[2]:
```
from matplotlib import pyplot as plt
import numpy as np
from scipy import signal as sig
import struct
import random
import ipywidgets
import itertools

import colorednoise

np.set_printoptions(linewidth=240)
```

[3]:
```
%matplotlib widget
```

[4]:
```
sampling_rate = 10 # sp/s
```

[5]:
```
#colorednoise.powerlaw_psd_gaussian(1, int(1e4))
```

[6]:
```
# From https://github.com/mubeta06/python/blob/master/signal_processing/sp/gold.
 ↪py
preferred_pairs = {5:[[2],[1,2,3]], 6:[[5],[1,4,5]], 7:[[4],[4,5,6]],
                       8:[[1,2,3,6,7],[1,2,7]], 9:[[5],[3,5,6]],
                       10:[[2,5,9],[3,4,6,8,9]], 11:[[9],[3,6,9]]}

def gen_gold(seq1, seq2):
    print(seq1.shape, seq2.shape)
    gold = [seq1, seq2]
    for shift in range(len(seq1)):
        gold.append(seq1 ^ np.roll(seq2, -shift))
    return gold

def gold(n):
    n = int(n)
    if not n in preferred_pairs:
        raise KeyError('preferred pairs for %s bits unknown' % str(n))
    t0, t1 = preferred_pairs[n]
    (seq0, _st0), (seq1, _st1) = sig.max_len_seq(n, taps=t0), sig.
 ↪max_len_seq(n, taps=t1)
    return gen_gold(seq0, seq1)
```

[7]:
```
fig, ax = plt.subplots()
ax.matshow(gold(5))
```

```
(31,) (31,)
```

[7]:
```
<matplotlib.image.AxesImage at 0x7ff8d9616610>
```

[8]:
```
def modulate(data, nbits=5):
    # 0, 1 -> -1, 1
    mask = np.array(gold(nbits))*2 - 1

    sel = mask[data>>1]
    data_lsb_centered = ((data&1)*2 - 1)

    return (np.multiply(sel, np.tile(data_lsb_centered, (2**nbits-1, 1)).T).
 ↪flatten() + 1) // 2
```

[9]:
```
data = np.array(list(range(16)))

mask = np.array(gold(5))*2 - 1

sel = mask[data>>1]
data_lsb_centered = ((data&1)*2 - 1)
mask.shape, data.shape, sel.shape

#fig, ax = plt.subplots()
#ax.plot(
np.multiply(sel, np.tile(data_lsb_centered, (2**5-1, 1)).T).flatten()
```

```
(31,) (31,)
```

[9]:
```
array([-1, -1, -1, -1, -1,  1,  1,  1, -1, -1,  1, -1, -1, -1,  1, -1,  1, -1,
       1,  1,  1,  1, -1,  1,  1, -1,  1, -1, -1,  1,  1,  1,  1,  1,  1,  1, -1, -1,
      -1,  1,  1, -1,  1,  1,  1, -1,  1, -1,  1, -1, -1, -1, -1,  1, -1, -1,  1, -1,
         1,  1, -1, -1, -1, -1, -1, -1, -1,  1,  1, -1,  1,  1, -1, -1,  1,  1,
       1,  1, -1,  1, -1, -1,  1, -1,  1, -1,  1,  1,  1, -1, -1, -1,  1,  1,  1,  1,
       1,  1, -1, -1,  1, -1, -1,  1,  1, -1, -1, -1, -1,  1, -1,  1,  1, -1,  1, -1,
         1, -1, -1, -1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1,
      -1,  1, -1, -1,  1, -1, -1, -1, -1, -1,  1, -1, -1, -1,  1, -1,  1,  1,  1, -1,
       1, -1, -1, -1, -1, -1, -1, -1,  1,  1,  1,  1, -1,  1,  1, -1,  1,  1,  1,  1,
         1, -1,  1,  1,  1, -1,  1, -1, -1, -1,  1, -1,  1,  1,  1,  1, -1,  1,
      -1,  1, -1,  1, -1, -1, -1, -1,  1,  1,  1,  1, -1,  1, -1,  1,  1,  1,  1, -1,
      -1,  1,  1,  1, -1, -1, -1, -1, -1,  1, -1,  1, -1,  1, -1,  1,  1,  1,  1, -1,
        -1, -1, -1,  1, -1,  1, -1, -1, -1, -1,  1,  1, -1, -1, -1,  1,  1,  1,
       1, -1, -1, -1,  1,  1,  1,  1,  1, -1, -1, -1, -1, -1, -1,  1,  1, -1,  1, -1,
      -1,  1,  1,  1, -1,  1, -1, -1, -1, -1, -1, -1,  1,  1,  1, -1, -1, -1, -1, -1,
         1,  1,  1,  1,  1,  1, -1, -1,  1, -1,  1,  1, -1, -1, -1,  1, -1,  1,
       1,  1,  1,  1, -1, -1,  1,  1,  1, -1,  1, -1,  1, -1, -1,  1,  1,  1, -1, -1,
      -1,  1, -1,  1, -1,  1, -1,  1, -1, -1,  1, -1, -1, -1, -1,  1,  1, -1, -1, -1,
         1, -1,  1, -1,  1,  1, -1, -1, -1,  1,  1,  1, -1,  1, -1,  1, -1,  1,
      -1,  1,  1, -1,  1,  1,  1, -1, -1,  1, -1,  1, -1, -1, -1, -1,  1, -1,  1, -1,
      -1,  1,  1,  1,  1, -1,  1,  1, -1, -1, -1,  1,  1,  1,  1, -1, -1, -1,  1,  1,
```

```
       -1,  1, -1,  1,  1,  1,  1, -1,  1, -1,  1,  1, -1, -1, -1, -1,  1, -1,
-1,  1,  1,  1, -1, -1, -1, -1,  1,  1, -1, -1,  1, -1, -1, -1, -1,  1, -1, -1,
1,  1, -1,  1, -1, -1, -1, -1, -1,  1,  1,  1,  1, -1, -1, -1, -1,  1,  1, -1,
       -1,  1,  1, -1,  1,  1,  1,  1, -1,  1,  1, -1, -1,  1, -1,  1,  1,  1,
1,  1, -1, -1, -1, -1,  1,  1,  1,  1, -1, -1,  1,  1])
```

[10]:
```
def correlate(sequence, nbits=5, decimation=1, mask_filter=lambda x: x):
    mask = np.tile(np.array(gold(nbits))[:,:,np.newaxis]*2 - 1, (1, 1,␣
 ↪decimation)).reshape((2**nbits + 1, (2**nbits-1) * decimation))

    sequence -= np.mean(sequence)

    return np.array([np.correlate(sequence, row, mode='full') for row in mask])
```

[11]:
```
nbits = 5
decimation = 10

foo = np.repeat(modulate(np.array(list(range(4))), nbits).astype(float),␣
 ↪decimation)
bar = np.repeat(modulate(np.array(list(range(4))), nbits) * 2.0 - 1,␣
 ↪decimation) * 1e-3
print('shapes', foo.shape, bar.shape)

mask = np.tile(np.array(gold(nbits))[:,:,np.newaxis]*2 - 1, (1, 1, decimation)).
 ↪reshape((2**nbits + 1, (2**nbits-1) * decimation))
print('mask', mask.shape)

fig, (ax1, ax2) = plt.subplots(2, figsize=(16, 5))
fig.tight_layout()
corr_m = np.array([np.correlate(foo, row, mode='full') for row in mask])
#corr_m = np.array([row for row in mask])
ax1.matshow(corr_m, aspect='auto')
#ax.matshow(foo.reshape(32, 31)[::2,:])
ax2.matshow(correlate(bar, decimation=decimation), aspect='auto')
```

```
(31,) (31,)
(31,) (31,)
shapes (1240,) (1240,)
(31,) (31,)
mask (33, 310)
```

```
(31,) (31,)
```

[11]:
```
<matplotlib.image.AxesImage at 0x7ff8d955afa0>
```

[12]:
```
decimation = 10
```

```
fig, (ax1, ax2) = plt.subplots(2, figsize=(12, 5))
fig.tight_layout()

#mask = np.tile(np.array(gold(nbits))[:,:,np.newaxis]*2 - 1, (1, 1,␣
↪decimation)).reshape((2**nbits + 1, (2**nbits-1) * decimation))
#mask_stretched = np.tile(np.array(gold(nbits))[:,:,np.newaxis]*2 - 1, (1, 1,␣
↪1)).reshape((2**nbits + 1, (2**nbits-1) * 1))

#ax1.matshow(mask)
#ax2.matshow(mask_stretched, aspect='auto')

foo = np.repeat(modulate(np.array(list(range(4)))).astype(float), 1).
↪reshape((4, 31))
foo_stretched = np.repeat(modulate(np.array(list(range(4)))).astype(float), 10).
↪reshape(4, 310)

ax1.matshow(foo)
ax2.matshow(foo_stretched, aspect='auto')
```

```
(31,) (31,)
(31,) (31,)
```

[12]:
```
<matplotlib.image.AxesImage at 0x7ff8d8eb9e20>
```

[13]:
```
decimation = 10
signal_amplitude = 2.0
nbits = 5

foo = np.repeat(modulate(np.array([0, 1, 0, 0, 1, 1, 1, 0]), nbits) * 2.0 - 1,␣
↪decimation) * signal_amplitude
noise = colorednoise.powerlaw_psd_gaussian(1, len(foo))

sosh = sig.butter(4, 0.01, btype='highpass', output='sos', fs=decimation)
sosl = sig.butter(6, 1.0, btype='lowpass', output='sos', fs=decimation)
filtered = sig.sosfilt(sosh, sig.sosfilt(sosl, foo + noise))
#filtered = sig.sosfilt(sosh, foo + noise)

fig, ((ax1, ax3), (ax2, ax4)) = plt.subplots(2, 2, figsize=(16, 9))
fig.tight_layout()

ax1.plot(foo + noise)
ax1.plot(foo)
ax1.set_title('raw')

ax2.plot(filtered)
ax2.plot(foo)
ax2.set_title('filtered')
```

```
ax3.plot(correlate(foo + noise, nbits=nbits, decimation=decimation))
ax3.set_title('corr raw')

ax3.grid()

ax4.plot(correlate(filtered, nbits=nbits, decimation=decimation))
ax4.set_title('corr filtered')
ax4.grid()

rms = lambda x: np.sqrt(np.mean(np.square(x)))
rms(foo), rms(noise)
```

```
(31,) (31,)
```

```
(31,) (31,)
(31,) (31,)
```

[13]:
```
(2.0, 1.0121324810255907)
```

[14]:
```
with open('/mnt/c/Users/jaseg/shared/raw_freq.bin', 'rb') as f:
    mains_noise = np.copy(np.frombuffer(f.read(), dtype='float32'))
    print('mean:', np.mean(mains_noise))
    mains_noise -= np.mean(mains_noise)
```

```
mean: 49.98625
```

[27]:
```
decimation = 10
signal_amplitude = 2.0e-3
nbits = 6

#test_data = np.random.randint(0, 2, 100)
#test_data = np.array([0, 1, 0, 0, 1, 1, 1, 0])
test_data = np.random.RandomState(seed=0xcbb3b8cf).randint(0, 2 * (2**nbits),␣
 ↪128)
#test_data = np.random.RandomState(seed=0).randint(0, 8, 64)
#test_data = np.array(list(range(8)) * 8)
#test_data = np.array([0, 1] * 32)
#test_data = np.array(list(range(64)))

foo = np.repeat(modulate(test_data, nbits) * 2.0 - 1, decimation) *␣
 ↪signal_amplitude
noise = np.resize(mains_noise, len(foo))
#noise = 0

sosh = sig.butter(3, 0.01, btype='highpass', output='sos', fs=decimation)
sosl = sig.butter(3, 0.8, btype='lowpass', output='sos', fs=decimation)
#filtered = sig.sosfilt(sosh, sig.sosfilt(sosl, foo + noise))
```

```
filtered = sig.sosfilt(sosh, foo + noise)

cor1 = correlate(foo + noise, nbits=nbits, decimation=decimation)
#cor2 = correlate(filtered, nbits=nbits, decimation=decimation)

#cor2_pe = correlate(filtered, nbits=nbits, decimation=decimation,␣
 ↪mask_filter=lambda mask: sig.sosfilt(sosh, sig.sosfiltfilt(sosl, mask)))

sosn = sig.butter(12, 4, btype='highpass', output='sos', fs=decimation)
#cor1_flt = sig.sosfilt(sosn, cor1)
#cor2_flt = sig.sosfilt(sosn, cor2)
#cor1_flt = cor1[1:] - cor1[:-1]
#cor2_flt = cor2[1:] - cor2[:-1]

fig, ((ax1, ax3), (ax2, ax4)) = plt.subplots(2, 2, figsize=(16, 9))
fig.tight_layout()

ax1.plot(foo + noise)
ax1.plot(foo)
ax1.set_title('raw')
ax1.grid(axis='y')

ax2.plot(filtered)
ax2.plot(foo)
ax2.set_title('filtered')
ax2.grid(axis='y')

for i in range(0, len(foo) + 1, decimation*(2**nbits - 1)):
    ax1.axvline(i, color='gray', alpha=0.5, lw=1)
    ax2.axvline(i, color='gray', alpha=0.5, lw=1)

for i, (color, trace) in enumerate(zip(plt.cm.winter(np.linspace(0, 1, cor1.
 ↪shape[0])), cor1.T)):
    if i%3 == 0:
        ax3.plot(trace + 0.5 * i, alpha=1.0, color=color)
ax3.set_title('corr raw')
ax3.grid()

#ax4.plot(cor2[:4].T)
#ax4.set_title('corr filtered')
#ax4.grid()
ax4.matshow(cor1, aspect='auto')

#ax5.plot(cor1_flt)
#ax5.set_title('corr raw (highpass)')
#ax5.grid()
```

```
#ax6.plot(cor2_flt)
#ax6.set_title('corr filtered (highpass)')
#ax6.grid()

#ax6.plot(cor2_pe[:4].T)
#ax6.set_title('corr filtered w/ mask preemphasis')
#ax6.grid()

rms = lambda x: np.sqrt(np.mean(np.square(x)))
rms(foo), rms(noise)
```

```
(63,) (63,)
(63,) (63,)
```

[27]:
```
(0.0020000000000000005, 0.014544699)
```

[23]:
```
fig, ax = plt.subplots()

seq = np.repeat(gold(6)[29]*2 -1, decimation)
sosh = sig.butter(3, 0.01, btype='highpass', output='sos', fs=decimation)
sosl = sig.butter(3, 0.8, btype='lowpass', output='sos', fs=decimation)
seq_filtered = sig.sosfilt(sosh, sig.sosfiltfilt(sosl, seq))
#seq_filtered = sig.sosfilt(sosh, seq)

ax.plot(seq)
ax.plot(seq_filtered)
```

```
(63,) (63,)
```

[23]:
```
[<matplotlib.lines.Line2D at 0x7ff8ad6f3b50>]
```

[24]:
```
fig, axs = plt.subplots(3, 1, figsize=(9, 7), sharex=True)
fig.tight_layout()
axs = axs.flatten()
for ax in axs:
    ax.grid()

seq = np.repeat(gold(6)[29]*2 -1, decimation)
sosh = sig.butter(3, 0.1, btype='highpass', output='sos', fs=decimation)
sosl = sig.butter(3, 0.8, btype='lowpass', output='sos', fs=decimation)
cor2_pe_flt = sig.sosfilt(sosh, cor2_pe)
cor2_pe_flt2 = sig.sosfilt(sosh, sig.sosfiltfilt(sosl, cor2_pe))

axs[0].plot(cor2_pe)
axs[1].plot(cor2_pe_flt)
axs[2].plot(cor2_pe_flt2)
```

```
#for idx in np.where(np.abs(cor2_pe_flt2) > 0.5)
```

```
(63,) (63,)
```

```
      ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-24-f158dfc14cca> in <module>
         8 sosh = sig.butter(3, 0.1, btype='highpass', output='sos',␣
↪fs=decimation)
         9 sosl = sig.butter(3, 0.8, btype='lowpass', output='sos',␣
↪fs=decimation)
   ---> 10 cor2_pe_flt = sig.sosfilt(sosh, cor2_pe)
        11 cor2_pe_flt2 = sig.sosfilt(sosh, sig.sosfiltfilt(sosl, cor2_pe))
        12


       NameError: name 'cor2_pe' is not defined
```

[25]:
```
fig, ax = plt.subplots()
nonlinear_distance = lambda x: 100**(2*np.abs(0.5-x%1)) / (np.abs(x)+3)**2
x = np.linspace(-1.5, 5.5, 10000)
ax.plot(x, nonlinear_distance(x))
```

[25]:
```
[<matplotlib.lines.Line2D at 0x7ff8a9b7a820>]
```

[29]:
```
threshold_factor = 4.0
power_avg_width = 1024
max_lookahead = 6.5

bit_period = (2**nbits) * decimation
peak_group_threshold = 0.1 * bit_period

cor_an = cor1

#fig, (ax1, ax2, ax3) = plt.subplots(3, figsize=(12, 12))
fig, (ax1, ax3) = plt.subplots(2, figsize=(12, 5))
fig.tight_layout()

#ax1.matshow(sig.cwt(cor_an, sig.ricker, np.arange(1, 31)), aspect='auto')

#for i in np.linspace(1, 10, 19):
```

```
#    offx = 5*i
#    ax2.plot(sig.cwt(cor_an, sig.ricker, [i]).flatten() + offx, color='red')
#
#    ax2.text(-50, offx, f'{i:.1f}',
#        horizontalalignment='right',
#        verticalalignment='center',
#        color='black')
#ax2.grid()

ax3.grid()
print('cor_an', cor_an.shape)

cwt_res = np.array([ sig.cwt(row, sig.ricker, [0.73 * decimation]).flatten()␣
 ↪for row in cor_an ])
ax3.plot(cwt_res.T)
#def update(w = 1.0 * decimation):
#    line.set_ydata(sig.cwt(cor_an, sig.ricker, [w]).flatten())
#    fig.canvas.draw_idle()
#ipywidgets.interact(update)

print('cwt_res', cwt_res.shape)
th = np.array([ np.convolve(np.abs(row), np.ones((power_avg_width,))/
 ↪power_avg_width, mode='same') for row in cwt_res ])
ax1.plot(th.T)
print('th', th.shape)

def compare_th(elem):
    idx, (th, val) = elem
    #print('compare_th:', th.shape, val.shape)
    return np.any(np.abs(val) > th*threshold_factor)

print([ (a.shape, b.shape) for a, b in zip(th.T, cwt_res.T) ][:5])

peaks = [ list(group) for val, group in itertools.groupby(enumerate(zip(th.T,␣
 ↪cwt_res.T)), compare_th) if val ]
print('peaks:', len(peaks))
peak_group = []
for group in peaks:
    pos = np.mean([idx for idx, _val in group])
    pol = np.mean([max(val.min(), val.max(), key=abs) for _idx, (_th, val) in␣
 ↪group])
    pol_idx = np.argmax(np.bincount([ np.argmax(np.abs(val)) for _idx, (_th,␣
 ↪val) in group ]))
    #print(f'group', pos, pol, pol_idx)
    #for pol, (_idx, (_th, val)) in zip([max(val.min(), val.max(), key=abs) for␣
 ↪_idx, (_th, val) in group], group):
```

```
    #    print('    ', pol, val)
    ax3.axvline(pos, color='cyan', alpha=0.3)

    if not peak_group or pos - peak_group[-1][1] > peak_group_threshold:
        if peak_group:
            peak_pos = peak_group[-1][3]
            ax3.axvline(peak_pos, color='red', alpha=0.6)
            #ax3.text(peak_pos-20, 2.0, f'{0 if pol < 0 else 1}',␣
↪horizontalalignment='right', verticalalignment='center', color='black')

        peak_group.append((pos, pos, pol, pos, pol_idx))
        #ax3.axvline(pos, color='cyan', alpha=0.5)

    else:
        group_start, last_pos, last_pol, peak_pos, last_pol_idx = peak_group[-1]

        if abs(pol) > abs(last_pol):
            #ax3.axvline(pos, color='magenta', alpha=0.5)
            peak_group[-1] = (group_start, pos, pol, pos, pol_idx)
        else:
            #ax3.axvline(pos, color='blue', alpha=0.5)
            peak_group[-1] = (group_start, pos, last_pol, peak_pos,␣
↪last_pol_idx)

avg_peak = np.mean(np.abs(np.array([last_pol for _1, _2, last_pol, _3, _4 in␣
↪peak_group])))
print('avg_peak', avg_peak)

noprint = lambda *args, **kwargs: None
def mle_decode(peak_groups, print=print):
    peak_groups = [ (pos, pol, idx) for _1, _2, pol, pos, idx in peak_groups ]
    candidates = [ (0, [(pos, pol, idx)]) for pos, pol, idx in peak_groups if␣
↪pos < bit_period*2.5 ]

    while candidates:
        chain_candidates = []
        for chain_score, chain in candidates:
            pos, ampl, _idx = chain[-1]
            score_fun = lambda pos, npos, npol: abs(npol)/avg_peak +␣
↪nonlinear_distance((npos-pos)/bit_period)
            next_candidates = sorted([ (score_fun(pos, npos, npol), npos, npol,␣
↪nidx) for npos, npol, nidx in peak_groups if pos < npos < pos +␣
↪bit_period*max_lookahead ], reverse=True)

            print(f'    candidates for {pos}, {ampl}:')
            for score, npos, npol, nidx in next_candidates:
```

```
                print(f'        {score:.4f} {npos:.2f} {npol:.2f} {nidx:.2f}')

            nch, cor_len = cor_an.shape
            if cor_len - pos < 1.5*bit_period or not next_candidates:
                score = sum(score_fun(opos, npos, npol) for (opos, _opol,␣
 ↪_oidx), (npos, npol, _nidx) in zip(chain[:-1], chain[1:])) / len(chain)
                yield score, chain

            else:
                print('extending')
                for score, npos, npol, nidx in next_candidates[:3]:
                    if score > 0.5:
                        new_chain_score = chain_score * 0.9 + score * 0.1
                        chain_candidates.append((new_chain_score, chain +␣
 ↪[(npos, npol, nidx)]))
        print('chain candidates:')
        for score, chain in sorted(chain_candidates, reverse=True):
            print('    ', [(score, [(f'{pos:.2f}', f'{pol:.2f}') for pos, pol,␣
 ↪_idx in chain])])
        candidates = [ (chain_score, chain) for chain_score, chain in␣
 ↪sorted(chain_candidates, reverse=True)[:10] ]

res = sorted(mle_decode(peak_group, print=noprint), reverse=True)
#for i, (score, chain) in enumerate(res):
#    print(f'Chain {i}@{score:.4f}: {chain}')
(_score, chain), *_ = res

def viz(chain):
    last_pos = None
    for pos, pol, nidx in chain:
        if last_pos:
            delta = int(round((pos - last_pos) / bit_period))
            if delta > 1:
                print(f'skipped {delta} symbols at {pos}')
            for i in range(delta-1):
                yield None
        ax3.axvline(pos, color='blue', alpha=0.5)
        decoded = nidx*2 + (0 if pol < 0 else 1)
        yield decoded
        ax3.text(pos-20, 0.0, f'{decoded}', horizontalalignment='right',␣
 ↪verticalalignment='center', color='black')

        last_pos = pos

decoded = list(viz(chain))
print('decoding [ref|dec]:')
failures = 0
```

```
for i, (ref, found) in enumerate(itertools.zip_longest(test_data, decoded)):
    print(f'{ref or -1:>3d}|{found or -1:>3d} {" " if ref==found else " " if␣
 ↪found else " "}', end='    ')
    if ref != found:
        failures += 1
    if i%8 == 7:
        print()
print(f'Symbol error rate e={failures/len(test_data)}')
print(f'maximum bitrate r={sampling_rate / decimation / (2**nbits) * nbits * (1␣
 ↪- failures/len(test_data)) * 3600} b/h')
#ax3.plot(th)
```

```
cor_an (65, 81269)
cwt_res (65, 81269)
th (65, 81269)
[((65,), (65,)), ((65,), (65,)), ((65,), (65,)), ((65,), (65,)), ((65,), (65,))]
peaks: 1852
avg_peak 1.6610203317347632
skipped 3 symbols at 42209.0
decoding [ref|dec]:
 10| 10      69| 69     124|124     102|102       2|  2       3|  3
78| 78      29| 29
122|123      73| 73      98| 98      34| 34      -1| -1      97| 97
7|  7      97| 97
 86| 86     120|120      95| 95      90| 90      49| 49      89| 89
83| 83      19| 19
 84| 84     117|117      92| 92     119|119      16| 16      45| 45
23| 23      16| 16
111|111       9|  9      89| 89      18| 18      36| 36       2|  2
115|115      40| 40
100|100     105|105      93| 93      85| 85     107|107      90| 90
62| 62     116|116
 42| 42     123|123      40| 40      -1| -1      77| 77      40| 40
57| 57     110|110
 29| 29      94| 94       1|  1      29| 29      71| 71     119|119
15| 15     115|115
120| -1      70| -1      50| 50      71| 71      50| 50      61| 61
38| 38       4|  4
  3|  3     124|124      95| 95      27| 27      48| 48     116|116
3|  3      63| 63
 19| 19      79| 79       2|  2      43| 43      92| 92       8|  8
65| 65      35| 35
 30| 30      73| 73      73| 73      38| 38      58| 58      49| 49
45| 45      58| 58
 46| 46     116|116     101|101       5|  5      78| 78     126|126
105| 76     108|108
 59| 59      46| 46      27| 27      14| 14      57| 57      81| 81
```

```
3|  3       9|  9
126|126      18| 55      76| 76     101|101     124|124       4|  4
3|  3     102|102
 79| 79     121|121     103|103      92| 92      30| 30       4|  4
103|103      59| 58
Symbol error rate e=0.046875
maximum bitrate r=321.6796875 b/h
```

[30]:
```
fig, axs = plt.subplots(2, 1, figsize=(9, 7))
fig.tight_layout()
axs = axs.flatten()
for ax in axs:
    ax.grid()

axs[0].plot(cor2_pe_flt2[1::10] - cor2_pe_flt2[:-1:10])
a, b = cor2_pe_flt2[1::10] - cor2_pe_flt2[:-1:10], np.array([0.0, -0.5, 1.0, -0.
↪5, 0.0])
axs[1].plot(np.convolve(a, b, mode='full'))
```

```
      ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-30-968181501cb1> in <module>
         5     ax.grid()
         6
   ----> 7 axs[0].plot(cor2_pe_flt2[1::10] - cor2_pe_flt2[:-1:10])
         8 a, b = cor2_pe_flt2[1::10] - cor2_pe_flt2[:-1:10], np.array([0.0, -0.
↪5, 1.0, -0.5, 0.0])
         9 axs[1].plot(np.convolve(a, b, mode='full'))


       NameError: name 'cor2_pe_flt2' is not defined
```