin{cases}
0 & \text{if } y_1 \geq 0, y_2 \geq 0 \\
1 & \text{if } y_1 < 0, y_2 \geq 0 \\
2 & \text{if } y_1 < 0, y_2 < 0 \\
3 & \text{if } y_1 \geq 0, y_2 < 0,
\end{cases}
\]

which is the minimum-distance decision rule. The decoding region for \(s_0\) is the first quadrant, for \(s_1\) the second quadrant, etc.

To calculate the probability of error, notice that when \(H = 0\), the decoder makes the correct decision if \(\{Z_1 > -\frac{d}{2}\} \cap \{Z_2 \geq -\frac{d}{2}\}\). This is the intersection of independent events. Hence the probability of the intersection is the product of the probability of each event, i.e.

\[
P_c(0) = \left[\Pr\left\{Z_i \geq -\frac{d}{2}\right\}\right]^2 = Q^2\left(-\frac{d}{2\sigma}\right) = \left[1 - Q\left(\frac{d}{2\sigma}\right)\right]^2.
\]

By symmetry, for all \(i\), \(P_c(i) = P_c(0)\). Hence,

\[
P_e = P_e(0) = 1 - P_c(0) = 2Q\left(\frac{d}{2\sigma}\right) - Q^2\left(-\frac{d}{2\sigma}\right).
\]

b. def generate_data(n = 10000, d=2):

# onehot is the one-hot representation of H.
# For example, if H = 2, then onehot = [0,0,1,0].
h = np.random.randint(0,4,size=n)
onehot = np.zeros((n,4))
onehot[np.arange(n), h] = 1
x = np.zeros((n,2))
for i in range(n):
    if h[i] == 0:
        x[i, :] = [-d/2,-d/2]
    if h[i] == 1:
        x[i, :] = [-d/2,d/2]
    if h[i] == 2:
        x[i, :] = [d/2,-d/2]
    if h[i] == 3:
        x[i, :] = [d/2,d/2]
z = np.random.normal(0,1,size=(n,2))
y = x + z
return y, onehot

c. Here is the code snippet for the function construct_graph, which is also used for
part d.

def construct_graph(self):
    self.x = tf.placeholder(tf.float32, [None, 2])
    h = self.x
    for layer_idx in range(self.num_layer):
        with tf.variable_scope("layer{}").format(layer_idx):
            h = self.layer(h, self.num_neuron)
    with tf.variable_scope("proj_layer"):
        hidden_shape = h.get_shape()
        W = tf.get_variable("weights",
                            [hidden_shape[-1], 4],
                            initializer = tf.truncated_normal_initializer(stddev = 0.1))
        b = tf.get_variable("bias",
                            [4],
                            initializer = tf.truncated_normal_initializer(stddev = 0.1))
        y = tf.nn.softmax(tf.matmul(h, W) + b)

    self.prediction = y
    self.y_ = tf.placeholder(tf.float32, [None, 4])
    self.loss = tf.reduce_mean(-tf.reduce_sum(self.y_ * tf.log(y), reduction_indices=[1]))
    self.train_step = tf.train.GradientDescentOptimizer(0.5).minimize(self.loss)

Here is the code snippet for training the neural network.

data = generate_data(n = 10000,d=2)
tf.reset_default_graph()
my_model = model(num_layer = 3, num_neuron = 100)
my_model.construct_graph()
sess = tf.InteractiveSession()
my_model.train(sess, data)

d. Here is the code snippet for testing the neural network.

test_data = generate_data(n = 1000,d=2)
y = test_data[0]
H = test_data[1]
prediction = my_model.test(sess, y)
correct_prediction = tf.equal(tf.argmax(prediction,1), tf.argmax(H,1))
p_error = 1 - tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

The empirical probability of error of the neural network is 0.286, which is lower but comparable to the maximum likelihood decoder probability of error 0.292.

e. We use the same code as d = 2 except that we pass d = 3 as the argument to the generate_data function and we use only 1 hidden layer. The error do not differ much between 1-4 hidden layers. The empirical probability of error is 0.131, which is higher but comparable to the maximum likelihood decoder probability of error 0.129.

2. Vector neural estimator. Consider a channel with vector inputs $X$ and vector outputs $Y$. The channel is given in the illustration below in figure 3.

The input of the channel is a 6-dimensional vector $X \in \mathbb{R}^6$, and the output is $Y \in \mathbb{R}^5$. Each node performs an addition operation of the incoming nodes plus an iid noise. For example, $N_{11} = X_1 + X_2 + Z_{11}$ and $Y_1 = N_{21} + N_{22} + Z_{31}$. $X$ and $Z_{ij}$'s are all
Figure 3: Vector channel model

independent. In this exercise you will implement a neural network estimator for this vector channel.

a. Write a function to generate training data. Assume \( X \sim \mathcal{N}(0, I_{6 \times 6}) \) and i.i.d. noise \( Z_{ij} \sim \text{Exp}(0.1) \).

b. Train a neural network estimator which takes input \( Y \) and outputs an estimated \( \hat{X} \). Tune the parameters, including but not limited to depth, number of neurons, non-linearity, initialization and so on. What’s the best set of parameters you get?

c. Calculate the covariance matrix for random vector \((X_1, \cdots, X_6, Y_1, \cdots, Y_5)\). Find the optimal linear estimator.

d. Compare the neural network with the linear estimator. Report their root mean-square errors (RMSE).

**SOLUTION (20 points)**

a. Here is the code snippet to generate training data.

```python
import numpy as np

def generate_data(n = 10000):
    x = np.random.normal(0, 1, size=(n, 6))
    l1 = np.matrix([[1,0,0,0,0], [1,1,0,0,0], [0,1,1,0,0], [0,0,1,1,0], [0,0,0,1,1], [0,0,0,0,1]])
    z1 = np.random.exponential(0.1, size=(n, 5))
    n1 = np.dot(x,l1) + z1

    z2 = np.random.exponential(0.1, size=(n, 6))
    n2 = np.dot(n1,np.transpose(l1)) + z2
```

```
z3 = np.random.exponential(0.1, size=(n, 5))
y = np.dot(n2,l1) + z3

return y, x

b. Here is the code snippet for parts b., c., and d.

def generate_batch(data, batch_size = 100):
x, y = data
indices = np.random.choice(len(x), batch_size, replace=False)
return x[indices], y[indices]

class model:
def __init__(self, num_layer = 1, num_neuron = 100):
    self.num_layer = num_layer
    self.num_neuron = num_neuron
def layer(self, x, num_neuron):
inpu
#layer takes input x, with num_neuron many neurons. Relu is used for activation.
W = tf.get_variable("W", [input_shape[1], num_neuron])
b = tf.get_variable("b", [num_neuron])
y = tf.matmul(x,W) + b
return tf.nn.relu(y)
def construct_graph(self):
    self.x = tf.placeholder(tf.float32, [None, 5])
h = self.x
for layer_idx in range(self.num_layer):
    #add intermediate layers
    with tf.variable_scope("layer{0}".format(layer_idx)):
        h = self.layer(h, self.num_neuron)

    #add projection layer
    with tf.variable_scope("proj_layer"):
        hidden_shape = h.get_shape()
        W_end = tf.get_variable("W", [hidden_shape[1],6])
b_end = tf.get_variable("b", [6])
y = tf.matmul(h,W_end) + b_end

self.prediction = y
self.y_ = tf.placeholder(tf.float32, [None, 6])
self.loss = tf.reduce_mean(tf.square(y - self.y_))
self.train_step = tf.train.AdamOptimizer(1e-4).minimize(self.loss)
self.saver = tf.train.Saver()

def train(self, sess, data, val):
    train_error = np.empty([100])
    val_error = np.empty([100])
tf.global_variables_initializer().run()
if not os.path.isfile("./model.ckpt"):
    batch_size = 100
for i in range(10000):
    batch_xs, batch_ys = generate_batch(data)
    _, train_loss = sess.run([self.train_step, self.loss], feed_dict={self.x: batch_xs,
                                                                     self.y_: batch_ys})
    if i % 100 == 0:
        pred = sess.run(self.prediction, feed_dict={self.x: val[0]})
        val_loss = np.mean(np.square(pred - val[1]))
        train_error[i/100] = train_loss
        val_error[i/100] = val_loss
    save_path = self.saver.save(sess, "./model.ckpt")
return train_error, val_error

def test(self, sess, t):
    self.saver.restore(sess, "./model.ckpt")
    return sess.run(self.prediction, feed_dict={self.x: t[0]})

if __name__ == '__main__':
data = generate_data()
t = generate_data(n = 100)
my_model = model(num_layer = 4, num_neuron = 100)
my_model.construct_graph()
sess = tf.InteractiveSession()
train_error, val_error = my_model.train(sess, data, t)
y = my_model.test(sess, t)
Cxy = np.matrix([[2,1,0,0,0],[3,3,1,0,0],[1,3,3,1,0],[0,0,1,3,3],[0,0,0,1,2]])
Cy = np.matrix([[13.72,13.63,5.59,0.58,-0.36],[13.63,19.6,14.56,5.52,0.58],[5.59,14.56,19.6,14.56,5.59],[0.58,5.52,14.56,19.6,13.63],[0.58,5.52,14.56,19.6,13.63]])
W = np.dot(Cxy, np.linalg.inv(Cy))
W = np.dot(Cxy, np.linalg.inv(Cy))
b = -np.dot(W, np.matrix([[0.6],[0.7],[0.7],[0.7],[0.6]]))
lin_est = np.dot(t[0], np.transpose(W)) + np.transpose(b)
lin_error = np.mean(np.square(lin_est - t[1]))

c. We obtain

\[ Y_1 = 2X_1 + 3X_2 + X_3 + 2Z_{11} + Z_{12} + Z_{13} + Z_{21} + Z_{22} + Z_{31} \]
\[ Y_2 = X_1 + 3X_2 + 3X_3 + X_4 + Z_{11} + 2Z_{12} + Z_{13} + Z_{22} + Z_{23} + Z_{32} \]
\[ Y_3 = X_2 + 3X_3 + 3X_4 + X_5 + Z_{12} + 2Z_{13} + Z_{14} + Z_{23} + Z_{24} + Z_{33} \]
\[ Y_4 = X_3 + 3X_4 + 3X_5 + X_6 + Z_{13} + 2Z_{14} + Z_{15} + Z_{24} + Z_{25} + Z_{34} \]
\[ Y_5 = X_4 + 3X_5 + 2X_6 + 2Z_{14} + Z_{15} + Z_{25} + Z_{26} + Z_{35}. \]
The covariance matrix is
\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 3 & 3 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 3 & 3 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 3 & 3 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 3 & 3 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 13 & 72 & 13 & 63 \\
0 & 0 & 0 & 0 & 0 & 0 & 13 & 63 & 19 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 13 & 63 & 19 & 6 \\
\end{bmatrix}
\]

Since \(X\) has zero mean but \(Y\) has non-zero mean,
\[
\hat{X} = \Sigma_Y^{-1}Y - E(Y).
\]

The code snippet to calculate the optimal linear estimator is shown in part b.

d. The RMSE of the neural network is 0.534. The RMSE of the linear estimator is 0.537.

3. **Convergence to a random variable.** Consider a coin with random bias \(P \sim F_P(p)\). Flip the coin \(n\) times independently to generate \(X_1, X_2, \ldots, X_n\), where \(X_i = 1\) if the \(i\)-th outcome is heads and \(X_i = 0\) otherwise. Let \(S_n = \frac{1}{n} \sum_{i=1}^{n} X_i\) be the sample average. Show that \(S_n\) converges to \(P\) in mean square.

**SOLUTION** (15 points)

\[
E((S_n - P)^2) = E_P(E((S_n - P)^2|P))
\]
\[
= E_P(\text{Var}(S_n|P))
\]
\[
= E_P\left(\frac{1}{n^2} \text{Var}\left(\sum_{i=1}^{n} X_i|P\right)\right)
\]
\[
= E_P\left(\frac{1}{n^2}\left(nP(1-P)\right)\right)
\]
\[
= \frac{1}{n}\left(E(P) - E(P^2)\right).
\]

Therefore \(\lim_{n \to \infty} E((S_n - P)^2) = 0\) and \(S_n\) converges to \(P\) in mean square.

4. **Convergence examples.** Consider the following sequences of random variables, where
\( \omega \) is uniformly distributed in the interval \([0, 1]\).

\[
X_n(\omega) = \begin{cases} 
\frac{1}{n} & \omega < \frac{1}{n} \\
0 & \text{otherwise}
\end{cases}
\]

\[
Y_n(\omega) = \begin{cases} 
2^n & \omega < \frac{1}{n} \\
0 & \text{otherwise}
\end{cases}
\]

\[
Z_n(\omega) = \begin{cases} 
1 & \omega \in S_n \\
0 & \text{otherwise}
\end{cases}
\]

where \( \{S_n\} \) denotes the following sequence of intervals:

\[
\{S_n\} = \{[0, 1], [0, 1/2), (1/3, 1], (1/3, 2/3), (2/3, 1], \ldots \}.
\]

Which of these sequences converges to zero (a) with probability one, (b) in mean square, and/or (c) in probability? Justify your answers.

**SOLUTION (10 points)**

First consider \( X_n \). Since \( X_n(\omega) \) is either 0 or \( \frac{1}{n} \),

\[
0 \leq X_n(\omega) \leq \frac{1}{n}
\]

for every \( \omega \).

Therefore \( \lim_{n \to \infty} X_n(\omega) = 0 \) for every \( \omega \in \Omega \). Thus

\[
P\{\omega: \lim_{n \to \infty} X_n(\omega) = 0\} = P(\Omega) = 1,
\]

which shows that \( X_n \to 0 \) with probability one. This implies that \( X_n \to 0 \) in probability.

Now consider convergence in mean square.

\[
E((X_n - 0)^2) = \frac{1}{n} P\{\omega < \frac{1}{n}\}
\]

\[
= \frac{1}{n^2}
\]

tends to zero as \( n \) grows to infinity so \( X_n \to 0 \) in mean square.

Next consider \( Y_n \). Let us fix a constant \( 0 < \epsilon < 1 \), then:

\[
P\{|Y_n| < \epsilon \ \forall n \geq m\} = P\{Y_n = 0 \ \forall n \geq m\}
\]

\[
= P\{Y_m = 0\}
\]

\[
= P\{\omega \geq \frac{1}{m}\}
\]

\[
= \frac{1}{m},
\]

which tends to zero as \( m \) tends to infinity. Thus, \( Y_n \) converges to zero with probability one, and as a result in probability.

It does not converge to zero in mean square, since:

\[
E((Y_n - 0)^2) = 2^n P\{\omega < \frac{1}{n}\}
\]

\[
= \frac{2^n}{n},
\]
which tends to infinity as \( n \) grows.

Finally consider \( Z_n \). For any value of \( \omega \) and any \( n \), there exists a value of \( m > n \) such that \( \omega \in S_m \), which implies \( Z_m(\omega) = 1 \). Thus:

\[
P\{ \omega : \lim_{n \to \infty} Z_n(\omega) = 0 \} = 0,
\]
so \( Z_n \) does not converge to zero with probability one.

Now consider convergence in mean square. Note that if \( S_n \) has length \( 1/k \) then \( P\{Z_n \in S_n\} = 1/k \). Let us assume that \( S_n \) has length \( 1/k \), then we can bound \( n \) by 

\[
1 + 2 + 3 + \cdots + (k - 1) + k \quad \text{(we are adding the \( k \) sets with length \( 1/k \), the \( k - 1 \) sets of length \( 1/(k-1) \) and so on, down to the set that has length one)}.
\]

This sum is equal to \( k(k+1)/2 \), which is smaller than \( k^2 \). As a result \( n \leq k^2 \), which means that 

\[
\frac{1}{\sqrt{n}} \geq \frac{1}{k}
\]

and consequently \( P\{Z_n \in S_n\} \leq \frac{1}{\sqrt{n}} \). Using this fact we obtain:

\[
E((Z_n - 0)^2) = P\{Z_n \in S_n\} \leq \frac{1}{\sqrt{n}}
\]

which tends to 0 as \( n \) grows to infinity, so \( Z_n \) converges to zero in mean square. As a result, it also converges to zero in probability.

5. **Convergence in probability implies convergence in distribution.** Prove that convergence in probability implies convergence in distribution following these steps:

a. Show that for any random variable \( A \), any real number \( a \) and any \( \epsilon > 0 \),

\[
P(A \leq a - \epsilon) - P(|A_n - A| > \epsilon) \leq P(A_n \leq a) \leq P(A \leq a + \epsilon) + P(|A_n - A| > \epsilon)
\]

b. Assume that \( A_n \to A \) in probability. Use the expression derived in (a) to prove that \( A_n \to A \) in distribution. Hint: Remember that \( A_n \to A \) in distribution if 

\[
\lim_{n \to \infty} F_{A_n}(a) = F_A(a)
\]

for points \( a \) at which \( F_A \) is continuous, which implies

\[
\lim_{\epsilon \to 0} F_A(a \pm \epsilon) = F_A(a).
\]

**Solution**

a. If \( A_n > a \) and \( |A_n - A| \leq \epsilon \) then \( A > a - \epsilon \), so that

\[
\{A_n > a\} \cap \{|A_n - A| \leq \epsilon\} \subseteq \{A > a - \epsilon\}.
\]

For any sets \( S_1 \) and \( S_2 \), \( S_1 \subseteq S_2 \) implies \( S_2^c \subseteq S_1^c \), so the above equation becomes

\[
\{A \leq a - \epsilon\} \subseteq \{A_n \leq a\} \cup \{|A_n - A| > \epsilon\},
\]

after applying the De Morgan’s laws. Finally, applying the union bound we obtain

\[
P(A \leq a - \epsilon) \leq P(A_n \leq a) + P(|A_n - A| > \epsilon).
\]
This proves the left inequality. For the right inequality, note that if \( A > a + \epsilon \) and \( |A_n - A| \leq \epsilon \) then \( A_n > a \). Following the same reasoning as before, this implies
\[
\{A_n \leq a\} \subseteq \{A \leq a + \epsilon\} \cup \{|A_n - A| > \epsilon\},
\]
which again by the union bound allows us to conclude
\[
P(A_n \leq a) \leq P(A \leq a + \epsilon) + P(|A_n - A| > \epsilon).
\]

b. Taking the limit as \( n \to \infty \) in the expression obtained in (a) and using the fact that since \( A_n \to A \) in probability \( \lim_{n \to \infty} P(|A_n - A| > \epsilon) = 0 \) yields
\[
P(A \leq a - \epsilon) \leq \lim_{n \to \infty} P(A_n \leq a) \leq P(A \leq a + \epsilon).
\]
Now, we take the limit as \( \epsilon \) tends to zero and assume that \( F_A \) is continuous at \( a \) to obtain
\[
F_A(a) = P(A \leq a) \leq \lim_{n \to \infty} F_{A_n}(a) \leq P(A \leq a) = F_A(a).
\]
This implies that for any \( a \) such that \( F_A \) is continuous at \( a \) \( \lim_{n \to \infty} F_{A_n}(a) = F_A(a) \), which establishes convergence in distribution.

6. **Convergence experiments.** The purpose of this problem is to demonstrate different types of convergence and laws of large numbers.

a. Use MATLAB to generate 200 samples \( X_1, \ldots, X_{200} \) of i.i.d. zero mean, unit variance Gaussian random variables. Plot the sample average sequence \( S_n = \frac{1}{n} \sum_{i=1}^{n} X_i \) as a function of \( n \). Hint: use `randn` and `cumsum`.

b. Generate 5000 sequences each consisting of 200 samples of i.i.d. zero mean, unit variance Gaussian random variables:
\[
X_{i,1}, X_{i,2}, \ldots, X_{i,200}, \quad i = 1, 2, \ldots, 5000
\]

Compute the sample average sequences \( S_{i,1}, S_{i,2}, \ldots, S_{i,200} \) for \( i = 1, \ldots, 5000 \). You can generate a \( 5000 \times 200 \) matrix of i.i.d. Gaussians using the `randn` command.

c. **Strong law of large numbers.** To explore convergence with probability 1, let \( N_m \) be the number of \( S_{i,n} \) sequences with \( |S_{i,n} - E(X)| > 0.1 \) for some \( n \geq m \). Use \( E_m = N_m/5000 \) as an estimate of the probability \( P_m = P(|S_n - E(X)| > 0.1 \) for some \( n \geq m \)\}. Plot \( E_m \) vs. \( m \) for \( 1 \leq m \leq 200 \).

d. **Mean square convergence.** To explore convergence in mean square, compute
\[
M_n = \frac{1}{5000} \sum_{i=1}^{5000} (S_{i,n} - E(X))^2,
\]
which are estimates of the mean square. Plot \( M_n \) vs. \( n \) for \( 1 \leq n \leq 200 \).

e. **Weak law of large numbers.** To explore convergence in probability, let \( N_n \) be the number of \( S_{i,n} \) sequences for which \( |S_{i,n} - E(X)| > 0.1 \). Use \( E_n = N_n/5000 \) as an
estimate of the probability $P_n = P\{|S_n - E(X)| > 0.1\}$. Plot $E_n$ and $P_n$ vs. $n$ for $1 \leq n \leq 200$.

**Solution (15 points)**

The MATLAB code is below. The output is shown in Figure 4.

```matlab
clear all;
cf;

% Part (a)
% Generate 200 samples (X_1 to X_200) of i.i.d. zero mean, unit variance Gaussian random variables.
% Hint: Use randn and cumsum.

n = 1:200;

% WRITE MATLAB CODE HERE

X = randn( 1, 200 );
S = cumsum( X );

% Do the divide by n part.
S = S./n;

subplot( 4, 1, 1 );
plot( n, S );
xlabel( 'n' );
ylabel( 'S_n' );
title( '2(a) Sample average sequence' );

% Part (b)
% Now generate 5000 such sequences.
% Hint: use randn and cumsum (be careful!) again

% WRITE MATLAB CODE HERE

X = randn( 5000, 200 );
S = cumsum( X, 2 );

S = S./repmat( n, 5000, 1 );

% Part (c) Strong Law of Large Numbers (this loop will run for a minute)

E_m = zeros(200,1);
for m = 0:199,

% N_m should be the number of rows in S that have an entry whose absolute value is > 0.1 in columns m+1 through 200.

% WRITE MATLAB CODE HERE
```
N_m = sum(sum((abs(S(:, m+1:200)) > 0.1), 2) > 0);
E_m(m+1) = N_m/5000;

end;
subplot( 4, 1, 2 );
plot( E_m );
xlabel( 'n' );
ylabel( 'E_m' );
title( '2(c) Strong Law of Large Numbers' );

% Part (d) Convergence in Mean Square

% S_squared is the square of the S matrix. EX = 0.
S_squared = S.^2;

% WRITE MATLAB CODE HERE

M = 1/5000*sum( S_squared );

subplot( 4, 1, 3 );
plot( n, M );
xlabel( 'n' );
ylabel( 'Mn' );
title( '2(d) Mean square convergence' );

% Part (e) Weak Law of Large Numbers

% Count the number of times |S_i,n| > 0.1 in each column.

% WRITE MATLAB CODE HERE

N = sum( abs( S ) > 0.1 );

% Find E by dividing by 5000.
E = N/5000;

% Plot Pn and En vs. n. Since EX = 0, Pn is the probability
% that |Sn| > 0.1. Sn has zero mean and a standard deviation
% sigma_Sn. Hint: sigma_Sn is a function of n.
% Therefore P( |Sn| > 0.1 ) = 2*Q( 0.1/sigma_Sn )
% erf(x) = 2/sqrt(pi) * integral from x to inf of exp( -t^2 ) dt
% So Q( x ) = 1/2*erfc( x/sqrt( 2 ) );
% Find sigma_Sn and Pn. Hint: Pn and En look quite similar.

% WRITE MATLAB CODE HERE

sigma_Sn = 1./sqrt(n);
P = erfc( ( 0.1/sigma_Sn )/sqrt(2) );

subplot( 4, 1, 4 );
plot( n, P, 'r--' );
hold on;
plot( n, E );
axis( [ 0 200 0 1 ] );
xlabel( 'n' );
ylabel( 'En (solid), Pn (dashed)' );
title( '2(e) Convergence in probability' );

% Produce hardcopy
orient tall
print hw6q7

Figure 4: Output of convergence experiments