Calculation of Different Jitter Types in Sinusoidal Signals in Matlab

Oğuzhan Oğuz
National Magnetic Resonance Research Center
oguzhan.oguz@metu.edu.tr

***Abstract* - This article aims to analyze the calculation of absolute, relative, period, n-period and cycle-to-cycle jitter effects in sinusoidal signals. This article will include the theoretical and simulation process and results.**

Index Terms – Random jitter, Absolute jitter, Relative Jitter, Period Jitter, N Period Jitter, Cycle-to-cycle Jitter

I. Introduction

The amount of deviation of the zero crossing points of a periodic signal compared to the ideal zero crossing points is called jitter. The most important point of the term jitter is that, unlike phase noise, generally refers to a time domain process. It may contain a deterministic effect, unlike phase noise [1].

II. Creating Signal and Random Jitter

```
f0 = 100; % MHz
fs = 300; % MHz
tj = 3e-4; % us
tmax = 50; % us
dt = 1/fs; % us
t = 0:dt:tmax; % us
N = size(t,2);
tn = tj*randn(1,N);
fenvelope = chebwin(N,180)';
f = fenvelope.*sin(2*pi*f0*(t+tn)); % jittered signal
fx = fenvelope.*sin(2*pi*f0*t); % ideal signal
```

Fig. 1. Ideal signal and jittered signal generation code

Frequency domain analysis will also be performed to observe the jitter effect in the generated signals. In order to perform frequency domain operations correctly, signals produced with the chebwin function were used. Figure 1 shows how signals are generated.

III. Absolute Jitter

Absolute jitter can be defined as the difference between the positive edges of a jittered signal clock and the positive edges of the clock of the ideal state of the signal. Absolute jitter can be defined as a discrete-time random sequence. The k^{th} element is denoted a_k, and represents the time difference between the kth positive edge point of the jittered clock and the corresponding positive edge point of the ideal signal [2]. This definition can be seen in figure 2.

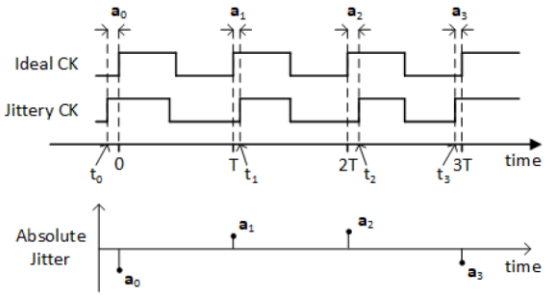

Fig. 2. Absolute jitter definition [3]

$$a_k = t_k - kT \quad (1)$$

However, the signals examined do not have clock information. Therefore, zero crossing points of the signals must be found and calculations must be made using these points.

IV. Determining Zero Crossing Values in Matlab

```
zero_crossings = [];

interval_start = 10; % us
interval_end = 10.05;

for k = 1:N-1

  if t(k) >= interval_start && t(k) <= interval_end

    if f(k) * f(k+1) < 0

      zero_crossing = t(k) – f(k) * (t(k+1) - t(k)) /
(f(k+1))
```

```
        zero_crossings = [zero_crossings, zero_crossing];
    end
  end
end
```

Fig. 3. Calculating zero crossing points in MATLAB

In this code, the expression t(k) >= interval_start && t(k) <= interval_end checks whether the t value under consideration is within the given interval. The expression f(k) * f(k+1) < 0 checks whether there is a sign change between the points f(k) and f(k+1). The expression zero_crossing = t(k) - f(k) * (t(k+1) - t(k)) / (f(k+1) - f(k)) uses linear interpolation to find the exact zero crossings. Using the code in figure 3, a result like in figure 4 is obtained.

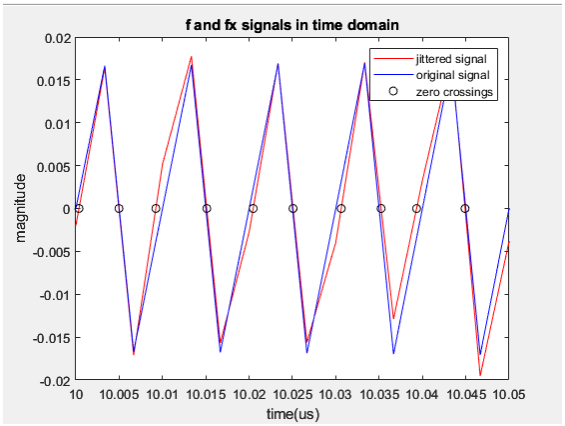

Fig. 4. Zero crossing points of the jittered signal

Zero crossing points are tried to be used like clocks. The rising edges of the clock are used when calculating absolute jitter. Likewise, among the zero crossings of the signal, only zero crossings that occur when the signal is increasing should be used. The result obtained in figure 4 includes both rising and falling zero crossing values. The code in figure 5 can be used to find only rising zero crossings.

```
zero_crossings = [];
for k = 1:length(t_limited)-1
    if f_limited(k) * f_limited(k+1) < 0 && f_limited(k+1) > f_limited(k)
        zero_crossing = t_limited(k) - f_limited(k) * (t_limited(k+1) - t_limited(k)) / (f_limited(k+1) - f_limited(k));
        zero_crossings = [zero_crossings, zero_crossing];
end end
```

Fig. 5. Calculating zero crossing points only for rising part in MATLAB

By adding the code f_limited(k+1) > f_limited(k), only rising zero crossings can be selected as figure 6.

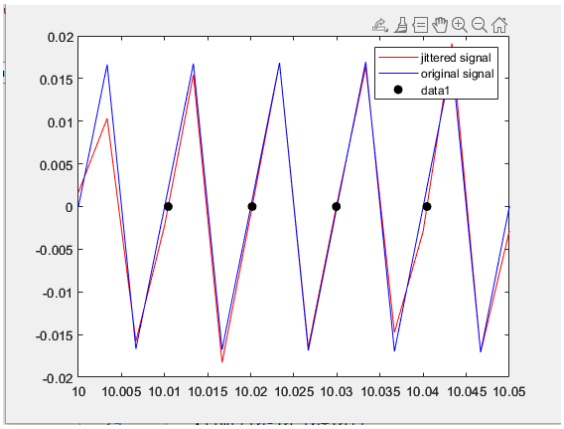

Fig. 6. Rising zero crossing points of the jittered signal

Linear interpolation is a method to estimate an unknown value within two known values on a linearly. In this case, linear interpolation ensures that the approximated zero crossing points are more exact.

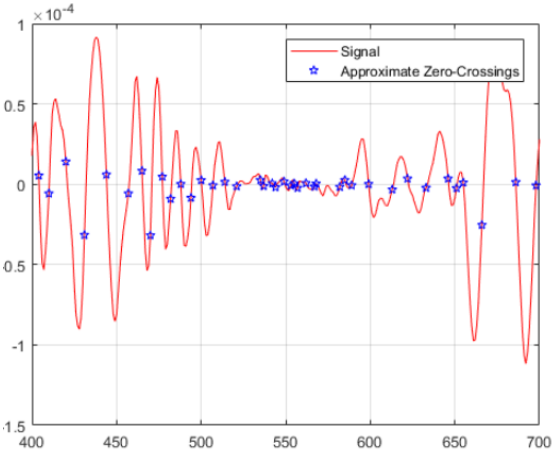

Fig. 7. A zero crossing approximation example without interpolation [4]

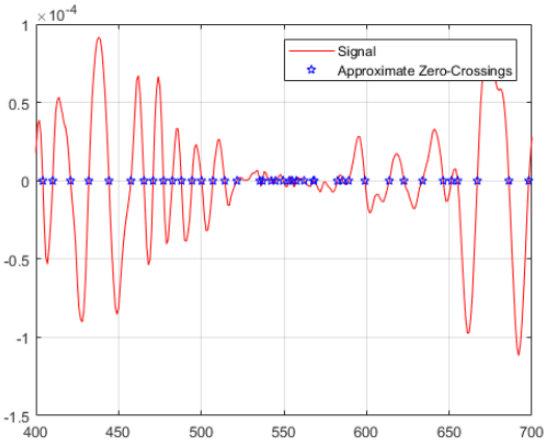

Fig. 8. A zero crossing approximation example with interpolation [4]

First, the desired range is determined. The "t" value is used for consecutive time vectors and the "f" value is used for signal values. The distance between two points can be calculated as:

$$m = \frac{f(k+1)-f(k)}{t(k+1)-t(k)} \quad (2)$$

m value gives the rate of change of the signal between t(k) and t(k+1). Then, the equation of the line connecting two points can be written as:

$$f(t) = f(k) + m * (t - t(k)) \quad (3)$$

In order to find the zero crossing values, f(t) = 0 values should be examined.

$$0 = f(k) + m * (t - t(k)) \quad (4)$$

$$t = t(k) - \frac{f(k)}{m} \quad (5)$$

Substituting the slope m:

$$t = t(k) - \frac{f(k)*[t(k+1)-t(k)]}{f(k+1)-f(k)} \quad (6)$$

In MATLAB:

zero_crossing = t_limited(k) - f_limited(k) * (t_limited(k+1) - t_limited(k)) / (f_limited(k+1) - f_limited(k));

This process provides a more accurate estimate of the zero-crossing time than using only discrete sampling points.

V. Calculating Absolute Jitter

How to calculate absolute jitter can be seen in equation (1). The same formula was applied to MATLAB.

```
T = 1/f0;
k_values = 1:(length(zero_crossings) - 2); % k values for the marked zero crossings excluding the first and last
a_k = zeros(size(k_values));

for i = 1:length(k_values)
    kT = interval_start + k_values(i) * T;
    [~, idx] = min(abs(zero_crossings - kT));
    t_k = zero_crossings(idx);
    if t_k >= interval_end
        continue
    end
    a_k(i) = t_k - kT;
end
```

Fig. 9. Calculating absolute jitter in MATLAB

Figure 9 shows how the absolute jitter value is calculated. With the formula k_values = 1:(length(zero_crossings) - 2), the a_k values at the boundary are not calculated. This is because these two limit value calculations give errors very often. The kT values in (1) are calculated with the command kT = interval_start + k_values(i) * T . With the command [~, idx] = min(abs(zero_crossings - kT)), the zero crossing value corresponding to the kT value is found. For this, the absolute difference between each zero crossing and kT is calculated and then the index of the minimum difference is found. Then, t_k values are found with this index and the t_k = zero_crossings(idx) command. If the found t_k values are within the interval, it is calculated as a_k(i) = t_k - kT.

For example, for values interval_start = 10, interval_end = 10.5, the following results can be obtained:

```
f0 = 100; % MHz
fs = 300; % MHz
tj = 3e-4; % us
interval_start = 10;
interval_end = 10.5;
```

Fig. 10. Values for the given example

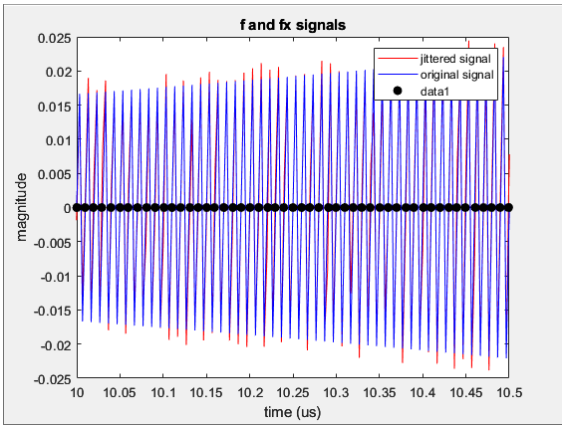

Fig. 11. Jittered and original signals plot for given interval

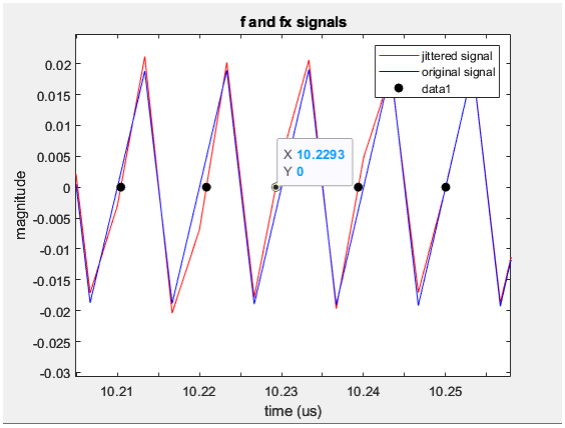

Fig. 12. Zero crossing value of jittered signal near the t = 10.23 point

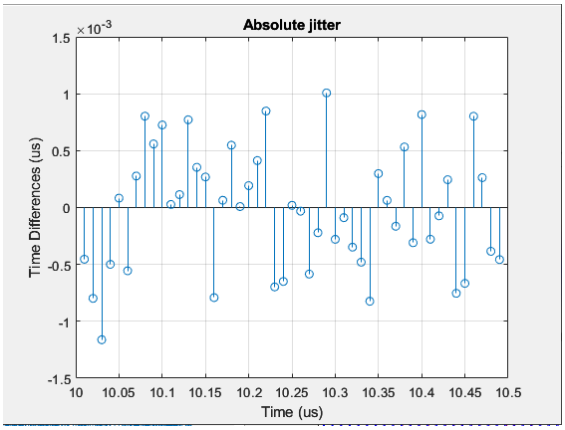

Fig. 13. Absolute jitter result for jittered and original signals for given interval

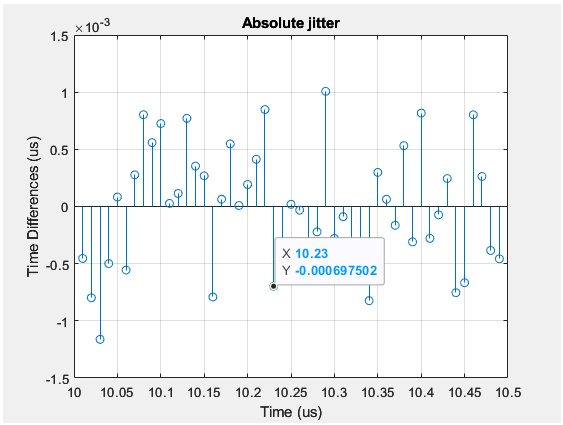

Fig. 14. Absolute jitter result at t = 10.23

The absolute jitter results obtained can be seen in figure 13. Original and jittered signals can be seen in figure 11. A random point was chosen to see the accuracy of the approximation. This point is point t = 10.23. In figure 12, the zero crossing point near t = 10.23 can be seen. This value is 10.2293. The value that should be in the ideal signal is 10.23. Therefore absolute jitter will be:

$$10.2293 - 10.23 = -0.0007 \quad (7)$$

In figure 14, this value is -0.000697502. The margin of error is -0.000002498. The margin of error as a percentage is 0.3568%. The error rate is very small, so it is a pretty good approximation.

VI. Relative Jitter

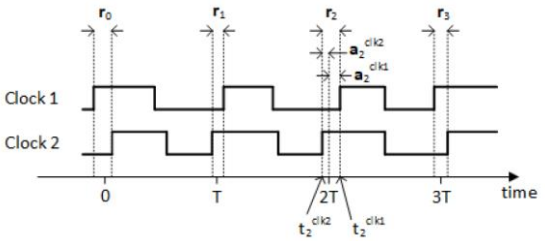

Fig. 15. Relative jitter definition [3]

Relative jitter can be defined as the difference between the positive edges of first jittered signal clock and the positive edges of the second jittered signal clock. An example of this situation can be seen in figure 15. Relative jitter, like absolute jitter, is a discrete-time random signal.

$r_k = t_k(\text{clock1}) - t_k(\text{clock2}) = a_k(\text{clock1}) - a_k(\text{clock2})$ (8)

The kth element of relative jitter can be calculated as in (8).

VII. Calculating Relative Jitter

```
tn1 = tj*randn(1,N);
tn2 = tj*randn(1,N);
f1 = fenvelope.*sin(2*pi*f0*(t+tn1)); %first jittered signal
f2 = fenvelope.*sin(2*pi*f0*(t+tn2)); %second jittered signal
```

Fig. 16. Two random jittered signals to be used in the calculation

This time, two random jittered signals are needed by definition. Therefore the f1 and f2 signals in figure 16 were created in matlab.

```
interval_start = 10;
interval_end = 10.5;
figure(1),plot(t,f1,'r',t,f2,'b'),legend("jittered signal 1","jittered signal 2");
xlim([interval_start, interval_end]);
title("f1 and f2 signals");
xlabel("time (us)");
ylabel("magnitude");
```

Fig. 17. Plotting the jittered signals

```
% Define the time range of interest
time_range_tart = interval_start;
time_range_nd = interval_end;

% Limit the time range of the signals for interpolation
time_indices = t >= time_range_start & t <= time_range_end;
t_limited = t(time_indices);
f1_limited = f1(time_indices);
f2_limited = f2(time_indices);

% Find zero crossings within the specified range for f1
zero_crossings_f1 = [];
for k = 1:length(t_limited)-1
   if f1_limited(k) * f1_limited(k+1) < 0 && f1_limited(k+1) > f1_limited(k)
      % Linear interpolation to find a more accurate zero crossing
      zero_crossing = t_limited(k) - f1_limited(k) * (t_limited(k+1) - t_limited(k)) / (f1_limited(k+1) - f1_limited(k));
      zero_crossings_f1 = [zero_crossings_f1, zero_crossing];
   end
end

% Find zero crossings within the specified range for f2
zero_crossings_f2 = [];
for k = 1:length(t_limited)-1
   if f2_limited(k) * f2_limited(k+1) < 0 && f2_limited(k+1) > f2_limited(k)
      % Linear interpolation to find a more accurate zero crossing
      zero_crossing = t_limited(k) - f2_limited(k) * (t_limited(k+1) - t_limited(k)) / (f2_limited(k+1) - f2_limited(k));
      zero_crossings_f2 = [zero_crossings_f2, zero_crossing];
   end
end

% Ensure the first zero crossings are at the interval start if necessary
if t_limited(1) == interval_start && abs(f1_limited(1)) < 1e-7
   zero_crossings_f1 = [interval_start, zero_crossings_f1];
end
if t_limited(1) == interval_start && abs(f2_limited(1)) < 1e-7
   zero_crossings_f2 = [interval_start, zero_crossings_f2];
end
```

Fig. 18. Estimating zero crossing points as in absolute jitter

Using the code in figure 18, the rising zero crossing points of both jitters are estimated. This process is the same as the process in absolute jitter.

```
% Calculate relative jitter r_k = a_k(CK1) -
a_k(CK2)
T = 1/f0;
k_values = 1:min(length(zero_crossings_f1),
length(zero_crossings_f2)) - 2; % k values for the
marked zero crossings excluding the first and last
r_k = zeros(size(k_values));

for i = 1:length(k_values)
    kT = interval_start + k_values(i) * T;
    [~, idx1] = min(abs(zero_crossings_f1 - kT));
    [~, idx2] = min(abs(zero_crossings_f2 - kT));
    t_k1 = zero_crossings_f1(idx1);
    t_k2 = zero_crossings_f2(idx2);
    if t_k1 >= interval_end || t_k2 >= interval_end
        continue
    end
    r_k(i) = (t_k1 - kT) - (t_k2 - kT);
end
```

Fig. 19. Relative jitter calculation

Using the code in Figure 19, tk1 and tk2 values are matched with rising edge zero crossing values. Then, for each value, the value of the second jittered signal is subtracted from the value of the first jittered signal and the relative jitter value is obtained. Interval start and end points have been removed from the calculation as they may cause problems.

```
% Plot the relative jitter
valid_indices = r_k ~= 0; % Filter out invalid
entries
figure(2);
stem(interval_start + k_values(valid_indices) * T,
r_k(valid_indices), '-o');
xlabel('Time (us)');
ylabel('Relative Time Differences (us)');
title("Relative jitter");
grid on;
```

Fig. 20. Plotting the relative jitter

The discrete-time signal result obtained can be plotted with the code in figure 20.

```
% Mark zero crossings on the original plot
figure(1);
hold on;
plot(zero_crossings_f1,
zeros(size(zero_crossings_f1)), 'ko',
'MarkerFaceColor', 'k');
plot(zero_crossings_f2,
zeros(size(zero_crossings_f2)), 'ko',
'MarkerFaceColor', 'g');
hold off;
```

Fig. 21. Marking the zero crossings

With the code in figure 21 and figure 17, zero crossing points can be clearly shown on the plot.

For example, for values interval_start = 10, interval_end = 10.5, the following results can be obtained:

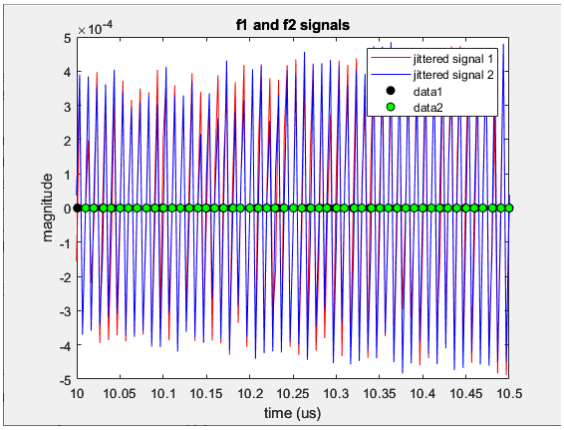

Fig. 22. f1 and f2 signals for given interval

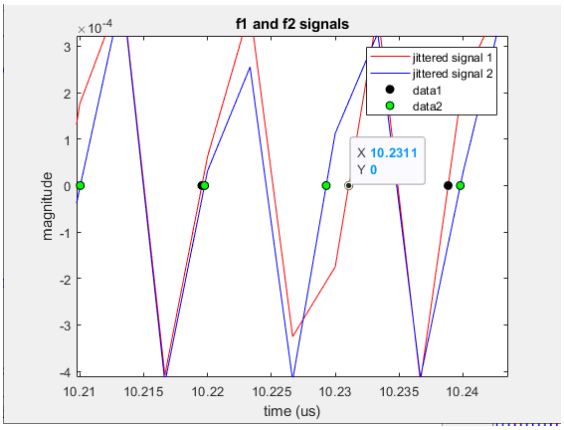

Fig. 23. Zero crossing point for first signal at t = 10.23

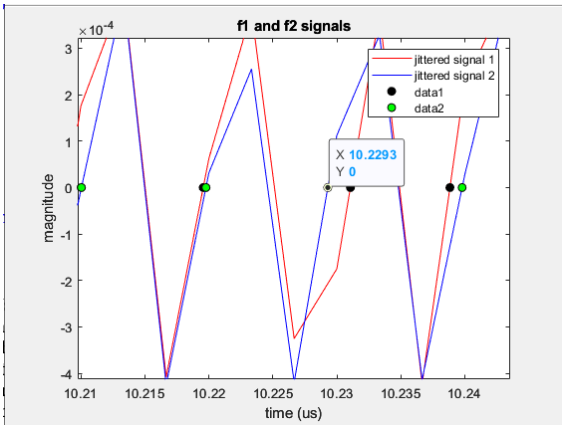

Fig. 24. Zero crossing point for second signal at t = 10.23

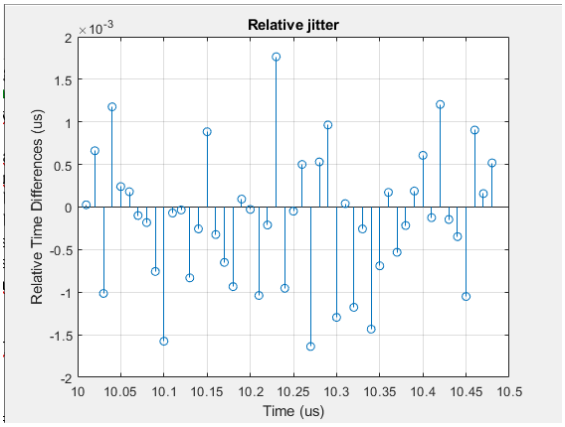

Fig. 25. Relative jitter results for f1 and f2 signals for given interval

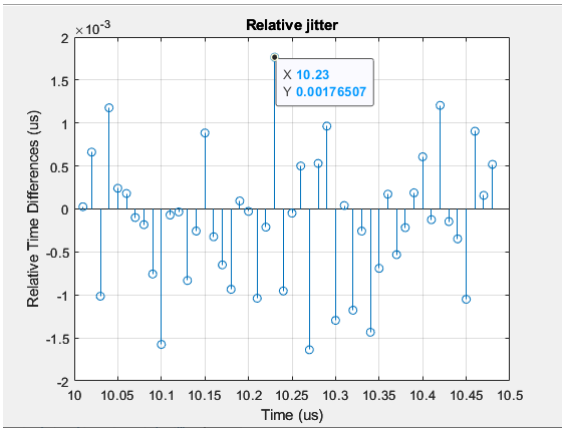

Fig. 26. Relative jitter result for f1 and f2 signals at t = 10.23

In figure 22, the obtained signals and the approximated zero crossing points can be seen. The result of the calculation made using these zero crossing points can be seen in figure 25.

To test this result, let's examine a random point. Let the random point be chosen as point t = 10.23. In figure 23, the zero crossing point of the first signal can be seen for the point t = 10.23. This value is 10.2311. In figure 24, the zero crossing point of the second signal can be seen for the point t = 10.23. This value is 10.2293. Therefore, relative jitter for this point should be:

$$10.2311 - 10.2293 = 0.0018 \quad (9)$$

When we look at figure 26, we see the result as 0.00176507. The margin of error is 0.00003493. If we look at it as a percentage, the margin of error is approximately 1.94%.

VIII. Period Jitter

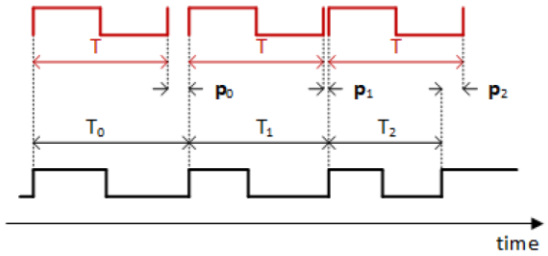

Fig. 27. Period (cycle) jitter definition [3]

Period jitter is defined as the difference between the non-ideal periods of the jitter signal and the nominal period. It is also called cycle jitter. Period jitter is also a discrete-time random signal. The calculation method can be seen in figure 27.

$$p_k = (t_{k+1} - t_k) - T = T_k - T \quad (10)$$

From (1), $t_k = a_k + kT$, so:

$$t_{k+1} = a_{k+1} + (k+1)T \quad (11)$$

$$t_k = a_k + kT \quad (12)$$

$$t_{k+1} - t_k = a_{k+1} - a_k + T \quad (13)$$

Finally,

$$t_{k+1} - t_k - T = a_{k+1} - a_k = p_k \quad (14)$$

Therefore:

$$p_k = a_{k+1} - a_k \quad (15)$$

It can be seen that from (15) period jitter can be easily calculated using absolute jitter.

IX. Calculating Period Jitter

The characteristics of the signals are as in figure 1. The operations up to Figure 9 are carried out exactly the same.

```
% Calculate period jitter p_k = a_{k+1} - a_k
p_k = diff(a_k);

% Adjust k_values to match the length of p_k
k_values_p = k_values(1:end-1);

figure(4);
stem(interval_start + k_values_p(valid_indices_p) * T, p_k(valid_indices_p), '-o');
xlabel('Time (us)');
ylabel('Time Differences (us)');
title("Period jitter");
grid on;
```

Fig. 28. Period jitter calculation in MATLAB

Period jitter can be calculated with the code in figure 28. Diff function computes the differences of consecutive elements. Specifically, diff(a_k) returns a_{k+1} - a_k. By k_values_p = k_values(1:end-1) code, length mismatch is prevented. Diff function creates one fewer element. Therefore such a procedure is needed. Let's examine the example for the t = (10,10.5) interval:

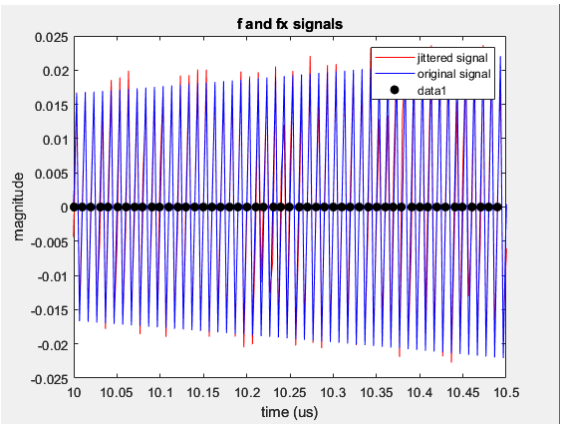

Fig. 29. f and fx signals for given interval

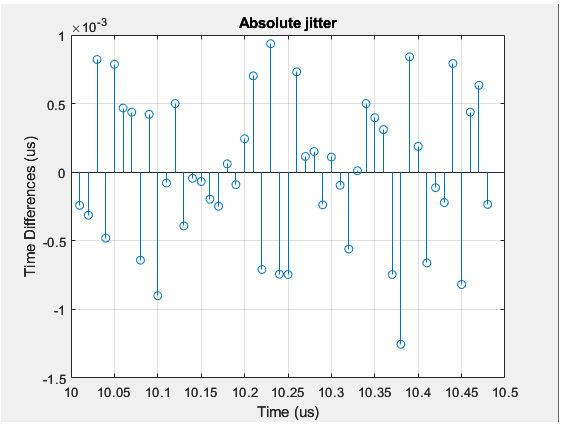

Fig. 30. Absolute jitter result for jittered and original signals for given interval

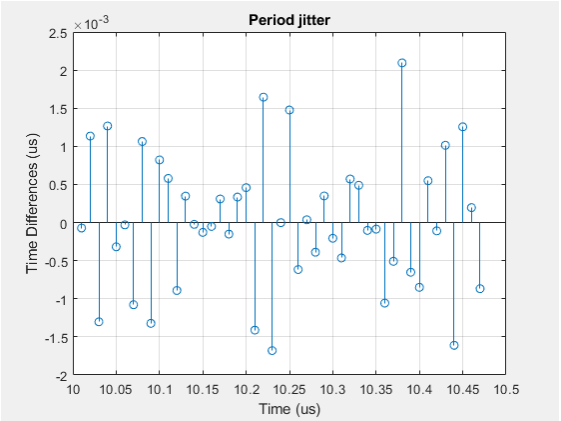

Fig. 31. Period jitter result for jittered and original signals for given interval

The signals obtained can be seen in Figure 29. Figure 30 shows the absolute jitter plot of these signals, and figure 31 shows the period jitter plot. Let's examine two points to check the accuracy of the calculations:

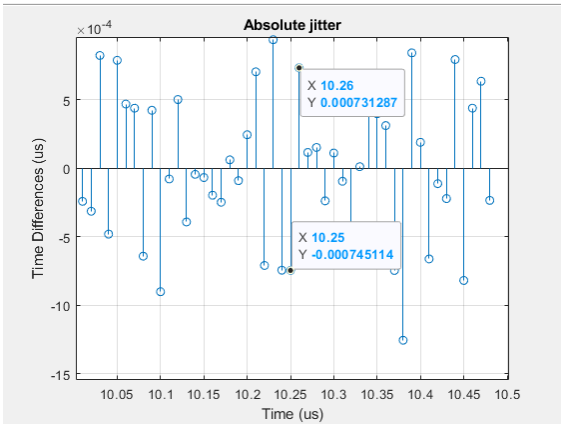

Fig. 32. Absolute jitter result for t = 10.25 and t = 10.26

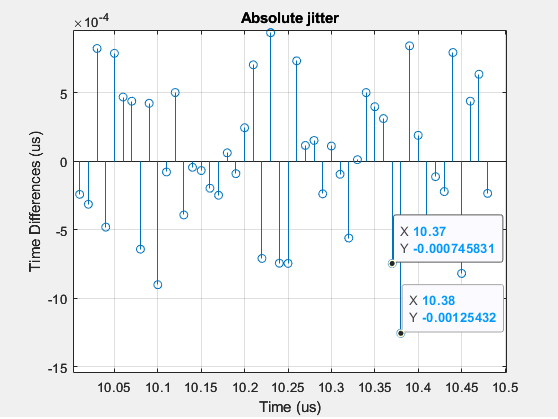

Fig. 33. Absolute jitter result for t = 10.37 and t = 10.38

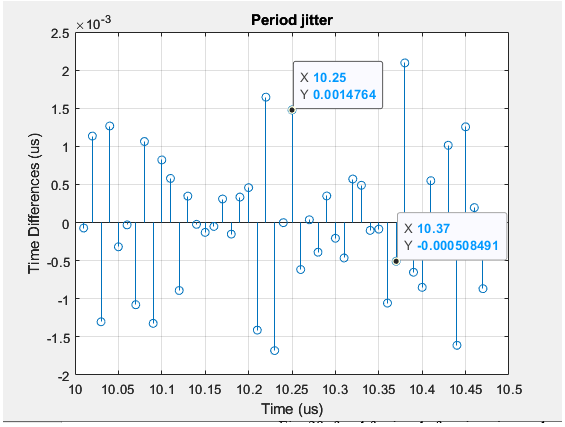

Fig. 34. Period jitter result for t = 10.25 and t = 10.37

If the values in Figures 32 and 33 are examined, for t = 10.25 and t = 10.37. From (15), $p_k = a_{k+1} - a_k$. Therefore for t = 10.25, $a_{k+1} = 0.000731287$ and $a_k = -0.000745114$. So:

$$p_{10.25} = a_{10.26} - a_{10.25} = 0.000731287+0.000745114 = 0.001476401 \quad (16)$$

For t = 10.37, a_{k+1} = -0.00125432 and a_k = -0.000745831. So:

$$p_{10.37} = a_{10.38} - a_{10.37} = -0.00125432 +0.000745831= -0.000508489 \quad (17)$$

From (16), it can be seen that the original value of $p_{10.25}$ is 0.001476401. In figure 34, the calculated value of $p_{10.25}$ was found to be 0.0014764. The error amount is 10^{-10}. So the error rate is approximately zero.

And from (17), it can be seen that the original value of $p_{10.37}$ is -0.000508489. In figure 34, the calculated value of $p_{10.37}$ was found to be -0.000508491. The error amount is $2*10^{-9}$. So the error rate is approximately zero again. As a result, the calculation is quite consistent.

X. N-Period Jitter

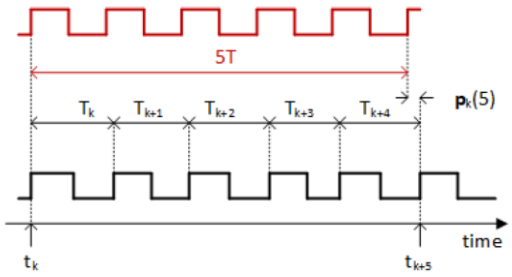

Fig. 35. N-Period (accumulation) jitter definition [3]

N-Period jitter is defined as the accumulation of period jitters in n consecutive nominals periods. It can be calculated as in figure 35.

$$p_k(N) = (t_{k+N} - t_k) - NT \quad (18)$$

Using the same calculations as for (15), the following result is obtained:

$$p_k(N) = a_{k+N} - a_k \quad (19)$$

XI. Calculating N-Period Jitter

```
% Calculate N-period jitter p_k(N) = a_{k+N} - a_k
N_period = 3;  % You can change this value to compute for different N
p_k_N = zeros(1, length(a_k) - N_period); % Preallocate the array for N-period jitter

for i = 1:length(p_k_N)
    p_k_N(i) = a_k(i+N_period) - a_k(i);
end

% Adjust k_values to match the length of p_k_N
k_values_N = k_values(1:end-N_period);

% Plot the N-period jitter
figure(4);
stem(interval_start + k_values_N * T, p_k_N, '-o');
xlabel('Time (us)');
ylabel('Time Differences (us)');
title(sprintf("N-Period jitter (N = %d)", N_period));
grid on;
```

Fig. 36. N-Period jitter calculation in MATLAB

Figure 36 shows how to calculate n-period jitter. The desired N value is selected with the N_period variable. "p_k_N = zeros(1, length(a_k) - N_period)" initializes an array to store the N-period jitter values. The length of p_k_N is length(a_k) - N_period since as can be seen in (17), a_{k+N} value is used for the p_k value. Therefore, for some k values at the end, a corresponding value will not be within the interval. Then, p_k values are calculated one by one with the for loop in the code. Finally, length match is made. Let's examine the example for the t = (10,10.5) interval:

For N = 3:

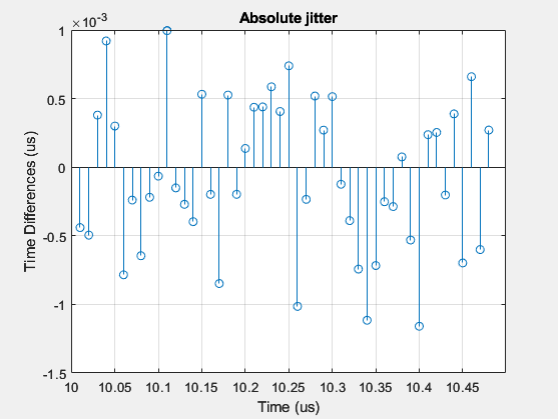

Fig. 37. Absolute jitter result for jittered and original signals for given interval

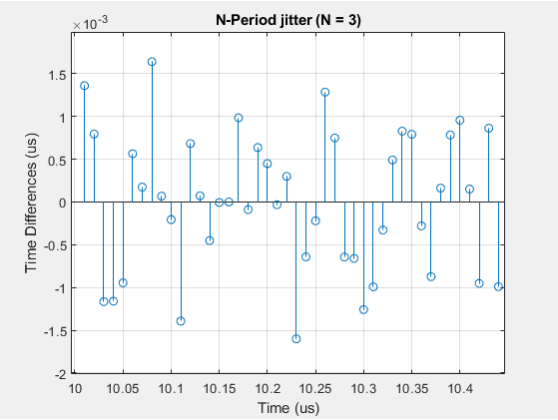

Fig. 38. N-period jitter result for jittered and original signals for given interval and for N = 3

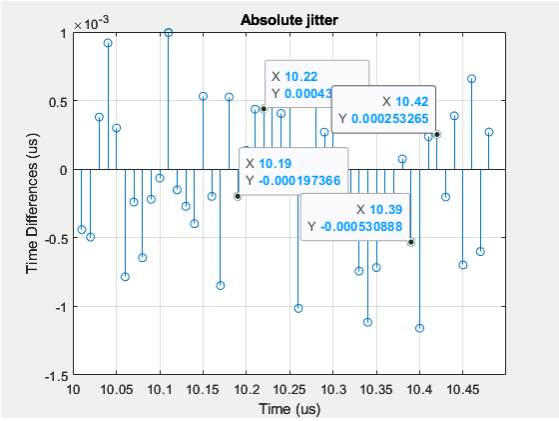

Fig. 39. Absolute jitter result for t = 10.19, t = 10.22, t = 10.39 and t = 10.42

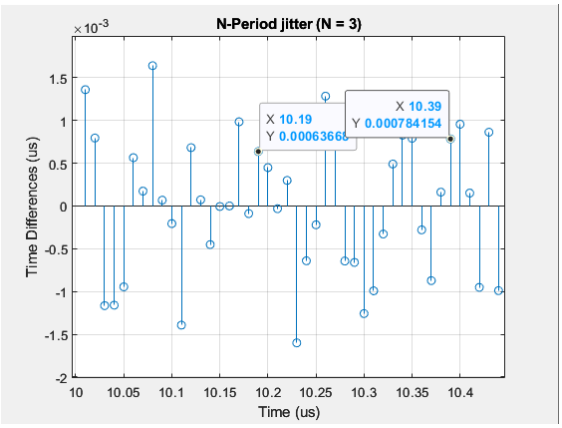

Fig. 40. N-period jitter result for t = 10.19 and t = 10.39

If the values given in figure 39 are examined it can be seen that for t = 10.19, a_k = -0.000197366, t = 10.22, a_{k+N} = 0.000439314, t = 10.39, a_k = -0.000530888 , t = 10.42, a_{k+N} = 0.000253265.

By using the formula in (19), the following results are obtained:

$$P_{10.19}(3) = a_{10.22} - a_{10.19} = 0.000439314 + 0.000197366 = 0.00063668 \quad (20)$$

$$P_{10.39}(3) = a_{10.42} - a_{10.39} = 0.000253265 + 0.000530888 = 0.000784153 \quad (21)$$

From (20), it can be seen that the original value of $p_{10.19}(3)$ is 0.00063668. In figure 40, the calculated value of $p_{10.19}$ was found to be 0.00063668. The error amount is 0. So the error rate is zero.

From (21), it can be seen that the original value of $p_{10.39}(3)$ is 0.000784153. In figure 40, the calculated value of $p_{10.39}$ was found to be 0.000784154. The error amount is -10^{-10}. So the error rate is approximately zero. As a result, the calculation is quite consistent.

For N = 5:

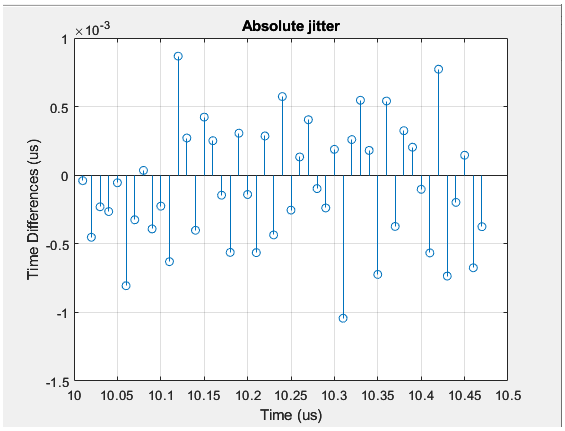

Fig. 41. Absolute jitter result for jittered and original signals for given interval

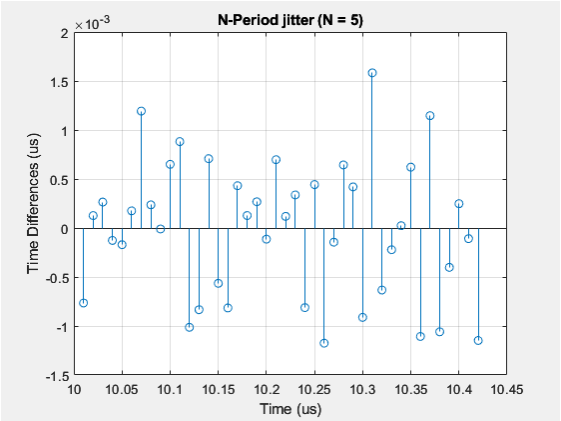

Fig. 42. N-period jitter result for jittered and original signals for given interval and for N = 5

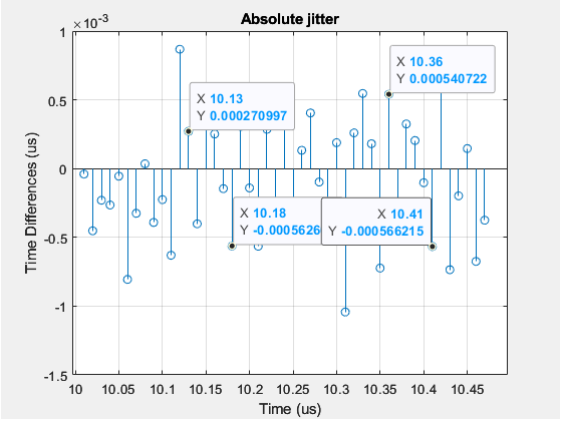

Fig. 43. Absolute jitter result for t = 10.13, t = 10.18, t = 10.36 and t = 10.41

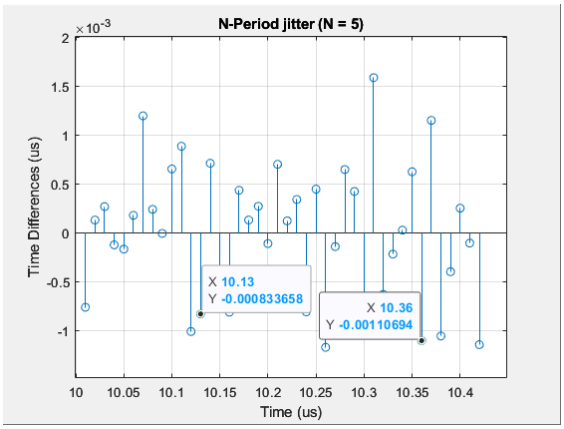

Fig. 44. N-period jitter result for t = 10.13 and t = 10.36

If the values given in figure 43 are examined it can be seen that for t = 10.13, a_k = 0.000270997, t = 10.18, a_{k+N} = -0.000582661, t = 10.36, a_k = 0.000540722 , t = 10.41, a_{k+N} = -0.000566215.

By using the formula in (19), the following results are obtained:

$$P_{10.13}(5) = a_{10.22} - a_{10.19} = -0.000582661 - 0.000270997 = -0.000853658 \quad (22)$$

$$P_{10.36}(5) = a_{10.42} - a_{10.39} = -0.000566215 - 0.000540722 = -0.001106937 \quad (23)$$

From (22), it can be seen that the original value of $p_{10.13}(5)$ is -0.000853658. In figure 44, the calculated value of $p_{10.13}$ was found to be -0.000833658. The error amount is 0. So the error rate is zero.

From (23), it can be seen that the original value of $p_{10.36}(5)$ is -0.001106937. In figure 44, the calculated value of $p_{10.36}$ was found to be -0.00110694. The error amount is $3*10^{-9}$. So the error rate is approximately zero. As a result, the calculation is quite consistent.

XII. Cycle-to-Cycle Jitter

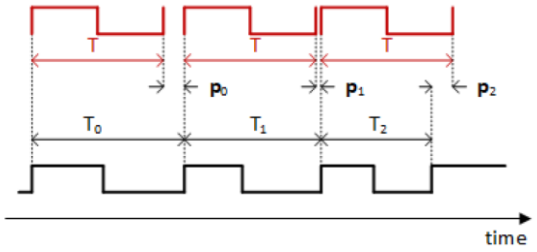

Fig. 45. Cycle-to-cycle jitter definition [3]

Cycle-to-cycle period jitter or cycle-to-cycle jitter is defined as the difference between two consecutive clock periods. And it shows how one period of the clock changes. [2] If we denote the k^{th} period with T_k, with the help of figure 45, cycle-to-cycle jitter is calculated as follows:

$$cc_k = T_{k+1} - T_k \quad (24)$$

From (10):

$$T_{k+1} = p_{k+1} + T \quad (25)$$

$$T_k = p_k + T \quad (26)$$

$$cc_k = T_{k+1} - T_k = p_{k+1} - p_k \quad (27)$$

From (15):

$$P_k = a_{k+1} - a_k \quad (28)$$

$$cc_k = p_{k+1} - p_k = a_{k+2} - a_{k+1} - (a_{k+1} - a_k) \quad (29)$$

$$cc_k = a_{k+2} - 2a_{k+1} + a_k \quad (30)$$

The equation in (27) shows how cycle-to-cycle jitter can be calculated using period jitter. Moreover, the equation in (30) shows how cycle-to-cycle jitter can be calculated using absolute jitter.

XIII. Calculating Cycle-to-Cycle Jitter With Using Absolute Jitter Result

```
% Calculate cycle-to-cycle jitter using absolute jitter: cck = ak+2 - 2*ak+1 + ak
cc_k = zeros(1, length(a_k) - 2); % Preallocate the array for cycle-to-cycle jitter

for i = 1:length(cc_k)
    cc_k(i) = a_k(i+2) - 2*a_k(i+1) + a_k(i);
end

% Adjust k_values to match the length of cc_k
k_values_cc = k_values(1:end-2);

% Plot the cycle-to-cycle jitter
figure(4);
stem(interval_start + k_values_cc * T, cc_k, '-o');
xlabel('Time (us)');
ylabel('Time Differences (us)');
title("Cycle-to-Cycle jitter");
grid on;
```

Fig. 46. Cycle-to-cycle jitter calculation with using absolute jitter in MATLAB

There was no process different from the previous processes. The formula derived in (30) was written in MATLAB. Length match was made and the result was plotted. Let's examine the accuracy of the calculation for the interval t (10, 10.5):

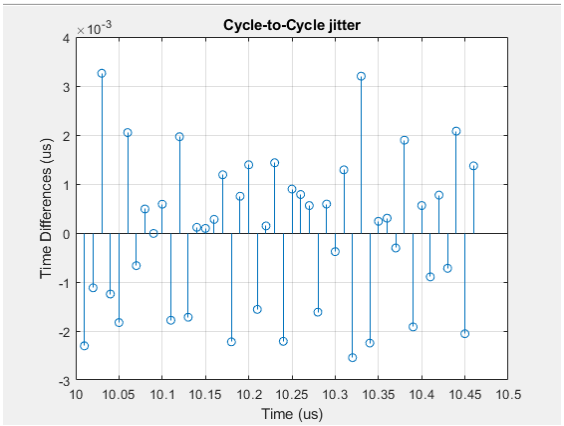

Fig. 47. Cycle-to-cycle jitter result for given interval

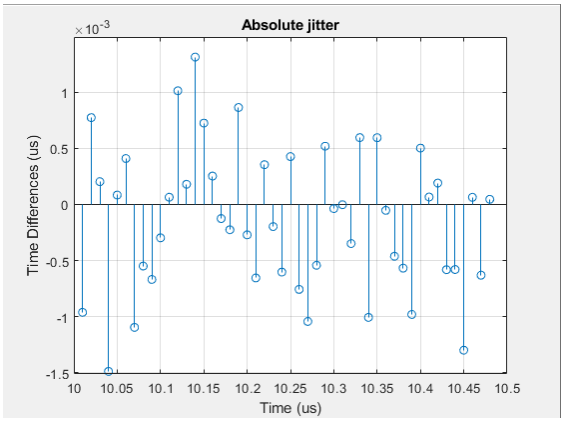

Fig. 48. Absolute jitter result for given interval

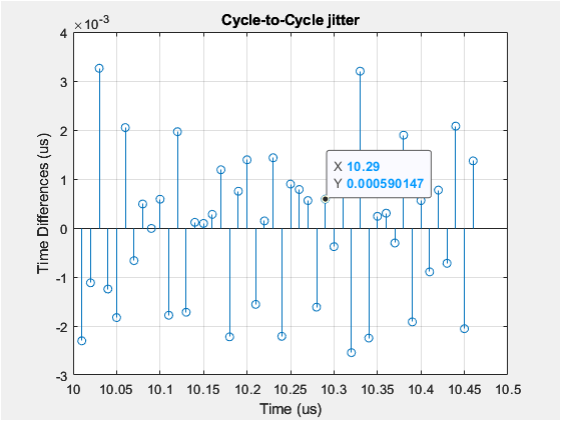

Fig. 49. Cycle-to-cycle jitter result for t = 10.29

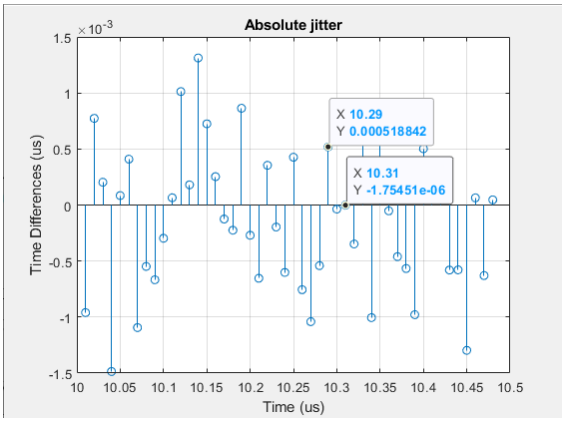

Fig. 50. Absolute jitter result for t = 10.29 and t = 10.31

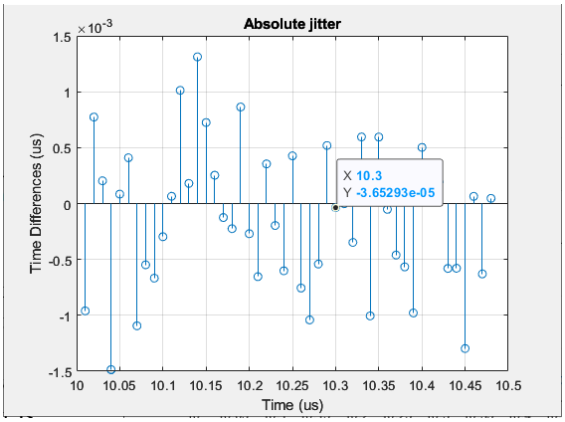

Fig. 51. Absolute jitter result for t = 10.3

From (30), $cc_k = a_{k+2} - 2a_{k+1} + a_k$. From figure 50 and 51, $a_{k+2} = 0.000518842$, $a_{k+1} = -0.0000365293$, $a_k = -0.00000175451$. Therefore:

$$cc_{10.29} = 0.000518842 + 2*0.0000365293 - 0.00000175451 = 0.00059014609 \quad (31)$$

From figure 49, $cc_{10.29} = 0.000590147$. Theoretically, $cc_{10.29}$ was found 0.00059014609 in (31). The error is $-9.1*10^{-10}$ which is almost zero.

As a result, the calculation is quite accurate.

XIV. Calculating Cycle-to-Cycle Jitter With Using Period Jitter Result

Period jitter was calculated in figure 28. Continuing from that point:

```
% Calculate cycle-to-cycle jitter using period jitter:
cck = p_{k+1} - p_k
cc_k = diff(p_k);

% Adjust k_values_p to match the length of cc_k
k_values_cc = k_values_p(1:end-1);

% Plot the cycle-to-cycle jitter
figure(5);
stem(interval_start + k_values_cc * T, cc_k, '-o');
xlabel('Time (us)');
ylabel('Time Differences (us)');
title("Cycle-to-Cycle jitter (using period jitter)");
grid on;
```

Fig. 52. Cycle-to-cycle jitter calculation with using period jitter in MATLAB

The same operations were carried out as in the previous processes. The formula derived in (27) was written in MATLAB. Length match was made and the result was plotted. Let's examine the accuracy of the calculation for the interval t (10, 10.5):

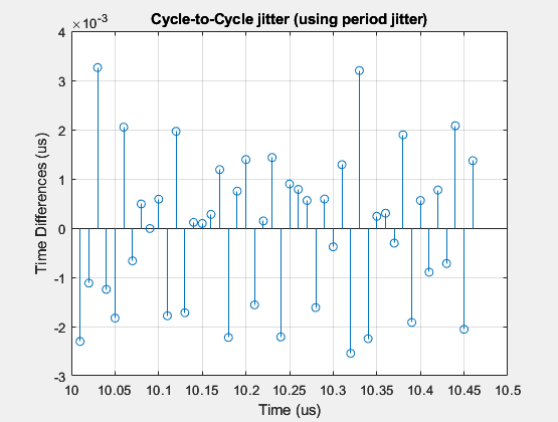

Fig. 53. Cycle-to-cycle jitter result for given interval

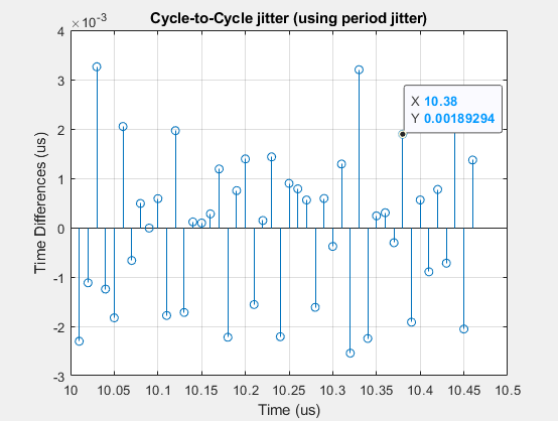

Fig. 54. Cycle-to-cycle jitter result for t = 10.38

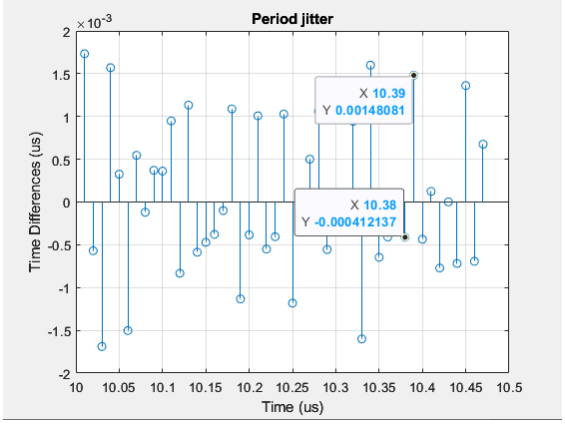

Fig. 55. Period jitter result for t = 10.38 and t = 10.39

From (27), $cc_k = p_{k+1} - p_k$. From figure 55, $p_{k+1} = 0.00148081$, $p_k = -0.000412137$

$$cc_{10.38} = 0.00148081+0.000412137 = 0.001892947 \quad (32)$$

From figure 54, $cc_{10.38} = 0.001829294$. Theoretically, $cc_{10.29}$ was found 0.001892947 in (32). The error is $-6.35*10^{-5}$. The error rate is 0.0335%

As a result, the calculation is quite accurate

XV. Conclusion

In this article, the absolute jitter, relative jitter, period jitter and n-period jitter phenomenons is examined both theoretically and in simulation. First, the concept of absolute jitter was explained theoretically. Then, two signals were created in MATLAB. While one of these signals was created as an ideal signal, the other signal was created as a randomly jittered version of the ideal signal. An attempt was made to make the theoretical calculation of absolute jitter suitable for the environment in MATLAB. Therefore, the zero crossing points of the jittered signal were tried to be approximated as precisely as possible. The approximation process and the interpolation concept used in approximation were explained in detail. Calculating absolute jitter using the zero crossing values found was explained in detail. The results obtained are shown.

Secondly, the concept of relative jitter was explained theoretically. Then, the difference between two different random jittered signals was calculated. In the calculation, processes in absolute jitter were used. Moreover the concepts of period jitter and n-period jitter were explained theoretically. Period jitter and n-period jitter results were obtained using the absolute jitter result. Finally, cycle-to-cycle jitter explained. Cycle-to-cycle jitter was calculated using period jitter and absolute jitter results. Since the calculations were based on approximation, the accuracy of all results was tested with the help of some values. It was observed that all calculations were quite consistent. Overall, this study provides a comprehensive framework for understanding and measuring jitters in sinusoidal signals.

REFERENCES

[1] Razavi, B. (2020). *Design of CMOS phase-locked loops: from circuit level to architecture level*. Cambridge University Press.

[2] Da Dalt, N., & Sheikholeslami, A. (2018). *Understanding jitter and phase noise: A circuits and systems perspective*. Cambridge University Press.

[3] Sheikholeslami, A. (2019). *Fundamental concepts in jitter and phase noise.*

[4] *Easy way of finding zero crossing of a function*. MATLAB Answers – MATLAB Central. (n.d.). https://www.mathworks.com/matlabcentral/answers/267222-easy-way-of-finding-zero-crossing-of-a-function