CS215: Assignment 1 Report

Y Harsha Vardhan 24b1069
M Rishi Vardhan 24b0969
S Vivek 24b0912

Contents

Instructions for Running the Code

Question-1:

To run the MATLAB code for this question, follow these steps:

- First upload the given code file in the MATLAB Workspace, it is named as Q1.m
- Next run this file using the Run option
- We will get 2 images, one when f (the percentage of corrupted points) is 30% and when it is 60%
- We will also get the RMSE values for the three filtered signals in the terminal.
- The obtained graphs are also present in the folder

Question-2:

The MATLAB code for this question is named Q2.m, it consists of the formulae only, to run this code successfully, we need to first intitate a set of data points, calculate their mean, median and standard deviations using inbuilt functions, then choose a new value to be added to this set, then we can call these functions to calculate the updated Mean, Median and Standard Deviation

1 Question 1: Moving Median, Average, and Quartile Filtering

1.1 Problem Restatement

Our goal is to recover the original sine wave, $y = 6.5 \sin(2.1x + \pi/3)$, after deliberately corrupting it. To simulate noise, we randomly replace 30% (and later 60%) of the data points with values between 100 and 120.
We then try three moving-window filters: median, average, and first quartile — to clean the signal. To check which one worked better, we compare them using RMSE values and also by plotting the filtered results with the original.

1.2 Approach

We are using a moving-window approach. For each point z(i) in the noisy signal, we looked at 8 neighbors on both sides (a window of 17 points). From this neighborhood, we calculate the output using either the median, mean, or first quartile. Each method smoothed the noise in its own way.

Moving Median

For the median filter, we took the 17-point neighborhood around each data point, sorted the values, and selected the middle value as the filtered output. We chose this method because the median is effective at ignoring extreme outliers, like the noise we added.

Moving Average

Our second approach was a moving average filter. Here, we simply calculated the arithmetic mean of all values in the neighborhood. We expected this to perform poorly since the mean is easily skewed by the large noise values.

Moving Quartile

Finally, we implemented a moving quartile filter by taking the 25th percentile of the values in each neighborhood. Like the median, this method is robust to outliers, but we anticipated it might introduce a downward bias in the signal.

1.3 MATLAB Implementation

Listing 1: MATLAB code for Moving Median, Average and Quartile Filtering

```
% Clearing workspace and closing all previously opened figures
clear;
clc;
close all;

% --- Step 1.1: Generating the clean sine wave ---
x = -3:0.02:3; % Range of x, from -3 to 3 in steps of 0.02
y = 6.5 * sin(2.1 * x + pi/3); % Sine Curve Equation

% --- Step 1.2: Generating the corrupted sine wave (f = 30%) ---
f = 0.30;
n = length(y); % Total number of data points
z = y; % Initializing z as a copy of the clean signal

% Calculating the number of points to corrupt (30% of the points)
num_corrupt = round(f * n);

% Selecting indices at random to corrupt using randperm
indices_to_corrupt = randperm(n, num_corrupt);

% Generating random noise values between 100 and 120 using rand
noise = 100 + (120 - 100) * rand(1, num_corrupt);

% Replacing the values at the selected indices with the noise generated
z(indices_to_corrupt) = noise;



% --Step 2: Applying the three filters (mean, median and first quartile
   ) --
w = 8; % The neighborhood width (8 to the left, 8 to the right)
y_median = zeros(1, n);
y_mean = zeros(1, n);
y_quartile = zeros(1, n);

for i = 1:n
    % Defining the start and end of the neighborhood
    % max/min functions handle the edges of the signal
    start_index = max(1, i - w);
    end_index = min(n, i + w);

    % Extracting the neighborhood from the corrupted signal z
    neighborhood = z(start_index:end_index);

    % Calculating the statistic for each filter
    y_median(i) = median(neighborhood);
    y_mean(i) = mean(neighborhood);
    y_quartile(i) = quantile(neighborhood, 0.25); % 0.25 for first
       quartile
end
```

```
% --- Step 3.1: Plotting the signals for f=30% ---
figure; % Creating a new figure window
hold on; % Allowing multiple plots on the same figure

% Plotting corrupted signal as white curve
plot(x, z, 'w-', 'DisplayName', 'Corrupted Signal');
plot(x, y, 'b-', 'LineWidth', 2, 'DisplayName', 'Original Signal');
plot(x, y_median,'r-','LineWidth',1.5,'DisplayName','Median Filtered');
plot(x, y_mean, 'g-', 'LineWidth', 1.5, 'DisplayName','Mean Filtered');
plot(x,y_quartile,'m-','LineWidth',1.5,'DisplayName','Quartile Filtered
    ');

title('Signal Filtering (f = 30% Corruption)');
xlabel('x');
ylabel('Signal Value');
legend('show'); % Displaying the legend
grid on;
hold off;

% --- Step 3.2: Calculating and displaying the RMSE ---
rmse_median = sum((y - y_median).^2) / sum(y.^2);
rmse_mean = sum((y - y_mean).^2) / sum(y.^2);
rmse_quartile = sum((y - y_quartile).^2) / sum(y.^2);

fprintf('--- Results for f = 30%% ---\n');
fprintf('RMSE Median Filter: %f\n', rmse_median);
fprintf('RMSE Mean Filter:   %f\n', rmse_mean);
fprintf('RMSE Quartile Filter: %f\n\n', rmse_quartile);
```

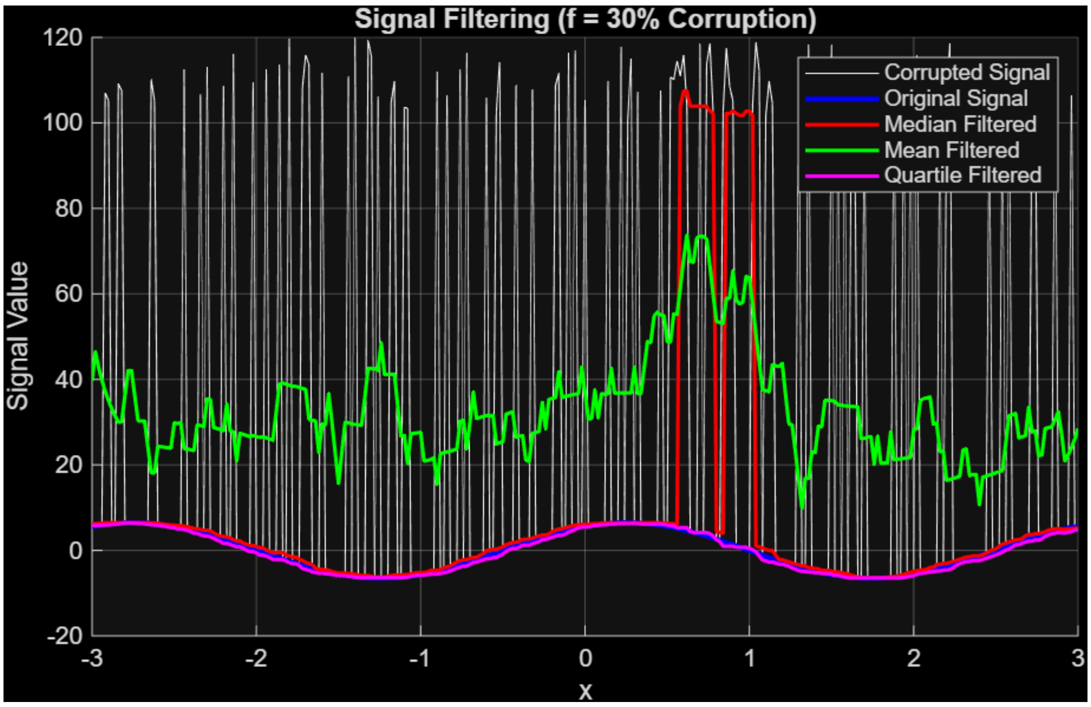

Figure 1: Original, Corrupted, and Filtered Signals for $f = 30\%$

```
% --- Repeating all steps for f = 60% ---
% --- Step 1: Generating the corrupted sine wave (f = 60%) ---
f = 0.60;
z = y; % Starting with a clean copy again

% Calculating the number of points to corrupt
num_corrupt = round(f * n);

% Getting random indices to corrupt using randperm
indices_to_corrupt = randperm(n, num_corrupt);

% Generating random noise values between 100 and 120
noise = 100 + (120 - 100) * rand(1, num_corrupt);

% Replacing the values with noise
z(indices_to_corrupt) = noise;

% -- Step 2: Applying the three filters (mean, median and first
   quartile)--
for i = 1:n
    start_index = max(1, i - w);
    end_index = min(n, i + w);
    neighborhood = z(start_index:end_index);

    y_median(i) = median(neighborhood);
    y_mean(i) = mean(neighborhood);
    y_quartile(i) = quantile(neighborhood, 0.25);
end

% --- Step 3.1: Plotting the signals for f=60% ---
figure;
hold on;
plot(x, z, 'w-', 'DisplayName', 'Corrupted Signal');
plot(x, y, 'b-', 'LineWidth', 2, 'DisplayName', 'Original Signal');
plot(x, y_median,'r-','LineWidth',1.5,'DisplayName','Median Filtered');
plot(x, y_mean, 'g-', 'LineWidth', 1.5, 'DisplayName','Mean Filtered');
plot(x,y_quartile,'m-','LineWidth',1.5,'DisplayName','Quartile Filtered
   ');

title('Signal Filtering (f = 60% Corruption)');
xlabel('x');
ylabel('Signal Value');
legend('show');
grid on;
hold off;

% --- Step 3.2: Calculating and showing the RMSE ---
rmse_median = sum((y - y_median).^2) / sum(y.^2);
rmse_mean = sum((y - y_mean).^2) / sum(y.^2);
rmse_quartile = sum((y - y_quartile).^2) / sum(y.^2);

fprintf('--- Results for f = 60%% ---\n');
fprintf('RMSE Median Filter: %f\n', rmse_median);
fprintf('RMSE Mean Filter:   %f\n', rmse_mean);
fprintf('RMSE Quartile Filter: %f\n', rmse_quartile);
```

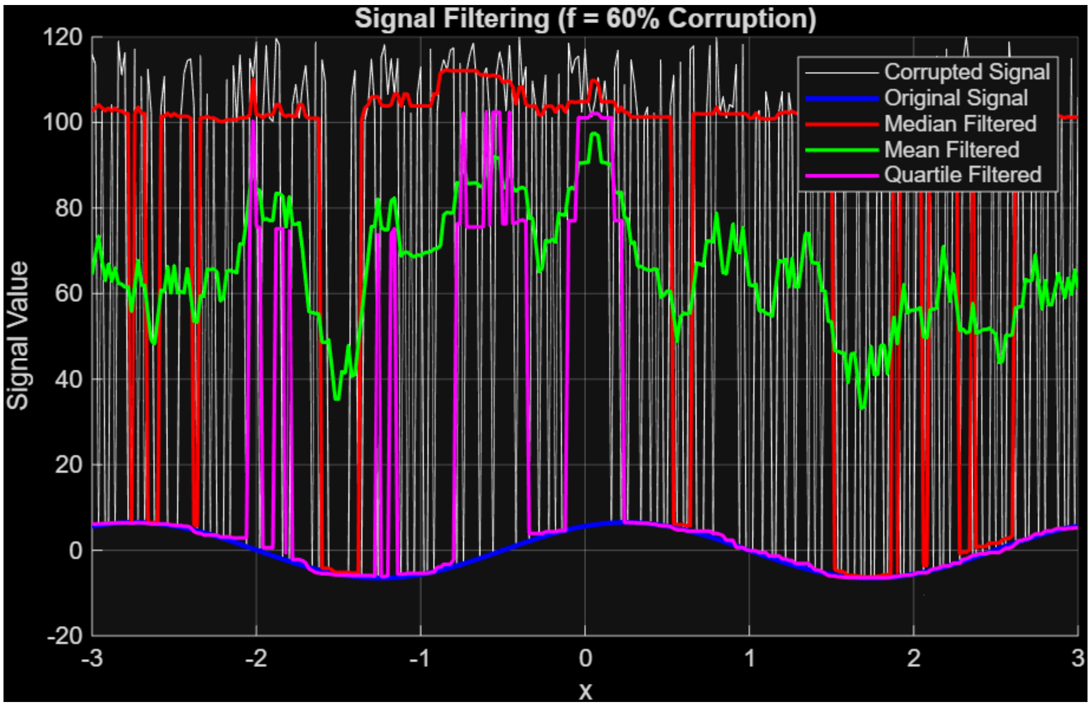

Figure 2: Original, Corrupted, and Filtered Signals for $f = 60\%$

1.4 Relative Mean Squared Errors

Method	**RMSE** ($f = 30\%$)	**RMSE** ($f = 60\%$)
Median	1.710584	454.934342
Average	55.267187	212.891854
Quartile	0.012188	21.086695

Table 1: RMSE comparison of different methods for two values of f.

1.5 Analysis

Looking at Table 1, the quartile filter turned out to work best in both the 30% and 60% corruption cases, which was a bit unexpected. Normally, the median handles impulse noise well, but at 60% corruption it struggled — which makes sense since its breakdown point is 50%. The quartile filter handled the heavy noise better and kept the error lower. As expected, the moving average did the worst because the large noise values pulled the mean away from the true signal.

2 Question 2: Updating Mean, Median, and Std. Deviation

2.1 Formula Derivations

Updating Mean

Given that the original dataset has n numbers, $A_1, A_2, \ldots, A_n$. The original mean, $\bar{A}_{old}$, is given by:

$$\bar{A}_{old} = \frac{\sum_{i=1}^{n} A_i}{n}$$

From this, we can express the sum of the original numbers as:

$$\sum_{i=1}^{n} A_i = n \cdot \bar{A}_{old}$$

When a new data value, x, is added to the set, the new number of elements becomes $n+1$. The new sum is the old sum plus the new value:

$$\text{New Sum} = \left(\sum_{i=1}^{n} A_i\right) + x = n \cdot \bar{A}_{old} + x$$

The new mean $\bar{A}_{new}$ is just the updated sum divided by $n+1$:

$$\bar{A}_{new} = \frac{n \cdot \bar{A}_{old} + x}{n+1}$$

This gives a simple update rule for the mean without storing all data again

Updating Median

The update rule for the median depends on whether the original number of elements, n, is odd or even. We can assume that the array A is sorted and all values are unique.

Case 1: n is odd
Let $n = 2k+1$. The old median is the middle element, $\text{OldMedian} = A_{k+1}$.
The new size of the array is $n+1 = 2k+2$ (even). The new median will be the average of the two new middle elements at positions $k+1$ and $k+2$.
Let the new value be x. The logic requires comparing x to the old median and its immediate neighbors.

- If $x < \text{OldMedian}$, the new median is $\frac{\text{OldMedian}+\max(x, A_{(n+1)/2-1})}{2}$.
- If $x > \text{OldMedian}$, the new median is $\frac{\text{OldMedian}+\min(x, A_{(n+1)/2+1})}{2}$.

Case 2: n is even
Let $n = 2k$. The old median is the average of the two middle elements, $\text{OldMedian} = \frac{A_k + A_{k+1}}{2}$.
The new size is $n+1 = 2k+1$ (odd). The new median will be the single middle element at position $k+1$.

- If $x < A_k$, the new median is A_k.
- If $x > A_{k+1}$, the new median is A_{k+1}.
- If $A_k < x < A_{k+1}$, the new median is x.

Updating Standard Deviation

We can derive the update formula for the sample variance, s^2, and then take the square root to get the formula for the updated standard deviation.
The sample standard deviation, s_n, for n values is given by:

$$s_n^2 = \frac{\sum_{i=1}^{n}(A_i - \bar{A}_n)^2}{n-1}$$

A useful formula for the sum of squared differences is $\sum(A_i - \bar{A})^2 = \sum A_i^2 - n\bar{A}^2$. From this, we can find the sum of the squares of the original values, $\sum A_i^2$:

$$(n-1)s_{old}^2 = \sum_{i=1}^{n} A_i^2 - n\bar{A}_{old}^2 \implies \sum_{i=1}^{n} A_i^2 = (n-1)s_{old}^2 + n\bar{A}_{old}^2$$

When a new value, x, is added, the new sum of squares for the $n+1$ values is:

$$\sum_{i=1}^{n+1} A_i^2 = \left(\sum_{i=1}^{n} A_i^2\right) + x^2 = (n-1)s_{old}^2 + n\bar{A}_{old}^2 + x^2$$

The new sample variance, s_{new}^2, for the set of $N = n+1$ values is:

$$s_{new}^2 = \frac{\left(\sum_{i=1}^{n+1} A_i^2\right) - (n+1)\bar{A}_{new}^2}{(n+1)-1} = \frac{\left[(n-1)s_{old}^2 + n\bar{A}_{old}^2 + x^2\right] - (n+1)\bar{A}_{new}^2}{n}$$

The new standard deviation is the square root of the new variance:

$$s_{new} = \sqrt{\frac{(n-1)s_{old}^2 + n\bar{A}_{old}^2 + x^2 - (n+1)\bar{A}_{new}^2}{n}}$$

2.2 MATLAB Functions

Listing 2: MATLAB functions for updating statistics

```
%####################################################################%
function newMean = UpdateMean(OldMean, NewDataValue, n)
    newMean = (OldMean * n + NewDataValue) / (n + 1);
end
%####################################################################%
function newMedian = UpdateMedian(OldMedian, NewDataValue, A, n)
    if mod(n, 2) == 1
        if n == 1
            newMedian = (A(1) + NewDataValue) / 2;
            return;
        end
        median_index = (n + 1) / 2;
        A_before = A(median_index - 1);
        A_after = A(median_index + 1);

        if NewDataValue < OldMedian
            newMedian = (OldMedian + max(NewDataValue, A_before)) / 2;
        else
            newMedian = (OldMedian + min(NewDataValue, A_after)) / 2;
        end
    else
        m1_index = n / 2;
        m2_index = n / 2 + 1;
        m1 = A(m1_index);
        m2 = A(m2_index);

        if NewDataValue < m1
            newMedian = m1;
        elseif NewDataValue > m2
            newMedian = m2;
        else
            newMedian = NewDataValue;
        end
    end
end
%######################################################################%
function newStd = UpdateStd(OldMean, OldStd, NewMean, NewDataValue, n)
    if n == 0
        newStd = 0;
        return;
    end

    sum_sq_old = (n - 1) * OldStd^2 + n * OldMean^2;
    sum_sq_new = sum_sq_old + NewDataValue^2;

    N = n + 1;

    new_variance = (sum_sq_new - N * NewMean^2) / n;

    if new_variance < 0
        new_variance = 0;
    end

    newStd = sqrt(new_variance);
end
%######################################################################%
```

2.3 Histogram Update Explanation

Updating a histogram is a two-step process. First, we need to find which bin the new data point belongs to by comparing it to the bin edges. Once the correct bin is found, we just increment that bin's frequency count by one.

1. **Identify the Bin:** First, you must determine which bin the new data value x, belongs to. This involves comparing x to the predefined bin edges. For example, if the bins are $[e_0, e_1), [e_1, e_2), \ldots, [e_{k-1}, e_k)$, you find the index j such that $e_{j-1} \leq x < e_j$.

2. **Increment the Count:** Once the correct bin j is identified, we simply increment the frequency count for that bin by one.

If the new value lies outside the histogram range (i.e., $x < e_0$ or $x \geq e_k$), we just extend the range by adding new bins. For example, if x is larger than the last bin, we create a new bin after it and set its count to 1.

3 Question 3: Probability Bound

Proof

We are given two events A and B such that:

$$P(A) \geq 1 - q_1, \quad \text{and} \quad P(B) \geq 1 - q_2.$$

We need to show that:

$$P(A, B) \geq 1 - (q_1 + q_2)$$

where $P(A, B)$ denotes the joint probability of A and B.

From the *Principle of Inclusion and Exclusion*, we have:

$$P(A \cup B) = P(A) + P(B) - P(A, B).$$

Since $P(A \cup B) \leq 1$, it follows that:

$$P(A, B) \geq P(A) + P(B) - 1.$$

Using the given inequalities:

$$P(A, B) \geq (1 - q_1) + (1 - q_2) - 1$$

$$\implies P(A, B) \geq 1 - (q_1 + q_2).$$

So the inequality holds, and the bound is established.

4 Question 4: Bayes' Theorem – Bus Accident Problem

Solution

Let the events be:

- R : the bus *is* red.
- B : the bus *is* blue.
- S_R : the witness *observes/labels* the bus as red.

From the problem:

$$P(R) = \frac{1}{100}, \qquad P(B) = \frac{99}{100}.$$

The ophthalmologist's test gives

$$P(S_R \mid R) = 0.99, \qquad P(S_R \mid B) = 0.02.$$

We want the posterior probability that the bus was really red given that the witness observed red. Let's use Bayes' theorem:

$$P(R \mid S_R) = \frac{P(S_R \mid R)\,P(R)}{P(S_R)} = \frac{P(S_R \mid R)\,P(R)}{P(S_R \mid R)\,P(R) + P(S_R \mid B)\,P(B)}.$$

Substituting the numbers, we get:

$$P(R \mid S_R) = \frac{0.99 \cdot \frac{1}{100}}{0.99 \cdot \frac{1}{100} + 0.02 \cdot \frac{99}{100}} = \frac{\frac{99}{10000}}{\frac{99}{10000} + \frac{198}{10000}} = \frac{99}{297} = \frac{1}{3}.$$

Hence, we get that:

$$P(R \mid S_R) = \frac{1}{3} \approx 0.3333 \quad \text{(i.e. 33.33\%)}.$$

Interpretation:

Even though the witness is fairly reliable for red objects (99%), because red buses are so rare (1 in 100), the posterior probability that the bus actually was red given the witness's statement is only one third.
In other words, it is twice as likely that the bus was blue (posterior probability 2/3) despite the witness saying "red."
This should be the defense's main argument: the eyewitness testimony alone does not strongly establish that the bus was red.

5 Question 5: Exit Poll Accuracy – Known Percentages

Solution

We are given that each resident votes for candidate A with probability 0.95 and for candidate B with probability 0.05. An exit poll samples 3 residents *with replacement* and declares the candidate with the majority in the sample as the poll's prediction.

Let X be the number of sampled voters (out of 3) who voted for A. Since sampling is with replacement and each sampled voter independently votes for A with probability $p = 0.95$

The exit poll declares A the winner iff at least 2 of the 3 sampled voters chose A. Thus the accuracy (probability the poll declares A) is

$$\Pr(\text{poll declares } A) = \Pr(X \geq 2) = \Pr(X = 2) + \Pr(X = 3).$$

Computing these probabilities, we have:

$$\Pr(X = 2) = \binom{3}{2} p^2(1-p) = 3 \cdot (0.95)^2 \cdot 0.05,$$
$$\Pr(X = 3) = \binom{3}{3} p^3(1-p)^0 = (0.95)^3.$$

Substituting $p = 0.95$:

$$\Pr(X = 2) = 3 \cdot 0.9025 \cdot 0.05 = 3 \cdot 0.045125 = 0.135375,$$
$$\Pr(X = 3) = 0.95^3 = 0.857375.$$

Therefore

$$\Pr(\text{poll declares } A) = 0.135375 + 0.857375 = 0.99275.$$

So the exit poll's accuracy is

$$\Pr(\text{correct prediction for } A) = 0.99275 \approx 99.275\%.$$

Second part (when population = 10,000).

The sampling is again stated to be *with replacement*, so the draws remain independent with the same success probability $p = 0.95$. Hence the distribution of X and the probability computed above do not depend on the population size. Even with 10,000 voters, the accuracy stays exactly the same since the sampling is with replacement.:

$$\Pr(\text{poll declares } A) = 0.99275 \approx 99.275\%.$$

6 Question 6: Exit Poll Accuracy – General Case

Given that there are m voters in the village, of which a fraction $p = \frac{k}{m}$ prefer A over B.
The exit poll samples n voters *with replacement*, uniformly at random.
For the i-th draw let

$$x_i = \begin{cases} 1, & \text{if the sampled voter voted for } A, \\ 0, & \text{otherwise,} \end{cases}$$

Given that, q(S) is the proportion of voters from S (a randomly chosen subset of n people) who voted for A out of n = $|S|$:

$$q(S) = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

The expectation of x$_i$ is given by:

$$E[x_i] = (p)(1) + (1-p)(0) = p$$

The variance of x$_i$ is given by:

$$Var(x_i) = E(x_i^2) - (E(x_i))^2$$

$$E(x_i^2) = (p)(1)^2 + (1-p)(0)^2 = p$$

$$Var(x_i) = p - p^2 = p(1-p)$$

Since there are m^n possible ordered n-tuples ("subsets" in the problem), averaging over all S is the same as taking expectation, E[q(S)].

$$\frac{1}{m^n}\sum_S q(S) = \mathbb{E}[q(S)]$$

6.1 Part (a)

$$\frac{1}{m^n}\sum_S q(S) = \mathbb{E}[q(S)] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} x_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[x_i] = \frac{1}{n}\cdot np = p.$$

6.2 Part (b)

As x$_i$ are independent and identically distributed random variables (or uncorrelated),

$$\mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\mathrm{Var}(x_i) = \frac{1}{n^2}\cdot n\,p(1-p) = \frac{p(1-p)}{n}.$$

Therefore,

$$\frac{1}{m^n}\sum_S q(S)^2 = \mathbb{E}[q(S)^2] = \mathrm{Var}(q(S)) + \big(\mathbb{E}[q(S)]\big)^2 = \frac{p(1-p)}{n} + p^2 = \frac{p}{n} + \frac{p^2(n-1)}{n}.$$

6.3 Part (c)

$$\frac{1}{m^n}\sum_S \big(q(S)-p\big)^2 = \mathbb{E}\big[(q(S)-p)^2\big] = \mathrm{Var}(q(S)) = \frac{p(1-p)}{n}.$$

6.4 Part (d)

By Chebyshev's inequality,

$$\Pr(\,|q(S) - p| > \delta\,) \leq \frac{\mathrm{Var}(q(S))}{\delta^2} = \frac{p(1-p)}{n\,\delta^2}.$$

Since each S is equally likely among the m^n possible samples, the left-hand side equals the *proportion* of n-sized samples for which $|q(S) - p| > \delta$. Hence that proportion is at most

$$\boxed{\frac{p(1-p)}{n\,\delta^2}}.$$

Significance. The sample proportion $q(S)$ stays close to the true value p, and large deviations (more than δ) become less likely as n grows. Interestingly, this doesn't depend on the total population size m. That's why even a small sample can give a fairly accurate poll.