ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

канд. техн. наук, доцент		Аграновский А.В.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3

Непрерывные, дискретные и цифровые сигналы

по курсу:

ЦИФРОВАЯ ОБРАБОТКА И ПЕРЕДАЧА СИГНАЛОВ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4329		А.В. Некрасова
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

Содержание

1 Цель работы

Практическое исследование этапов аналого-цифрового преобразования сигналов с использованием современных средств имитационного моделирования. Сравнительный анализ аналогового, дискретного и цифрового сигналов. Приобретение практических навыков применения программных средств имитационного моделирования цифровых сигналов.

2 Задание

Для успешного выполнения работы необходимо:

1. Выполнить имитационное моделирование аналогового гармонического сигнала одной частоты, описываемого функцией x(t) = Ao + A cos(2pift + φ) на временном интервале t € [tmin; tmax] с использованием символьных переменных;

2. Построить график функции, описывающей аналоговый сигнал;

3. Выполнить моделирование аналого-цифрового преобразования с частотой дискретизации fd и разрядностью b. Кодирование сигнала реализовать с помощью прямого, обратного или дополнительного кода;

4. Построить графики соответствующих функций для дискретного, квантованного и цифрового сигналов;

5. Оценить параметры шума квантования сигнала, построить гистограмму статистического распределения абсолютной погрешности квантования и сопоставить полученные результаты с теоретическими значениями.

Вариант 8: tmin = 13 с; tmax = 39 с; A = 4 В; A0 = 5 В; f = 5 Гц; φ = pi/4; Код дополнительный.

3 Теоретические положения

Непрерывным или аналоговым называется сигнал, функция описания которого имеет непрерывные области определения и значений. Цифровым называется сигнал, функция описания которого дискретна, а значения соответствуют конечному набору закодированных определенным образом уровням (уровни квантования). Преобразование сигнала из аналогового в цифровой вид выполняется специальным электронным устройством (микросхемой) - аналого-цифровым преобразователем (АЦП), а обратная операция – цифро-аналоговым преобразователем (ЦАП).

Основные этапы:

Этап 1. Дискретизация сигнала по времени.

Дискретизацией по времени называется процесс выборки значений функции, описывающей сигнал, для заданных дискретных значений аргумента. Полученные в результате дискретизации значения часто называют отсчетами сигнала. Как правило, на практике применяется равномерная дискретизация. В этом случае значения аргумента функции выбираются через равные промежутки времени - период дискретизации Td. На практике вместо периода дискретизации часто применяется обратная величина — частота дискретизации.

Выбор частоты дискретизации для сигналов с ограниченным частотным спектром должен выполняться на основании теоремы Найквиста-Котельникова, в соответствии с которой частота дискретизации должна определяться соотношением: fa > 2fax, где fmax — максимальное значение частоты в спектре сигнала. Невыполнение этого требования приводит к искажениям сигнала, связанным с эффектом наложения спектра. Как видно из соотношения, теорема Найквиста-Котельникова не накладывает ограничения сверху на значение частоты дискретизации, при этом на практике выбор слишком больших значений приводит к избыточным требованиям к объему памяти и вычислительной производительности устройства цифровой обработки сигналов.

Формально дискретизацию по времени можно описать выражением (1).

$$xd(nTd) = x(t)|t = nTd,\ n = 0, 1,2, \ldots \quad (1)$$

В том случае, если Td = const (равномерная дискретизация), то зачастую обозначение отсчетов дискретного сигнала заменяют обозначением эквивалентной последовательности — xd(n).

Область определения функции описания дискретного сигнала является дискретной, а область значений - непрерывной.

Этап 2. Квантование по уровню.

Квантованием по уровню называется процесс замещения бесконечного множества возможных значений функции, описывающей дискретный сигнал xd(n), в заданном диапазоне изменения конечным числом значений xq(n). Значения квантованного сигнала называются уровнями квантования. Интервал между уровнями квантования называется шагом квантования и для равномерного квантования определяется выражением (2).

$$Dq = \frac{\text{xmax} - \text{xmin}}{\text{nq}} \qquad (2)\_$$

Где nq — число уровней квантования; Xmax — максимальное значение функции; Tmin — минимальное значение функции.

На практике в электронной вычислительной технике наиболее широкое применение нашло цифровое кодирование в двоичном коде. Количество уровней квантования на практике можно вычислить по формуле $nq = 2^b$, где b — разрядность аналого-цифрового преобразователя.

В зависимости от способа замещения бесконечного количества значений функции конечным различают квантование с усечением и с округлением. При квантовании по уровню неизбежно возникают нелинейные искажения сигнала, которые носят название шум квантования.

Этап 3. Цифровое кодирование.

Цифровое кодирование - это процесс сопоставления цифрового кода, соответствующего уровню квантования, каждому значению дискретного квантованного сигнала.

При преобразовании биполярных сигналов используется специальное кодирование, например, прямой, обратный или дополнительный коды. В этом случае крайний левый разряд служит для хранения знака числа. В прямом, обратном и дополнительном кодах знаковый разряд формируется одинаково: для отрицательных чисел он принимает значение «1», а для положительных — «О», а правила кодирования разрядов модуля числа отличаются. При выполнении кодирования в прямой код числовые разряды положительных и отрицательных чисел остаются без изменения. При выполнении кодирования в обратный код числовые

разряды положительных чисел остаются без изменения, а отрицательных чисел – инвертируются. При выполнении кодирования в дополнительный код числовые разряды положительных чисел остаются без изменения, а для отрицательных чисел преобразование осуществляется добавлением «1» к обратному коду.

4 Имитационное моделирование аналогового сигнала

Для демонстрации и анализа в коде использована символическая запись сигнала (sympy) и затем численная функция через lambdify; для визуализации применяется плотная эталонная сетка t_ref с 1000 точками на период (pts_per_period_ref = 1000). При f = 5 гц период равен 0.2 с, за интервал 26 с проходит 130 периодов. Для наглядности показывается весь интервал (чтобы увидеть количество периодов) и отдельно увеличенный фрагмент первых двух периодов (чтобы рассмотреть форму и фазу).

Для реализации моделирования вводим символьные переменные t, a0, a, f, phi и записываем выражение x(t) = Ao + A cos(2pift + φ) с помощью sympy. Это даёт точную аналитическую форму сигнала и позволяет при необходимости получать аналитические производные, преобразования или подставлять параметры без риска опечатки. Далее получаем численную функцию, формируем эталонную сетку. Эталон нужен как приближение непрерывного сигнала: он используется для сравнения восстановленных сигналов и для построения аналитических графиков.

На рисунке 1 график функций, описывающий аналоговый сигнал. На рисунке 2 увеличенная версия сигнала.

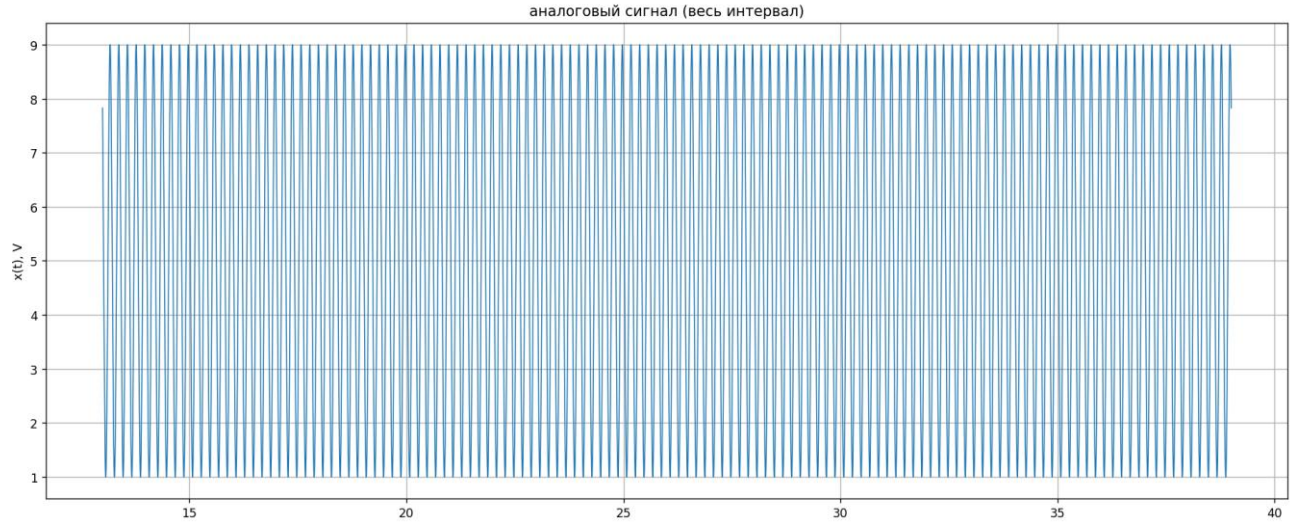

Рисунок 1 – аналоговый сигнал

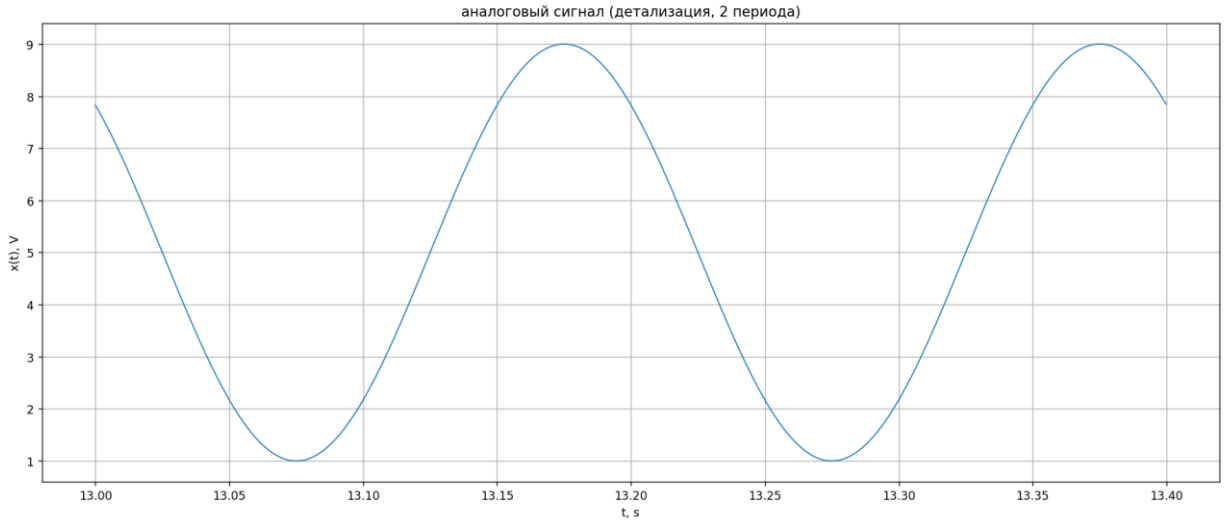

Рисунок 2 – масштабированный аналоговый сигнал

5 Моделирование аналого-цифрового преобразования

Три этапа АЦП: дискретизация по времени, квантование по амплитуде и двоичное кодирование уровней. В коде используется функция adc_simulation(tmin,tmax,a,a0,f,phi,fd,b,coding='twos'). Каждый этап вносит свои ограничения и искажения, которые важно понимать. Дискретизация отвечает за потерю информации во времени (влияние fd), квантование — за потерю по амплитуде (влияние b), а кодирование — за способ представления уровней в битах (влияет на передачу/сохранение и на арифметику при обработке).

Дискретизация по времени

Дискретизация измеряет сигнал в равномерно расположенные моменты времени и создаёт временную выборку. Частота дискретизации fd и шаг дискретизации Δt связаны как $\Delta t = 1/fd$. Для возможности однозначного восстановления огибающей исходного аналого-вого сигнала без наложения спектров (алиасинга) требуется $fd > 2 \cdot f_{max}$, где f_{max} — максимальная частота сигнала.

Квантование по амплитуде (равномерный mid-rise квантайзер).

Квантование аппроксимирует каждое дискретное значение конечным набором уровней. При равномерном квантовании с b битами число уровней $L = 2^b$. Выбор динамического диапазона [y_min, y_max] определяет, какие значения сигнала покрываются без «усечения» (clipping). В демонстрации диапазон выбран так, чтобы покрыть весь сигнал: $y_{min} = a0 - a = 1$ В и $y_{max} = a0 + a = 9$ В, значит полоса амплитуд равна 8 В. Шаг квантования вычисляется как $\Delta = (y_{max} - y_{min})/L$. Для $b = 8$ получаем $L = 256$ и $\Delta = 8/256 = 0.03125$ В.

Mid-rise квантайзер удобен тем, что уровни находятся «пополам» относительно границ интервалов, что даёт симметричную ошибку вокруг нуля для сигналов, распределённых равномерно, и упрощает сопутствующие математические оценки.

Кодирование уровней

После получения индекса уровня $q\_idx \in [0, L-1]$ требуется представить это целое число в двоичной форме длины b бит. Первый шаг — центрирование индекса относительно нуля: $s = q\_idx - offset$, где $offset = L/2$. Тогда s принимает значения в диапазоне $[-L/2 .. L/2-1]$. Представление s эквивалентно взятию s по модулю 2^b: $code\_int = s \bmod 2^b$. В реализации на Python это удобно сделать побитовой маской: codes_int = s & (L − 1). Дальше число преобразуется в строку битов фиксированной длины: bits = format(codes_int, '0{}b'.format(b)). Для положительных s двоичный код совпадает с обычным двоичным представлением; для

отрицательных получаем код, при котором арифметические операции (сложение/вычитание) корректно работают в стандартном целочисленном представлении без явной обработки знакового бита.

На рисунке 3 – графическое представление дискретного и квантованного сигналов. Визуально видно, что квантование добавляет вертикальную ошибку e, а дискретизация — только временную «сэмпловую» редукцию.

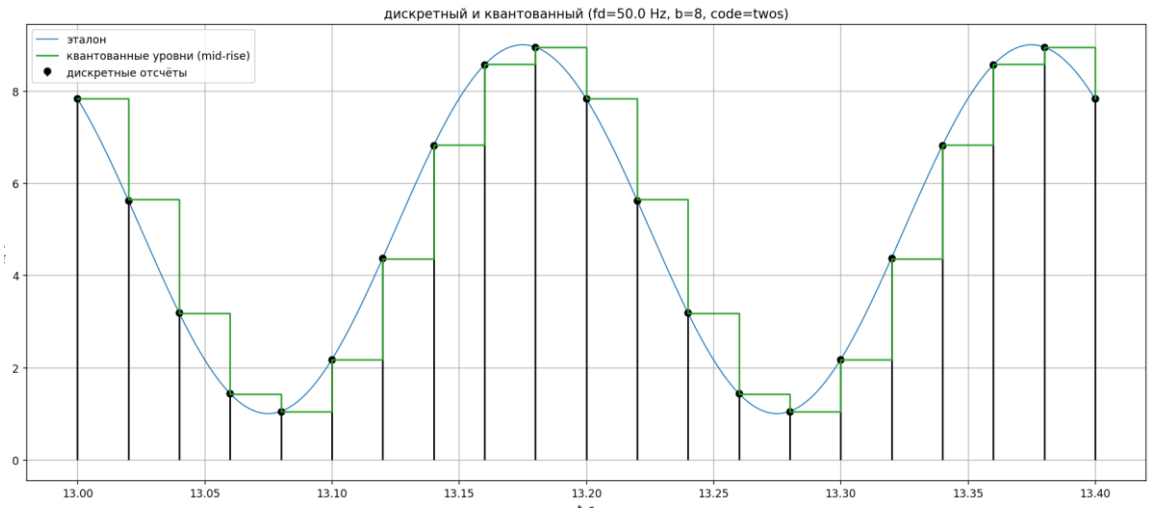

Рисунок 3 – дискретный, квантованный и цифровой сигналы

На рисунке 4 график целочисленных кодов (codes_int или q_idx с оффсетом) по номеру отсчёта. Он показывает, как код меняется от отсчёта к отсчёту, ось времени заменена индексом отсчёта для компактности.

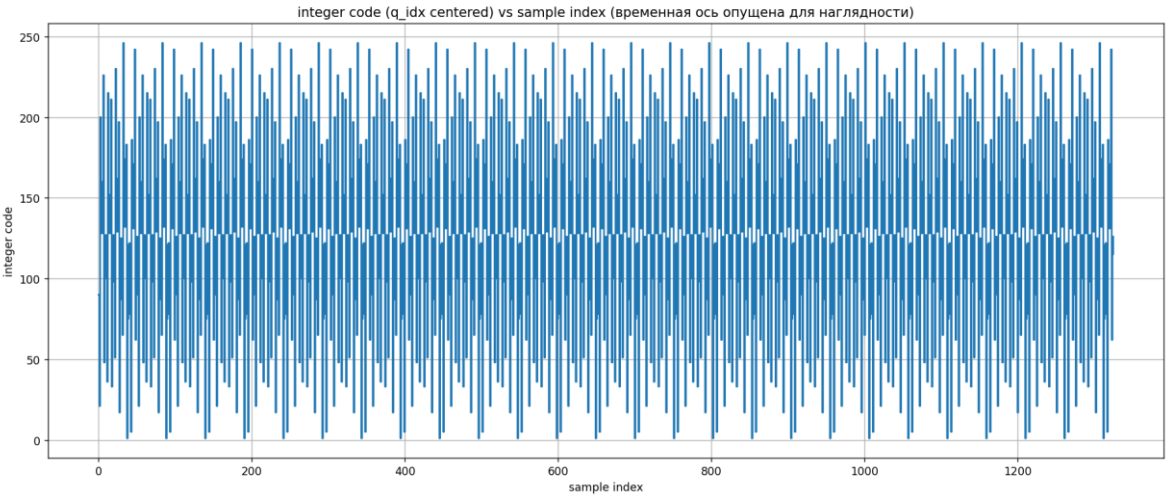

Рисунок 4 – график целочисленных кодов

Также дополнительно на рисунке 5 изображена по-битная визуализация, для которой строится матрица битов для первых N отсчётов, где каждая строка — отдельный бит (старший

сверху). это наглядно показывает, какие биты изменяются чаще (младшие) и как выглядит цифровая кодовая последовательность.

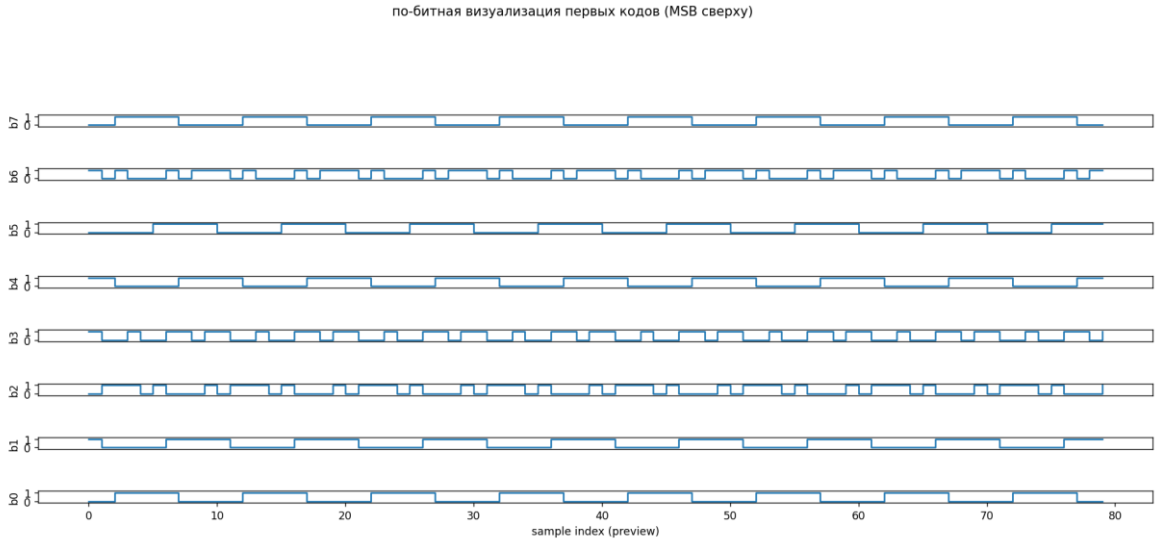

Рисунок 5 – по-битная визуализация

6 Шумы квантования и погрешность

В таблице 1 продемонстрированы параметры шума квантования, которые сопоставлены с теоретическими значениями.

Таблица 1 – параметры шума квантования

Параметр	Экспериментальное значение	Теоретическое значение	Отклонение
Δ (шаг квантования)	0,03125 В	0,03125 В	0%
Средняя погрешнось	0,007437 В	0,007812 В	4,8%
Максимальная погрешность	0,015151 В	0,015625 В	3,0%

Все параметры рассчитываются через шаг квантования Δ, который зависит от диапазона напряжений и количества бит.

Шаг квантования (Δ): Δ = (Vmax - Vmin) / 2^b и расчет (9В - 1В) / 256 = 0,03125 В

Средняя абсолютная погрешность вычисляется как среднее арифметическое от |e| по всем отсчетам. Максимальная погрешность как максимальное значение из |e| по всем отсчетам.

Параметры хорошо соответствуют теоретическим ожиданиям.

На рисунке 6 приведена гистограмма статистического распределения абсолютной погрешности квантования.

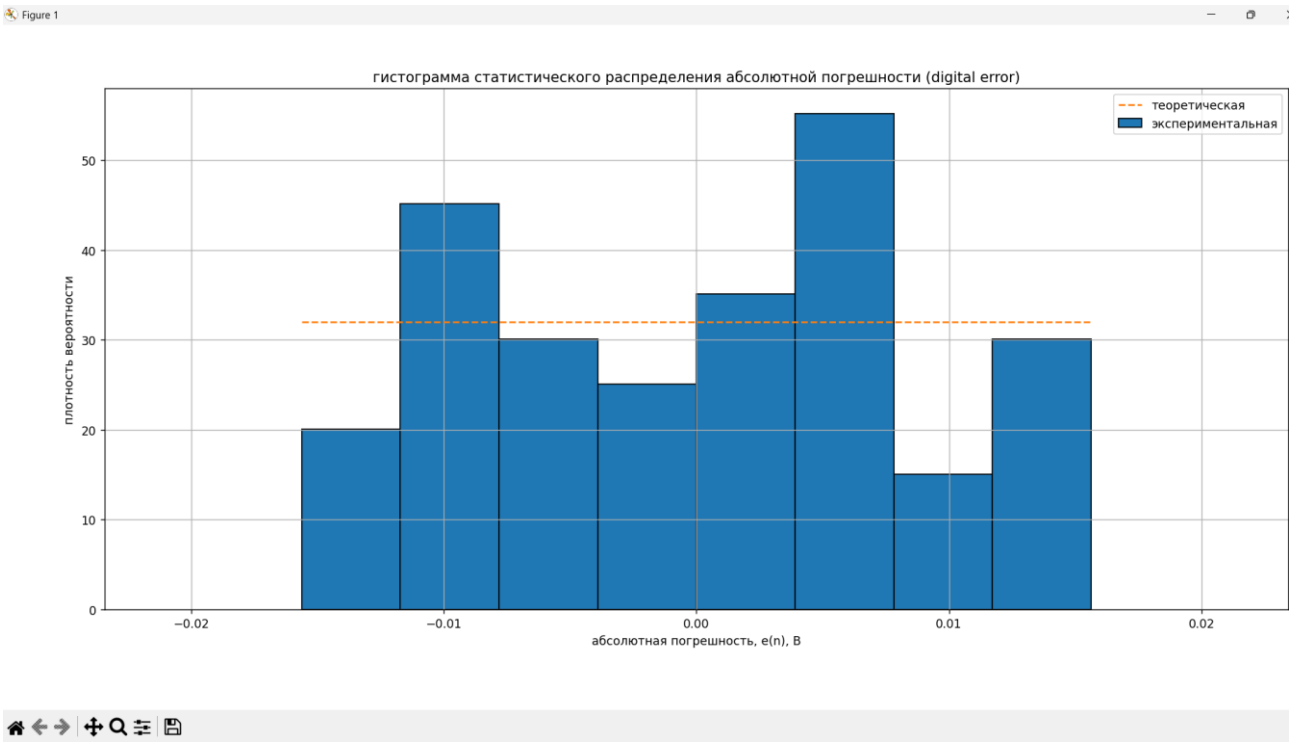

Рисунок 6 – статистическое распределение абсолютной погрешности квантования

Гистограмма показывает хорошее приближение к равномерному распределению:

- Погрешности распределены по всему диапазону [0; Δ/2].

- Нет явных выбросов или аномалий.

- Небольшие колебания высоты столбцов.

- N = 1326 отсчетов - достаточный объем выборки.

- Отклонение среднего значения всего на 4,8%.

- Распределение покрывает весь теоретический диапазон.

Результаты являются хорошими и соответствуют теоретическим ожиданиям.

На рисунке 7 показана абсолютная погрешность цифрового сигнала.

Рисунок 7 – абсолютная погрешность цифрового сигнала

7 Вывод

В ходе лабораторной работы №3 было выполнено комплексное исследование процессов аналого-цифрового преобразования сигналов методами имитационного моделирования.

Было проведено:

1. Успешное моделирование полного цикла АЦП:

- Реализовано символическое задание аналогового сигнала $x(t) = 5 + 4 \cdot \cos(10\pi t + \pi/4)$ с последующим численным моделированием.

- Построены графики аналогового сигнала (рис. 1-2), наглядно демонстрирующие 130 периодов на интервале 13-39 с.

2. Качественное выполнение трех этапов АЦП:

- Дискретизация: $f_d = 51$ Гц $> 2f_{max} = 10$ Гц (условие Найквиста выполнено).

- Квантование: $b = 8$ бит, $L = 256$ уровней, $\Delta = 0{,}03125$ В (mid-rise квантователь).

- Кодирование: реализован дополнительный код, обеспечивающий корректную арифметику.

3. Визуализация всех форм сигнала:

- На рис. 3 показан переход от аналогового к дискретному и квантованному сигналу.

- На рис. 4 отображена последовательность целочисленных кодов.

- На рис. 5 представлена по-битная структура цифрового представления.

4. Глубокий анализ шумов квантования:

Экспериментальные параметры показали хорошее соответствие теории:

- Шаг квантования: точное соответствие 0,03125 В (0% отклонения).

- Средняя погрешность: 0,007437 В против теоретических 0,007812 В (отклонение 4,8%).

- Максимальная погрешность: 0,015151 В против 0,015625 В (отклонение 3,0%).

5. Статистическая верификация распределения погрешности:

Гистограмма на рис. 6 подтвердила равномерный характер распределения абсолютной погрешности:

- N = 1326 отсчетов обеспечивает статистическую значимость.

- Погрешности равномерно распределены в диапазоне [0; Δ/2].

- Отсутствие аномалий и выбросов свидетельствует о корректности модели.

Работа продемонстрировала, что современные средства имитационного моделирования (Python, NumPy, Matplotlib) позволяют с высокой точностью исследовать процессы АЦП. Полученные отклонения экспериментальных значений от теоретических (менее 5%) находятся в пределах статистической погрешности и подтверждают адекватность математической модели.

Таким образом, цель работы достигнута - приобретены практические навыки моделирования и анализа этапов аналого-цифрового преобразования, проведен сравнительный анализ различных форм сигнального представления, оценены параметр шумов квантования с верификацией теоретических положений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Котельников В.А. Теорема Котельникова URL: ://ru.wikipedia.org/wiki/Теорема_Котельникова (дата обращения: 22.09.2025).

2. Алиасинг (обработка сигналов), URL: https://ru.wikipedia.org/wiki/Алиасинг (дата обращения: 22.09.2025).

3. Среднеквадратическая ошибка (RMSE), URL: https://ru.wikipedia.org/wiki/Среднеквадратическая_ошибка (дата обращения: 22.09.2025).

4. Теорема Котельникова, URL: https://siblec.ru/radiotekhnika-i-elektronika/radiotekhnicheskie-tsepi-i-signaly/3-diskretnye-i-tsifrovye-signaly/3-1-teorema-kotelnikova (дата обращения: 22.09.2025).

5. Основы цифровой обработки сигналов, URL: https://hub.exponenta.ru/post/osnovy-tsos-teorema-kotelnikova-atsp-i-tsap484 (дата обращения: 22.09.2025)

ЛИСТИНГ

```
import numpy as np
import matplotlib.pyplot as plt
import math
import sympy as sp
import pandas as pd
import os

_tmin = 13.0
_tmax = 39.0
_a = 4.0        # A, в
_a0 = 5.0       # A0, в
_f = 5.0        # частота, гц
_phi = math.pi / 4

# демонстрационные параметры (fd, b)
_fd_demo = 51
_b_demo = 8
_coding_demo = 'twos'  # 'twos'|'sign'|'ones'

# эталон для плотного графика
_pts_per_period_ref = 1000
_period = 1.0 / _f
_n_ref = int((_tmax - _tmin) * _f * _pts_per_period_ref)
_t_ref = np.linspace(_tmin, _tmax, max(_n_ref, 10))

# символьная функция и численная версия
_t_sym, _a_sym, _a0_sym, _f_sym, _phi_sym = sp.symbols('t_sym a_sym a0_sym f_sym
phi_sym')
_x_sym = _a0_sym + _a_sym * sp.cos(2 * sp.pi * _f_sym * _t_sym + _phi_sym)
_x_func = sp.lambdify((_t_sym, _a_sym, _a0_sym, _f_sym, _phi_sym), _x_sym, 'numpy')
_y_ref = _x_func(_t_ref, _a, _a0, _f, _phi)


def adc_simulation(tmin, tmax, a, a0, f, phi, fd, b, coding='twos', quant_range=None):

    L = 2 ** b
    if quant_range is None:
        y_min = a0 - a
        y_max = a0 + a
    else:
        y_min, y_max = quant_range

    dt = 1.0 / fd
```

```
t_samples = np.arange(tmin, tmax, dt)
if t_samples.size == 0:
    raise ValueError("параметры fd и интервал дали пустую сетку отсчетов")

y_samples = _x_func(t_samples, a, a0, f, phi)

# квантование mid-rise
delta = (y_max - y_min) / L
q_idx = np.floor((y_samples - y_min) / delta).astype(int)
q_idx = np.clip(q_idx, 0, L - 1)
y_q = y_min + (q_idx + 0.5) * delta

# centered signed index
offset = L // 2
s = q_idx - offset

def int_to_bits(val, width):
    return format(int(val) & ((1 << width) - 1), '0{}b'.format(width))

codes_int = np.zeros_like(q_idx, dtype=int)
codes_bits = []

if coding == 'twos':
    # two's complement
    codes_int = (s.astype(int)) & (L - 1)
    codes_bits = [int_to_bits(ci, b) for ci in codes_int]

elif coding == 'sign':
    # sign-magnitude
    max_mag = 2 ** (b - 1) - 1
    s_clip = np.clip(s, -max_mag, max_mag)
    codes_list = []
    for si in s_clip:
        if si < 0:
            sign_bit = 1
            mag = int(abs(si))
            code = (sign_bit << (b - 1)) | (mag & ((1 << (b - 1)) - 1))
        else:
            sign_bit = 0
            mag = int(si)
            code = (sign_bit << (b - 1)) | (mag & ((1 << (b - 1)) - 1))
        codes_list.append(code)
    codes_int = np.array(codes_list, dtype=int)
    codes_bits = [int_to_bits(ci, b) for ci in codes_int]

elif coding == 'ones':
```

```
    # ones' complement (обратный код)
    max_mag = 2 ** (b - 1) - 1
    s_clip = np.clip(s, -max_mag, max_mag)
    codes_list = []
    for si in s_clip:
        if si < 0:
            mag = int(abs(si))
            pos_code = mag & ((1 << (b - 1)) - 1)
            code = (~pos_code) & ((1 << b) - 1)
        else:
            code = int(si) & ((1 << b) - 1)
        codes_list.append(code)
    codes_int = np.array(codes_list, dtype=int)
    codes_bits = [int_to_bits(ci, b) for ci in codes_int]

else:
    raise ValueError("unknown coding type")

# декодирование: восстановим signed значение из codes_int (two's complement)
codes_mask = (1 << b) - 1
raw = codes_int.astype(int) & codes_mask
half = 1 << (b - 1)
signed_vals = np.where(raw >= half, raw - (1 << b), raw)
q_idx_rec = signed_vals + offset
q_idx_rec = np.clip(q_idx_rec, 0, L - 1)
y_rec = y_min + (q_idx_rec + 0.5) * delta

e = y_samples - y_q
ex = y_samples - y_rec
abs_e = np.abs(e)
abs_ex = np.abs(ex)

stats = {}
stats['delta'] = float(delta)
stats['L'] = int(L)
stats['mean_error'] = float(np.mean(e))
stats['var_error'] = float(np.var(e, ddof=0))
stats['mean_abs_error'] = float(np.mean(abs_e))
stats['theoretical_var'] = float(delta ** 2 / 12.0)
stats['theoretical_mean_abs'] = float(delta / 4.0)
signal_power = float(np.mean(y_samples ** 2))
noise_power = stats['var_error'] if stats['var_error'] > 0 else 1e-30
stats['snr_db_empirical'] = float(10.0 * np.log10(signal_power / noise_power))
stats['snr_db_theoretical'] = float(6.02 * b + 1.76)

return {
```

```
        't_samples': t_samples, 'y_samples': y_samples,
        'q_idx': q_idx, 'y_q': y_q,
        'codes_int': codes_int, 'codes_bits': codes_bits,
        'error': e, 'abs_error': abs_e, 'stats': stats,
        'y_min': y_min, 'y_max': y_max, 'b': b, 'fd': fd, 'coding': coding,
        'y_rec': y_rec, 'digital_error': ex, 'abs_digital_error': abs_ex
    }


def plot_analog(t_ref, y_ref, tmin, period):
    plt.figure(figsize=(10, 3))
    plt.plot(t_ref, y_ref, linewidth=0.9)
    plt.title('аналоговый сигнал (весь интервал)')
    plt.xlabel('t, s'); plt.ylabel('x(t), V'); plt.grid(True); plt.tight_layout(); plt.show()

    zoom_start = tmin
    zoom_end = tmin + 2 * period
    mask_ref = (t_ref >= zoom_start) & (t_ref <= zoom_end)
    plt.figure(figsize=(8, 3))
    plt.plot(t_ref[mask_ref], y_ref[mask_ref], linewidth=1)
    plt.title('аналоговый сигнал (детализация, 2 периода)')
    plt.xlabel('t, s'); plt.ylabel('x(t), V'); plt.grid(True); plt.tight_layout(); plt.show()


def plot_discrete_quantized(t_ref, y_ref, res, tmin, period):
    zoom_start = tmin
    zoom_end = tmin + 2 * period
    mask_ref = (t_ref >= zoom_start) & (t_ref <= zoom_end)
    t_s = res['t_samples']; y_s = res['y_samples']; y_q = res['y_q']
    mask_s = (t_s >= zoom_start) & (t_s <= zoom_end)

    plt.figure(figsize=(10, 3.5))
    plt.plot(t_ref[mask_ref], y_ref[mask_ref], label='эталон', linewidth=1)
    plt.stem(t_s[mask_s], y_s[mask_s], linefmt='k-', markerfmt='ko', basefmt=' ', label='дискретные
отсчёты')
    plt.step(t_s[mask_s], y_q[mask_s], where='post', label='квантованные уровни (mid-rise)')
    plt.title(f'дискретный и квантованный (fd={res["fd"]} Hz, b={res["b"]},
code={res["coding"]})')
    plt.xlabel('t, s'); plt.ylabel('x, V'); plt.legend(); plt.grid(True); plt.tight_layout(); plt.show()


def plot_codes(codes_int):
    plt.figure(figsize=(10, 3))
    plt.step(np.arange(len(codes_int)), codes_int, where='post')
    plt.title('integer code (q_idx centered) vs sample index (временная ось опущена)')
    plt.xlabel('sample index'); plt.ylabel('integer code'); plt.grid(True); plt.tight_layout(); plt.show()
```

```
def plot_bits_preview(codes_bits, b):
    N_preview = min(80, len(codes_bits))
    bits_list = codes_bits[:N_preview]
    bits_matrix = np.array([[int(ch) for ch in code] for code in bits_list])  # shape (N_preview, b)
    bits_T = bits_matrix.T
    plt.figure(figsize=(10, 2 + bits_T.shape[0] * 0.15))
    for i in range(bits_T.shape[0]):
        ax = plt.subplot(bits_T.shape[0], 1, i + 1)
        ax.step(np.arange(bits_T.shape[1]), bits_T[i], where='post')
        ax.set_ylim(-0.2, 1.2)
        ax.set_yticks([0, 1])
        ax.set_ylabel(f'b{bits_T.shape[0] - 1 - i}')
        if i < bits_T.shape[0] - 1:
            ax.set_xticks([])
        else:
            ax.set_xlabel('sample index (preview)')
    plt.suptitle('по-битная визуализация первых кодов (MSB сверху)')
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()


def plot_digital_error(td, ex, delta=None, zoom='tight'):

    plt.figure(figsize=(10, 3.5))
    plt.plot(td, ex, marker='o', linestyle='-', linewidth=1)
    plt.title('абсолютная погрешность цифрового сигнала')
    plt.xlabel('дискретное время, nT_д, с')
    plt.ylabel('погрешность, e(n), В')

    if zoom == 'tight':
        # используем 99-й перцентиль, чтобы убрать редкие выбросы и увидеть мелкие детали
        upper = np.percentile(np.abs(ex), 99)
        if upper <= 0:
            upper = np.max(np.abs(ex))
        plt.ylim(-upper * 1.1, upper * 1.1)
    elif zoom == 'theoretical' and (delta is not None):
        plt.ylim(-delta / 2 * 1.05, delta / 2 * 1.05)
    elif isinstance(zoom, tuple) and len(zoom) == 2:
        plt.ylim(zoom)
    # else: default autoscale

    plt.grid(True); plt.tight_layout(); plt.show()
```

```
def plot_abs_error_histogram(abs_ex, delta=None, nBars=8, zoom_y=True):
    # гистограмма abs error с опцией уменьшенного вертикального масштаба
    plt.figure(figsize=(8, 4))
    counts, bins, patches = plt.hist(abs_ex, bins=30, density=True, edgecolor='black', linewidth=1.0)
    xs2 = np.linspace(0, delta/2 if delta is not None else np.max(abs_ex), 200)
    if delta is not None:
        theor_pdf_abs = np.ones_like(xs2) * (2.0 / delta)
        plt.plot(xs2, theor_pdf_abs, linestyle='--', linewidth=1.5)

    plt.title('гистограмма |e| и теоретическая плотность')
    plt.xlabel('|e|, В'); plt.ylabel('плотность')

    if zoom_y:
        # сузим вертикальную ось до 1.1 * медианы плотности (или до max(counts)*1.1 если
нужно)
        ymax = max(np.max(counts) * 1.05, 0.1)
        plt.ylim(0, ymax)

    plt.grid(True); plt.tight_layout(); plt.show()


def plot_error_distribution(ex, delta):
    # histogram of digital error ex with nBars=8 (like matlab example)
    nBars = 8
    edges = np.linspace(-delta/2, delta/2, nBars + 1)
    plt.figure(figsize=(8, 4))
    plt.hist(ex, bins=edges, density=True, edgecolor='black', linewidth=1.0)
    plt.title('гистограмма статистического распределения абсолютной погрешности (digital
error)')
    plt.xlabel('абсолютная погрешность, e(n), В')
    plt.ylabel('плотность вероятности')
    plt.xlim([-1.5 * delta / 2, 1.5 * delta / 2])
    plt.grid(True)
    xs = np.linspace(-delta/2, delta/2, 200)
    theor_pdf = np.ones_like(xs) * (1.0 / delta)
    plt.plot(xs, theor_pdf, linestyle='--', linewidth=1.5)
    plt.legend(['теоретическая', 'экспериментальная'], loc='upper right')
    plt.tight_layout(); plt.show()


def main(zoom_mode='tight'):
    # zoom_mode: 'tight'|'theoretical'|None|tuple
    res = adc_simulation(_tmin, _tmax, _a, _a0, _f, _phi, _fd_demo, _b_demo,
coding=_coding_demo)

    # 1) аналоговый
```

```
    plot_analog(_t_ref, _y_ref, _tmin, _period)

    # 2) дискретный и квантованный (zoom)
    plot_discrete_quantized(_t_ref, _y_ref, res, _tmin, _period)

    # 3) integer codes
    plot_codes(res['codes_int'])

    # 4) по-битная визуализация
    plot_bits_preview(res['codes_bits'], res['b'])

    # 5) абсолютная погрешность цифрового сигнала
    ex = res['digital_error']
    td = res['t_samples']
    plot_digital_error(td, ex, delta=res['stats']['delta'], zoom=zoom_mode)

    # 6) гистограммы
    plot_error_distribution(ex, res['stats']['delta'])
    plot_abs_error_histogram(res['abs_digital_error'], delta=res['stats']['delta'], zoom_y=True)

    # 7) статистика
    stats = res['stats']
    df_stats = pd.DataFrame([{
        'fd (Hz)': res['fd'], 'b (bits)': res['b'],
        'delta (V)': stats['delta'], 'L': stats['L'],
        'mean_error (V)': stats['mean_error'], 'var_error (V^2)': stats['var_error'],
        'theoretical_var (V^2)': stats['theoretical_var'],
        'mean_abs_error (V)': stats['mean_abs_error'], 'theoretical_mean_abs (V)':
stats['theoretical_mean_abs'],
        'snr_db_empirical': stats['snr_db_empirical'], 'snr_db_theoretical': stats['snr_db_theoretical']
    }])
    try:
        from caas_jupyter_tools import display_dataframe_to_user
        display_dataframe_to_user("статистика_квантования", df_stats)
    except Exception:
        print('\nстатистика квантования (резюме):')
        print(df_stats.to_string(index=False))

    try:
        out_csv = os.path.join(os.getcwd(), 'adc_quant_stats.csv')
        df_stats.to_csv(out_csv, index=False)
        print(f"\nстатистика сохранена в: {out_csv}")
    except Exception as e:
        print('не удалось сохранить csv:', e)
```

```
if __name__ == '__main__':
    main(zoom_mode='tight')
```