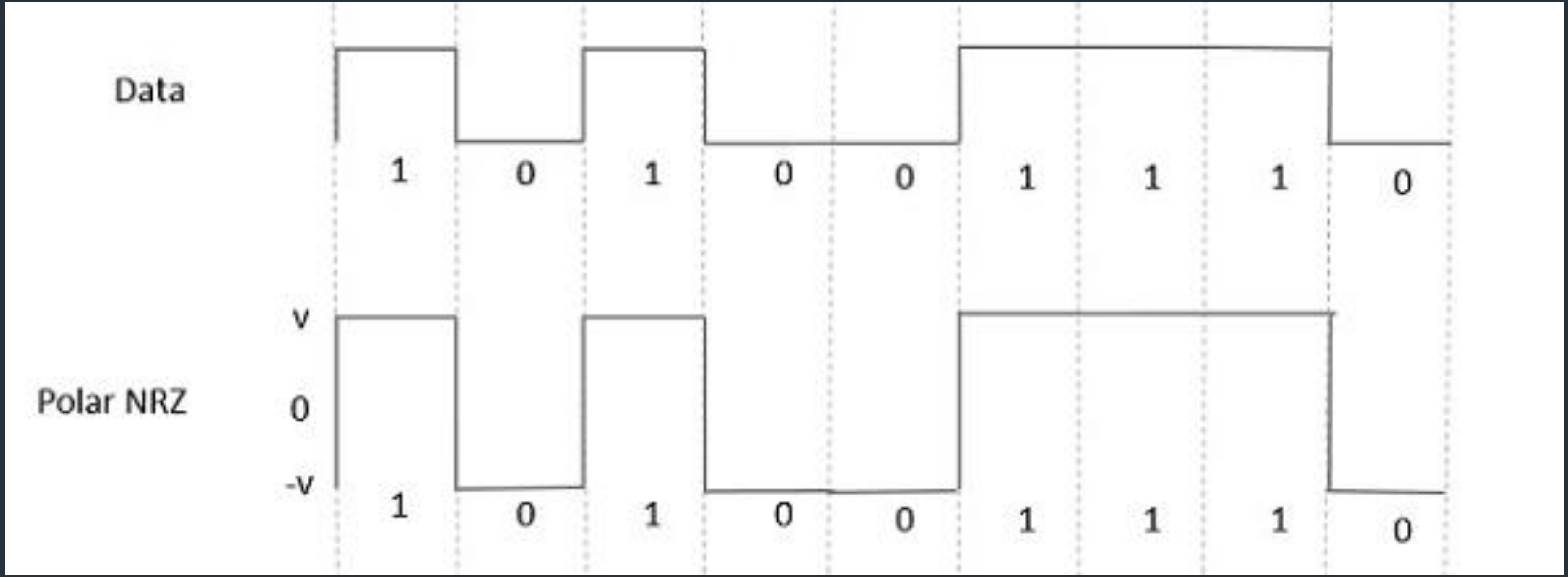
Polar Line Coding

- ❑Polar NRZ – level
- ❑Polar NRZ – invert
- ❑Polar RZ

❑ Polar NRZ – Level

As its name suggests polar which means it will have both positive and negative values for voltages or amplitude, it is quite like the NRZ scheme but, here we have NRZ-L (i.e. NRZ-Level) and NRZ-I (i.e. NRZInvert).

Let's see how these are represented:



In the above diagram, we can simply notice that the high volt is for logical 0 and the low volt is for logical 1. This is the representation of NRZ-Level.

Step 1: Define the Binary Sequence

```
binary_sequence = [1 0 1 1 0 0 1 0]; % Example sequence
```

- ✓ This is the input sequence of bits you want to encode in the polar NRZ format.
- ✓ Each 1 will be represented by a high signal (e.g., 1), and each 0 by a low signal (e.g., -1).
- ✓ Replace this array with your desired binary data.

Step 2: Set Parameters

```
bit_duration = 1; % Duration of each bit in seconds
fs = 1000; % Sampling frequency in Hz
t = 0:1/fs:bit_duration; % Time vector for one bit
```

- ✓ bit_duration: Specifies how long each bit lasts in seconds.
- ✓ fs: Sampling frequency determines how many samples are taken per second. A higher fs provides smoother curves.
- ✓ t: Creates a time vector for one bit, with intervals of 1/fs.

Step 3: Initialize the Signal

```
polar_nrz_signal = [];
```

- ✓ nrz_signal: This will store the final signal. Initially, it’s empty and will be filled bit by bit.

Step 4: Loop Through Each Bit to Generate the Signal

```
for bit = binary_sequence
    if bit == 1
        polar_bit = ones(1, length(t_bit)); % Positive level for '1'
    else
        polar_bit = -ones(1, length(t_bit)); % Negative level for '0'
    end
    polar_nrz_signal = [polar_nrz_signal, polar_bit]; % Append to the signal
end
```

- ✓ Loop through each bit in binary_sequence.
- ✓ If the bit is 1, append a high-level signal (array of ones) to nrz_signal.
- ✓ If the bit is 0, append a low-level signal (array of zeros) to nrz_signal.
- ✓ The use of ones and zeros ensures the signal has a constant level for the duration of each bit.

Step 5: Create the Time Axis

```
total_time = length(binary_sequence) * bit_duration; % Total signal duration
time_axis = linspace(0, total_time, length(polar_nrz_signal));
```

- ✓ total_time: Calculate the total duration of the signal by multiplying the number of bits by bit_duration.
- ✓ time_axis: Create a time array that spans the entire signal, ensuring proper alignment for plotting.

Step 6: Plot the Signal

```
figure;
plot(time_axis, nrz_signal, 'LineWidth', 2);
ylim([-0.5, 1.5]);
xlabel('Time (s)');
ylabel('Amplitude');
title(‘Polar Non-Return-to-Zero (NRZ) Signal');
grid on;
```

- ✓ plot: Plots the NRZ signal against the time axis.
- ✓ ylim: Limits the y-axis to make the plot easier to interpret.
- ✓ Labels (xlabel, ylabel, title) and grid are added for clarity.

Step 7: Annotate the Binary Sequence

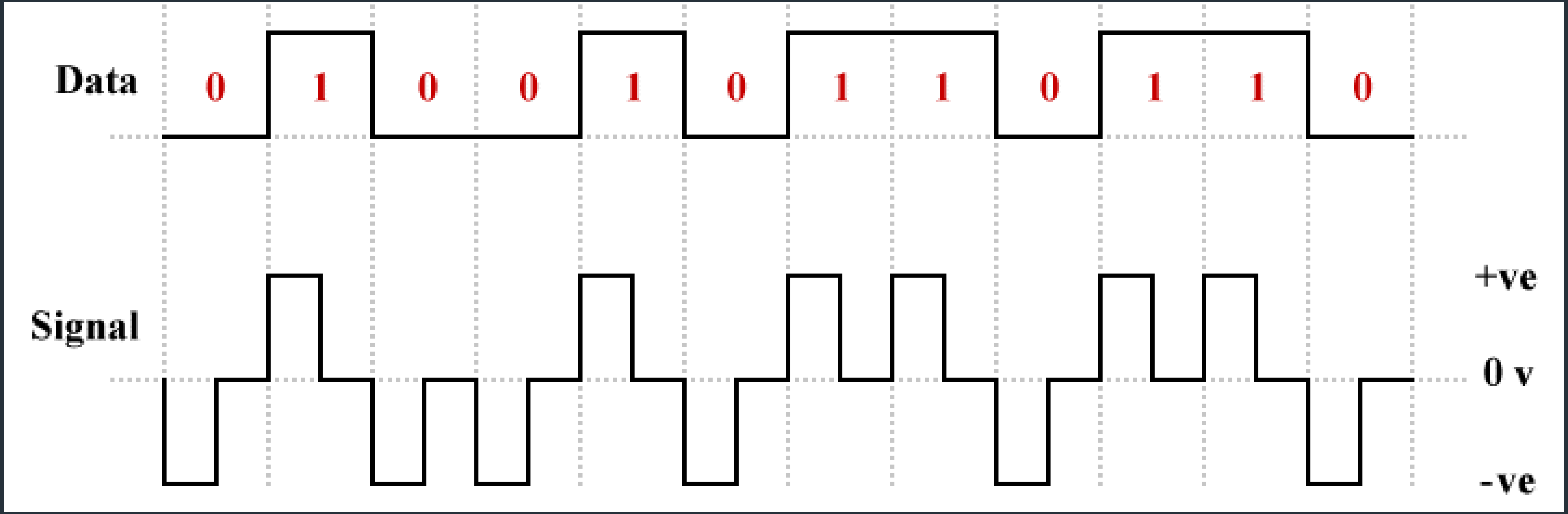
```
binary_labels = arrayfun(@num2str, binary_sequence, 'UniformOutput', false);
for i = 1:length(binary_sequence)
   text((i-0.5)*bit_duration, 1.2, binary_labels{i}, 'FontSize', 12, 'HorizontalAlignment', 'center');
end
```

- ✓ arrayfun: Converts each bit to a string so you can use it as a label.
- ✓ text: Places the bit value above each corresponding bit in the plot.

Complete Code:

```
clc;clear all;close all;
binary_sequence = [1 0 1 1 0 0 1 0];

bit_duration = 1; % Duration of each bit in seconds
sampling_rate = 1000; % Samples per second
t_bit = linspace(0, bit_duration, sampling_rate);

polar_nrz_signal = [];

for bit = binary_sequence
    if bit == 1
        polar_bit = ones(1, length(t_bit)); % Positive level for '1'
    else
        polar_bit = -ones(1, length(t_bit)); % Negative level for '0'
    end
    polar_nrz_signal = [polar_nrz_signal, polar_bit];
end

total_time = length(binary_sequence) * bit_duration;
time_axis = linspace(0, total_time, length(polar_nrz_signal));

figure;
plot(time_axis, polar_nrz_signal, 'LineWidth', 2);
ylim([-1.5, 1.5]); % Adjust y-axis limits for better visualization
xlabel('Time (s)');
ylabel('Amplitude');
title('Polar Non-Return-to-Zero (NRZ) Signal');
grid on;

binary_labels = arrayfun(@num2str, binary_sequence, 'UniformOutput', false);
for i = 1:length(binary_sequence)
    text((i-0.5)*bit_duration, 1.2, binary_labels{i}, 'FontSize', 12, 'HorizontalAlignment', 'center');
end
```

Polar RZ



- ✓ One solution to NRZ problem is the RZ scheme, which uses three values positive, negative, and zero.
- ✓ In this scheme signal goes to 0 in the middle of each bit.
- ✓ Note – The logic we are using here to represent data is that for bit 1 half of the signal is represented by +V and half by zero voltage and for bit 0 half of the signal is represented by -V and half by zero voltage.

1. Define the Binary Sequence

```
binary_sequence = [1 0 1 1 0 0 1 0];
```

- ✓ Input the binary data you want to encode in polar RZ format.
- ✓ Modify this sequence to test with different binary values.

2. Set Parameters

```
bit_duration = 1;
sampling_rate = 1000;
t_bit = linspace(0, bit_duration, sampling_rate);
```

- ✓ bit_duration: Time allocated to each bit in the signal.
- ✓ sampling_rate: Determines the resolution of the signal.
- ✓ t_bit: Creates a time vector for one bit.

3. Initialize the Signal

```
polar_rz_signal = [];
```

- ✓ An empty array to store the entire polar RZ signal.

4. Generate the RZ Signal

```
for bit = binary_sequence
    if bit == 1
        polar_bit = [ones(1, length(t_bit)/2), zeros(1, length(t_bit)/2)]; % Positive for first half, zero for second
    else
        polar_bit = [-ones(1, length(t_bit)/2), zeros(1, length(t_bit)/2)]; % Negative for first half, zero for second
    end
    polar_rz_signal = [polar_rz_signal, polar_bit]; % Append to the signal
end
```

- ✓ Loop through each bit in the binary sequence:
- ✓ For 1: High level for the first half, then return to zero.
- ✓ For 0: Always low level.
- ✓ Append the generated signal for each bit to the final signal array.

5. Create the Time Axis

```
total_time = length(binary_sequence) * bit_duration;
time_axis = linspace(0, total_time, length(polar_rz_signal));
```

- ✓ Compute the total duration of the signal.
- ✓ Create a time array for plotting the entire signal.

6. Plot the RZ Signal

```
figure;
plot(time_axis, polar_rz_signal, 'LineWidth', 2);
ylim([-1.5, 1.5]); % Adjust y-axis limits for better visualization
xlabel('Time (s)');
ylabel('Amplitude');
title('Polar Return-to-Zero (RZ) Signal');
grid on;
```

- ✓ Plot the signal against the time axis.
- ✓ Use ylim to control the range of the y-axis for clear visualization.

7. Annotate the Binary Sequence

```
binary_labels = arrayfun(@num2str, binary_sequence, 'UniformOutput', false);
for i = 1:length(binary_sequence)
   text((i-0.5)*bit_duration, 1.2, binary_labels{i}, 'FontSize', 12, 'HorizontalAlignment', 'center');
end
```

- ✓ Add text above each bit interval to show the corresponding binary value.

Complte Code:

```
clc;clear all;close all;

binary_sequence = [1 0 1 1 0 0 1 0];
bit_duration = 1;
sampling_rate = 1000;
t_bit = linspace(0, bit_duration, sampling_rate);

polar_rz_signal = [];

for bit = binary_sequence
    if bit == 1
        polar_bit = [ones(1, length(t_bit)/2), zeros(1, length(t_bit)/2)];
    else
        polar_bit = [-ones(1, length(t_bit)/2), zeros(1, length(t_bit)/2)];
    end
    polar_rz_signal = [polar_rz_signal, polar_bit];
end

total_time = length(binary_sequence) * bit_duration;
time_axis = linspace(0, total_time, length(polar_rz_signal));

figure;
plot(time_axis, polar_rz_signal, 'LineWidth', 2);
ylim([-1.5, 1.5]); % Adjust y-axis limits for better visualization
xlabel('Time (s)');
ylabel('Amplitude');
title('Polar Return-to-Zero (RZ) Signal');
grid on;

binary_labels = arrayfun(@num2str, binary_sequence, 'UniformOutput', false);
for i = 1:length(binary_sequence)
    text((i-0.5)*bit_duration, 1.2, binary_labels{i}, 'FontSize', 12, 'HorizontalAlignment', 'center');
end
```

Thank You!