```
#
# ============================================================================
# --- 1. INSTALL LIBRARIES ---
# Installs ffmpeg for audio conversion and pydub for handling audio files.
# The output is suppressed to keep the notebook clean.
#
# ============================================================================
!apt-get install -y ffmpeg > /dev/null 2>&1
!pip install pydub tqdm -q

#
# ============================================================================
# --- 2. IMPORT LIBRARIES ---
#
# ============================================================================
import numpy as np
from scipy.io.wavfile import write, read
import matplotlib.pyplot as plt
from tqdm import tqdm
from IPython.display import Audio, display
from google.colab import output
from base64 import b64decode
from pydub import AudioSegment
import io

#
# ============================================================================
# --- 3. CONFIGURATION PARAMETERS ---
# These are "safe" settings that should produce a clear result.
#
# ============================================================================
RECORD_SECONDS = 5
# Note: AUDIO_SAMPLE_RATE is no longer a global constant for processing.
# The actual sample rate is determined dynamically from the recording.
FILENAME_BROWSER_RECORDING = "recorded_audio_from_browser.webm"
FILENAME_ORIGINAL = "original_audio.wav"
FILENAME_RECEIVED = "received_audio.wav"

# Signal processing parameters
CARRIER_FREQ = 4000
BIT_RATE = 800
RF_SAMPLE_RATE = 10 * CARRIER_FREQ
AMPLITUDE = 1.0

# Set a high SNR for a clear initial test. Change to 10 or 5 to hear more noise.
SNR_DB = 30

#
# ============================================================================
# --- 4. HELPER FUNCTIONS ---
#
# ============================================================================
```

```
def record_audio_colab(filename, seconds):
"""Records audio from the browser microphone in Colab using JavaScript."""
RECORD_JS = f"""
const sleep = time => new Promise(resolve => setTimeout(resolve, time));
const b2text = blob => new Promise(resolve => {{
const reader = new FileReader();
reader.onloadend = e => resolve(e.target.result);
reader.readAsDataURL(blob);
}});
var record = time => new Promise(async resolve => {{
stream = await navigator.mediaDevices.getUserMedia({{ audio: true }});
recorder = new MediaRecorder(stream);
chunks = [];
recorder.ondataavailable = e => chunks.push(e.data);
recorder.start();
await sleep(time);
recorder.onstop = async ()=> {{
blob = new Blob(chunks);
text = await b2text(blob);
resolve(text);
}};
recorder.stop();
}});
"""
try:
print(f"🎤 Starting {seconds} second recording... Speak into your microphone.")
# This JS captures the audio and returns it as a base64 string
display(output.eval_js(RECORD_JS, timeout_sec=seconds + 2))
audio_b64 = output.eval_js(f'record({seconds * 1000})')
audio_bytes = b64decode(audio_b64.split(',')[1])
# Write the raw browser output (likely WebM) to a file
with open(filename, 'wb') as f:
f.write(audio_bytes)
print(f"✅ Browser audio recorded and saved as '{filename}'")
return True
except Exception as e:
print(f"\n❌ Audio recording failed. Please make sure you clicked 'Allow' for microphone access.")
print(f"Error: {e}")
return False

def audio_to_bits(audio_file):
"""
Reads a WAV file, converts it to a bit stream, and crucially, returns
the actual sample rate of the file.
"""
print("🔄 Converting audio to bit stream...")
sample_rate, audio_data = read(audio_file)
if audio_data.ndim > 1:
audio_data = audio_data.mean(axis=1)
bit_stream = ''.join([format(sample.view(np.uint16), '016b') for sample in
audio_data.astype(np.int16)])
print(f"✅ Conversion complete. Detected sample rate: {sample_rate} Hz")
return bit_stream, audio_data.astype(np.int16), sample_rate
```

```
def modulate_bpsk(bit_stream):
"""Modulates a bit stream using BPSK."""
print("🔄 Modulating bit stream using BPSK...")
samples_per_bit = int(RF_SAMPLE_RATE / BIT_RATE)
t = np.linspace(0, len(bit_stream) / BIT_RATE, len(bit_stream) * samples_per_bit, endpoint=False)
carrier = AMPLITUDE * np.cos(2 * np.pi * CARRIER_FREQ * t)
# Creates the +1/-1 signal that gets multiplied with the carrier
modulating_signal = np.repeat([1 if bit == '0' else -1 for bit in bit_stream], samples_per_bit)
modulated_signal = carrier * modulating_signal
print("✅ Modulation complete.")
return modulated_signal, modulating_signal

def simulate_channel(signal):
"""Adds Additive White Gaussian Noise to the signal."""
print(f"📡 Simulating noisy channel with SNR = {SNR_DB} dB...")
signal_power = np.mean(signal**2)
snr_linear = 10**(SNR_DB / 10.0)
noise_power = signal_power / snr_linear
noise = np.random.normal(0, np.sqrt(noise_power), len(signal))
received_signal = signal + noise
print("✅ Channel simulation complete.")
return received_signal

def demodulate_bpsk(received_signal):
"""Demodulates a BPSK signal to recover the bit stream."""
print("🔄 Demodulating received signal...")
samples_per_bit = int(RF_SAMPLE_RATE / BIT_RATE)
reference_carrier = np.cos(2 * np.pi * CARRIER_FREQ * np.linspace(0, 1/BIT_RATE,
samples_per_bit, endpoint=False))
recovered_bits = ""
num_bits = len(received_signal) // samples_per_bit
for i in tqdm(range(num_bits), desc="Demodulating"):
segment = received_signal[i*samples_per_bit:(i+1)*samples_per_bit]
correlation = np.sum(segment * reference_carrier)
recovered_bits += '0' if correlation > 0 else '1'
print("✅ Demodulation complete.")
return recovered_bits

def bits_to_audio(bit_stream, output_file, sample_rate):
"""
Converts a bit stream back into a WAV file using the correct, original
sample rate to ensure the speed and duration are correct.
"""
print(f"🔄 Converting bit stream back to audio at {sample_rate} Hz...")
extra_bits = len(bit_stream) % 16
if extra_bits != 0: bit_stream = bit_stream[:-extra_bits]
audio_samples = [np.array(int(bit_stream[i:i+16], 2), dtype=np.uint16).view(np.int16) for i in
range(0, len(bit_stream), 16)]
recovered_audio = np.array(audio_samples, dtype=np.int16)
write(output_file, sample_rate, recovered_audio)
print(f"✅ Recovered audio saved as '{output_file}'")
return recovered_audio

def visualize(original_audio, bit_stream, modulated_signal, received_signal, recovered_audio,
```

```
sample_rate):
"""Plots the signals at various stages of the process."""
print("🎨 Generating visualizations...")
plt.style.use('seaborn-v0_8-darkgrid')
fig, axs = plt.subplots(5, 1, figsize=(12, 18), constrained_layout=True)
fig.suptitle('Audio Transmission via BPSK Digital Modulation', fontsize=16)

# 1. Original Audio Waveform
time_audio = np.linspace(0, len(original_audio) / sample_rate, num=len(original_audio))
axs[0].plot(time_audio, original_audio, color='blue')
axs[0].set_title(f'1. Original Audio Waveform ({sample_rate} Hz)')
axs[0].set_xlabel('Time (s)'); axs[0].set_ylabel('Amplitude')

# 2. Modulated Signal (Zoomed In)
samples_to_show = int(5 * RF_SAMPLE_RATE / BIT_RATE)
time_modulated = np.linspace(0, 5/BIT_RATE, samples_to_show, endpoint=False)
axs[2].plot(time_modulated, modulated_signal[:samples_to_show], color='green')
axs[2].set_title('3. BPSK Modulated Signal (Zoomed In)')
axs[2].set_xlabel('Time (s)'); axs[2].set_ylabel('Amplitude')

# 3. Digital Bit Stream (first 100 bits)
bits_to_show = 100
axs[1].step(range(bits_to_show), [int(b) for b in bit_stream[:bits_to_show]], where='post',
color='purple')
axs[1].set_title(f'2. Digital Bit Stream (First {bits_to_show} Bits)')
axs[1].set_xlabel('Bit Index'); axs[1].set_ylabel('Value (0 or 1)'); axs[1].set_ylim(-0.1, 1.1);
axs[1].set_yticks([0, 1])

# 4. Received Signal with Noise (same zoomed view)
axs[3].plot(time_modulated, received_signal[:samples_to_show], color='orange')
axs[3].set_title(f'4. Received Signal with AWGN (SNR = {SNR_DB} dB)')
axs[3].set_xlabel('Time (s)'); axs[3].set_ylabel('Amplitude')

# 5. Recovered Audio Waveform
time_recovered = np.linspace(0, len(recovered_audio) / sample_rate, num=len(recovered_audio))
axs[4].plot(time_recovered, recovered_audio, color='cyan')
axs[4].set_title('5. Recovered Audio Waveform')
axs[4].set_xlabel('Time (s)'); axs[4].set_ylabel('Amplitude')

plt.show()

#
================================================================================
# --- 5. MAIN EXECUTION ---
#
================================================================================
if __name__ == "__main__":
# Step 1: Record audio from the browser.
if record_audio_colab(FILENAME_BROWSER_RECORDING, RECORD_SECONDS):

# Step 2: Convert the recorded file (e.g., WebM) to a proper WAV format.
print(f"🔄 Converting '{FILENAME_BROWSER_RECORDING}' to a proper WAV file...")
sound = AudioSegment.from_file(FILENAME_BROWSER_RECORDING)
sound.export(FILENAME_ORIGINAL, format="wav")
```

```
print(f"✅ Conversion successful.")

# --- DIAGNOSTIC STEP ---
# Play the initial recording to ensure it worked correctly.
print("\n--- 🔊 DIAGNOSTIC: Playing the source audio file ---")
display(Audio(FILENAME_ORIGINAL))

# Step 3: Start the simulation, capturing the actual sample rate.
bit_stream, original_audio_data, actual_sample_rate = audio_to_bits(FILENAME_ORIGINAL)

# Step 4: Run the modulation-demodulation chain.
modulated_signal, _ = modulate_bpsk(bit_stream)
received_signal = simulate_channel(modulated_signal)
recovered_bit_stream = demodulate_bpsk(received_signal)

# Step 5: Convert bits back to audio, passing the correct sample rate.
recovered_audio_data = bits_to_audio(recovered_bit_stream, FILENAME_RECEIVED,
actual_sample_rate)

# --- FINAL REPORT ---
print("\n--- 🏁 Process Complete 🏁 ---")
errors = sum(1 for a, b in zip(bit_stream, recovered_bit_stream) if a != b)
ber = errors / len(bit_stream) if len(bit_stream) > 0 else 0
print(f"Bit Error Rate (BER): {ber:.6f} ({errors} errors out of {len(bit_stream)} bits)")

# Step 6: Visualize results and play the final audio.
visualize(original_audio_data, bit_stream, modulated_signal, received_signal,
recovered_audio_data, actual_sample_rate)
print("\n🔊 Received Audio (after transmission and noise):")
display(Audio(FILENAME_RECEIVED))
```