OSSE

March 27, 2020

Contents

[1]:
```
%pylab inline
#%matplotlib qt
import sk_dsp_comm.sigsys as ss
import sk_dsp_comm.digitalcom as dc
import scipy.signal as signal
import rs_fec_conv.fec_conv as rs_fec
import sk_dsp_comm.fec_conv as fec
import numpy as np
from numpy.random import randint
import matplotlib.pyplot as plt
from IPython.display import Audio, display
from IPython.display import Image, SVG
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
pylab.rcParams['savefig.dpi'] = 100 # default 72
#pylab.rcParams['figure.figsize'] = (6.0, 4.0) # default (6,4)
#%config InlineBackend.figure_formats=['png'] # default for inline viewing
#%config InlineBackend.figure_formats=['svg'] # SVG inline viewing
%config InlineBackend.figure_formats=['pdf'] # render pdf figs for LaTeX
```

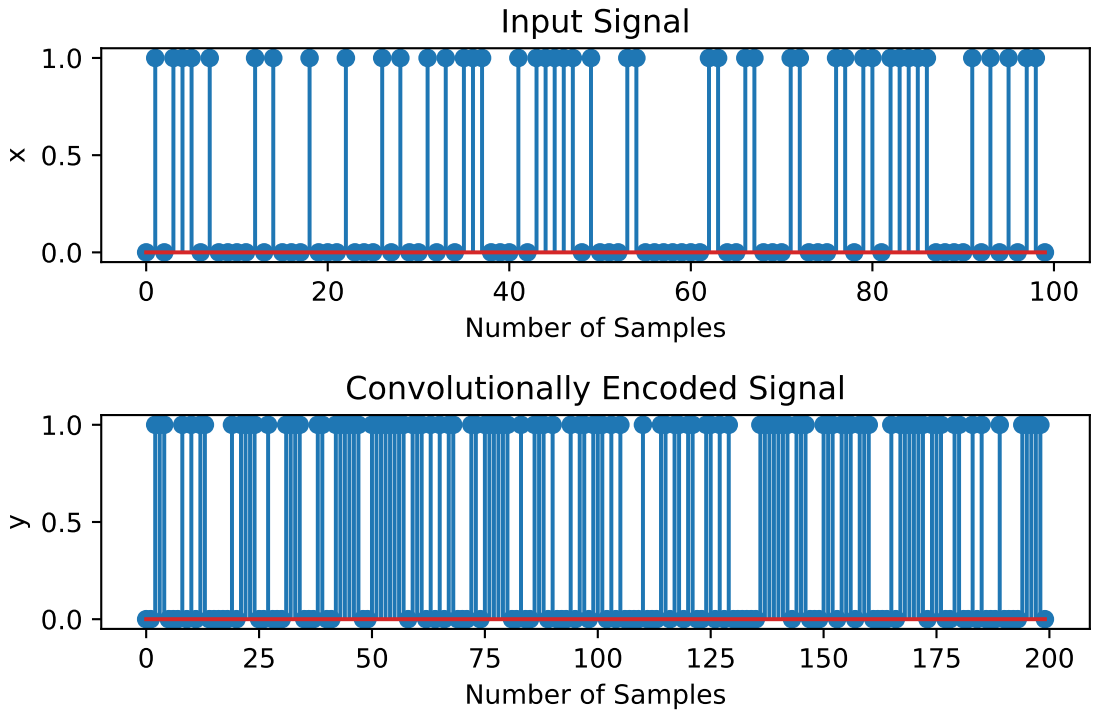
conv_encoder and viterbi_decoder

Function Comparison

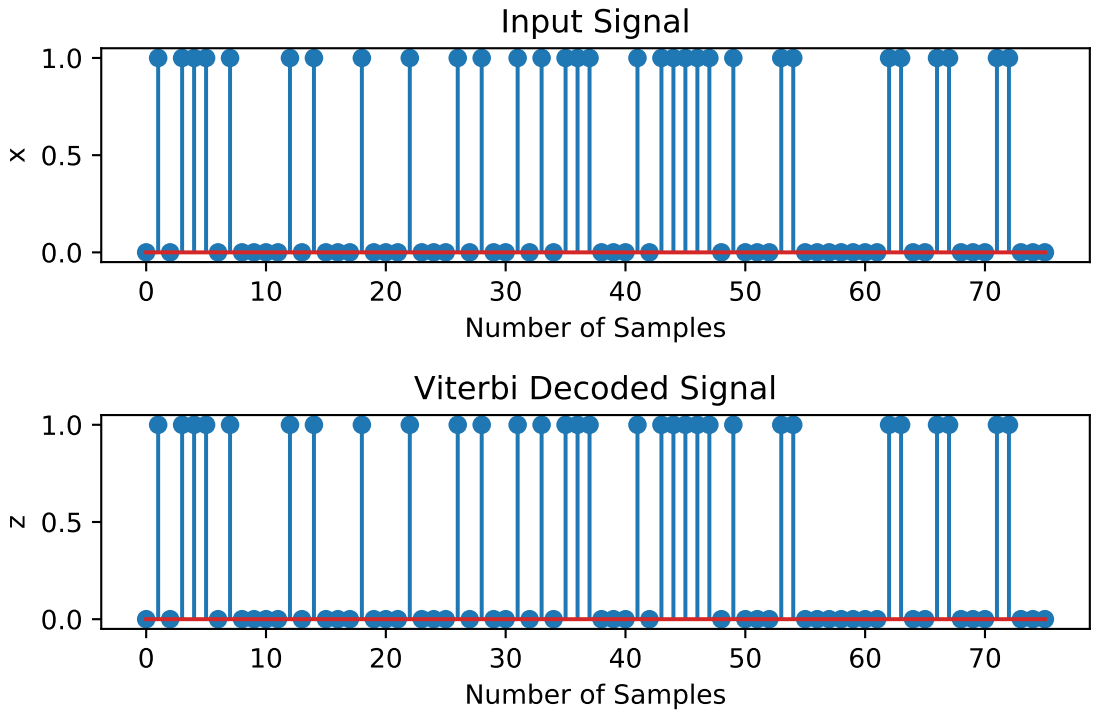
Compare the outputs of the convolutional encoder and viterbi decoder between Python and Rust functions.

[3]:
```
# Generate random data
N = 100
x = randint(0,2,N)

# Initialize fec_conv object with either G length 2 or 3
# depth = 10
# G =('111','101')

# depth = 25
# G = ('11111','11011','10101')

depth = 25
G = ('1111001','1011011')

cc1 = rs_fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state = ''
for i in range(len(G[0]) - 1):
    state += '0'

# Convolutionally Encode Signal
y,state = cc1.conv_encoder(x,state)

# Plot input signal
subplot(211)
stem(x, use_line_collection=True)
xlabel('Number of Samples')
ylabel('x')
title('Input Signal')

# Plot convolutionally encoded signal
subplot(212)
stem(y, use_line_collection=True)
xlabel('Number of Samples')
ylabel('y')
title('Convolutionally Encoded Signal')
tight_layout()
savefig('conv_enc.png')
```

```
Rate 1/2 Object
```



[4]:
```
# Viterbi decode
z = cc1.viterbi_decoder(y.astype(int), 'hard', 3)
z

# Plot input signal
subplot(211)
# stem(x[:11], use_line_collection=True)
stem(x[:-(depth-1)], use_line_collection=True)
xlabel('Number of Samples')
ylabel('x')
title('Input Signal')
# xlim([0,10])

# Plot viterbi decoded signal
subplot(212)
stem(z, use_line_collection=True)
xlabel('Number of Samples')
ylabel('z')
title('Viterbi Decoded Signal')
# xlim([0,10])
tight_layout()
savefig('viterbi_dec.png')
```



Code length 3. Rate 1/2

[5]:
```
# Python
N = 50
x = randint(0,2,N)

depth = 10
G =('111','101')
cc1 = fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state1 = ''
for i in range(len(G[0]) - 1):
    state1 += '0'
y1,state1 = cc1.conv_encoder(x,state1)

# Viterbi decode
z1 = cc1.viterbi_decoder(y1.astype(int), 'hard', 3)

# print results
print(y1)
print(state1)
print(z1)
```

```
Rate 1/2 Object
[1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0.
 1. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0.
 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0.
 0. 0. 1. 0.]
01
[1. 1. 0. 1. 1. 0. 1. 1. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0.
 1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1.]
```

[6]:
```
# Rust
N = 50
#x = randint(0,2,N)

depth = 10
G =('111','101')
cc2 = rs_fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state2 = ''
for i in range(len(G[0]) - 1):
    state2 += '0'
y2,state2 = cc2.conv_encoder(x,state2)

# Viterbi decode
z2 = cc2.viterbi_decoder(y2.astype(int), 'hard', 3)

# print results
print(y2)
print(state2)
print(z2)
```

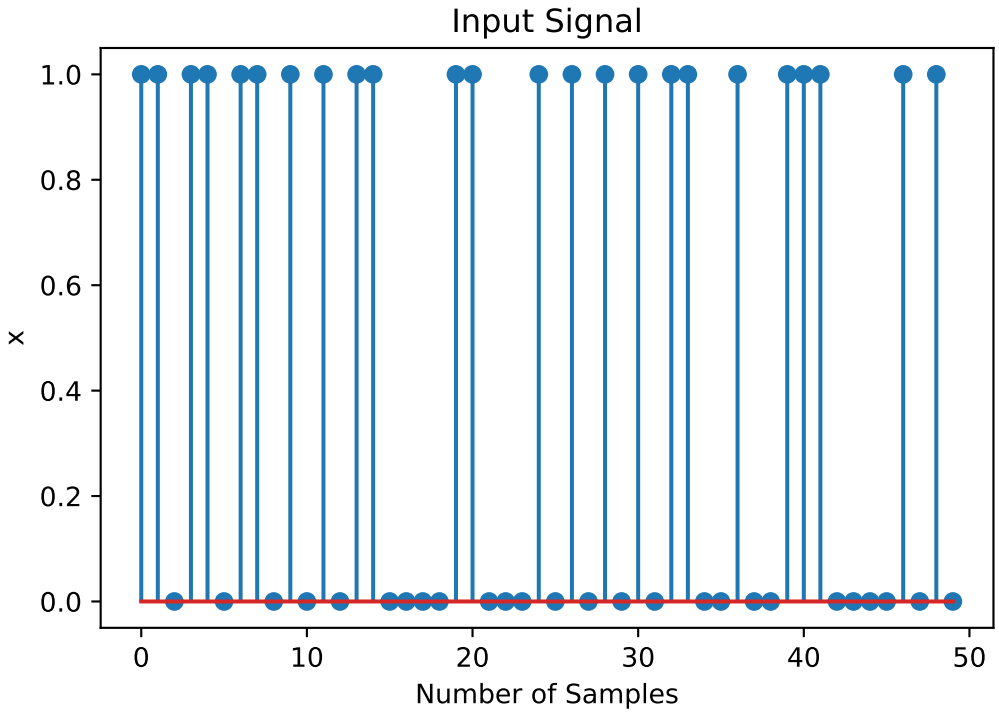
```
Rate 1/2 Object
[1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0.
 1. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0.
 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0.
 0. 0. 1. 0.]
01
[1. 1. 0. 1. 1. 0. 1. 1. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0.
 1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 0. 0. 1. 0. 0. 1. 1.]
```

Plot the outputs from Rust and Python.

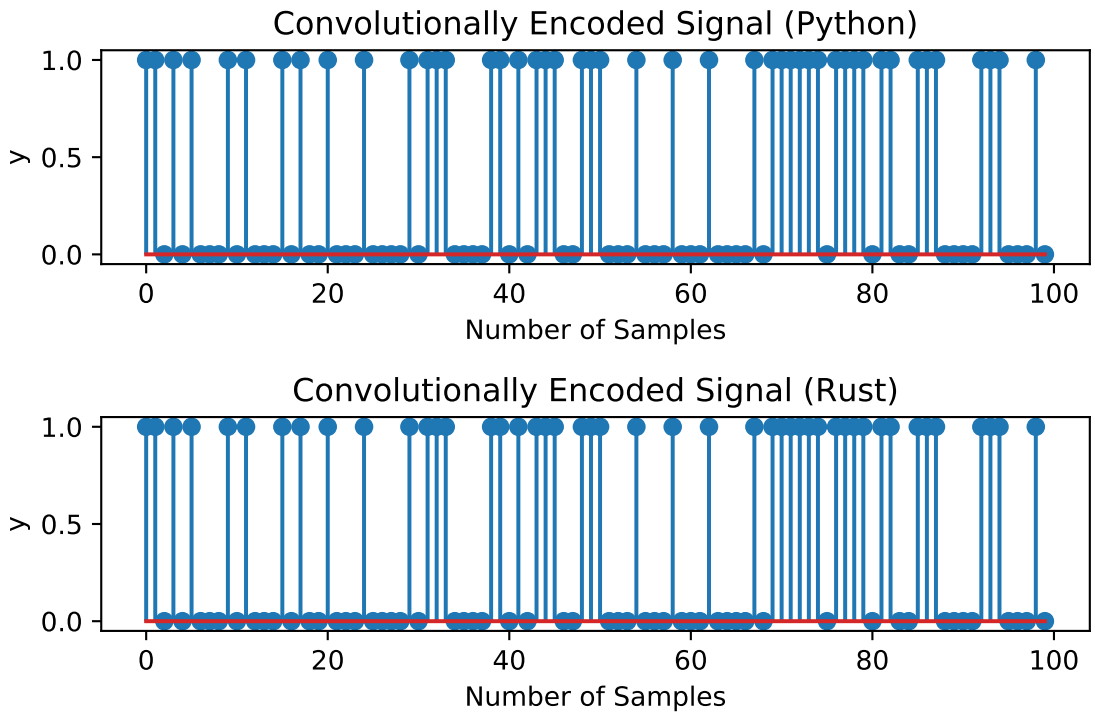
[7]:
```
# plot input
stem(x, use_line_collection=True)
xlabel('Number of Samples')
ylabel('x')
```

```
title('Input Signal')
```

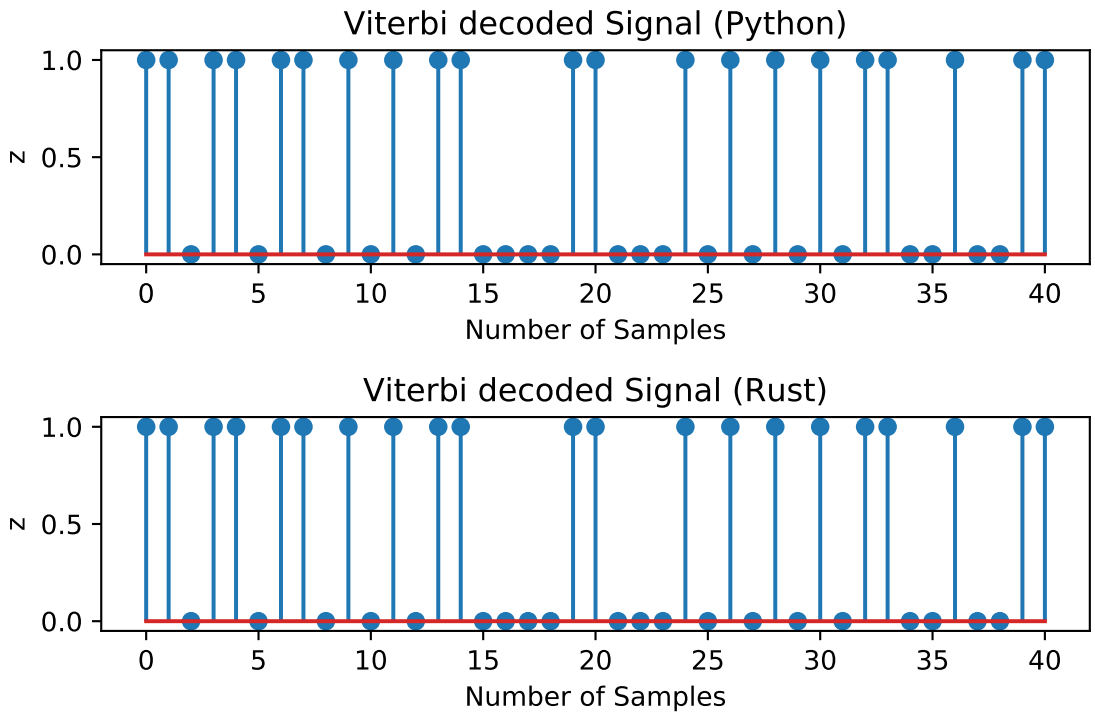
[7]: `Text(0.5, 1.0, 'Input Signal')`



[8]:
```
# plot conv encoded signal
subplot(211)
stem(y1, use_line_collection=True)
xlabel('Number of Samples')
ylabel('y')
title('Convolutionally Encoded Signal (Python)')

subplot(212)
stem(y2, use_line_collection=True)
xlabel('Number of Samples')
ylabel('y')
title('Convolutionally Encoded Signal (Rust)')
tight_layout()
```



[9]:
```
# plot viterbi decoded signals
subplot(211)
stem(z1, use_line_collection=True)
xlabel('Number of Samples')
ylabel('z')
title('Viterbi decoded Signal (Python)')

subplot(212)
stem(z2, use_line_collection=True)
xlabel('Number of Samples')
ylabel('z')
title('Viterbi decoded Signal (Rust)')
tight_layout()
```



Code length 5. Rate 1/3

[10]:
```
# Python
N = 50
x = randint(0,2,N)

depth = 25
G = ('11111','11011','10101')
cc1 = fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state1 = ''
for i in range(len(G[0]) - 1):
    state1 += '0'
y1,state1 = cc1.conv_encoder(x,state1)

# Viterbi decode
z1 = cc1.viterbi_decoder(y1.astype(int), 'hard', 3)

# print results
print(y1)
print(state1)
print(z1)
```

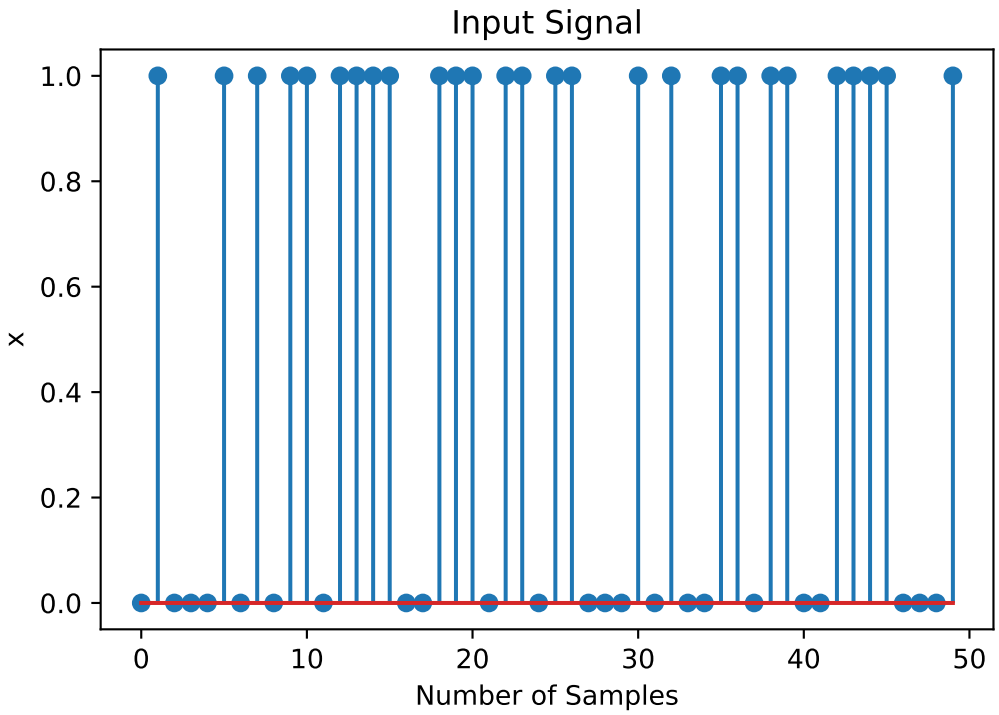
```
Rate 1/3 Object
[0. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0.
 0. 0. 0. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0.
 0. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1. 0. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0.
 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1. 1. 0.
 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0.
 0. 0. 1. 0. 0. 0.]
1000
[0. 1. 0. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1.
 0. 1.]
```

[11]:
```
# Rust
N = 50
#x = randint(0,2,N)

depth = 25
G = ('11111','11011','10101')
cc2 = rs_fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state2 = ''
for i in range(len(G[0]) - 1):
    state2 += '0'
y2,state2 = cc2.conv_encoder(x,state2)

# Viterbi decode
z2 = cc2.viterbi_decoder(y2.astype(int), 'hard', 3)

# print results
print(y2)
print(state2)
print(z2)
```

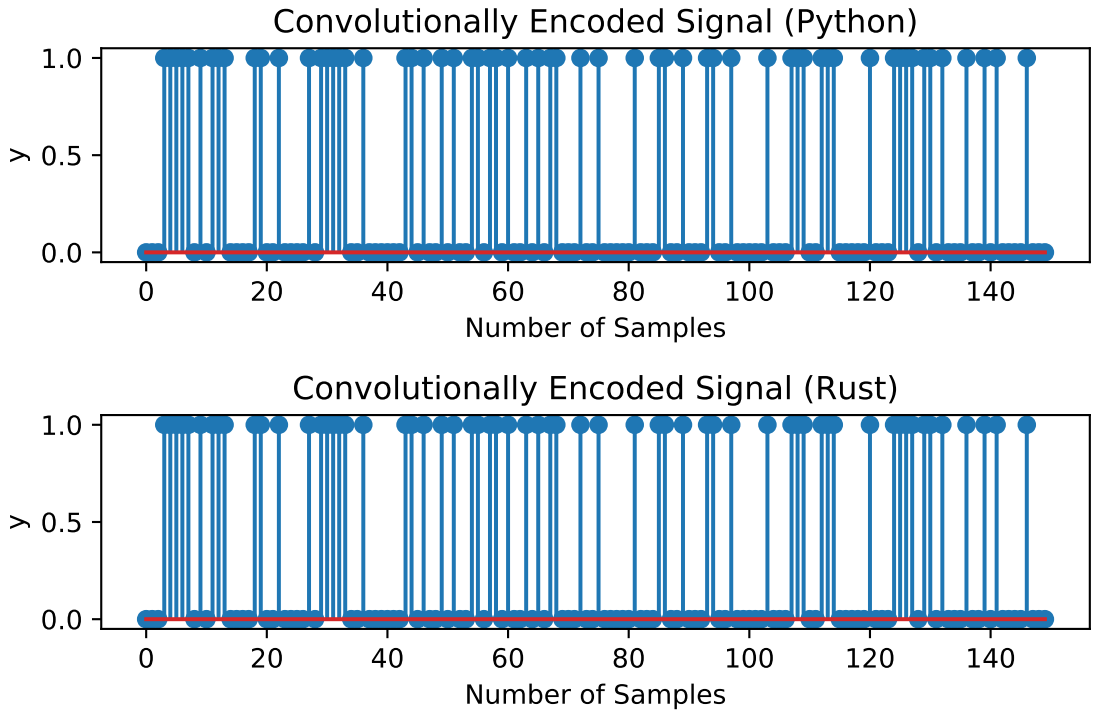
```
Rate 1/3 Object
[0. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0.
 0. 0. 0. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0.
 0. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1. 0. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 0. 0.
 1. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1. 1. 0.
 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0.
 0. 0. 1. 0. 0. 0.]
1000
[0. 1. 0. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1.
 0. 1.]
```

[12]:
```
# plot input
stem(x, use_line_collection=True)
xlabel('Number of Samples')
ylabel('x')
title('Input Signal')
```

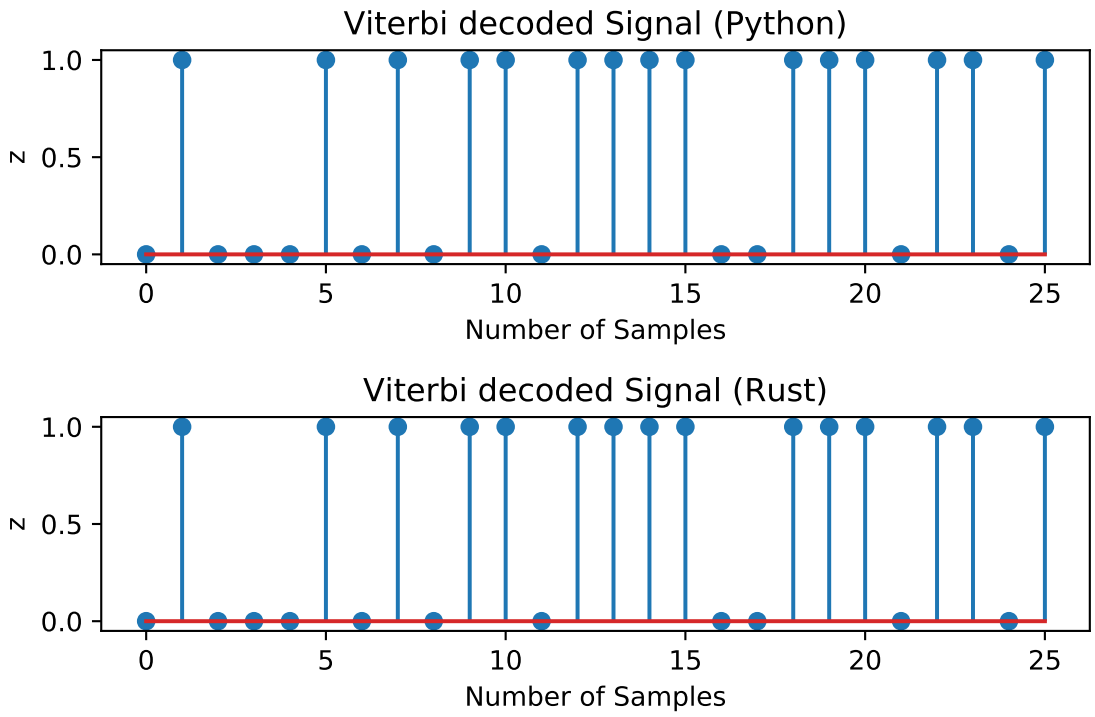
[12]:
```
Text(0.5, 1.0, 'Input Signal')
```



[13]:
```
# plot conv encoded signal
subplot(211)
stem(y1, use_line_collection=True)
xlabel('Number of Samples')
ylabel('y')
title('Convolutionally Encoded Signal (Python)')

subplot(212)
stem(y2, use_line_collection=True)
xlabel('Number of Samples')
ylabel('y')
title('Convolutionally Encoded Signal (Rust)')
tight_layout()
```



[14]:
```
# plot viterbi decoded signals
subplot(211)
stem(z1, use_line_collection=True)
xlabel('Number of Samples')
ylabel('z')
title('Viterbi decoded Signal (Python)')

subplot(212)
stem(z2, use_line_collection=True)
xlabel('Number of Samples')
ylabel('z')
title('Viterbi decoded Signal (Rust)')
tight_layout()
```



Code length 7. Rate 1/2

[15]:
```
# Python
N = 50
x = randint(0,2,N)

depth = 25
G = ('1111001','1011011')
cc1 = fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state1 = ''
for i in range(len(G[0]) - 1):
    state1 += '0'
y1,state1 = cc1.conv_encoder(x,state1)

# Viterbi decode
z1 = cc1.viterbi_decoder(y1.astype(int), 'hard', 3)

# print results
print(y1)
print(state1)
print(z1)
```

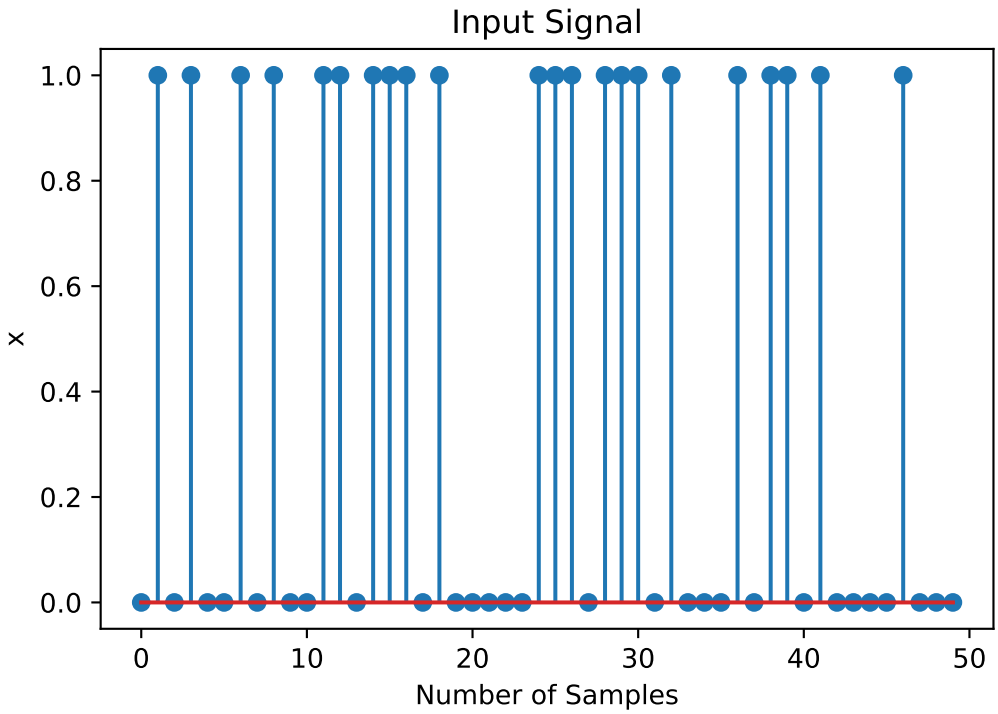
```
Rate 1/2 Object
[0. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 0. 1.
 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 1.
 0. 0. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1.
 0. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1.
 1. 1. 1. 1.]
000100
[0. 1. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0.
 1. 1.]
```

[16]:
```
# Rust
N = 50
#x = randint(0,2,N)

depth = 25
G = ('1111001','1011011')
cc2 = rs_fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state2 = ''
for i in range(len(G[0]) - 1):
    state2 += '0'
y2,state2 = cc2.conv_encoder(x,state2)

# Viterbi decode
z2 = cc2.viterbi_decoder(y2.astype(int), 'hard', 3)

# print results
print(y2)
print(state2)
print(z2)
```

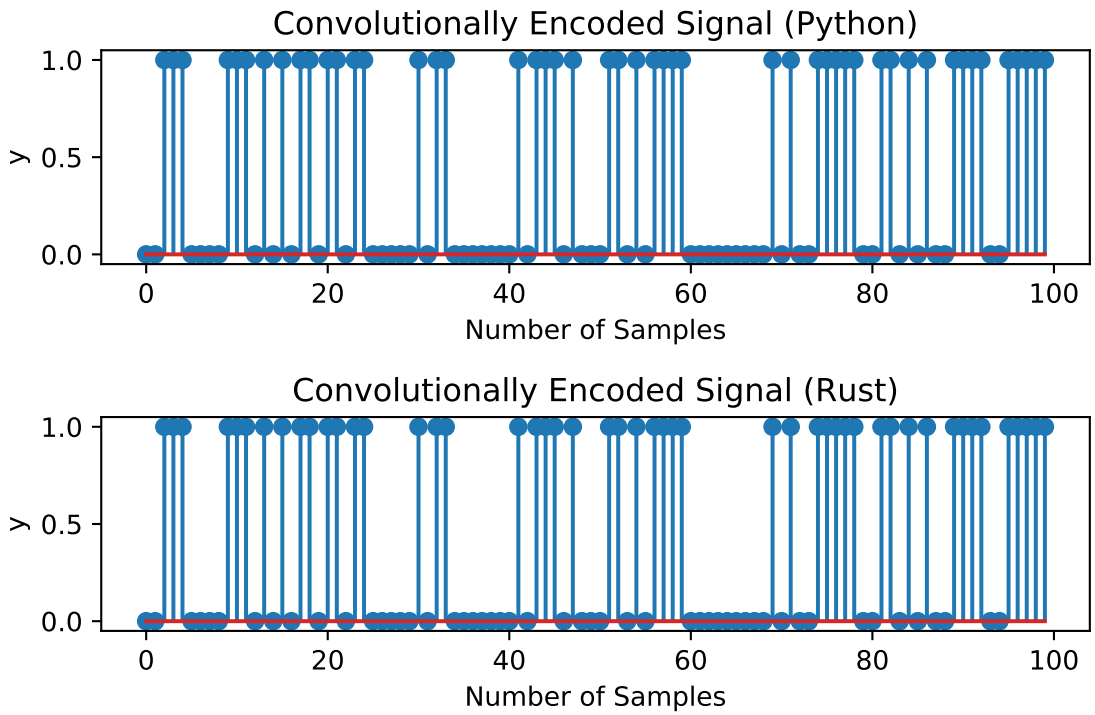
```
Rate 1/2 Object
[0. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 0. 1.
 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 1.
 0. 0. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1.
 0. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1.
 1. 1. 1. 1.]
000100
[0. 1. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0.
 1. 1.]
```

[17]:
```
# plot input
stem(x, use_line_collection=True)
xlabel('Number of Samples')
ylabel('x')
title('Input Signal')
```

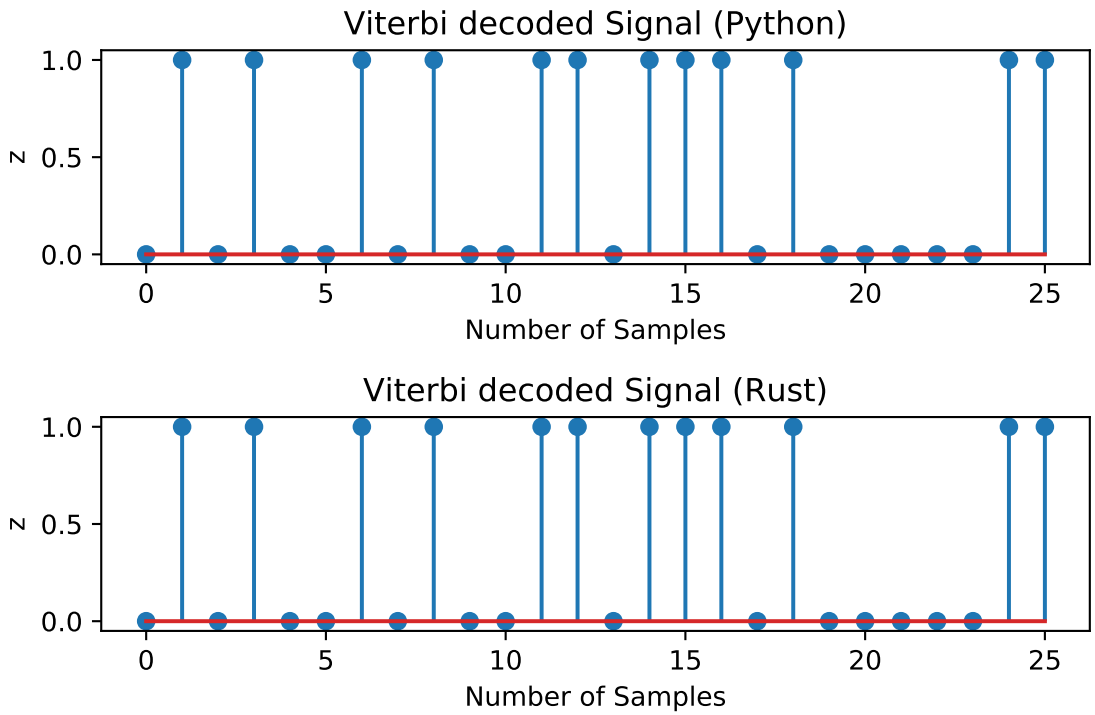
[17]: Text(0.5, 1.0, 'Input Signal')



[18]:
```
# plot conv encoded signal
subplot(211)
stem(y1, use_line_collection=True)
xlabel('Number of Samples')
ylabel('y')
title('Convolutionally Encoded Signal (Python)')

subplot(212)
stem(y2, use_line_collection=True)
xlabel('Number of Samples')
ylabel('y')
title('Convolutionally Encoded Signal (Rust)')
tight_layout()
```



[19]:
```
# plot viterbi decoded signals
subplot(211)
stem(z1, use_line_collection=True)
xlabel('Number of Samples')
ylabel('z')
title('Viterbi decoded Signal (Python)')

subplot(212)
stem(z2, use_line_collection=True)
xlabel('Number of Samples')
ylabel('z')
title('Viterbi decoded Signal (Rust)')
tight_layout()
```



The Rust and Python match for convolutionally encoding and viterbi decoding.

Timing Comparison

The Python versions of conv_encoder and viterbi_decoder functions are compared with the Rust versions of the functions with respect to time.

Code length 3. Rate 1/2

[ ]:
```
%%timeit
# Python
N = 100
x = randint(0,2,N)

depth = 10
G =('111','101')

cc1 = fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state1 = ''
for i in range(len(G[0]) - 1):
    state1 += '0'
y1,state1 = cc1.conv_encoder(x,state1)
```

```
# Viterbi decode
z1 = cc1.viterbi_decoder(y1.astype(int), 'hard', 3)
```

[ ]:
```
%%timeit
# Rust
N = 100
#x = randint(0,2,N)

depth = 10
G =('111','101')

cc2 = rs_fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state2 = ''
for i in range(len(G[0]) - 1):
    state2 += '0'
y2,state2 = cc2.conv_encoder(x,state2)

# Viterbi decode
z2 = cc2.viterbi_decoder(y2.astype(int), 'hard', 3)
```

Code length 5. Rate 1/3

[ ]:
```
%%timeit
# Python
N = 100
x = randint(0,2,N)

depth = 25
G = ('11111','11011','10101')

cc1 = fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state1 = ''
for i in range(len(G[0]) - 1):
    state1 += '0'
y1,state1 = cc1.conv_encoder(x,state1)

# Viterbi decode
z1 = cc1.viterbi_decoder(y1.astype(int), 'hard', 3)
```

[ ]:
```
%%timeit
# Rust
```

```
N = 100
#x = randint(0,2,N)

depth = 25
G = ('11111','11011','10101')

cc2 = rs_fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state2 = ''
for i in range(len(G[0]) - 1):
    state2 += '0'
y2,state2 = cc2.conv_encoder(x,state2)

# Viterbi decode
z2 = cc2.viterbi_decoder(y2.astype(int), 'hard', 3)
```

Code length 7. Rate 1/2

[ ]:
```
%%timeit
# Python
N = 100
x = randint(0,2,N)

depth = 25
G = ('1111001','1011011')

cc1 = fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state1 = ''
for i in range(len(G[0]) - 1):
    state1 += '0'
y1,state1 = cc1.conv_encoder(x,state1)

# Viterbi decode
z1 = cc1.viterbi_decoder(y1.astype(int), 'hard', 3)
```

[ ]:
```
%%timeit
# Rust
N = 100
#x = randint(0,2,N)

depth = 25
G = ('1111001','1011011')
```

```
cc2 = rs_fec.fec_conv(G,depth)

# Encode with shift register starting state of '0000'
state2 = ''
for i in range(len(G[0]) - 1):
    state2 += '0'
y2,state2 = cc2.conv_encoder(x,state2)

# Viterbi decode
z2 = cc2.viterbi_decoder(y2.astype(int), 'hard', 3)
```

20 bits

Python: 3.81 ms ± 126 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

Rust: 696 µs ± 44.8 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

The convolutional encoder and viterbi decoder functions built in Rust run significantly 5 times faster than the pure Python functions.

Viterbi BER Simulation

Function Comparison

[20]:
```
# Python
# Soft decision rate 1/2 simulation
N_bits_per_frame = 100000
EbN0 = 4
total_bit_errors = 0
total_bit_count = 0
cc1 = fec.fec_conv(('11101','10011'),25)

# Encode with shift register starting state of '0000'
state = '0000'
while total_bit_errors < 100:
    # Create 100000 random 0/1 bits
    x = randint(0,2,N_bits_per_frame)
    y,state = cc1.conv_encoder(x,state)

    # Add channel noise to bits, include antipodal level shift to [-1,1]
    yn_soft = dc.cpx_AWGN(2*y-1,EbN0-3,1) # Channel SNR is 3 dB less for rate 1/
 ↪2
    yn_hard = ((np.sign(yn_soft.real)+1)/2).astype(int)
    z = cc1.viterbi_decoder(yn_hard,'hard')

    # Count bit errors
    bit_count, bit_errors = dc.bit_errors(x,z)
    total_bit_errors += bit_errors
```

```
    total_bit_count += bit_count
    print('Bits Received = %d, Bit errors = %d, BEP = %1.2e' %\
          (total_bit_count, total_bit_errors,\
           total_bit_errors/total_bit_count))

print('**************************************************')
print('Bits Received = %d, Bit errors = %d, BEP = %1.2e' %\
      (total_bit_count, total_bit_errors,\
       total_bit_errors/total_bit_count))
```

```
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99976, Bit errors = 875, BEP = 8.75e-03
**************************************************
Bits Received = 99976, Bit errors = 875, BEP = 8.75e-03
```

[21]:
```
# Rust
# Soft decision rate 1/2 simulation
N_bits_per_frame = 100000
EbN0 = 4
total_bit_errors = 0
total_bit_count = 0
cc1 = rs_fec.fec_conv(('11101','10011'),25)

# Encode with shift register starting state of '0000'
state = '0000'
while total_bit_errors < 100:
    # Create 100000 random 0/1 bits
    x = randint(0,2,N_bits_per_frame)
    y,state = cc1.conv_encoder(x,state)

    # Add channel noise to bits, include antipodal level shift to [-1,1]
    yn_soft = dc.cpx_AWGN(2*y-1,EbN0-3,1) # Channel SNR is 3 dB less for rate 1/
 ↪2
    yn_hard = ((np.sign(yn_soft.real)+1)/2).astype(int)
    z = cc1.viterbi_decoder(yn_hard,'hard')

    # Count bit errors
    bit_count, bit_errors = dc.bit_errors(x,z)
    total_bit_errors += bit_errors
    total_bit_count += bit_count
    print('Bits Received = %d, Bit errors = %d, BEP = %1.2e' %\
          (total_bit_count, total_bit_errors,\
           total_bit_errors/total_bit_count))

print('**************************************************')
print('Bits Received = %d, Bit errors = %d, BEP = %1.2e' %\
```

```
      (total_bit_count, total_bit_errors,\
       total_bit_errors/total_bit_count))
```

```
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99976, Bit errors = 723, BEP = 7.23e-03
******************************************************
Bits Received = 99976, Bit errors = 723, BEP = 7.23e-03
```

The two simulations provide similar BERs althought since they are using random inputs for each loop the results are not exactly the same.

Timing Comparison

[22]:
```
%%timeit
# Python
# Soft decision rate 1/2 simulation
N_bits_per_frame = 100000
EbN0 = 4
total_bit_errors = 0
total_bit_count = 0

depth = 10
G =('111','101')

# depth = 25
# G = ('11111','11011','10101')

# depth = 25
# G = ('1111001','1011011')
cc1 = fec.fec_conv(G, depth)

# Encode with shift register starting state of '0000'
state = ''
for i in range(len(G[0]) - 1):
    state += '0'

while total_bit_errors < 100:
    # Create 100000 random 0/1 bits
    x = randint(0,2,N_bits_per_frame)
    y,state = cc1.conv_encoder(x,state)

    # Add channel noise to bits, include antipodal level shift to [-1,1]
    yn_soft = dc.cpx_AWGN(2*y-1,EbN0-3,1) # Channel SNR is 3 dB less for rate 1/
 ↪2
    yn_hard = ((np.sign(yn_soft.real)+1)/2).astype(int)
    z = cc1.viterbi_decoder(yn_hard,'hard')
```

```
    # Count bit errors
    bit_count, bit_errors = dc.bit_errors(x,z)
    total_bit_errors += bit_errors
    total_bit_count += bit_count
    print('Bits Received = %d, Bit errors = %d, BEP = %1.2e' %\
          (total_bit_count, total_bit_errors,\
           total_bit_errors/total_bit_count))

print('**************************************************')
print('Bits Received = %d, Bit errors = %d, BEP = %1.2e' %\
      (total_bit_count, total_bit_errors,\
       total_bit_errors/total_bit_count))
```

```
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1132, BEP = 1.13e-02
**************************************************
Bits Received = 99991, Bit errors = 1132, BEP = 1.13e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1315, BEP = 1.32e-02
**************************************************
Bits Received = 99991, Bit errors = 1315, BEP = 1.32e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1122, BEP = 1.12e-02
**************************************************
Bits Received = 99991, Bit errors = 1122, BEP = 1.12e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1219, BEP = 1.22e-02
**************************************************
Bits Received = 99991, Bit errors = 1219, BEP = 1.22e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1303, BEP = 1.30e-02
**************************************************
Bits Received = 99991, Bit errors = 1303, BEP = 1.30e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1347, BEP = 1.35e-02
**************************************************
Bits Received = 99991, Bit errors = 1347, BEP = 1.35e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1218, BEP = 1.22e-02
**************************************************
```

```
Bits Received = 99991, Bit errors = 1218, BEP = 1.22e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1316, BEP = 1.32e-02
**************************************************
Bits Received = 99991, Bit errors = 1316, BEP = 1.32e-02
42.5 s ± 2.08 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[23]:
```
%%timeit
# Rust
# Soft decision rate 1/2 simulation
N_bits_per_frame = 100000
EbN0 = 4
total_bit_errors = 0
total_bit_count = 0

depth = 10
G =('111','101')

# depth = 25
# G = ('11111','11011','10101')

# depth = 25
# G = ('1111001','1011011')
cc1 = rs_fec.fec_conv(G, depth)

# Encode with shift register starting state of '0000'
state = ''
for i in range(len(G[0]) - 1):
    state += '0'

while total_bit_errors < 100:
    # Create 100000 random 0/1 bits
    x = randint(0,2,N_bits_per_frame)
    y,state = cc1.conv_encoder(x,state)

    # Add channel noise to bits, include antipodal level shift to [-1,1]
    yn_soft = dc.cpx_AWGN(2*y-1,EbN0-3,1) # Channel SNR is 3 dB less for rate 1/
 ↪2
    yn_hard = ((np.sign(yn_soft.real)+1)/2).astype(int)
    z = cc1.viterbi_decoder(yn_hard,'hard')

    # Count bit errors
    bit_count, bit_errors = dc.bit_errors(x,z)
    total_bit_errors += bit_errors
    total_bit_count += bit_count
    print('Bits Received = %d, Bit errors = %d, BEP = %1.2e' %\
```

```
          (total_bit_count, total_bit_errors,\
           total_bit_errors/total_bit_count))

print('*****************************************************')
print('Bits Received = %d, Bit errors = %d, BEP = %1.2e' %\
      (total_bit_count, total_bit_errors,\
       total_bit_errors/total_bit_count))
```

```
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1224, BEP = 1.22e-02
*****************************************************
Bits Received = 99991, Bit errors = 1224, BEP = 1.22e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1344, BEP = 1.34e-02
*****************************************************
Bits Received = 99991, Bit errors = 1344, BEP = 1.34e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1342, BEP = 1.34e-02
*****************************************************
Bits Received = 99991, Bit errors = 1342, BEP = 1.34e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1186, BEP = 1.19e-02
*****************************************************
Bits Received = 99991, Bit errors = 1186, BEP = 1.19e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1339, BEP = 1.34e-02
*****************************************************
Bits Received = 99991, Bit errors = 1339, BEP = 1.34e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1436, BEP = 1.44e-02
*****************************************************
Bits Received = 99991, Bit errors = 1436, BEP = 1.44e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1202, BEP = 1.20e-02
*****************************************************
Bits Received = 99991, Bit errors = 1202, BEP = 1.20e-02
Rate 1/2 Object
kmax =  0, taumax = 0
Bits Received = 99991, Bit errors = 1209, BEP = 1.21e-02
*****************************************************
```

```
Bits Received = 99991, Bit errors = 1209, BEP = 1.21e-02
633 ms ± 12.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

100000 bits

Python: 1min 20s ± 3.82 s per loop (mean ± std. dev. of 7 runs, 1 loop each)

Rust: 2.73 s ± 135 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

The convolutional encoder and decoder functions built in Rust run faster than the Python versions. The time to process the Viterbi decoder runs about 30 times faster.