Name: Abdelrahman abyad , Amr Mohsen, Daniel Fadi **Tut 2**

Project Report

```
% Load video
V = VideoReader('highway.avi');
a = read(V);
frames = get(V,'NumFrames');
% Initialize frame storage
ArrayOfFrames(frames) = struct('cdata', []);
ArrayOfFrames(1).cdata = a(:,:,:,1); % Store first frame
% Get frame dimensions
s = size(ArrayOfFrames(1).cdata);
bits_per_frame = s(1)*s(2)*8*3; % Total bits per frame (144×176×8×3 = 608256)
% Define coding parameters with only 20 values
error_probs = [0.001, 0.005, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, ...
0.09, 0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.18, 0.2];
% Define all trellis structures
trellises = struct(...
'rate_1_2', poly2trellis(3, [5 7]), ... % Rate 1/2 code
'rate_1_3', poly2trellis(3, [5 7 7]), ... % Rate 1/3 code
'rate_1_4', poly2trellis(3, [5 7 7 7]), ... % Rate 1/4 code
'rate_2_3', poly2trellis([2 2], [3 1 3; 1 2 2]) ... % Rate 2/3 code with K=3
);
% Define all puncturing patterns
puncture_patterns = struct(...
'rate_8_9', [1 1 1 0 1 0 1 0 0 1 1 0 1 0 1 0], ... % Keeps 9/16
'rate_4_5', [1 1 1 0 1 0 1 0 1 1 1 0 1 0 1 0], ... % Keeps 10/16
'rate_2_3', [1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0] ... % Keeps 12/16
);
throughput_rates = [1, 1/2, 1/3, 1/4, 8/9, 4/5, 2/3]; % Theoretical rates
% Initialize BER storage arrays
BER_uncoded = zeros(size(error_probs));
BER_conv_1_2 = zeros(size(error_probs));
BER_conv_1_3 = zeros(size(error_probs));
BER_conv_1_4 = zeros(size(error_probs));
BER_conv_2_3 = zeros(size(error_probs));
BER_punc_8_9 = zeros(size(error_probs));
BER_punc_4_5 = zeros(size(error_probs));
BER_punc_2_3 = zeros(size(error_probs));
% Process for each error probability
for p_idx = 1:length(error_probs)
fprintf("looping \n")
p = error_probs(p_idx);
```

```
% Initialize errors for this probability
errors_uncoded = 0;
errors_conv_1_2 = 0;
errors_conv_1_3 = 0;
errors_conv_1_4 = 0;
errors_conv_2_3 = 0;
errors_punc_8_9 = 0;
errors_punc_4_5 = 0;
errors_punc_2_3 = 0;
if(p == 0.001 || p == 0.1)
% Initialize output videos with p in filename
output_no_coding = VideoWriter(sprintf('no_coding_p%.6f.avi', p), 'Uncompressed
AVI');
output_conv_1_2 = VideoWriter(sprintf('conv_1_2_p%.6f.avi', p), 'Uncompressed AVI');
output_conv_1_3 = VideoWriter(sprintf('conv_1_3_p%.6f.avi', p), 'Uncompressed AVI');
output_conv_1_4 = VideoWriter(sprintf('conv_1_4_p%.6f.avi', p), 'Uncompressed AVI');
output_conv_2_3 = VideoWriter(sprintf('conv_2_3_p%.6f.avi', p), 'Uncompressed AVI');
output_punc_8_9 = VideoWriter(sprintf('punc_8_9_p%.6f.avi', p), 'Uncompressed AVI');
output_punc_4_5 = VideoWriter(sprintf('punc_4_5_p%.6f.avi', p), 'Uncompressed AVI');
output_punc_2_3 = VideoWriter(sprintf('punc_2_3_p%.6f.avi', p), 'Uncompressed AVI');
% Open all video writers
open(output_no_coding);
open(output_conv_1_2);
open(output_conv_1_3);
open(output_conv_1_4);
open(output_conv_2_3);
open(output_punc_8_9);
open(output_punc_4_5);
open(output_punc_2_3);
end
for i = 1:frames
% Extract current frame
ArrayOfFrames(i).cdata = a(:,:,:,i);
% Convert frame to binary
R = ArrayOfFrames(i).cdata(:,:,1);
G = ArrayOfFrames(i).cdata(:,:,2);
B = ArrayOfFrames(i).cdata(:,:,3);
% Convert the pixel values from uint8 to double for binary conversion
Rdouble = double(R);
Gdouble = double(G);
Bdouble = double(B);
% Convert each pixel value to its 8-bit binary representation
% The result is a matrix where each row corresponds to a pixel,
% and each column is a bit (LSB to MSB)
Rbin = de2bi(Rdouble, 8);
Gbin = de2bi(Gdouble, 8);
Bbin = de2bi(Bdouble, 8);
```

```
% Reshape the binary matrices into 3D arrays:
% Dimensions are [Height, Width, 8 bits] corresponding to image structure
% Assuming image size is 144 (rows) x 176 (columns)
R_bin_reshaped = reshape(Rbin, [144, 176, 8]);
G_bin_reshaped = reshape(Gbin, [144, 176, 8]);
B_bin_reshaped = reshape(Bbin, [144, 176, 8]);
% Flatten the 3D binary arrays into column vectors (streams of bits)
R_stream = reshape(R_bin_reshaped, [], 1);
G_stream = reshape(G_bin_reshaped, [], 1);
B_stream = reshape(B_bin_reshaped, [], 1);
binary_stream = [R_stream; G_stream; B_stream];
% Calculate number of 1024-bit packets (608256/1024 = 594 exactly)
num_packets = bits_per_frame / 1024;
% Initialize arrays to store processed packets
no_coding_bits = zeros(bits_per_frame, 1);
conv_1_2_bits = zeros(bits_per_frame, 1);
conv_1_3_bits = zeros(bits_per_frame, 1);
conv_1_4_bits = zeros(bits_per_frame, 1);
conv_2_3_bits = zeros(bits_per_frame, 1);
punc_8_9_bits = zeros(bits_per_frame, 1);
punc_4_5_bits = zeros(bits_per_frame, 1);
punc_2_3_bits = zeros(bits_per_frame, 1);
% Initializing the errors across frames
errors_uncoded = 0;
errors_conv_1_2 = 0;
errors_conv_1_3 = 0;
errors_conv_1_4 = 0;
errors_conv_2_3 = 0;
errors_punc_8_9 = 0;
errors_punc_4_5 = 0;
errors_punc_2_3 = 0;
% Process each packet
for pkt = 1:num_packets
% Extract current packet (1024 bits)
current_packet = binary_stream((pkt-1)*1024+1 : pkt*1024);
% 1. No channel coding
received_packet = bsc(current_packet, p); % 1×1024
no_coding_bits((pkt-1)*1024+1 : pkt*1024) = received_packet;
errors_uncoded = errors_uncoded + sum(current_packet ~= received_packet);
% 2. Conventional coding (all rates)
% Rate 1/2
encoded_1_2 = convenc(current_packet, trellises.rate_1_2);

% Simulate transmission over a Binary Symmetric Channel (BSC) with bit-flip
probability 'p'

noisy_1_2 = bsc(encoded_1_2, p);
```

```
% Decode the received noisy bitstream using Viterbi decoding

% trellises.rate_1_2 is the same trellis used for encoding % 34 is the traceback
depth

% 'trunc' mode resets the decoder state between packets (suitable for independent
packets)

% 'hard' indicates hard decision decoding (binary values)


decoded_1_2 = vitdec(noisy_1_2, trellises.rate_1_2, 34, 'trunc', 'hard');
% Truncate the decoded output to match the original packet size (1024 bits)
decoded_pkt_1_2 = decoded_1_2(1:1024);
% Store the decoded bits in the pre-allocated array at the correct position
conv_1_2_bits((pkt-1)*1024+1 : pkt*1024) = decoded_pkt_1_2;
% Calculate and accumulate the number of bit errors by comparing the original and
decoded packets
errors_conv_1_2 = errors_conv_1_2 + sum(current_packet ~= decoded_pkt_1_2);
% Rate 1/3
encoded_1_3 = convenc(current_packet, trellises.rate_1_3);
noisy_1_3 = bsc(encoded_1_3, p);
decoded_1_3 = vitdec(noisy_1_3, trellises.rate_1_3, 34, 'trunc', 'hard');
decoded_pkt_1_3 = decoded_1_3(1:1024);
conv_1_3_bits((pkt-1)*1024+1 : pkt*1024) = decoded_pkt_1_3;
errors_conv_1_3 = errors_conv_1_3 + sum(current_packet ~= decoded_pkt_1_3);
% Rate 1/4
encoded_1_4 = convenc(current_packet, trellises.rate_1_4);
noisy_1_4 = bsc(encoded_1_4, p);
decoded_1_4 = vitdec(noisy_1_4, trellises.rate_1_4, 34, 'trunc', 'hard');
decoded_pkt_1_4 = decoded_1_4(1:1024);
conv_1_4_bits((pkt-1)*1024+1 : pkt*1024) = decoded_pkt_1_4;
errors_conv_1_4 = errors_conv_1_4 + sum(current_packet ~= decoded_pkt_1_4);
% Rate 2/3 (with padding)
if mod(length(current_packet),2)~=0
padded_packet = [current_packet; 0];
else
padded_packet = current_packet;
end
encoded_2_3 = convenc(padded_packet, trellises.rate_2_3);
noisy_2_3 = bsc(encoded_2_3, p);
decoded_2_3 = vitdec(noisy_2_3, trellises.rate_2_3, 34, 'trunc', 'hard');
decoded_pkt_2_3 = decoded_2_3(1:1024);
conv_2_3_bits((pkt-1)*1024+1 : pkt*1024) = decoded_pkt_2_3;
errors_conv_2_3 = errors_conv_2_3 + sum(current_packet ~= decoded_pkt_2_3);
% 3. Punctured coding (all on rate-1/2 base)
% Prepare base encoded
encoded_base = convenc(current_packet, trellises.rate_1_2);
```

```
% Punc 8/9
pattern = puncture_patterns.rate_8_9;
punctured_8_9 = convenc(current_packet, trellises.rate_1_2, pattern);
noisy_punc_8_9 = bsc(punctured_8_9, p);
decoded_punc_8_9 = vitdec(noisy_punc_8_9, trellises.rate_1_2, 34, 'trunc', 'hard',
pattern);
% Extract the first 1024 bits from the decoded output to match the original packet
size
decoded_pkt_p8_9 = decoded_punc_8_9(1:1024);
% Store the decoded bits in the appropriate section of the preallocated array
punc_8_9_bits((pkt-1)*1024+1 : pkt*1024) = decoded_pkt_p8_9;
% Calculate and accumulate the number of bit errors by comparing original and decoded
packets
errors_punc_8_9 = errors_punc_8_9 + sum(current_packet ~= decoded_pkt_p8_9);
% Punc 4/5
pattern = puncture_patterns.rate_4_5;
punctured_4_5 = convenc(current_packet, trellises.rate_1_2, pattern);
noisy_punc_4_5 = bsc(punctured_4_5, p);
decoded_punc_4_5 = vitdec(noisy_punc_4_5, trellises.rate_1_2, 34, 'trunc', 'hard',
pattern);
decoded_pkt_p4_5 = decoded_punc_4_5(1:1024);
punc_4_5_bits((pkt-1)*1024+1 : pkt*1024) = decoded_pkt_p4_5;
errors_punc_4_5 = errors_punc_4_5 + sum(current_packet ~= decoded_pkt_p4_5);
% Punc 2/3
pattern = puncture_patterns.rate_2_3;
punctured_2_3 = convenc(current_packet, trellises.rate_2_3, pattern);
noisy_punc_2_3 = bsc(punctured_2_3, p);
decoded_punc_2_3 = vitdec(noisy_punc_2_3, trellises.rate_2_3, 34, 'trunc', 'hard',
pattern);
decoded_pkt_p2_3 = decoded_punc_2_3(1:min(1024, length(decoded_punc_2_3)));
punc_2_3_bits((pkt-1)*1024+1 : pkt*1024) = decoded_pkt_p2_3;
errors_punc_2_3 = errors_punc_2_3 + sum(current_packet ~= decoded_pkt_p2_3);
end
% Reconstruct frames
no_coding_frame = reconstruct_frame(no_coding_bits, s);
conv_1_2_frame = reconstruct_frame(conv_1_2_bits, s);
conv_1_3_frame = reconstruct_frame(conv_1_3_bits, s);
conv_1_4_frame = reconstruct_frame(conv_1_4_bits, s);
conv_2_3_frame = reconstruct_frame(conv_2_3_bits, s);
punc_8_9_frame = reconstruct_frame(punc_8_9_bits, s);
punc_4_5_frame = reconstruct_frame(punc_4_5_bits, s);
punc_2_3_frame = reconstruct_frame(punc_2_3_bits, s);
if(p==0.001 || p==0.1)
% Write to all output videos (using your exact frame reconstruction)
writeVideo(output_no_coding, no_coding_frame);
writeVideo(output_conv_1_2, conv_1_2_frame);
writeVideo(output_conv_1_3, conv_1_3_frame);
```

```
writeVideo(output_conv_1_4, conv_1_4_frame);
writeVideo(output_conv_2_3, conv_2_3_frame);
writeVideo(output_punc_8_9, punc_8_9_frame);
writeVideo(output_punc_4_5, punc_4_5_frame);
writeVideo(output_punc_2_3, punc_2_3_frame);
end
end
% Store BERs for this probability
BER_uncoded(p_idx) = errors_uncoded / (bits_per_frame * frames);
BER_conv_1_2(p_idx) = errors_conv_1_2 / (bits_per_frame * frames);
BER_conv_1_3(p_idx) = errors_conv_1_3 / (bits_per_frame * frames);
BER_conv_1_4(p_idx) = errors_conv_1_4 / (bits_per_frame * frames);
BER_conv_2_3(p_idx) = errors_conv_2_3 / (bits_per_frame * frames);
BER_punc_8_9(p_idx) = errors_punc_8_9 / (bits_per_frame * frames);
BER_punc_4_5(p_idx) = errors_punc_4_5 / (bits_per_frame * frames);
BER_punc_2_3(p_idx) = errors_punc_2_3 / (bits_per_frame * frames);
if(p==0.001 || p==0.1)
% Close all video writers
close(output_no_coding);
close(output_conv_1_2);
close(output_conv_1_3);
close(output_conv_1_4);
close(output_conv_2_3);
close(output_punc_8_9);
close(output_punc_4_5);
close(output_punc_2_3);
end
end
% Plot all BER results in one figure
figure;
semilogy(error_probs, BER_uncoded, 'k-o', 'LineWidth', 1.5, 'DisplayName',
'Uncoded'); hold on;
semilogy(error_probs, BER_conv_1_2, 'b-*', 'LineWidth', 1.5, 'DisplayName', 'Conv
1/2');
semilogy(error_probs, BER_conv_1_3, 'r-+', 'LineWidth', 1.5, 'DisplayName', 'Conv
1/3');
semilogy(error_probs, BER_conv_1_4, 'g-x', 'LineWidth', 1.5, 'DisplayName', 'Conv
1/4');
semilogy(error_probs, BER_conv_2_3, 'm-s', 'LineWidth', 1.5, 'DisplayName', 'Conv
2/3');
semilogy(error_probs, BER_punc_8_9, 'c-d', 'LineWidth', 1.5, 'DisplayName', 'Punc
8/9');
semilogy(error_probs, BER_punc_4_5, 'y-^', 'LineWidth', 1.5, 'DisplayName', 'Punc
4/5');
semilogy(error_probs, BER_punc_2_3, 'k-v', 'LineWidth', 1.5, 'DisplayName', 'Punc
2/3');
hold off;
```

```
grid on;
xlabel('Channel Error Probability (p)');
ylabel('Bit Error Rate (BER)');
title('BER Performance of Different Coding Schemes');
legend('Location', 'northwest');
% Calculate throughput for each scheme
throughput_uncoded = 1 - BER_uncoded; % Uncoded has rate 1
throughput_conv_1_2 = (1 - BER_conv_1_2); % Rate 1/2
throughput_conv_1_3 = (1 - BER_conv_1_3); % Rate 1/3
throughput_conv_1_4 = (1 - BER_conv_1_4); % Rate 1/4
throughput_conv_2_3 = (1 - BER_conv_2_3); % Rate 2/3
throughput_punc_8_9 = (1 - BER_punc_8_9); % Rate 8/9
throughput_punc_4_5 = (1 - BER_punc_4_5); % Rate 4/5
throughput_punc_2_3 = (1 - BER_punc_2_3); % Rate 2/3
% Plot all throughput results in one figure
figure;
plot(error_probs, throughput_uncoded, 'k-o', 'LineWidth', 1.5, 'DisplayName',
'Uncoded (1)'); hold on;
plot(error_probs, throughput_conv_1_2, 'b-*', 'LineWidth', 1.5, 'DisplayName', 'Conv
1/2');
plot(error_probs, throughput_conv_1_3, 'r-+', 'LineWidth', 1.5, 'DisplayName', 'Conv
1/3');
plot(error_probs, throughput_conv_1_4, 'g-x', 'LineWidth', 1.5, 'DisplayName', 'Conv
1/4');
plot(error_probs, throughput_conv_2_3, 'm-s', 'LineWidth', 1.5, 'DisplayName', 'Conv
2/3');
plot(error_probs, throughput_punc_8_9, 'c-d', 'LineWidth', 1.5, 'DisplayName', 'Punc
8/9');
plot(error_probs, throughput_punc_4_5, 'y-^', 'LineWidth', 1.5, 'DisplayName', 'Punc
4/5');
plot(error_probs, throughput_punc_2_3, 'k-v', 'LineWidth', 1.5, 'DisplayName', 'Punc
2/3');
hold off;
grid on;
xlabel('Channel Error Probability (p)');
ylabel('Throughput (Effective Rate)');
title('Throughput Performance of Different Coding Schemes');
legend('Location', 'best');
% Frame reconstruction after receiving the bits
function reconstructed_frame = reconstruct_frame(binary_stream, s)
% Split back into color channels
bits_per_channel = s(1)*s(2)*8;
R_stream_received = binary_stream(1:bits_per_channel);
G_stream_received = binary_stream(bits_per_channel+1:2*bits_per_channel);
B_stream_received = binary_stream(2*bits_per_channel+1:3*bits_per_channel);
% Reshape back to original dimensions
R_bin_received = reshape(R_stream_received, [s(1), s(2), 8]);
```

```
G_bin_received = reshape(G_stream_received, [s(1), s(2), 8]);
B_bin_received = reshape(B_stream_received, [s(1), s(2), 8]);
% Convert back to pixel values
R_recovered = uint8(bi2de(reshape(R_bin_received, [], 8)));
G_recovered = uint8(bi2de(reshape(G_bin_received, [], 8)));
B_recovered = uint8(bi2de(reshape(B_bin_received, [], 8)));
% Reconstruct frame
reconstructed_frame = zeros(s(1), s(2), 3, 'uint8');
reconstructed_frame(:,:,1) = reshape(R_recovered, [s(1), s(2)]);
reconstructed_frame(:,:,2) = reshape(G_recovered, [s(1), s(2)]);
reconstructed_frame(:,:,3) = reshape(B_recovered, [s(1), s(2)]);
end
```

We begin by loading a sample video file (`highway.avi`) and extracting its frames. Each frame is converted into a binary bitstream by separating the RGB channels and transforming pixel values into 8-bit binary. The resulting stream is divided into packets of 1024 bits.

These packets are then transmitted through a Binary Symmetric Channel (BSC) with different error probabilities ranging from 0.001 to 0.2. We simulate 8 different scenarios:

- **Uncoded transmission** (no error correction)
- **Convolutional codes** with rates: 1/2, 1/3, 1/4, and 2/3
- **Punctured convolutional codes** with rates: 8/9, 4/5, and 2/3

Each scheme uses MATLAB's `convenc` for encoding and `vitdec` for decoding using the Viterbi algorithm. After decoding, the bits are reshaped back into RGB frames. We compare the decoded frames to the original ones to count bit errors and calculate BER and throughput.

Key Results

- **BER Analysis:**
  - Lower-rate convolutional codes like **1/3 and 1/4** perform **best in error correction** due to **higher redundancy**.
  - **Punctured codes** (e.g., **8/9, 4/5**) show **worst BER performance**, consistent with **lower redundancy**.
- **Throughput Analysis:**
  - The **highest overall throughput** (across all error probabilities) is for the **uncoded scheme** (as expected, since there's no redundancy).

    - Among the **coded schemes**, the **highest throughput** is seen for **Conv 1/3**, followed by **Conv 1/2**, then **Conv 2/3**.
    - **Conv 1/4** has **lower throughput** than Conv 1/3 and 1/2, confirming that **higher redundancy reduces throughput**.
    - The **worst throughput** is for **Punc 2/3**, **Punc 8/9**, and **Punc 4/5**, especially as the error probability increases.
- **Balanced Schemes:**
    - **Conv 1/2 and Conv 2/3** represent a **good trade-off** between BER and throughput — offering **reasonable redundancy** with **better throughput** than low-rate codes like 1/4.

In conclusion, This project highlights the trade-off between **reliability and efficiency** in video transmission. Stronger coding (like conv 1/4) is suitable for noisy environments but uses more bandwidth, while high-rate schemes (like punc 8/9) are better for cleaner channels. By plotting BER and throughput across various error rates, we gain a deeper understanding of how to select the right coding method depending on channel conditions.

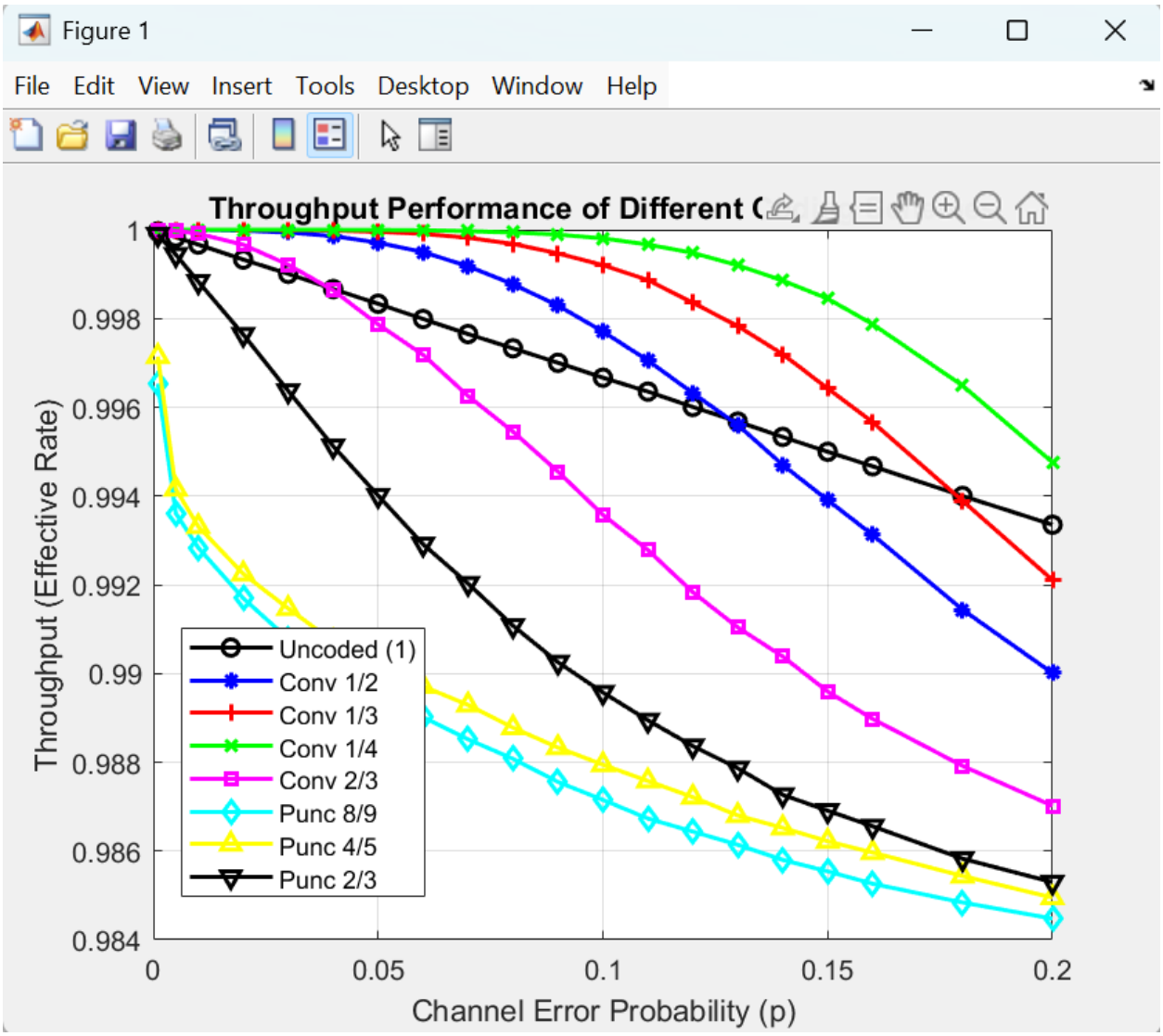
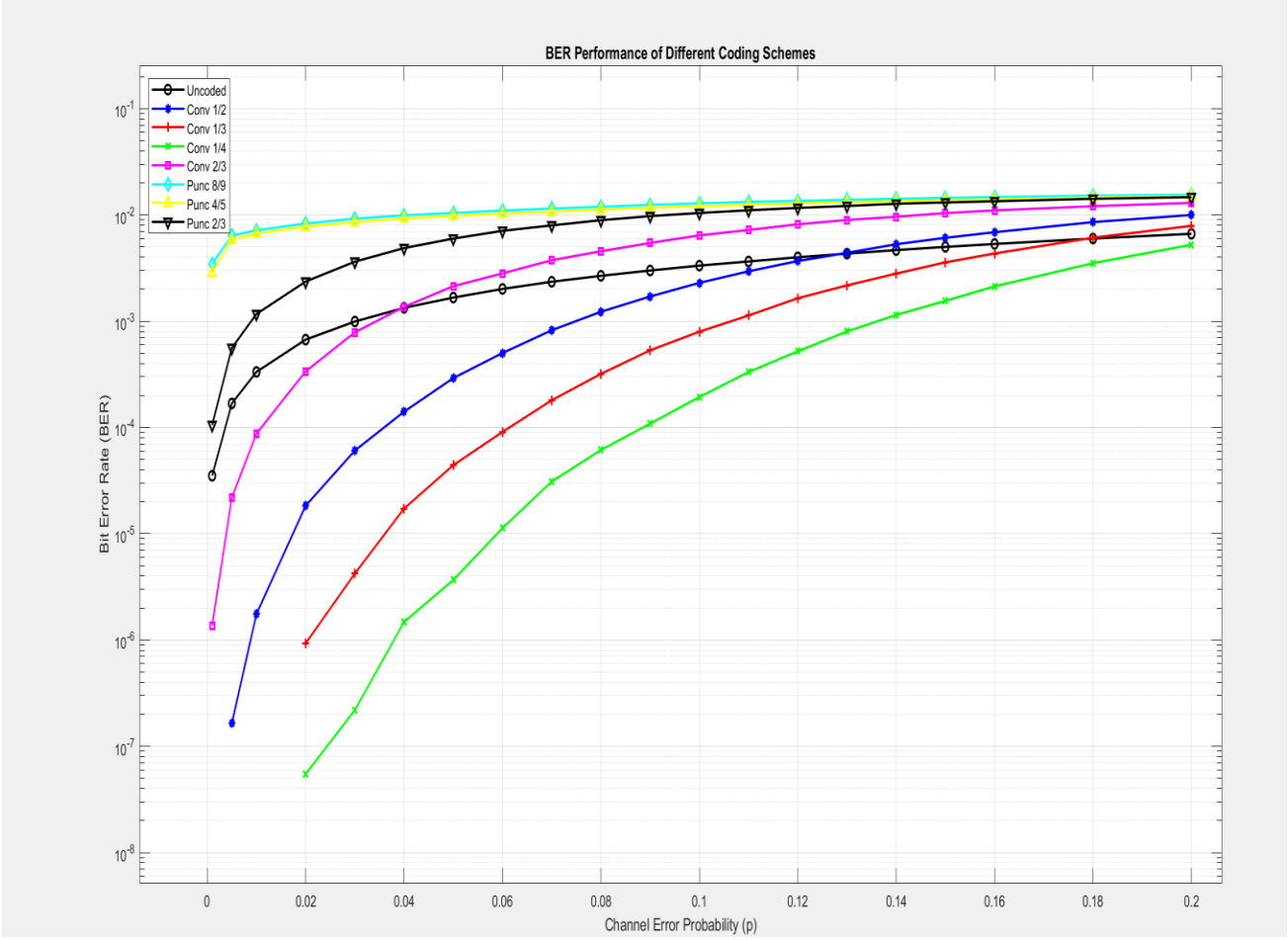


Figure 1
File Edit View Insert Tools Desktop Window Help
Throughput Performance of Different C
Throughput (Effective Rate)
Channel Error Probability (p)
1
0.998
0.996
0.994
0.992
0.99
0.988
0.986
0.984
0
0.05
0.1
0.15
0.2
Uncoded (1)
Conv 1/2
Conv 1/3
Conv 1/4
Conv 2/3
Punc 8/9
Punc 4/5
Punc 2/3


BER Performance of Different Coding Schemes
Uncoded
Conv 1/2
Conv 1/3
Conv 1/4
Conv 2/3
Punc 8/9
Punc 4/5
Punc 2/3
Bit Error Rate (BER)
Channel Error Probability (p)
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
10^-7
10^-8
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2