Домашнее задание 1

Домашнее задание 1 по курсу РСПИ

- вариант: 16
- группа: РЛ1-93
- выполнил: Шаповалов Иван
- github
- collab

Условие:

1. Напишите согласно вариантам из приложения следующие функции для оценки характеристик схемы компандирования:
- функция компрессора, на вход которой передается нормированный с нулевым математическим ожиданием вещественный сигнал (|s| <= 1) и настроечный параметр для заданной схемы сжатия;
- функция равномерного квантования, на вход которой передается сигнал с выхода компрессора, а на выходе формируются целочисленные отсчеты с заданной разрядностью b;
- функция экспандера для заданного закона компрессора и таких же параметрах, формирующая из целочисленных значений сигнала на входе вещественный сигнал с нулевым математическим ожиданием.
1. Изобразите в draw.io функциональную схему системы компандирования, включающую источник сигнала, функции преобразования и оконечную схему сравнения восстановленного сигнала и исходного. Добавьте ее в отчёт.
2. Сгенерируйте непериодический массив отсчетов входного сигнала (более 10 OOO отсчетов) по форме согласно варианту. Используйте иррациональные числа для нормированных частот повторения заданных последовательностей.
3. Сделайте преобразование над полученным сигналом по схеме из п.2 и вьтчислите отношение С/Ш по формуле (SQNR):
4. Постройке график зависимости SQNR от параметра компрессора и найдите оптимальное значение параметра по критерию максимума SQNR. Изобразите на одном графиках осциллограммы сигналов с и без компандирования при оптимальном и неоптимальном значении параметра сжатия. Сделайте выводы по полученным результатам.

Исходные данные

N	Схема сжатия	Разрядность АЦП,b	Вид сигнала
16	A-law	5	Коррелированный гауссовский случайный процесс с параметрами (0, 1)

Модель коррелированный гауссовский случайный процесс:

$$s(n) = a_1 s(n-1) + b_0 x(n) + b_1 x(n-1);$$

где $x(n)$ - гауссовский случайный процесс с параметрами $N(0,1)$.

Подключим требуемые библиотеки.

```
from math import sqrt, pi, exp, log, copysign, log10
from numpy import linspace, vectorize, random
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal as sg
import copy
```

Внесем требуемые константы.

```
b = 5
A = 87.56
1/A
```

```
0.011420740063956145
```

Отладка основных компонентов схемы обработки

Прежде чем собрать требуемую схему обработки, произведем отладку её компонентов. Убедимся, что они работают нормально.

Формирование входного сигнала

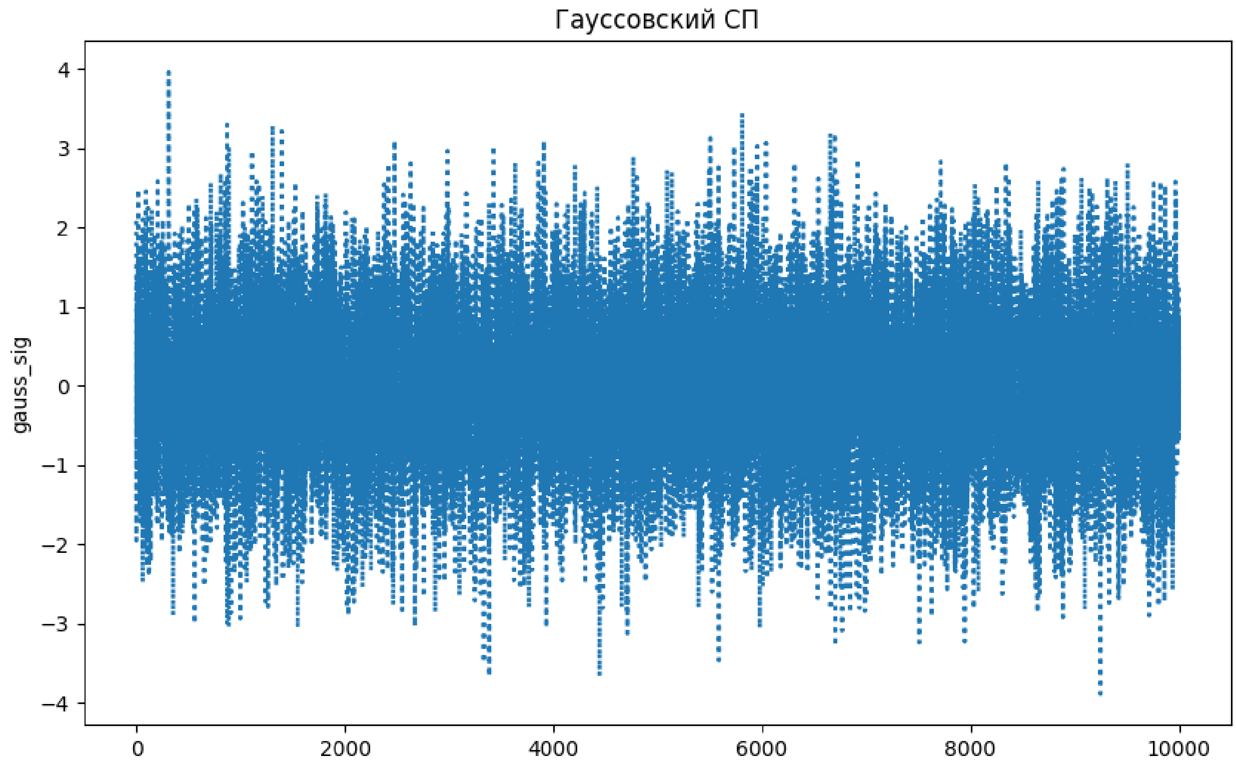
Для начала сформируем гауссовский СП с нулевым мат. ожиданием и стандартным отклонением = 1, используя пакет Numpy.

```
mean = 0
std = 1
size = int(1e4)
gauss_sig = np.random.normal(mean, std, size)
```

Получим следующую форму сигнала

```
fig, ax = plt.subplots()

plt.plot(gauss_sig,
        linewidth=2.0,
        linestyle=':',)
fig.set_size_inches(10, 6)

plt.title("Гауссовский СП")
plt.xlabel("")
```

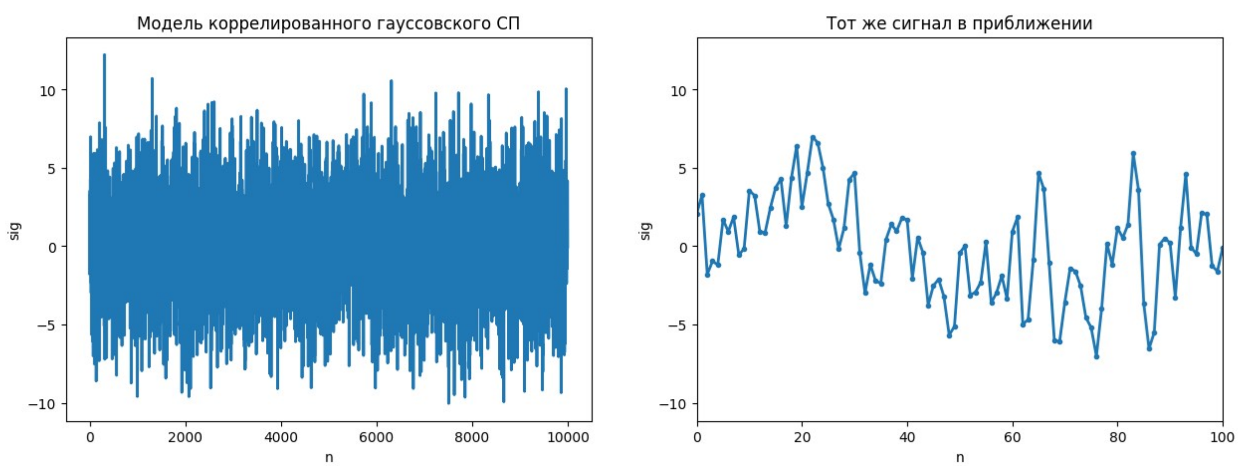
```
plt.ylabel("gauss_sig")

plt.show()
```



Теперь на основе полученных выборок сигнала сформируем сигнал требуемого, в условии задания, вида

```
def s(n, gauss_sig):
  a_1, b_0, b_1 = 0.5, 1, 2 #параметры принимаем такими так как не
заданно другое значение
  try:
    previous_val = b_0*gauss_sig[0] #храним предыдущее значение,
избавляемся от рекурсии
    returned_sig = [] #возвращаемый массив данных
    for n_i in range(n)[1:]:
      returned_sig.append(previous_val)
      previous_val = a_1*previous_val + b_0*gauss_sig[n_i] +
b_1*gauss_sig[n_i - 1]

    returned_sig.append(previous_val)
    return returned_sig
  except IndexError:
    print("Некорректный массив гауссовых значений")

n = np.arange(size)
sig = s(size, gauss_sig)
```

```python
# plot
fig, ax = plt.subplots(1,2)

plt.subplot(1, 2, 1)
plt.plot(n, sig, linewidth=2.0)

plt.title("Модель коррелированного гауссовского СП")
plt.xlabel("n")
plt.ylabel("sig")

plt.subplot(1, 2, 2)
plt.plot(n, sig, linewidth=2.0, marker=".")

plt.title("Тот же сигнал в приближении")
plt.xlabel("n")
plt.ylabel("sig")
plt.xlim(0,1e2)


fig.set_size_inches(15, 5)
plt.show()
```

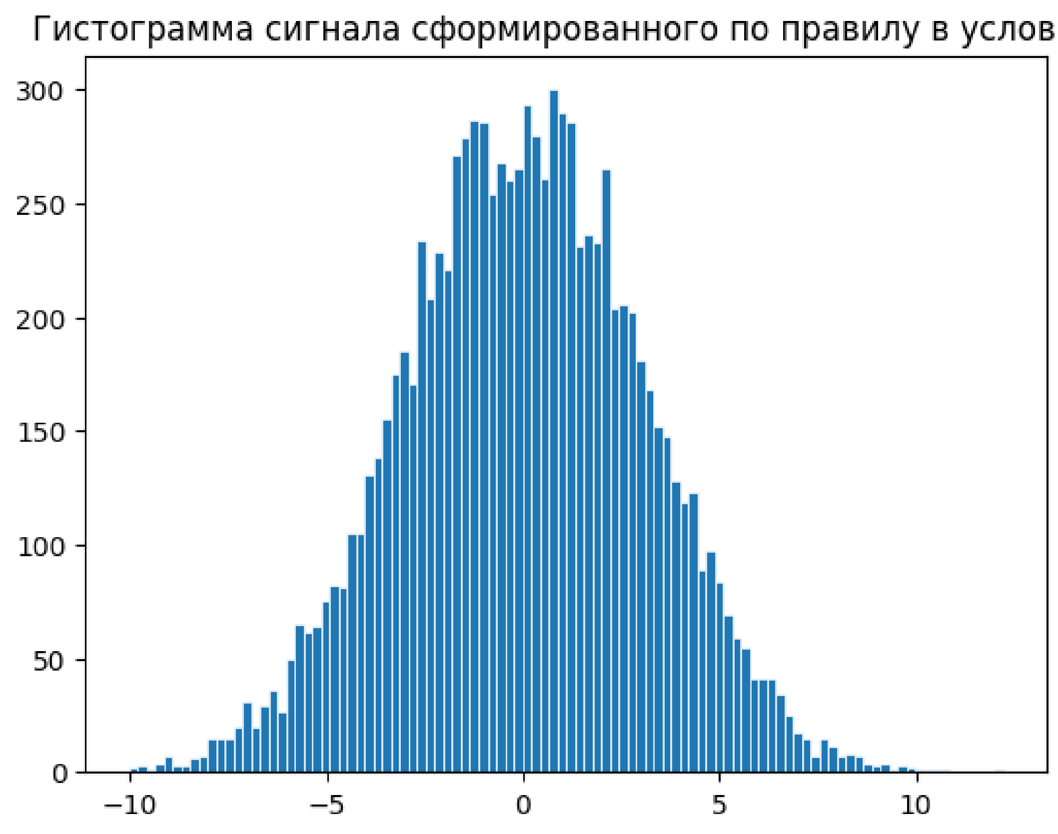


Изучим статистические характиристики полученного сигнала.

```python
print("Для ненормированного \"исходного\" сигнала:")
print("Среднее значение:", np.mean(sig))
print("Медиана:", np.median(sig))
print("Дисперсия:", np.var(sig))
print("Стандартное отклонение:", np.std(sig))
print("Минимальное значение:", np.min(sig))
print("Максимальное значение:", np.max(sig))
```

```
Для ненормированного "исходного" сигнала:
Среднее значение: 0.0020743114105561274
```

```
Медиана: 0.019997629954317836
Дисперсия: 9.428083847690443
Стандартное отклонение: 3.0705184981840516
Минимальное значение: -10.03724679343398
Максимальное значение: 12.22358312059983

fig, ax = plt.subplots()

ax.hist(sig, bins=int(size/100), linewidth=0.5, edgecolor="white")

# ax.set(xlim=(0, 8), xticks=np.arange(1, 8),
#        ylim=(0, 56), yticks=np.linspace(0, 56, 9))

plt.title("Гистограмма сигнала сформированного по правилу в условии")
plt.show()
```



Так как в условии требуеться нормированный сигнал с нулевым мат. ожиданием. Нормируем полученный сигнал

```
sig_norm = copy.copy(sig)
sig_norm = (sig_norm - np.mean(sig_norm))/np.std(sig_norm)
```

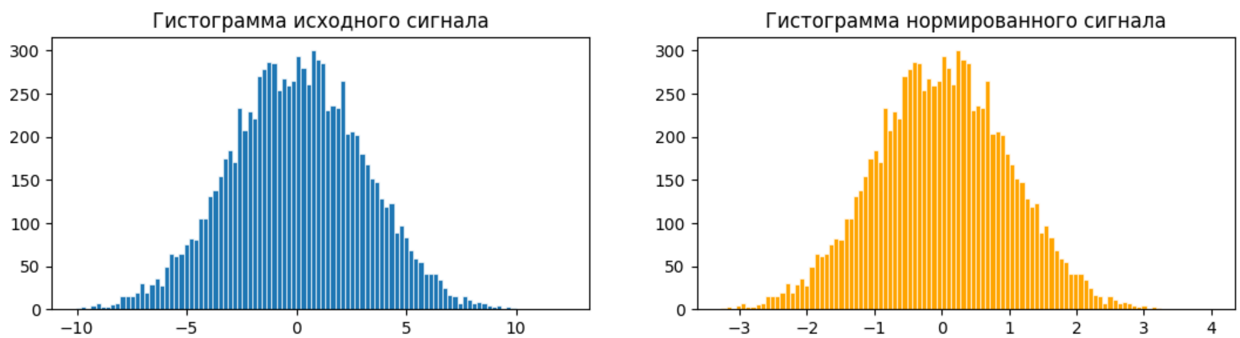
Теперь мы имеем действительно нормированный сигнал

```python
print("Для нормированного \"исходного\" сигнала:")
print("Среднее значение:", np.mean(sig_norm))
print("Медиана:", np.median(sig_norm))
print("Дисперсия:", np.var(sig_norm))
print("Стандартное отклонение:", np.std(sig_norm))
print("Минимальное значение:", np.min(sig_norm))
print("Максимальное значение:", np.max(sig_norm))
# print("===================")
```

```
Для нормированного "исходного" сигнала:
Среднее значение: 2.842170943040401e-18
Медиана: 0.005837228648634365
Дисперсия: 1.0
Стандартное отклонение: 1.0
Минимальное значение: -3.2695849612311187
Максимальное значение: 3.9802752585328007
```

```python
# %%capture
# plot
fig, (ax1, ax2, ax3) = plt.subplots(1,3)


ax1.plot(n, sig, linewidth=2.0)
ax1.set_title('Модель сигнала исходная')
ax1.set_xlabel('n')
ax1.set_ylabel('sig')

ax2.plot(n, sig_norm, linewidth=2.0, color="orange")
ax2.set_title('Модель сигнала нормированная ')
ax2.set_xlabel('n')
ax2.set_ylabel('sig_norm')

ax3.plot(n, sig, linewidth=2.0)
ax3.plot(n, sig_norm, linewidth=2.0, color="orange")
ax3.set_title('Модель исходная и нормированная ')
ax3.set_xlabel('n')




fig.set_size_inches(20, 5)
plt.show()
```


```
fig, (ax1, ax2) = plt.subplots(1,2)

ax1.hist(sig, bins=int(size/100), linewidth=0.5, edgecolor="white")
ax1.set_title('Гистограмма исходного сигнала')

ax2.hist(sig_norm, bins=int(size/100), linewidth=0.5,
edgecolor="white", color="orange")
ax2.set_title('Гистограмма нормированного сигнала')

fig.set_size_inches(13, 3)
plt.show()
```



Компрессирование

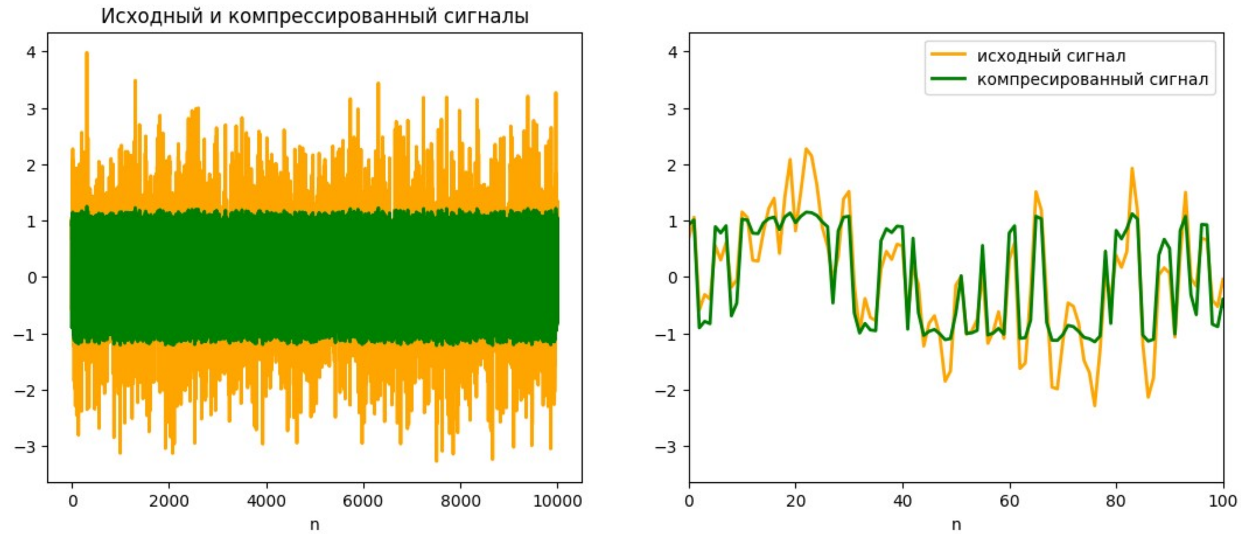
Теперь проведем **компрессирование** сигнала, по А-закону. По следующей формуле

\begin{equation} y = sgn(x) \begin{cases} \frac{A|x|}{1 + ln(A)}, |x| < \frac{1}{A}\ \frac{A|x|}{1 + ln(A)}, \frac{1}{A} \leq |x| \leq 1\ \end{cases},. \end{equation}

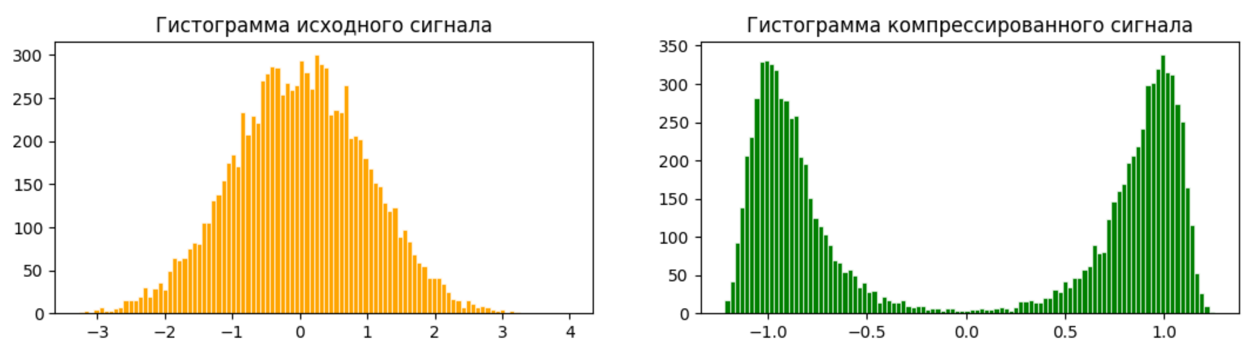
$A \approx 87.56$

```
def compessor(x, A):
  if 1/A <= abs(x) or abs(x) <= 1:
    return (1 + log(A*abs(x)))/(1 + log(A))*copysign(1,x)
  elif abs(x) < 1/A:
    return (A*abs(x))/(1 + log(A))*copysign(1,x)

compress_sig = vectorize(compessor)(sig_norm, A)
```

```python
# plot
fig, ax = plt.subplots(1, 2)


ax[0].plot(n, sig_norm,
        linewidth=2.0,
        label='исходный сигнал',
        color="orange")
ax[0].plot(n, compress_sig,
        linewidth=2.0,
        label='компресированный сигнал',
        color="green")
ax[0].set_title('Исходный и компрессированный сигналы')
ax[0].set_xlabel('n')



ax[1].plot(n, sig_norm,
        linewidth=2.0,
        label='исходный сигнал',
        color="orange")
ax[1].plot(n, compress_sig,
        linewidth=2.0,
        label='компресированный сигнал',
        color="green")
ax[1].set_xlabel('n')
ax[1].set_xlim(0,100)


# plt.xlabel("n")
# plt.ylabel("compress_sig")

plt.legend()
fig.set_size_inches(13, 5)
plt.show()
```




```
fig, (ax1, ax2) = plt.subplots(1,2)

ax1.hist(sig_norm, bins=int(size/100),
         linewidth=0.5,
         edgecolor="white",
         color="orange")
ax1.set_title('Гистограмма исходного сигнала')

ax2.hist(compress_sig, bins=int(size/100),
         linewidth=0.5,
         edgecolor="white",
         color="green")
ax2.set_title('Гистограмма компрессированного сигнала')

fig.set_size_inches(13, 3)
plt.show()
```



Стоит отметить, что на данный момент *трудно судить* о правильности компрессирования. С одной стороны, компрессирования ярко выражено в форме сигнала, но с другой на выходе компрессора должно быть равномерное распределние, что мы к сожалению не наблидаем.

Квантование сигнала

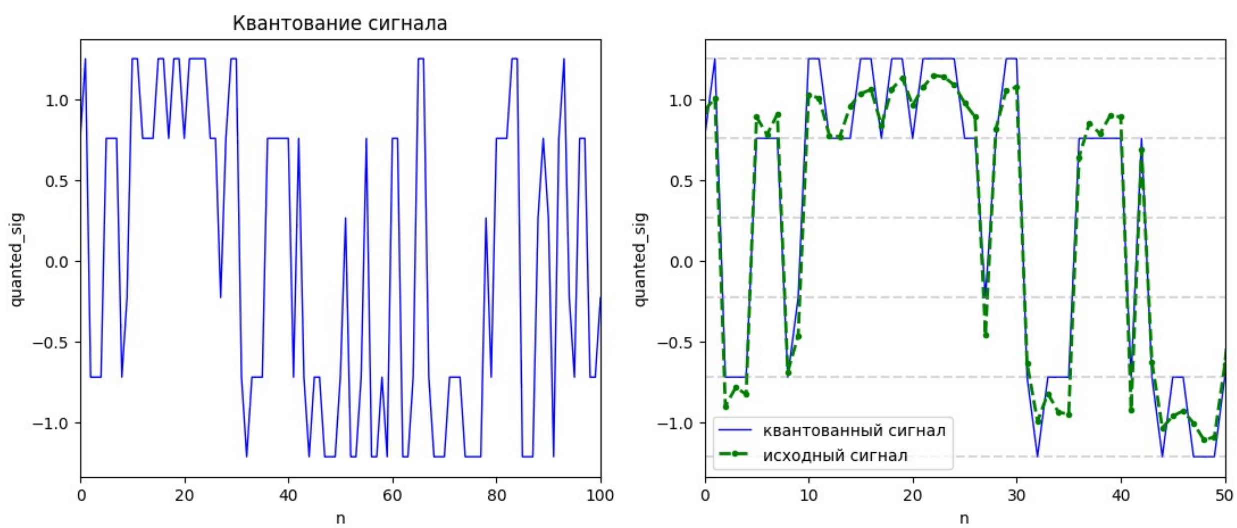
Чтобы не нарушать общности, перейдем к реализации блока **квантователя**. Квантователь реализован на идеи вычисления "расстояния" между уровнями квантования и текущим уровнем сигнала. После вычисления метрики, производится поиск ближайщего уровня квантования к текущему отсчету сигнала. Таким образом, получаем значание на выходе квантователя.

```
def quantizer(signal, N):
  quantized_signal = []
  step = (max(signal) - min(signal))/N
  levels = np.array([step*i for i in range(N + 1)]) + min(signal)
  for countdown in signal:
    distance = np.abs(levels - countdown)
    curent_level = levels[np.argmin(distance)]
    quantized_signal.append(curent_level)
  return quantized_signal, levels
```

Теперь проведем квантование полученного комперсиированного сигнала

```
quanted_sig, levels = quantizer(compress_sig, b)
# print(quanted_sig)

# plot
fig, ax = plt.subplots(1,2)

#=======================================

ax[0].plot(n, quanted_sig,
        linewidth=1.0,
        label="квантованный сигнал",
        color = "blue")

ax[0].set_title("Квантование сигнала")
ax[0].set_xlabel("n")
ax[0].set_ylabel("quanted_sig")
ax[0].set_xlim(0,1e2)

fig.set_size_inches(13, 5)
#=======================================

ax[1].plot(n, quanted_sig,
        linewidth=1.0,
        label="квантованный сигнал",
        color = "blue")

ax[1].set_xlabel("n")
ax[1].set_ylabel("quanted_sig")

ax[1].plot(n, compress_sig,
```

```
        linewidth=2.0,
        linestyle='--',
        marker='.',
        label="исходный сигнал",
        alpha=1,
        color="green")

for i_level in levels:
  ax[1].axhline(y=i_level,
              color='gray',
              linestyle='--',
              alpha = 0.3)

ax[1].legend()
ax[1].set_xlim(0,0.5e2)

#=======================================

plt.show()
```



Здесь можно наблюдать, что квантователь *выполянет свою функцию*.

Экспандирование сигнала

Теперь проведем операцию обратную компрессированию - **экспандирование**.
Преобразование будем проводить, что по следующей формуле

\begin{equation} x = sgn(y) \begin{cases} \frac{|y|(1 + ln(A))}{A}, |y| < \frac{1}{1 + ln(A)}\ \frac{e^{-1 + |y|(1 + ln(A))}}{A}, \frac{1}{1 + ln(A)} \leq |y| \leq 1\ \end{cases},. \end{equation}

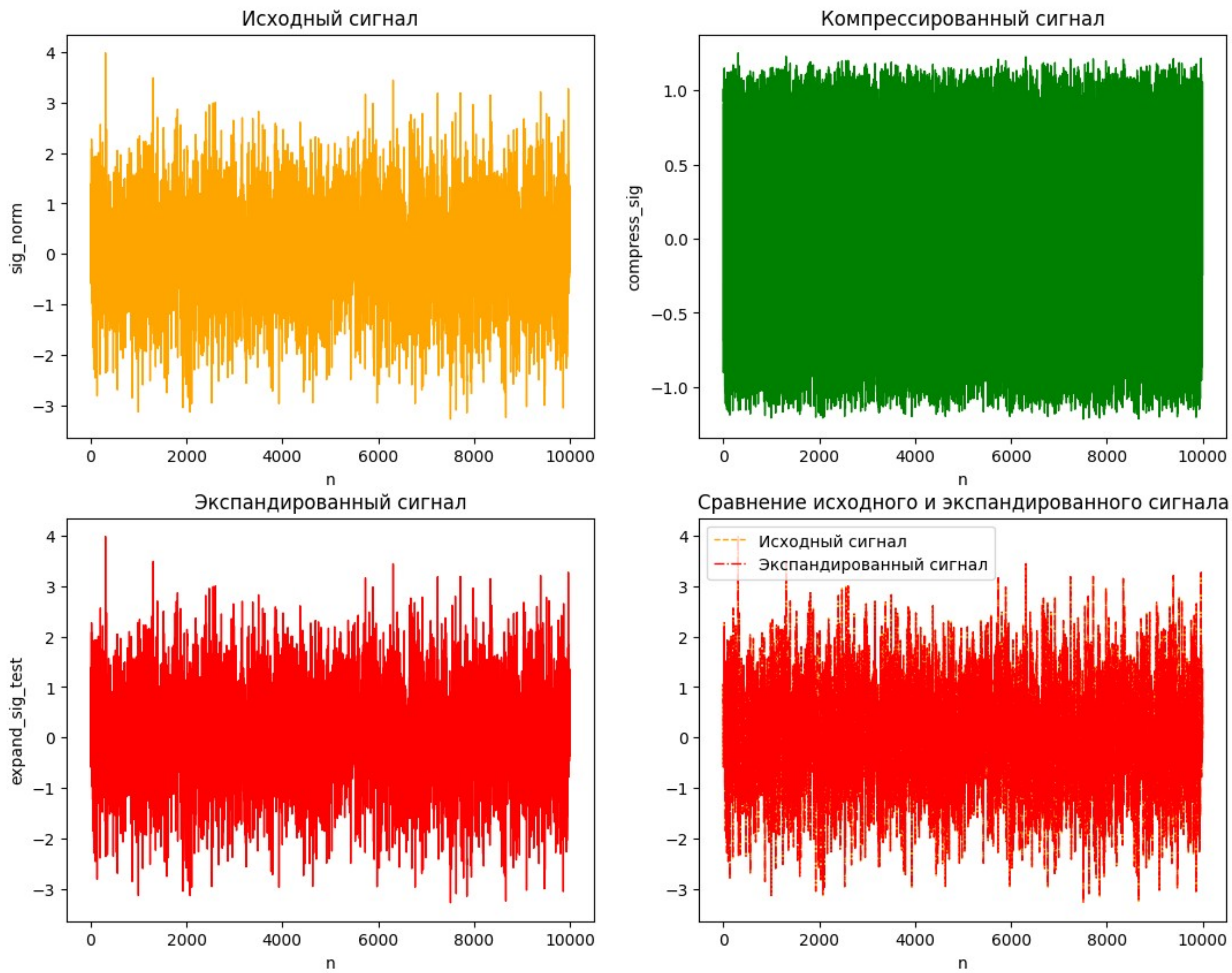
```
def expander(y, A):
  if abs(y) < 1/(1 + log (A)):
    return (abs(y)*(1 + log(A)))/A*copysign(1,y)
```

```
  elif 1/(1+log(A)) <= abs(y) or abs(y) <= 1:
    return exp(-1 + abs(y)*(1 + log(A)))/A*copysign(1,y)
```

!!Для **проверки** работы алгоритма, проведем преобразование **компресированного** сигнала

```
expand_sig_test = vectorize(expander)(compress_sig, A)


fig, axs = plt.subplots(2,2)

axs[0,0].plot(sig_norm,
         linewidth=1,
         color="orange")
axs[0,0].set_title('Исходный сигнал')
axs[0,0].set_xlabel("n")
axs[0,0].set_ylabel("sig_norm")

axs[0,1].plot(compress_sig,
         linewidth=1,
         color="green")
axs[0,1].set_title('Компрессированный сигнал')
axs[0,1].set_xlabel("n")
axs[0,1].set_ylabel("compress_sig")



axs[1,0].plot(expand_sig_test,
         linewidth=1,
         color="red")
axs[1,0].set_title('Экспандированный сигнал')
axs[1,0].set_xlabel("n")
axs[1,0].set_ylabel("expand_sig_test")


axs[1,1].plot(sig_norm,
         linewidth=1,
         color="orange",
        #  alpha=0.5,
         linestyle='--',
         label='Исходный сигнал')

axs[1,1].plot(expand_sig_test,
         linewidth=1,
         color="red",
        #  alpha=1.0,
         linestyle='-.',
         label='Экспандированный сигнал')
axs[1,1].legend()
axs[1,1].set_title('Сравнение исходного и экспандированного сигнала')
axs[1,1].set_xlabel("n")
```

```
fig.set_size_inches(13, 10)
plt.show()
```



Таким образом, можно судить, что операция компрессирование и экспандирование производят взаимо обраные операции, а значит алгоритмы работают *верно*.

Эксперименты с параметром компандирования

Вычисление SQNR

В общем виде схема обработки сигнал будет выглядеть следующим образом

Схема устройства

Сформируем требуемую схему обработки

```
def Process_Circuit(input_sig, A = A):
  compress_sig = vectorize(compessor)(input_sig, A)
  quanted_sig, levels = quantizer(compress_sig, b)
  expand_sig = vectorize(expander)(quanted_sig, A)
  return expand_sig

expand_sig = Process_Circuit(sig_norm)
```

Теперь найдем отношение сигнал шум.

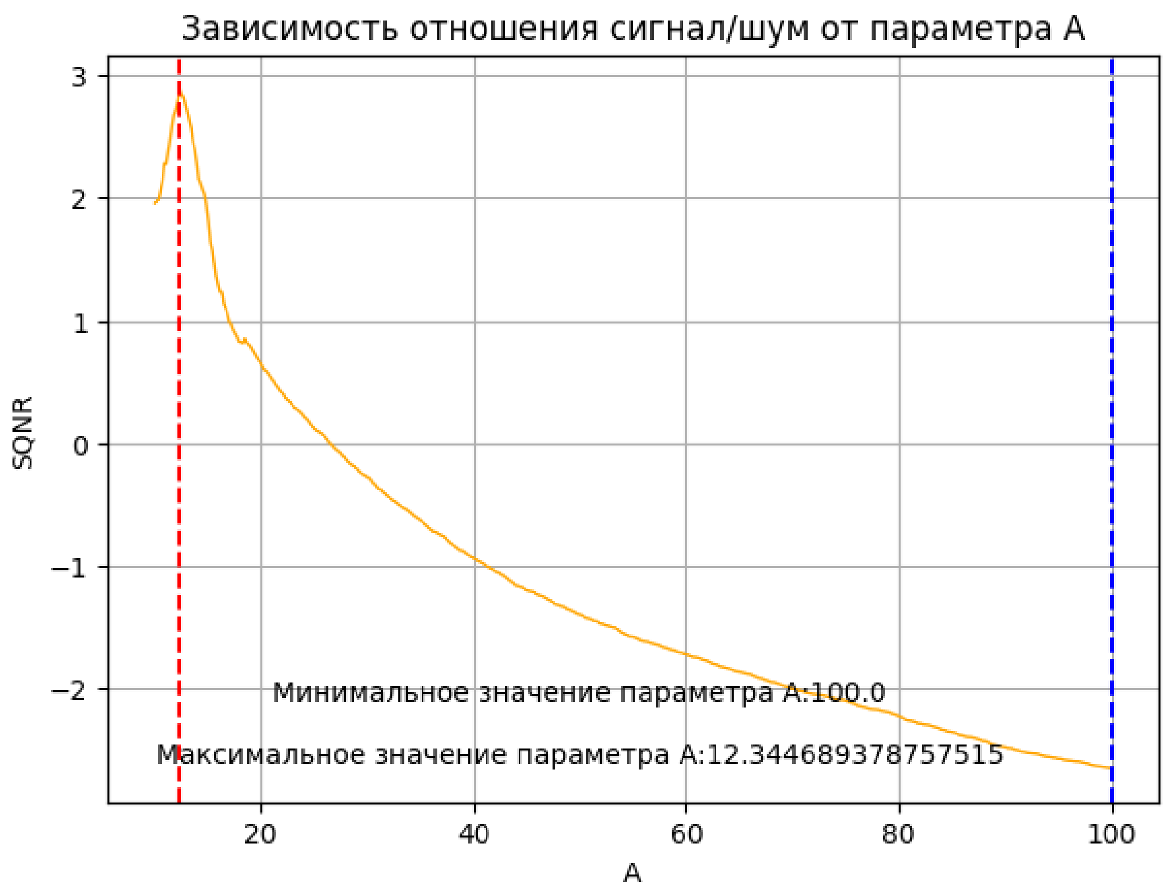
```
SQNR = lambda sig_norm, expand_sig:
10*log10(sum(sig_norm**2)/sum((sig_norm - expand_sig)**2))

f"Отношение сигнал/шум с установленным параметром A = 87.56 равен SQNR
= {SQNR(sig_norm, expand_sig)}"

{"type":"string"}
```

Зависимость SQNR от A

А теперь постоим зависимость SQNR от параметра A

```
A_array = linspace(10,100,500)

SQNR_array = [SQNR(sig_norm, Process_Circuit(sig_norm, A)) for A in
A_array]

maxA = A_array[np.argmax(SQNR_array)]
minA = A_array[np.argmin(SQNR_array)]

fig, ax = plt.subplots()
ax.plot(A_array,
        SQNR_array,
        linewidth=1,
        color="orange")

ax.set_xlabel('A')
ax.set_ylabel('SQNR')
ax.set_title('Зависимость отношения сигнал/шум от параметра A')

plt.axvline(maxA,
            color = "red",
            linestyle="--")

plt.axvline(minA,
            color = "blue",
            linestyle="--")


ax.text(max(A_array)/2,min(SQNR_array), f'Максимальное значение
```

```
параметра A:{maxA}', ha='center', va='bottom')
ax.text(max(A_array)/2,min(SQNR_array) + 0.5, f'Минимальное значение
параметра A:{minA}', ha='center', va='bottom')

plt.grid(True)
fig.set_size_inches(7, 5)
plt.show()
```

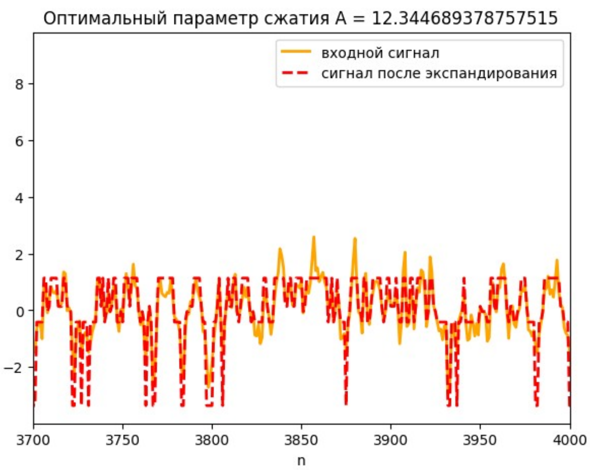


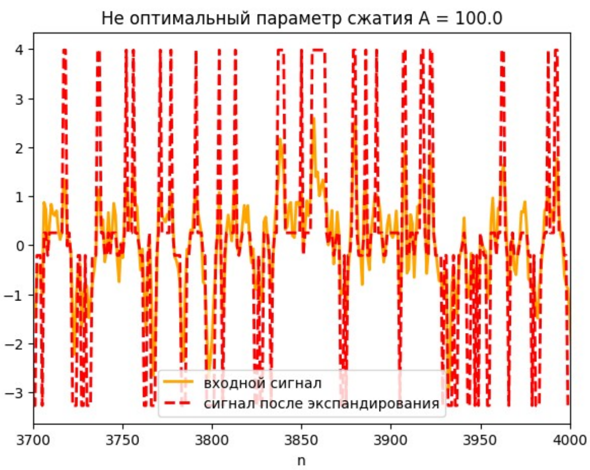
Сравнение оптимального и неоптимального значения SQNR

Тепрь изобразим на одном графике осциллограммы сигналов с и без компандирования при оптимальном(максимальное SQNR) и неоптимальном(минимальном SQNR) значении параметра сжатия.

```
expand_sig_opt = Process_Circuit(sig_norm, maxA)
expand_sig_bad = Process_Circuit(sig_norm, minA)

# plot
fig, ax = plt.subplots(1,2)

#====================================
```

```
ax[0].plot(n, sig_norm,
        linewidth=2.0,
        label="входной сигнал",
        color = "orange")

ax[0].plot(n, expand_sig_opt,
        linewidth=2.0,
        label="сигнал после экспандирования",
        linestyle='--',
        color = "red")

ax[0].set_title(f"Оптимальный параметр сжатия A = {maxA}")
ax[0].set_xlabel("n")
ax[0].legend()
ax[0].set_xlim(3700,4000)

#=======================================

ax[1].plot(n, sig_norm,
        linewidth=2.0,
        label="входной сигнал",
        color = "orange")


ax[1].plot(n, expand_sig_bad,
        linewidth=2.0,
        linestyle='--',
        # marker='.',
        # label="исходный сигнал",
        # alpha=1,
        label="сигнал после экспандирования",
        color="red")

ax[1].set_title(f"Не оптимальный параметр сжатия A = {minA}")
ax[1].legend()
ax[1].set_xlabel("n")
ax[1].set_xlim(3700,4000)

#=======================================

fig.set_size_inches(15, 5)
plt.show()
```





Вывод

При использовании неоптимального параметра компандирования сигнала происходят следующие изменения:

1. **Искажение сигнала**: Неоптимальный параметр компандирования может привести к искажению сигнала. Компандирование используется для увеличения динамического диапазона сигнала, но неправильно выбранный параметр может привести к искажениям и потере качества сигнала.

2. **Потеря деталей**: Неоптимальный параметр компандирования может привести к потере деталей в сигнале. Если параметр выбран слишком низким, то могут быть потеряны низкоуровневые сигналы и детали, что может привести к ухудшению качества воспроизведения.

3. **Увеличение шума**: Неоптимальный параметр компандирования может привести к увеличению уровня шума в сигнале. Если параметр выбран слишком высоким, то шумы и помехи могут быть усилены, что может негативно сказаться на качестве сигнала.

4. **Изменение динамического диапазона**: Неоптимальный параметр компандирования может изменить динамический диапазон сигнала. Если параметр выбран неправильно, то могут быть потеряны или искажены сигналы с низким или высоким уровнем амплитуды.

Важно выбирать оптимальные параметры компандирования сигнала, чтобы достичь наилучшего качества и минимизировать искажения и потери информации.