ECE 514: Monte Carlo Simulation

Arpad Voros

October 13th, 2020

1 Background

In wired, digital communications, a system sends a sequence of known and unknown symbols through a channel. During transmission, after filtering, and after sampling, the received sample, X can be modeled as

$$X_i = Cs_i + N_i \tag{1}$$

where C is the channel coefficient, s is the symbol, and N is noise. In this simulation, we are going to use X to estimate the channel coefficient C.

1.1 Specifications

- The simulation consists of two separate runs. One where the number of samples n is 5, and another where n is 10. Each time, there will be m number of simulated trials, where $m = 1000$
- We will be constructing 68.27% confidence intervals (CI) for X
- The true channel coefficient, C, equals 10
- Each symbol equals one when the symbol is known, and since all symbols are known, we know that
$$s_i = 1, \quad 0 \leq i \leq n-1 \tag{2}$$
- N_i is a sequence of i.i.d. Gaussian noise r.v.'s with the following parameters
$$N_i \sim N\left(0, \sigma_N^2\right), \quad 0 \leq i \leq n-1 \tag{3}$$
- The SNR, the ratio of the signal power to the noise power, for this simulated channel is -9dB, or 0.125. The SNR, for this model of X, can be calculated using
$$\text{SNR} = 0.125 = \frac{|C|^2}{\sigma_N^2} \tag{4}$$

2 Experiment

We will be using MATLAB to run the simulation. Instead of using the standard random number generator, we will be using the Mersenne Twister pseudo-random number generator with seed 1056, since the vowels in my name (Arpad Voros) are AAOO, which results in the seed being $16 + 16 + 512 + 512 = 1056$

To implement this in MATLAB, we add the following line at the beginning of our script

```
rng(1056, 'twister');
```

Before we construct the various CIs for X, we know that the distribution of X is simply a constant plus the noise N, meaning it follows the same distribution of N, shifted by Cs_i. Using (1), (2), (3), we know

$$X \sim N(C, \sigma_N^2) \tag{5}$$

And from (4), we can determine

$$\sigma_N^2 = \frac{|C|^2}{0.125} = 800 \tag{6}$$

so that

$$\mu_x = E[X] = 10, \quad \sigma_x = \sqrt{800} \tag{7}$$

2.1 Case 1: CI using known variance of X

Since we want to be 68.27% confident,

$$\alpha = 1 - 0.6827 = 0.3173 \tag{8}$$

And $y_{\alpha/2}$ is calculated using the Wald distribution, or similarly in MATLAB

```
y = norminv(1 - (alpha / 2));
```

And we know our CI for each trial depends on the sample mean M, as well as the range δ, where

$$\delta = y_{\alpha/2}\sqrt{\frac{\sigma_X^2}{n}} \tag{9}$$

where δ is consistent throughout all m trials.

2.2 Case 2: CI using estimated variance of X

For this CI, we are unable to use σ_X, but must use the sample variance S, calculated by

$$S = \frac{1}{n-1}\sum_{i=0}^{n-1}(X_i - M)^2 \tag{10}$$

We know that

$$S \approx \sigma_X^2 \tag{11}$$

meaning our CIs use the same $y_{\alpha/2}$ values, with the only term changing being the δ for each of the m trials, so that

$$\delta_j = y_{\alpha/2}\sqrt{\frac{S_j}{n}}, \quad 0 \le j \le m-1 \tag{12}$$

2.3 Case 3: CI using Student's T Distribution

By taking advantage of the fact that the noise N is normally distributed, we can use Student's T distribution to calculate the confidence interval. Our $y_{\alpha/2}$ value is computed using the inverse Student's T distribution, or similarly in MATLAB

```
y = tinv(1 - (alpha / 2), n - 1);
```

where n is the number of samples per trial (5 and 10), and $n-1$ is the degrees of freedom, commonly denoted by ν. Using the sample variance as before, we calculate the CI the same way as before, with the only difference being the $y_{\alpha/2}$ value

$$\delta_j = y_{\alpha/2}\sqrt{\frac{S_j}{n}}, \quad 0 \le j \le m-1 \tag{13}$$

2.4 Observations

n \ case	1	2	3
5	1	1	1.1417
10	1	1	1.0588

Table 1: All $y_{\alpha/2}$ values used in creating the CIs

It's clear that using the Student's T distribution in case 3 results in a larger confidence interval when compared to case 2. It also seems to decrease and approach 1 as n increases.

n \ case	1	2	3
5	12.6491	$\sqrt{\frac{S}{5}}$	$\sqrt{\frac{S}{5}}$
10	8.9443	$\sqrt{\frac{S}{10}}$	$\sqrt{\frac{S}{10}}$

Table 2: All standard deviations used in creating the CIs

As you can see, the standard deviations of the sample means M decrease as n increases, due to being inversely proportional. This also intuitively makes sense, since increasing the population will result in more accurate results. For the first case, we know for all trials $\sigma_X^2 = 800$, so that the standard deviation of M used in calculating the CI is $\sqrt{800/5}$ and $\sqrt{800/10}$ for n is 5 and 10, respectively. As for cases 2 and 3, the estimated variance of X, S, is used

3 Results

n \ case	1	2	3
5	69.9%	62.0%	68.0%
10	68.3%	65.0%	66.9%

Table 3: Percent of times for C fell into CI

The table above shows the amount if times the true mean of X, C, fell within the CI for all m trials. We can observe that all of these CIs are close to our theoretical confidence value of 68.27%. But more noticeably, we can see that using the true variance of X in case 1 results in almost perfect values. Using the estimated variance of X in case 2 drops in accuracy, due to the small sample size. But then in case 3, since we take advantage of the fact that the noise is normally distributed, we can observe more accurate values than in the previous case. This confirms our observation in Table 1, where we observed larger CIs being produced in case 3.

n	RMSE
5	27.7389
10	28.5263

Table 4: RMSE for all values of X_{ij}

The mean-square-error is simply calculated for all values of X_{ij} by

$$\text{MSE} = \frac{1}{mn} \sum_{j=0}^{m-1} \sum_{i=0}^{n-1} (X_{ij} - C)^2 \tag{14}$$

And the RMSE is simply the square root of the MSE. We observe that these values are awfully close to our true standard deviation $\sigma_X = \sqrt{800} \approx 28.2843$, which makes sense since the data, on average, deviates from the mean by this amount.

All the figures below plot the true mean of X ($C = 10$) shown in black, the sample mean M shown in blue, as well as the CI ($M \pm \delta$) shown in dashed red for each of the three cases. Only the first 10 trials are plotted. All the plots are to scale, labeled correctly, and titled appropriately. The top plot is for $n = 5$ while the bottom plot is for $n = 10$.

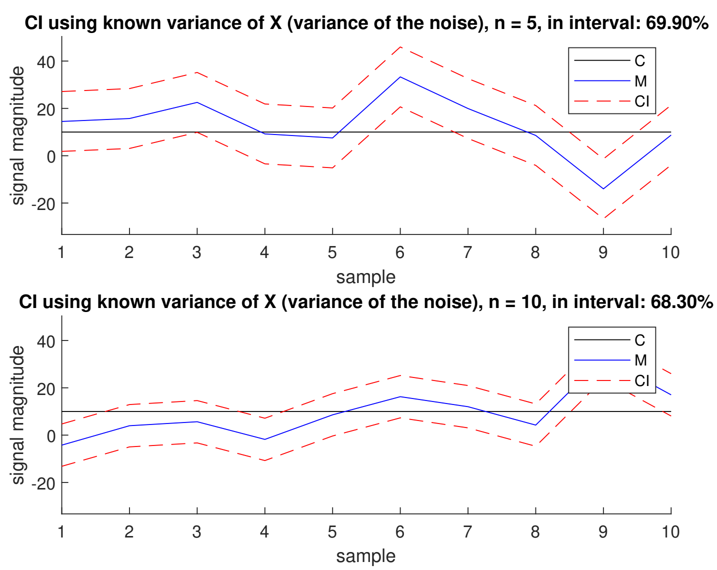

Figure 1: Case 1: CIs for the first 10 trials

The figure above shows the plots for case 1. It's evident that all the δ values stay consistent throughout all trials of m, as show in (9)

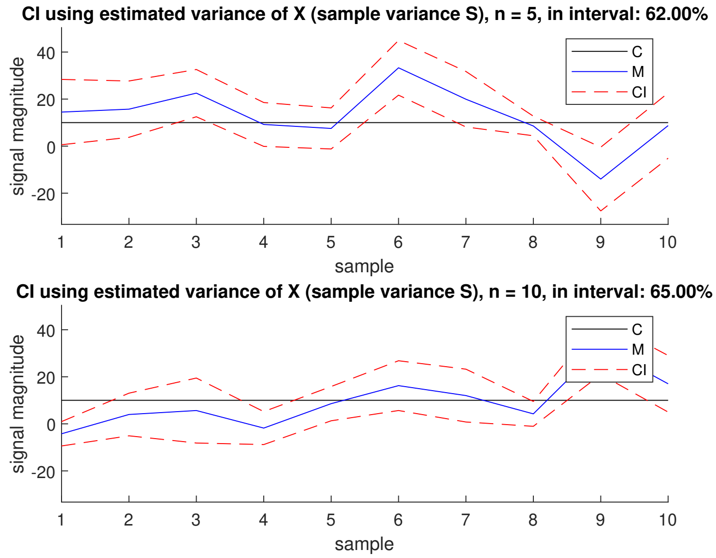

Figure 2: Case 2: CIs for the first 10 trials

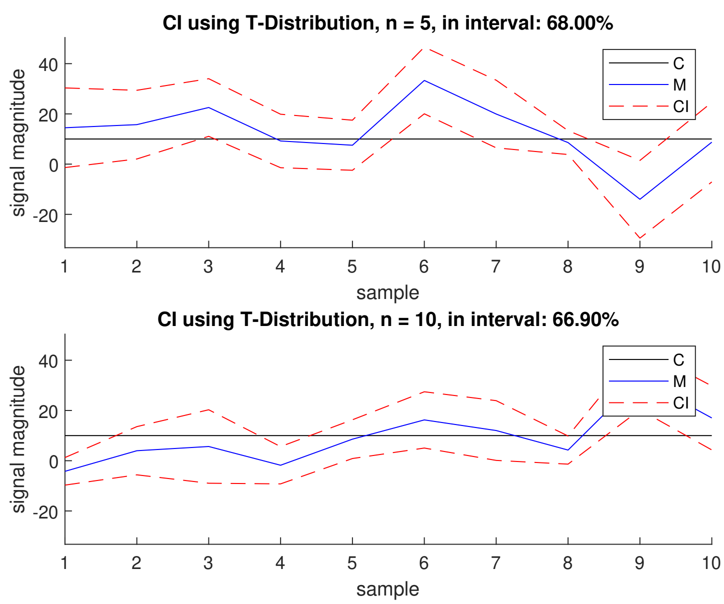

Figure 3: Case 3: CIs for the first 10 trials

The figures above shows the plots for case 2 and 3. It's evident that all the δ values change for each trial of m, as observed in (12) and (13). It can also be observed that the shapes of both CIs are the same, but the CIs in Figure 3 are slightly larger due to what we observed in Table 1. All in all, we have learned how to create more and more accurate CIs by knowing more information about the data, since case 1 had the closest accuracy to our 68.27% confidence, then case 3, and then case 2.

4 Conclusion

I have learned that even if you have a small sample size with a large variance, it is still possible to extract vital information regarding the data through statistics, analysis, and repeated trials. As the number of samples increased, its observed that the confidence intervals get closer to the required value of 68.27%. In addition, better methods for calculating confidence intervals will result in closer percentages to our required confidence, as we noticed that case 1 was the closest, then case 3, and then last was case 2. The performance of the $y_{\alpha/2}$ values increases as we know more about the distribution of our data, i.e., knowing the noise had a Gaussian distribution using the Student's T distribution for confidence intervals yielded higher accuracy.

5 Appendix

```
% seed: Arpad Voros => AAOO => 1056
rng(1056, 'twister');

% initialize some variables
% trials
m = 1000;
% recieved samples
n = [5, 10];
% signal to noise ratio
snr = 0.125;
% symbol
s = 1;
% true channel coefficient
C = 10;
% sigma_n
sig_n = sqrt((abs(C)^2) / snr);

% initialize CI variables
% how confident we want to be
confident = 0.6827;
% alpha coefficient
alpha = 1 - confident;
% y value for cases 1 and 2
y = norminv(1 - (alpha / 2));

% some dimensions
% number of n's
num_n = length(n);
% sizes of our data for each value of n
sizes = [n', repmat(m, num_n, 1)];

% result cell arrays, used for plotting
M_cell = cell(num_n, 1);
int_known_cell = cell(num_n, 1);
int_est_cell = cell(num_n, 1);
int_t_cell = cell(num_n, 1);

% allocating memory for cell arrays
for instance = 1:num_n
    M_cell{instance} = zeros(sizes(instance, :));
    int_known_cell{instance} = zeros(2, m);
    int_est_cell{instance} = zeros(2, m);
    int_t_cell{instance} = zeros(2, m);
```

```
end

% allocating memory for the percentage times the estimated value
% falls into the confidence interval
percent_in_known = zeros(num_n, 1);
percent_in_est = zeros(num_n, 1);
percent_in_t = zeros(num_n, 1);
rmse = zeros(num_n, 1);
y_t = zeros(num_n, 1);

% simulate for all values of n
for instance = 1:num_n
    % set up simulation, calculate sample means and variances
    % noise
    N = normrnd(0, sig_n, sizes(instance, :));
    % recieved signal
    X = C*s + N;
    % sample mean
    M = sum(X) / n(instance);
    % sample variance, unbiased estimator for sigma_x squared
    S = sum((X - repmat(M, n(instance), 1)).^2) / (n(instance) - 1);
    % root mean square error
    rmse(instance) = sqrt(mean(mean((X - C).^2)));

    % store M
    M_cell{instance} = M;

    % CI using KNOWN variance of X (sigma_n^2)
    % Δ
    Δ_known = y * sig_n / sqrt(n(instance));
    % confidence interval, upper and lower bounds
    int_known = zeros(2, m);
    int_known(1, :) = M - Δ_known;
    int_known(2, :) = M + Δ_known;
    % percentage the true channel coefficient falls into interval
    percent_in_known(instance) = sum((C ≥ int_known(1, :)) & (C ≤ int_known(2, ...
        :))) / m;
    % store values
    int_known_cell{instance} = int_known;

    % CI using ESTIMATED variance of X (sample variance S)
    % Δ
    Δ_est = y * sqrt(S / n(instance));
    % confidence interval, upper and lower bounds
    int_est = zeros(2, m);
    int_est(1, :) = M - Δ_est;
    int_est(2, :) = M + Δ_est;
    % percentage the true channel coefficient falls into interval
    percent_in_est(instance) = sum((C ≥ int_est(1, :)) & (C ≤ int_est(2, :))) ...
        / m;
    % store values
    int_est_cell{instance} = int_est;

    % CI using Student's T distribution
    % Δ
    y_t(instance) = tinv(1 - (alpha / 2), n(instance) - 1);
    Δ_t = y_t(instance) * sqrt(S / n(instance));
    % confidence interval, upper and lower bounds
    int_t = zeros(2, m);
    int_t(1, :) = M - Δ_t;
    int_t(2, :) = M + Δ_t;
    % percentage the true channel coefficient falls into interval
```

```
    percent_in_t(instance) = sum((C ≥ int_t(1, :)) & (C ≤ int_t(2, :))) / m;
    % store values
    int_t_cell{instance} = int_t;
end

% amount to plot
num_plot = 10;
samp_plot = 1:num_plot;

% minima and maxima value arrays, fot plot range
min_vals = zeros(num_n, 1);
max_vals = zeros(num_n, 1);

% find minima and maxima, for plot range
for instance = 1:num_n
    min_vals(instance) = min([min(min(int_known_cell{instance}(1:(2 * ...
        num_plot)))); min(min(int_est_cell{instance}(1:(2 * num_plot)))); ...
        min(min(int_t_cell{instance}(1:(2 * num_plot))))]);
    max_vals(instance) = max([max(max(int_known_cell{instance}(1:(2 * ...
        num_plot)))); max(max(int_est_cell{instance}(1:(2 * num_plot)))); ...
        max(max(int_t_cell{instance}(1:(2 * num_plot))))]);
end

% finding boundaries for figures
factor = 0.05;
bounds = [min(min_vals) - abs((max(max_vals) - min(min_vals))  * factor), ...
    max(max_vals) + abs((max(max_vals) - min(min_vals)) * factor)];

% close all already open figures
close all;
ci_colors = ["red", "red", "red"];
% plotting all figures
for instance = 1:num_n
    % KNOWN
    figure(1);
    subplot(num_n, 1, instance);
    hold on;
    plot(samp_plot, C * ones(size(samp_plot)), 'Color', 'black');
    plot(samp_plot, M_cell{instance}(samp_plot), 'Color', 'blue');
    plot(samp_plot, int_known_cell{instance}(1, samp_plot), 'Color', ...
        ci_colors(1), 'LineStyle', '--');
    plot(samp_plot, int_known_cell{instance}(2, samp_plot), 'Color', ...
        ci_colors(1), 'LineStyle', '--');
    title(sprintf('CI using known variance of X (variance of the noise), n = ...
        %d, in interval: %.2f%%', n(instance), 100 * percent_in_known(instance)));
    legend('C', 'M', 'CI');
    xlabel('sample');
    ylabel('signal magnitude');
    ylim(bounds);
    hold off;
    if instance == num_n
        print -depsc ci_known.eps
    end

    % ESTIMATED
    figure(2);
    subplot(num_n, 1, instance);
    hold on;
    plot(samp_plot, C * ones(size(samp_plot)), 'Color', 'black');
    plot(samp_plot, M_cell{instance}(samp_plot), 'Color', 'blue');
    plot(samp_plot, int_est_cell{instance}(1, samp_plot), 'Color', ...
        ci_colors(2), 'LineStyle', '--');
```

```
    plot(samp_plot, int_est_cell{instance}(2, samp_plot), 'Color', ...
        ci_colors(2), 'LineStyle', '--');
    title(sprintf('CI using estimated variance of X (sample variance S), n = ...
        %d, in interval: %.2f%%', n(instance), 100 * percent_in_est(instance)));
    legend('C', 'M', 'CI');
    xlabel('sample');
    ylabel('signal magnitude');
    ylim(bounds);
    hold off;
    if instance == num_n
        print -depsc ci_est.eps
    end

    % Student's T Dist
    figure(3);
    subplot(num_n, 1, instance);
    hold on;
    plot(samp_plot, C * ones(size(samp_plot)), 'Color', 'black');
    plot(samp_plot, M_cell{instance}(samp_plot), 'Color', 'blue');
    plot(samp_plot, int_t_cell{instance}(1, samp_plot), 'Color', ci_colors(3), ...
        'LineStyle', '--');
    plot(samp_plot, int_t_cell{instance}(2, samp_plot), 'Color', ci_colors(3), ...
        'LineStyle', '--');
    title(sprintf('CI using T-Distribution, n = %d, in interval: %.2f%%', ...
        n(instance), 100 * percent_in_t(instance)));
    legend('C', 'M', 'CI');
    xlabel('sample');
    ylabel('signal magnitude');
    ylim(bounds);
    hold off;
    if instance == num_n
        print -depsc ci_t.eps
    end
end
```