

lecture01

September 21, 2026

1 Лекция 1. Введение в обработку сигналов

Курс: Обработка сигналов · 4 курс · 01.03.02 «Прикладная математика и информатика»

Направленность: Искусственный интеллект и системная аналитика

Эта лекция — вход в курс. Мы почти не будем выводить формулы: наша задача — **увидеть**, что такое сигнал, почему его удобно рассматривать во времени и по частоте, и как из сырых данных датчиков получается решение, которое помогает человеку. Всё будем показывать картинками на Python.

1.1 Цели лекции

К концу лекции вы сможете:

1. Объяснить, чем отличаются **аналоговый, дискретный и цифровой** сигналы.
2. Смотреть на сигнал **во времени и в частотной области** и понимать, что даёт спектр.
3. Объяснить простыми словами **теорему Котельникова** и явление **алиасинга**.
4. Понимать, что такое **квантование, шум и отношение сигнал/шум (ОСШ)**.
5. Описать **сквозной проект** курса и роль обработки сигналов в системе поддержки принятия решений (КСППР).

1.2 1. Что такое сигнал

Сигнал — это физическая величина, которая меняется во времени (или в пространстве) и несёт информацию о состоянии объекта.

Сигналы вокруг нас:

- температура и вибрация насоса на заводе;
- звук речи оператора;
- ток и напряжение в сети;
- радиоволна, отражённая от объекта (RFID, радар, LiDAR);
- электрическая активность сердца (ЭКГ).

Нас интересует не сама кривая, а **что за ней стоит**: дефект, событие, команда, угроза. Обработка сигналов — это мост от сырых измерений к смыслу.

1.3 2. Классификация сигналов

- **Аналоговый** — значение определено в любой момент времени (непрерывный).
- **Дискретный** — значения берутся в отдельные моменты времени (квантование по времени), но уровень — любой.
- **Цифровой** — дискретный по времени, квантованный по уровню **и закодированный**: каждый отсчёт представляется кодом (двоичным числом).

Итого три операции: **дискретизация** → **квантование** → **кодирование**. Цифровой сигнал — это уже не *curve*, а поток чисел (кодов), с которым работает вычислитель:

датчик → *дискретизация* → *квантование* → *кодирование* (АЦП) → *обработка*.

```
[1]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

FIG = Path("figures")
FIG.mkdir(exist_ok=True)

plt.rcParams.update({
    "figure.figsize": (11, 3.6),
    "figure.dpi": 110,
    "font.size": 11,
    "axes.grid": True,
    "grid.alpha": 0.3,
})

def save(fig, name):
    fig.savefig(FIG / name, bbox_inches="tight")
    print("сохранено:", name)

print("numpy", np.__version__)
```

numpy 2.5.3

```
[2]: t = np.linspace(0, 1, 2000)
f0 = 4
analog = np.sin(2 * np.pi * f0 * t)

n = np.arange(16)
discrete = np.sin(2 * np.pi * f0 * n / 16)

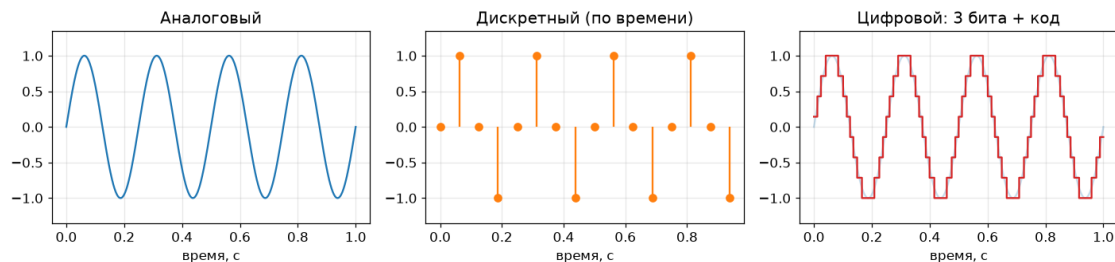
bits = 3
q = np.round((analog + 1) / 2 * (2**bits - 1)) / (2**bits - 1) * 2 - 1
```

```

fig, ax = plt.subplots(1, 3, figsize=(13, 3.2))
ax[0].plot(t, analog, color="C0")
ax[0].set_title("Аналоговый")
ax[1].stem(n / 16, discrete, basefmt=" ", linefmt="C1", markerfmt="C1o")
ax[1].set_title("Дискретный (по времени)")
ax[2].plot(t, analog, alpha=0.25, color="C0")
ax[2].step(t, q, where="mid", color="C3")
ax[2].set_title("Цифровой: 3 бита + код")
for a in ax:
    a.set_xlabel("время, с")
    a.set_ylim(-1.35, 1.35)
fig.tight_layout()
save(fig, "fig01_signal_types.png")
plt.show()

```

сохранено: fig01_signal_types.png



1.4 2а. Как вернуть аналог: ЦАП и ШИМ (затравка на схемотехнику)

Цифровой код нужно снова превратить в аналог — этим занимаются **ЦАП** и **выходной каскад усилителя мощности**. Есть два принципиально разных подхода:

- **Линейный режим — усилитель класса А («ЦАП»).** Транзистор всё время приоткрыт и работает как **управляемый резистор**: выход плавно повторяет вход, но на транзисторе рассеивается мощность → греется, КПД ~25%.
- **Ключевой режим — усилитель класса D (ШИМ).** Полевые транзисторы работают как **ключи**: либо закрыты, либо открыты. Сигнал задаётся **широтно-импульсной модуляцией (ШИМ)**, а на выходе **LC-фильтр** восстанавливает аналог. Потери малы, КПД >90%.

Фундаментальный инженерный вопрос: что удобнее? Ключ, который либо полностью закрыт, либо полностью открыт, почти не рассеивает мощность. Именно поэтому вся современная силовая электроника строит выходные каскады на **ключевом режиме**.

```

[3]: from scipy import signal as sps

fig = plt.figure(figsize=(13, 6.6))
gs = fig.add_gridspec(2, 2, height_ratios=[1, 1.15], hspace=0.4, wspace=0.12)

```

```

def chain(ax, blocks, title, note):
    ax.axis("off")
    wid, gap = 1.5, 0.42
    x = 0.0
    for i, b in enumerate(blocks):
        ax.add_patch(plt.Rectangle((x, 0.35), wid, 0.7, fc="#dbeafe",
        ec="#334155"))
        ax.text(x + wid / 2, 0.7, b, ha="center", va="center", fontsize=8)
        if i < len(blocks) - 1:
            ax.annotate("", xy=(x + wid + gap, 0.7), xytext=(x + wid, 0.7),
            arrowprops=dict(arrowstyle="->", color="#334155"))
            x += wid + gap
    ax.text((x - gap) / 2, 0.02, note, ha="center", va="top", fontsize=8.2,
    color="#475569")
    ax.set_title(title, fontsize=11)
    ax.set_xlim(-0.2, x + 0.2)
    ax.set_ylim(-0.5, 1.25)

chain(fig.add_subplot(gs[0, 0]),
    ["Вход", "Транзистор\nлинейно", "Выход"],
    "Класс А (ЦАП): линейный режим",
    "транзистор приоткрыт как резистор → потери, КПД ~25%, греется")
chain(fig.add_subplot(gs[0, 1]),
    ["ШИМ", "Полумост:\n2 ключа", "LC-фильтр", "Выход"],
    "Класс D (ШИМ): ключевой режим",
    "ключ либо закрыт, либо открыт → малые потери, КПД >90%")

a1 = fig.add_subplot(gs[1, 0])
tt = np.linspace(0, 1, 400)
ss = np.sin(2 * np.pi * 2 * tt)
bits = 4
qq = np.round((ss + 1) / 2 * (2**bits - 1)) / (2**bits - 1) * 2 - 1
a1.plot(tt, ss, alpha=0.3, color="C0", label="аналог")
a1.step(tt, qq, where="mid", color="C3", label="после ЦАП")
a1.set_title("ЦАП: ступенчатый выход", fontsize=11)
a1.set_xlabel("время")
a1.legend(fontsize=8, loc="lower right")

a2 = fig.add_subplot(gs[1, 1])
fs2 = 4000
tp = np.arange(0, 1, 1 / fs2)
ref = np.sin(2 * np.pi * 2 * tp)
carrier = sps.sawtooth(2 * np.pi * 25 * tp, 0.5)
pwm = (ref > carrier).astype(float)

```

```

b_lp, a_lp = sps.butter(4, 0.006)
filt = sps.filtfilt(b_lp, a_lp, pwm)
a2.plot(tp, ref, alpha=0.35, color="C0", label="аналог")
a2.plot(tp, pwm * 0.7 - 0.9, color="C3", lw=0.5, label="ШИМ (смещено вниз)")
a2.plot(tp, 2 * filt - 1, color="C2", label="после LC-фильтра")
a2.set_title("ШИМ + LC-фильтр восстанавливает аналог", fontsize=11)
a2.set_xlabel("время")
a2.legend(fontsize=8, loc="lower right")

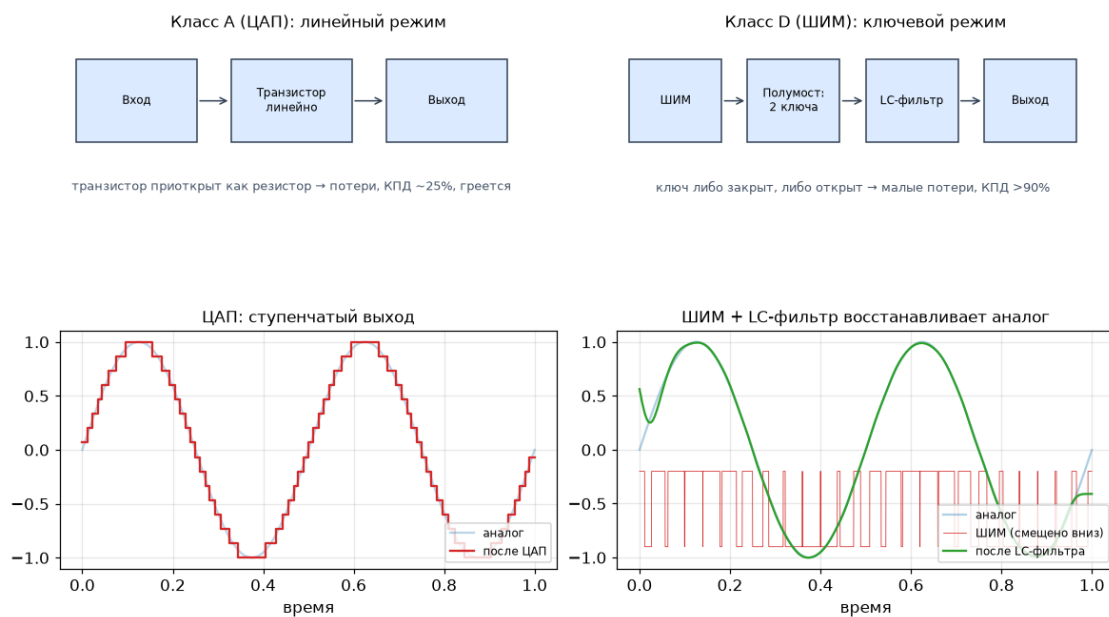
fig.tight_layout()
save(fig, "fig11_pwm_dac.png")
plt.show()

```

/tmp/ipykernel_2803807/3984476788.py:60: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.

fig.tight_layout()

сохранено: fig11_pwm_dac.png



Вывод: цифровой сигнал — это дискретизация, квантование и **кодирование**; обратный путь в аналог делают либо линейным ЦАП (класс А), либо ключевым ШИМ (класс D). Выбор режима транзистора — уже вопрос инженерной выгоды.

1.5 2b. Сигналы вокруг нас: шесть «суперсил» обработки

Мы разобрали базовые понятия. Теперь — короткая галерея из шести настоящих сигналов: пары бытовых, которыми человечество пользуется каждый день, и пары «извлекающих слабое из шума». Каждый пример показывает **одну суперсилу** обработки сигналов.

Смотрим сам сигнал во времени, а модулированные составляющие — отдельно.

1.5.1 УКВ, ТВ и передача данных

Пилот-тон стерео УКВ. В одном радиоканале умещаются сразу два звуковых канала. Сумму L+R передают внизу спектра, а разность L-R — на поднесущей 38 кГц (двойная боковая, без несущей). Чтобы приёмник знал фазу, рядом ставят **пилот-тон 19 кГц**. Спектр этой смеси (MPX) — наглядный пример «упаковки» нескольких сигналов в один.

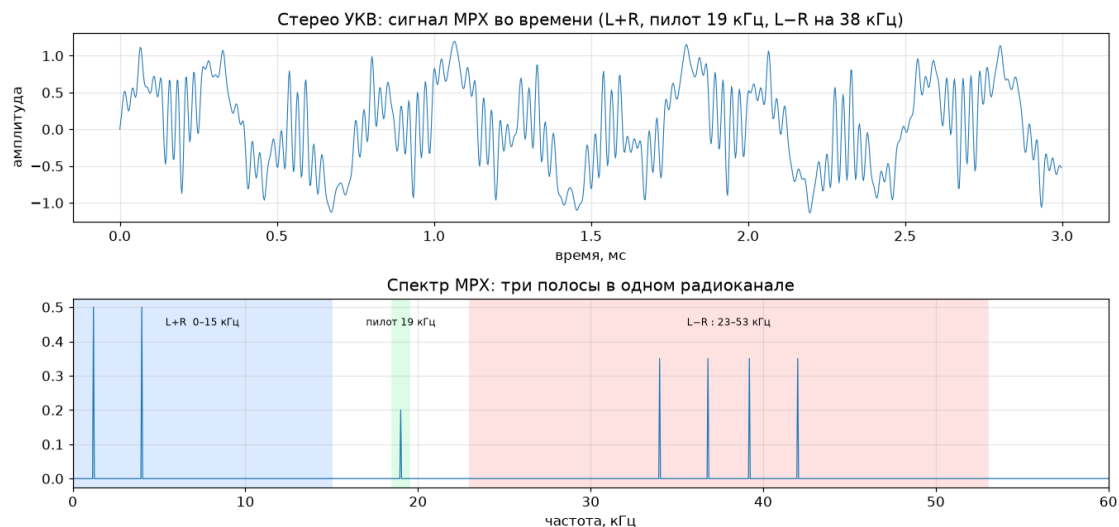
```
[4]: fs = 400e3
t = np.arange(0, 3e-3, 1 / fs)
L = 0.5 * np.sin(2 * np.pi * 1200 * t)
R = 0.5 * np.sin(2 * np.pi * 4000 * t)
mono = L + R
diff = L - R
pilot = 0.2 * np.sin(2 * np.pi * 19e3 * t)
dsb = 0.7 * diff * np.cos(2 * np.pi * 38e3 * t)
mpx = mono + pilot + dsb

tt = np.arange(0, 20e-3, 1 / fs)
mp = (0.5 * np.sin(2 * np.pi * 1200 * tt) + 0.5 * np.sin(2 * np.pi * 4000 * tt)
      + 0.2 * np.sin(2 * np.pi * 19e3 * tt)
      + 0.7 * (np.sin(2 * np.pi * 1200 * tt) - np.sin(2 * np.pi * 4000 * tt))
      * np.cos(2 * np.pi * 38e3 * tt))

fig, ax = plt.subplots(2, 1, figsize=(12.5, 6))
ax[0].plot(t * 1e3, mpx, lw=0.7)
ax[0].set_title("Стерео УКВ: сигнал MPX во времени (L+R, пилот 19 кГц, L-R на 38 кГц)")
ax[0].set_xlabel("время, мс")
ax[0].set_ylabel("амплитуда")

S = np.abs(np.fft.rfft(mp)) / len(mp) * 2
f = np.fft.rfftfreq(len(mp), 1 / fs)
ax[1].plot(f / 1e3, S, lw=0.8)
for x0, x1, lab, col in [(0, 15, "L+R 0–15 кГц", "#d8eafe"),
                        (18.5, 19.5, "пилот 19 кГц", "#d8fce7"),
                        (23, 53, "L-R : 23–53 кГц", "#fee2e2")]:
    ax[1].axvspan(x0, x1, color=col)
    ax[1].text((x0 + x1) / 2, ax[1].get_ylim()[1] * 0.85, lab, ha="center",
               fontsize=8)
ax[1].set_xlim(0, 60)
ax[1].set_title("Спектр MPX: три полосы в одном радиоканале")
ax[1].set_xlabel("частота, кГц")
fig.tight_layout()
save(fig, "fig14_fm_stereo.png")
plt.show()
```

сохранено: fig14_fm_stereo.png



Композитный видеосигнал PAL/SECAM. В аналоговом ТВ **уровень сигнала — это яркость:** белый — выше, чёрный — ниже, а ещё ниже лежат **синхроимпульсы** (вершина 0 В при белом 1 В). Цвет передают отдельно — **модулированными цветоразностными составляющими** на поднесущей 4.43 МГц (QAM из U и V), плюс короткая **цветовая вспышка** (burst) для фазировки. На отдельном графике покажем U, V и уже промодулированную поднесущую.

```
[5]: fs = 20e6
fsc = 4.43361875e6
t = np.arange(0, 64e-6, 1 / fs)
SYNC, BLANK, WHITE = 0.0, 0.3, 1.0

comp = np.full_like(t, BLANK)
comp[t < 4.7e-6] = SYNC
burst = (t >= 5.3e-6) & (t < 7.55e-6)
comp[burst] = BLANK + 0.15 * np.sin(2 * np.pi * fsc * t[burst])

active = (t >= 10.4e-6) & (t < 62.4e-6)
ta = t[active]
ramp = (ta - ta[0]) / (ta[-1] - ta[0])
Y = BLANK + (WHITE - BLANK) * ramp
U = 0.2 * np.sin(2 * np.pi * 0.5e6 * ta)
V = 0.2 * np.cos(2 * np.pi * 0.3e6 * ta)
comp[active] = Y + U * np.cos(2 * np.pi * fsc * ta) + V * np.sin(2 * np.pi * fsc
    ↳ * ta)

fig = plt.figure(figsize=(13, 9))
gs = fig.add_gridspec(3, 1, height_ratios=[1.15, 1, 1], hspace=0.6)
```

```

a = fig.add_subplot(gs[0])
a.plot(t * 1e6, comp, lw=0.8, color="C0")
for lvl in (WHITE, BLANK, SYNC):
    a.axhline(lvl, color="0.7", ls=":", lw=0.8)
a.text(65, WHITE, "белый 1.0 В", va="center", fontsize=8)
a.text(65, BLANK, "чёрный/гашение 0.3 В", va="center", fontsize=8)
a.text(65, SYNC, "синхровершина 0.0 В", va="center", fontsize=8)
a.annotate("синхроимпульс 4.7 мкс", xy=(2.3, 0.06), xytext=(4, 0.8),
           arrowprops=dict(arrowstyle="->", lw=0.8), fontsize=8)
a.annotate("цветовая вспышка", xy=(6.4, 0.3), xytext=(11, 1.02),
           arrowprops=dict(arrowstyle="->", lw=0.8), fontsize=8)
a.annotate("активная часть 52 мкс (яркость: чёрный → белый)", xy=(40, 0.7),
           xytext=(24, 0.1), arrowprops=dict(arrowstyle="->", lw=0.8),
           ↪ fontsize=8)
a.set_xlim(0, 80)
a.set_ylim(-0.1, 1.3)
a.set_xlabel("время, мкс")
a.set_ylabel("напряжение, В")
a.set_title("PAL/SECAM: одна строка композитного видеосигнала", fontsize=11)

env = np.sqrt(U**2 + V**2)
b = fig.add_subplot(gs[1])
b.plot(ta * 1e6, U, color="C1", lw=1.4, label="U (цветоразность B-Y)")
b.plot(ta * 1e6, V, color="C2", lw=1.4, label="V (цветоразность R-Y)")
b.plot(ta * 1e6, env, "k--", lw=0.8, label="оггибающая")
b.plot(ta * 1e6, -env, "k--", lw=0.8)
b.set_xlabel("время, мкс")
b.set_ylabel("цветоразность")
b.set_title("Цветоразностные сигналы U и V — до модуляции", fontsize=11)
b.legend(fontsize=8, ncol=3, loc="upper right")

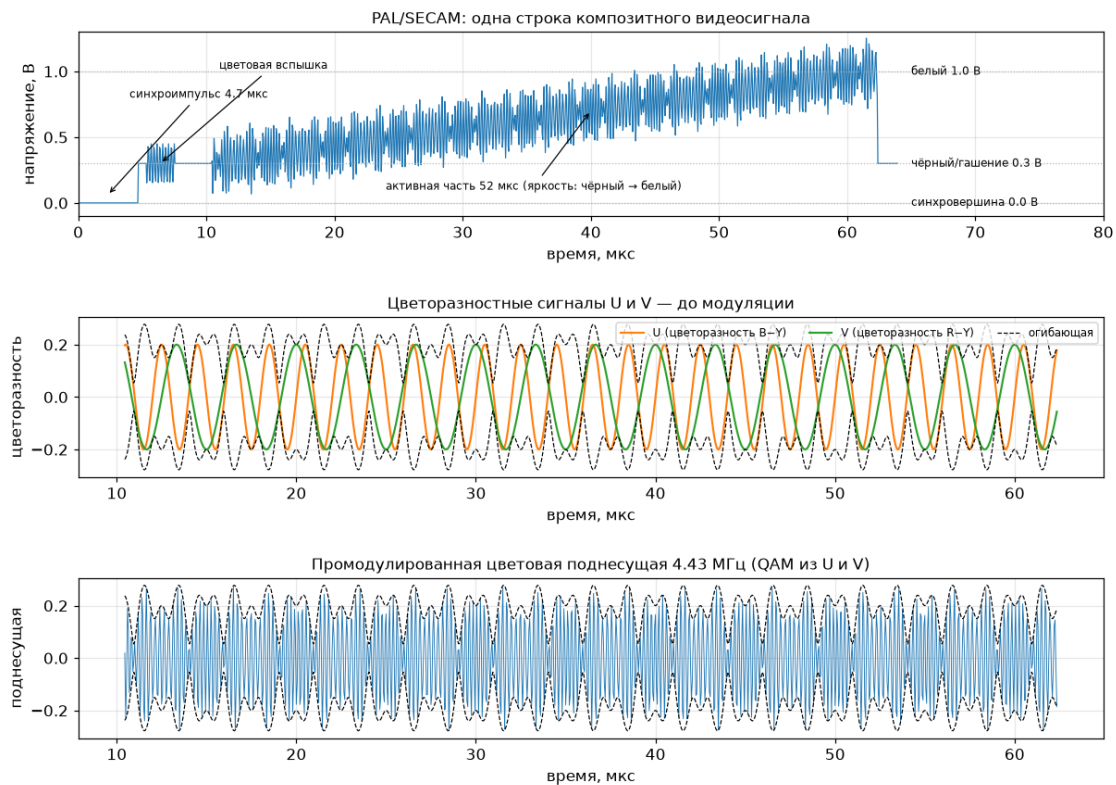
chroma = U * np.cos(2 * np.pi * fsc * ta) + V * np.sin(2 * np.pi * fsc * ta)
c = fig.add_subplot(gs[2])
c.plot(ta * 1e6, chroma, lw=0.6, color="C0")
c.plot(ta * 1e6, env, "k--", lw=0.8)
c.plot(ta * 1e6, -env, "k--", lw=0.8)
c.set_xlabel("время, мкс")
c.set_ylabel("поднесущая")
c.set_title("Промодулированная цветовая поднесущая 4.43 МГц (QAM из U и V)",
           ↪ fontsize=11)
fig.tight_layout()
save(fig, "fig12_pal_video.png")
plt.show()

```

/tmp/ipykernel_2803807/156912822.py:60: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.

```
fig.tight_layout()
```

сохранено: fig12_pal_video.png



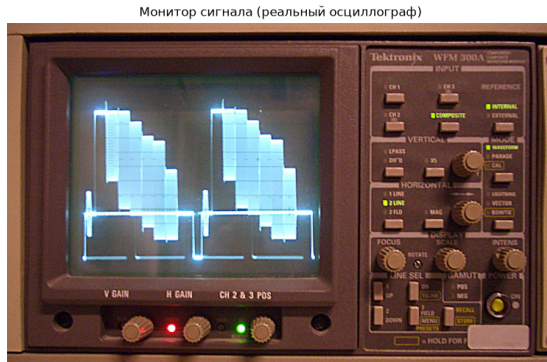
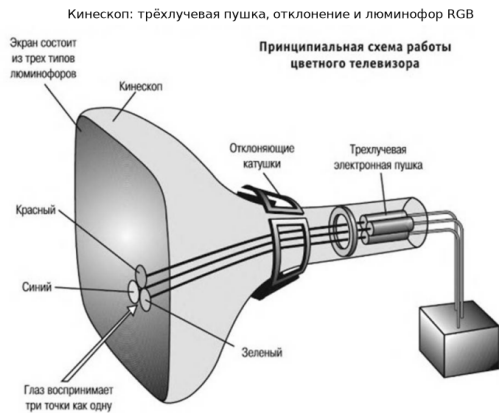
Так это выглядит на осциллографе. Слева — **кинескоп** цветного телевизора: трёхлучевая электронная пушка, отклоняющие катушки и экран из трёх люминофоров (**R, G, B**), три точки которых глаз сливает в одну. Справа — реальный **монитор сигнала** (waveform monitor) с цветными полосами: видны все уровни, от синхровершины до белого, и цветовая вспышка.

Фото справа: *Wikimedia Commons*, автор **grm_wnr**, лицензия **CC BY-SA 3.0** ([файл](#)). Схема кинескопа слева: сайт «Ремонт ТВ своими руками» ([remont-televizorov-na-domu.ru](#)).

```
[6]: from PIL import Image as PILImage
import matplotlib.pyplot as plt
fig, (ax_crt, ax_osc) = plt.subplots(1, 2, figsize=(14, 5.6))
crt = PILImage.open("figures/crt_color_tv.jpg")
ax_crt.imshow(crt)
ax_crt.set_title("Кинескоп: трёхлучевая пушка, отклонение и люминофор RGB",
    ↪fontsize=11)
ax_crt.axis("off")
osc = PILImage.open("figures/waveform_monitor.jpg")
ax_osc.imshow(osc)
ax_osc.set_title("Монитор сигнала (реальный осциллограф)", fontsize=11)
ax_osc.axis("off")
fig.tight_layout()
```

```
save(fig, "fig18_crt_monitor.png")
plt.show()
```

сохранено: fig18_crt_monitor.png



Скан-линии и растр. Телевизор рисует картинку **строкой за строкой**: луч бежит слева направо, затем возвращается и идёт ниже. Один проход — **поле**. Чтобы не мерцало, строки делят на два **чересстрочных** поля (нечётные и чётные), которые дополняют друг друга до полного кадра. Именно поэтому в сигнале и есть синхроимпульсы: каждый отмечает конец строки, чтобы приёмник начал следующую. Разверни сигнал во времени к растр — и получишь картинку.

```
[7]: H, W = 240, 360
yy, xx = np.mgrid[0:H, 0:W]
img = 0.5 + 0.35 * np.sin(2 * np.pi * xx / 90)
cx, cy, r = W // 2, H // 2, 85
mask = (xx - cx) ** 2 + (yy - cy) ** 2 < r ** 2
img = np.where(mask, 0.1, img)
img = np.clip(0.25 + 0.6 * img + 0.15 * (xx / W), 0, 1)

fs = 20e6
tline = np.arange(0, 64e-6, 1 / fs)

def video_line(row):
    s = np.full_like(tline, 0.3)
    active = (tline >= 10.4e-6) & (tline < 62.4e-6)
    ta = tline[active]
    lum = np.interp(np.linspace(0, 1, ta.size), np.linspace(0, 1, row.size), row)
    s[tline < 4.7e-6] = 0.0
    s[active] = 0.3 + 0.7 * lum
    return s
```

```

fig = plt.figure(figsize=(13, 8))
gs = fig.add_gridspec(3, 1, height_ratios=[1.3, 1, 1], hspace=0.55)

a = fig.add_subplot(gs[0])
a.imshow(img, cmap="gray", vmin=0, vmax=1, aspect="auto")
for y0 in np.linspace(2, H - 3, 22):
    a.axhline(y0, color="#22c55e", lw=0.9)
a.set_title("Растр: луч идёт строкой за строкой (зелёные линии – ход развёртки)")
a.set_xlabel("строка, x")
a.set_yticks([])

sig = np.concatenate([video_line(img[k]) for k in (30, 60, 90)])
tb = np.arange(sig.size) / fs * 1e6
b = fig.add_subplot(gs[1])
b.plot(tb, sig, lw=0.7, color="C0")
for k in range(4):
    b.axvline(k * 64, color="0.8", lw=0.6)
b.set_xlim(0, 3 * 64)
b.set_xlabel("время, мкс")
b.set_ylabel("напряжение, В")
b.set_title("Три строки в сигнале: синхроимпульс + активная часть = «лестница»  
↪растра")

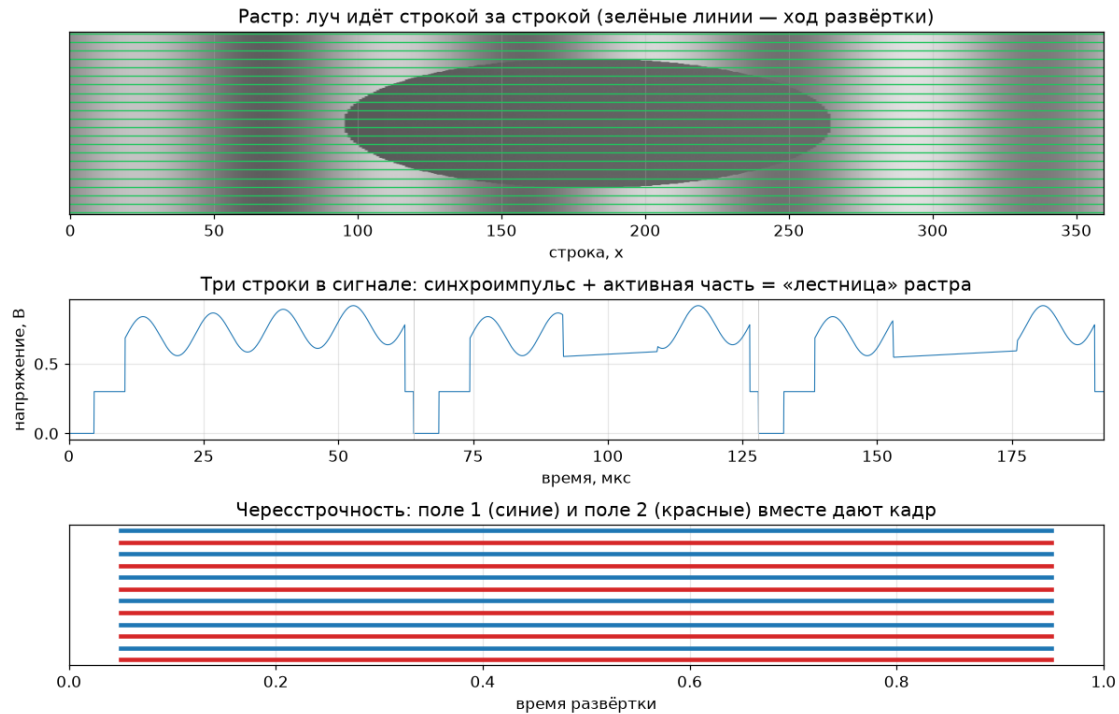
c = fig.add_subplot(gs[2])
c.set_xlim(0, 1)
c.set_ylim(-0.5, 11.5)
for k in range(12):
    col = "C0" if k % 2 == 0 else "C3"
    c.plot([0.05, 0.95], [k, k], color=col, lw=3)
c.invert_yaxis()
c.set_yticks([])
c.set_xlabel("время развёртки")
c.set_title("Чересстрочность: поле 1 (синие) и поле 2 (красные) вместе дают  
↪кадр")
fig.tight_layout()
save(fig, "fig17_scanlines.png")
plt.show()

```

/tmp/ipykernel_2803807/3040878500.py:55: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.

```
fig.tight_layout()
```

сохранено: fig17_scanlines.png



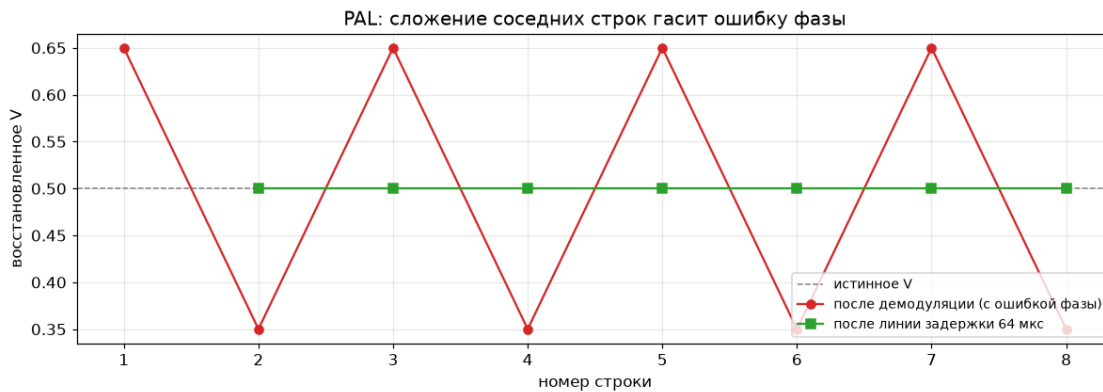
Линия задержки в PAL — суть стандарта. В PAL составляющую V на каждой второй строке передают с обратным знаком. Если фаза поднесущей «уехала» на малую ошибку, она даёт одинаковую добавку на обеих строках. Приёмник по **линии задержки 64 мкс** складывает текущую строку с предыдущей — и ошибка фазы **вычитается**, а цвет остаётся верным. Это уже настоящий фильтр с памятью.

```
[8]: lines = np.arange(1, 9)
V_true, Usig, delta = 0.5, 0.5, 0.3
V_recv = np.where(lines % 2 == 1, V_true + Usig * delta, -V_true + Usig * delta)
V_out = V_recv.copy()
V_out[lines % 2 == 0] *= -1
V_avg = 0.5 * (V_out + np.roll(V_out, 1))
V_avg[0] = np.nan

fig, ax = plt.subplots(figsize=(11, 4))
ax.axhline(V_true, color="0.5", ls="--", lw=1, label="истинное V")
ax.plot(lines, V_out, "o-", color="C3", label="после демодуляции (с ошибкой фазы)")
ax.plot(lines, V_avg, "s-", color="C2", label="после линии задержки 64 мкс")
ax.set_xticks(lines)
ax.set_xlabel("номер строки")
ax.set_ylabel("восстановленное V")
ax.set_title("PAL: сложение соседних строк гасит ошибку фазы")
ax.legend(fontsize=9, loc="lower right")
```

```
fig.tight_layout()
save(fig, "fig13_pal_delay.png")
plt.show()
```

сохранено: fig13_pal_delay.png



Цифровая связь: созвездие QPSK. В цифровых системах сигнал удобно задавать не кривой, а **точкой на плоскости I/Q** (синфазная и квадратурная составляющие). Каждое сочетание бит — своя точка; физически это фаза несущей, которая скачком меняется на границе символа. Так передают Wi-Fi, GPS, DVB.

OFDM — ортогональные поднесущие. Wi-Fi и LTE передают данные сразу на многих **ортогональных** поднесущих: на частоте каждой из них все остальные обращаются в нуль, поэтому они не мешают друг другу. Наш **ESP32-C3** гонит данные по Wi-Fi — а это и есть OFDM.

```
[9]: fs = 400e3
fc = 25e3
pairs = [(0, 0), (1, 0), (1, 1), (0, 1), (1, 0), (0, 0)]
sp = 0.25e-3
t = np.arange(0, len(pairs) * sp, 1 / fs)
sig = np.zeros_like(t)
for k, (b1, b0) in enumerate(pairs):
    I = 1.0 if b1 else -1.0
    Q = 1.0 if b0 else -1.0
    m = (t >= k * sp) & (t < (k + 1) * sp)
    sig[m] = np.cos(2 * np.pi * fc * t[m] + np.arctan2(Q, I))

fig = plt.figure(figsize=(13, 7.5))
gs = fig.add_gridspec(2, 2, hspace=0.4, wspace=0.25)

a = fig.add_subplot(gs[0, :])
a.plot(t * 1e3, sig, lw=0.7)
for k in range(len(pairs) + 1):
    a.axvline(k * sp * 1e3, color="C3", lw=0.8, ls="--")
```

```

a.set_title("QPSK: фаза несущей скачет на границе символов (пунктир)")
a.set_xlabel("время, мс")
a.set_ylabel("амплитуда")

b = fig.add_subplot(gs[1, 0])
for b1, b0 in pairs:
    I = 1.0 if b1 else -1.0
    Q = 1.0 if b0 else -1.0
    b.plot(I, Q, "o", ms=10, color="C0")
    b.text(I + 0.12, Q, f"{b1}{b0}", fontsize=9)
b.axhline(0, color="0.8")
b.axvline(0, color="0.8")
b.set_xlim(-1.6, 1.6)
b.set_ylim(-1.6, 1.6)
b.set_xlabel("I")
b.set_ylabel("Q")
b.set_title("Созвездие QPSK: каждые 2 бита — точка", fontsize=10)

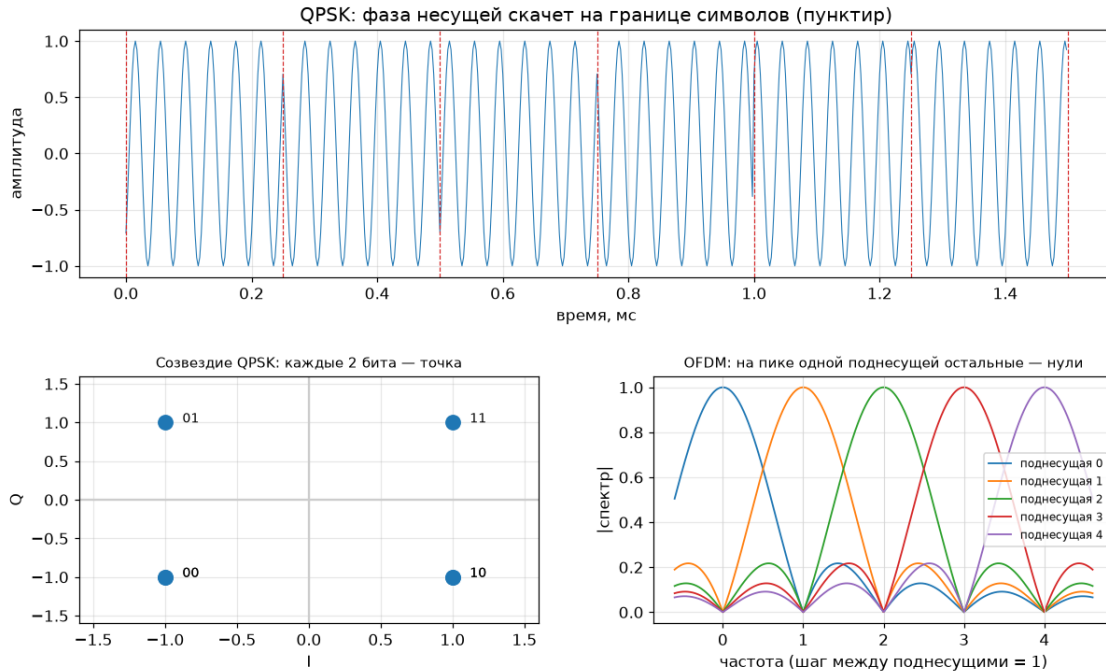
c = fig.add_subplot(gs[1, 1])
fq = np.linspace(-0.6, 4.6, 2000)
for k in range(5):
    c.plot(fq, np.abs(np.sinc(fq - k)), lw=1.2, label=f"поднесущая {k}")
    c.axvline(k, color="0.85", lw=0.6)
c.set_xlabel("частота (шаг между поднесущими = 1)")
c.set_ylabel("|спектр|")
c.set_title("OFDM: на пике одной поднесущей остальные — нули", fontsize=10)
c.legend(fontsize=8)
fig.tight_layout()
save(fig, "fig15_qpsk_ofdm.png")
plt.show()

```

/tmp/ipykernel_2803807/2536532730.py:47: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.

```
fig.tight_layout()
```

сохранено: fig15_qpsk_ofdm.png



Достать слабый сигнал из шума. Радар, сонар, LiDAR и RFID издавна решают одну задачу: обнаружить отражение, которое **слабее шума**. Помогает **сложный сигнал + согласованный фильтр**. У радиолокатора — длинный «чирп» (частота растёт линейно), а на приёме его сжимают в один узкий пик. Так же устроен приём GPS: полезный код тонет в шуме, но корреляция с его копией даёт **острый пик**, по задержке которого находят расстояние.

```
[10]: fs = 1e6
T = 5e-3
t = np.arange(0, T, 1 / fs)
f0, f1 = 10e3, 90e3
chirp = np.cos(2 * np.pi * (f0 * t + (f1 - f0) / (2 * T) * t**2))

fig = plt.figure(figsize=(13, 8))
gs = fig.add_gridspec(3, 1, hspace=0.6)

a = fig.add_subplot(gs[0])
a.plot(t[:2500] * 1e3, chirp[:2500], lw=0.7)
a.set_title("ЛЧМ-чирп: частота растёт со временем")
a.set_xlabel("время, мс")
a.set_ylabel("амплитуда")

mf = np.correlate(chirp, chirp, mode="full")
lags = (np.arange(len(mf)) - (len(chirp) - 1)) / fs
b = fig.add_subplot(gs[1])
b.plot(lags * 1e3, np.abs(mf), lw=0.8, color="C2")
```

```

b.set_title("Согласованный фильтр: длинный чирп сжался в узкий пик")
b.set_xlabel("задержка, мс")
b.set_ylabel("|корреляция|")

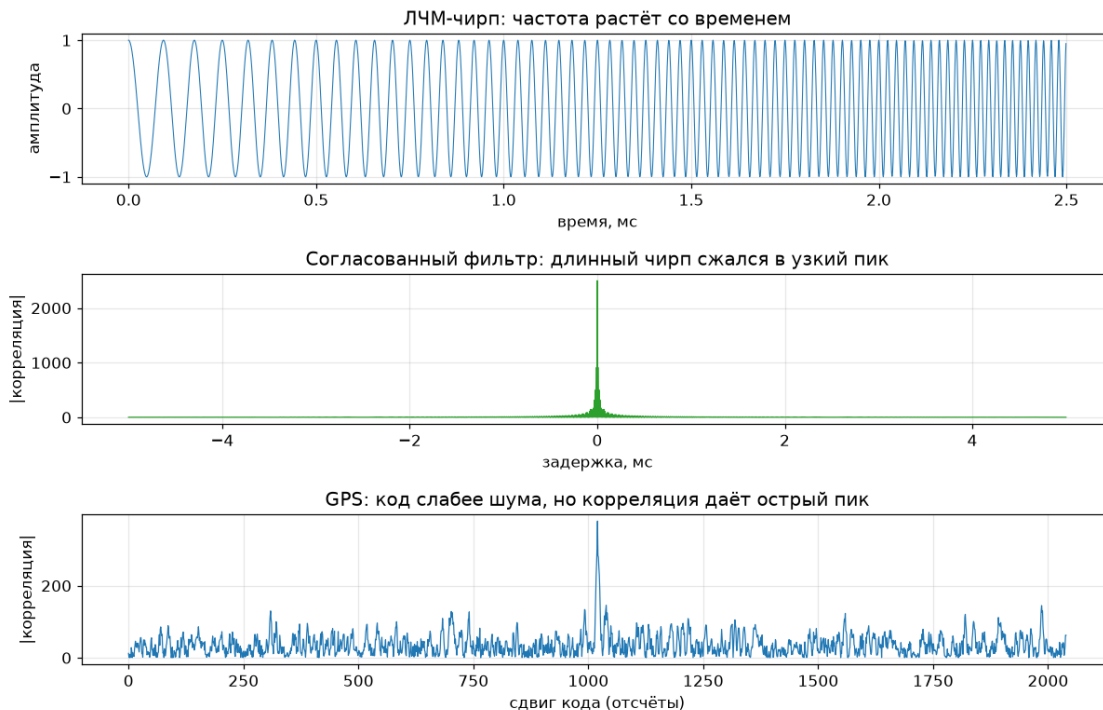
rng = np.random.default_rng(7)
nchips, sps_n = 255, 8
code = rng.choice([-1.0, 1.0], nchips)
tx = np.repeat(code, sps_n)
rx = 0.2 * tx + rng.standard_normal(len(tx))
corr = np.abs(np.correlate(rx, tx, mode="same"))
c = fig.add_subplot(gs[2])
c.plot(corr, lw=0.8, color="C0")
c.set_title("GPS: код слабее шума, но корреляция даёт острый пик")
c.set_xlabel("сдвиг кода (отсчёты)")
c.set_ylabel("|корреляция|")
fig.tight_layout()
save(fig, "fig16_chirp_gps.png")
plt.show()

```

/tmp/ipykernel_2803807/974088546.py:35: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.

fig.tight_layout()

сохранено: fig16_chirp_gps.png



Вывод раздела. Один и тот же аппарат — спектр, модуляция, корреляция, память — лежит в основе стерео-радио, телевидения, Wi-Fi, GPS, радаров и RFID. Освоив его, вы получаете инструмент, применимый почти в любой области.

1.6 3. Временная и частотная области

Один и тот же сигнал можно описывать двумя способами:

- **во времени** — как меняется значение;
- **по частоте** — какие колебания и с какой амплитудой в нём «спрятаны».

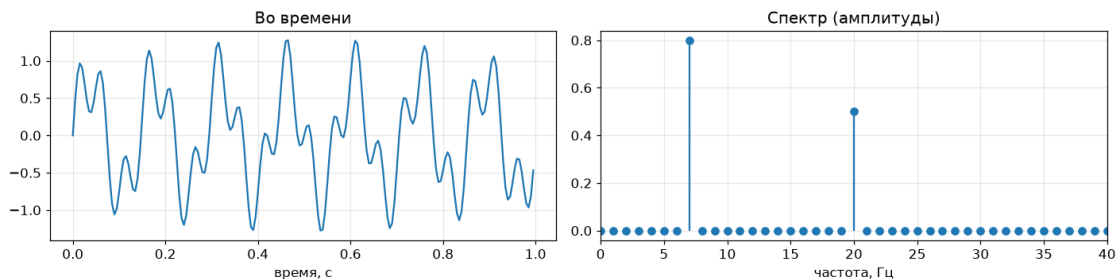
Спектр (преобразование Фурье) — это «рентген» сигнала: он показывает состав колебаний. Ниже — сигнал из двух гармоник. Во времени это сложная кривая, а в спектре — два чётких пика.

```
[11]: fs = 200
t = np.arange(0, 1, 1 / fs)
x = 0.8 * np.sin(2 * np.pi * 7 * t) + 0.5 * np.sin(2 * np.pi * 20 * t)

X = np.fft.rfft(x)
freqs = np.fft.rfftfreq(len(x), 1 / fs)

fig, ax = plt.subplots(1, 2, figsize=(13, 3.4))
ax[0].plot(t, x)
ax[0].set_title("Во времени")
ax[0].set_xlabel("время, с")
ax[1].stem(freqs, np.abs(X) / len(x) * 2, basefmt=" ")
ax[1].set_xlim(0, 40)
ax[1].set_title("Спектр (амплитуды)")
ax[1].set_xlabel("частота, Гц")
fig.tight_layout()
save(fig, "fig02_time_freq.png")
plt.show()
```

сохранено: fig02_time_freq.png



Вывод: по спектру сразу видно то, что во временной картинке «сливается»: две составляющие 7 Гц и 20 Гц. Именно так находят дефекты машин, помехи связи и частоты в

ЭКГ.

1.7 4. Дискретизация и теорема Котельникова

Чтобы обработать сигнал на компьютере, его измеряют через равные промежутки времени — **дискретизируют** с частотой f_d (сколько отсчётов в секунду).

Теорема Котельникова (Найквиста): сигнал можно точно восстановить, если

$$f_d \geq 2 \cdot f_{\text{макс}},$$

где $f_{\text{макс}}$ — самая высокая частота в сигнале.

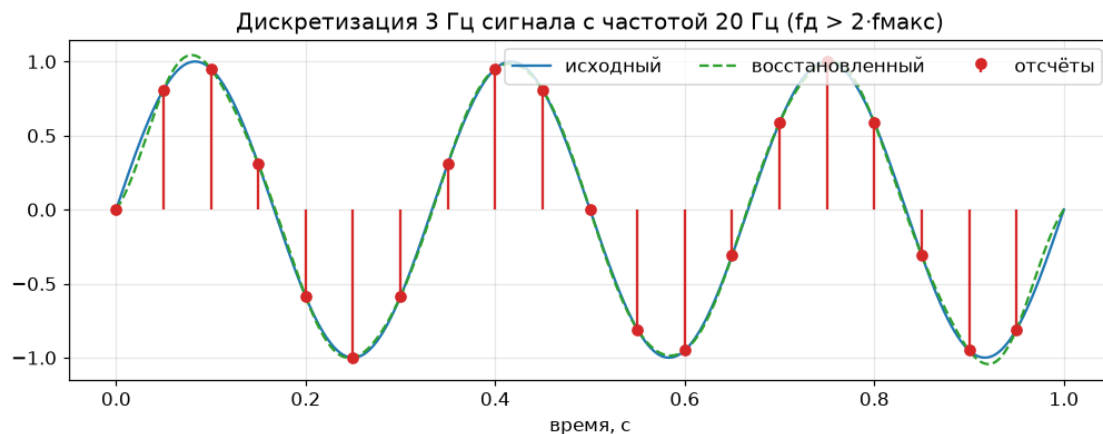
Если брать отсчёты слишком редко, разные сигналы становятся неразличимыми. Это явление — **алиасинг**.

```
[12]: fc = 3
t = np.linspace(0, 1, 3000)
x = np.sin(2 * np.pi * fc * t)

fsg = 20
ts = np.arange(0, 1, 1 / fsg)
xs = np.sin(2 * np.pi * fc * ts)
xr = sum(xs[k] * np.sinc(fsg * t - k) for k in range(len(xs)))

fig, ax = plt.subplots(figsize=(11, 3.6))
ax.plot(t, x, label="исходный", color="C0")
ax.plot(t, xr, "--", label="восстановленный", color="C2")
ax.stem(ts, xs, basefmt=" ", linefmt="C3", markerfmt="C3o", label="отсчёты")
ax.set_title("Дискретизация 3 Гц сигнала с частотой 20 Гц ( $f_d > 2 \cdot f_{\text{макс}}$ )")
ax.set_xlabel("время, с")
ax.legend(loc="upper right", ncol=3)
save(fig, "fig03_sampling.png")
plt.show()
```

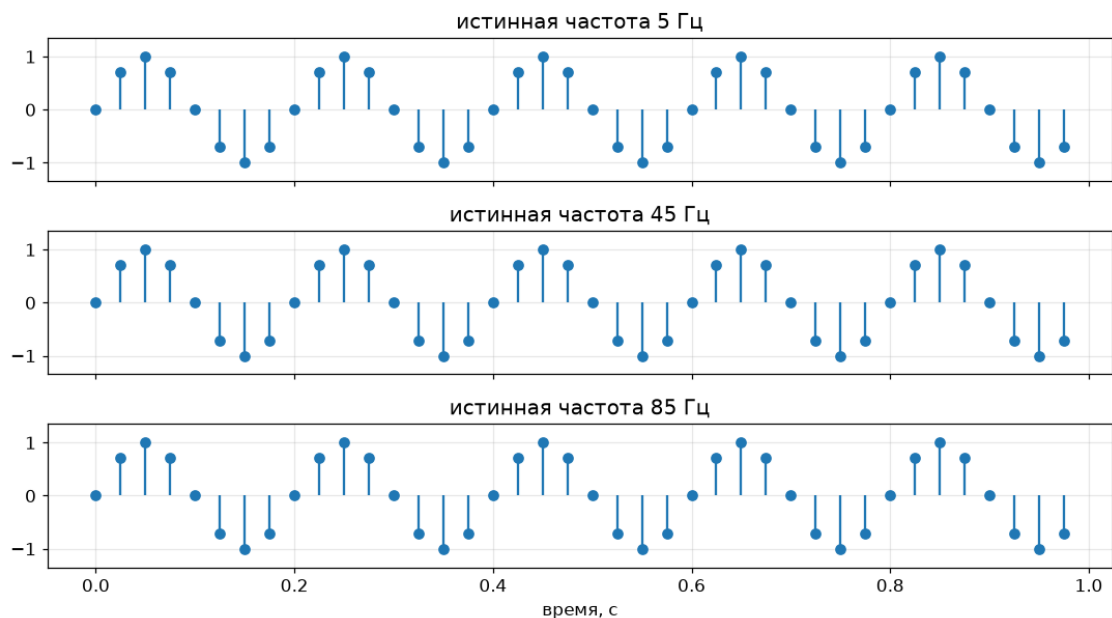
сохранено: fig03_sampling.png



```
[13]: fsg = 40
t = np.arange(0, 1, 1 / fsg)
fig, ax = plt.subplots(3, 1, figsize=(10, 6), sharex=True)
for a, f in zip(ax, [5, 45, 85]):
    a.stem(t, np.sin(2 * np.pi * f * t), basefmt=" ", linefmt="C0",
    ↪markerfmt="C0o")
    a.set_title(f"истинная частота {f} Гц")
    a.set_ylim(-1.35, 1.35)
ax[-1].set_xlabel("время, с")
fig.suptitle("Алиасинг: при fd = 40 Гц все три сигнала дают одинаковые отсчёты",
    ↪y=1.02)
fig.tight_layout()
save(fig, "fig04_aliasing.png")
plt.show()
```

сохранено: fig04_aliasing.png

Алиасинг: при $f_d = 40$ Гц все три сигнала дают одинаковые отсчёты



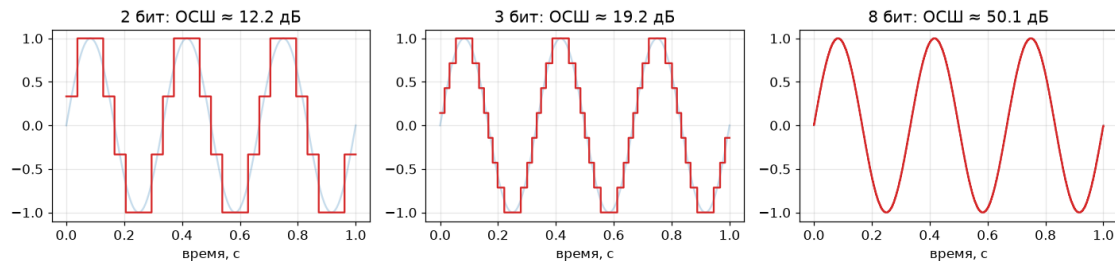
1.8 5. Квантование

Даже если отсчёты взяты правильно, значение хранится с ограниченной точностью — **разрядностью** (числом бит). Чем меньше бит, тем грубее «ступеньки» и тем больше ошибка квантования. Грубо: каждый бит добавляет около **6 дБ** к качеству.

```
[14]: t = np.linspace(0, 1, 1000)
x = np.sin(2 * np.pi * 3 * t)

fig, ax = plt.subplots(1, 3, figsize=(13, 3.2))
for a, bits in zip(ax, [2, 3, 8]):
    q = np.round((x + 1) / 2 * (2**bits - 1)) / (2**bits - 1) * 2 - 1
    err = x - q
    snr = 10 * np.log10(np.mean(x**2) / np.mean(err**2))
    a.plot(t, x, alpha=0.25, color="C0")
    a.step(t, q, where="mid", color="C3")
    a.set_title(f"{bits} бит: ОСШ ≈ {snr:.1f} дБ")
    a.set_xlabel("время, с")
fig.tight_layout()
save(fig, "fig05_quantization.png")
plt.show()
```

сохранено: fig05_quantization.png



1.9 6. Шум и отношение сигнал/шум

Реальные измерения всегда зашумлены. Мету «полезности» сигнала задаёт **отношение сигнал/шум (ОСШ, SNR)** в децибелах:

$$\text{ОСШ} = 10 \cdot \log_{10}(\text{мощность сигнала} / \text{мощность шума}).$$

Чем ниже ОСШ, тем труднее принять решение — и тем важнее правильная обработка.

```
[15]: rng = np.random.default_rng(0)
t = np.arange(0, 1, 1 / 500)
clean = np.sin(2 * np.pi * 5 * t)

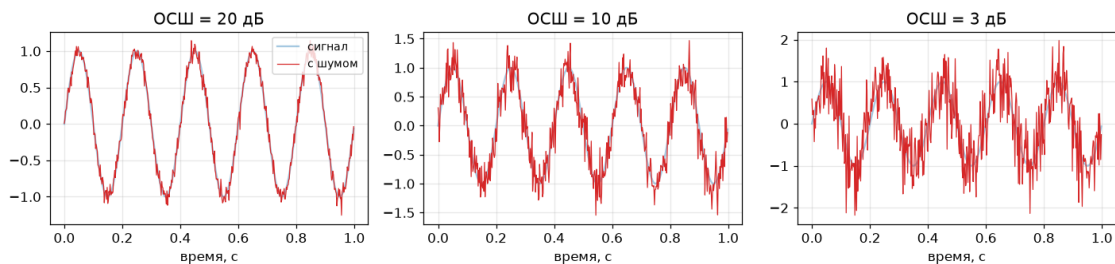
fig, ax = plt.subplots(1, 3, figsize=(13, 3.2))
for a, snr_db in zip(ax, [20, 10, 3]):
    noise = rng.standard_normal(len(t))
    noise *= np.std(clean) / np.std(noise) / 10 ** (snr_db / 20)
    noisy = clean + noise
    a.plot(t, clean, alpha=0.4, color="C0", label="сигнал")
    a.plot(t, noisy, color="C3", linewidth=0.8, label="с шумом")
```

```

a.set_title(f"ОСШ = {snr_db} дБ")
a.set_xlabel("время, с")
ax[0].legend(loc="upper right", fontsize=9)
fig.tight_layout()
save(fig, "fig06_noise_snr.png")
plt.show()

```

сохранено: fig06_noise_snr.png



1.10 7. Реальный пример: вибрация механизма

Смоделируем сигнал датчика вибрации: полезная низкочастотная составляющая, признак дефекта подшипника (характерная частота) и шум. Во времени дефект почти не виден, но на спектрограмме он проявляется как устойчивая полоса.

```

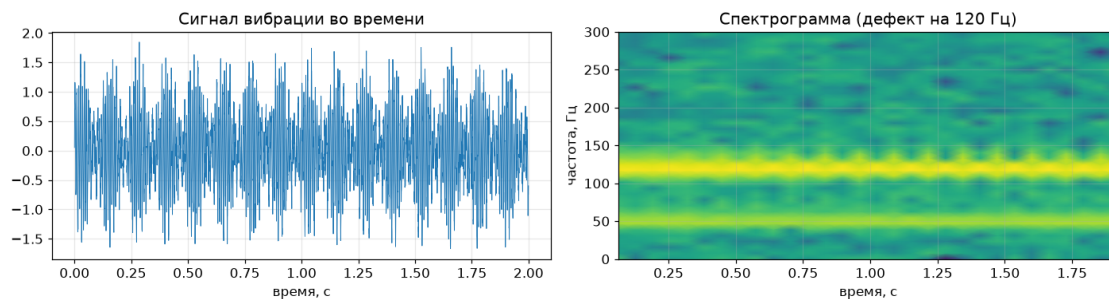
[16]: from scipy import signal as sps

fs = 2000
t = np.arange(0, 2, 1 / fs)
rng = np.random.default_rng(1)
base = 0.3 * np.sin(2 * np.pi * 50 * t)
defect = (1 + 0.6 * np.sin(2 * np.pi * 8 * t)) * np.sin(2 * np.pi * 120 * t)
vib = base + 0.8 * defect + 0.15 * rng.standard_normal(len(t))

fig, ax = plt.subplots(1, 2, figsize=(13, 3.6))
ax[0].plot(t, vib, linewidth=0.6)
ax[0].set_title("Сигнал вибрации во времени")
ax[0].set_xlabel("время, с")
f, tt, Sxx = sps.spectrogram(vib, fs, nperseg=256, noverlap=128)
ax[1].pcolormesh(tt, f, 10 * np.log10(Sxx + 1e-12), shading="gouraud")
ax[1].set_ylim(0, 300)
ax[1].set_title("Спектрограмма (дефект на 120 Гц)")
ax[1].set_xlabel("время, с")
ax[1].set_ylabel("частота, Гц")
fig.tight_layout()
save(fig, "fig07_vibration.png")
plt.show()

```

сохранено: fig07_vibration.png



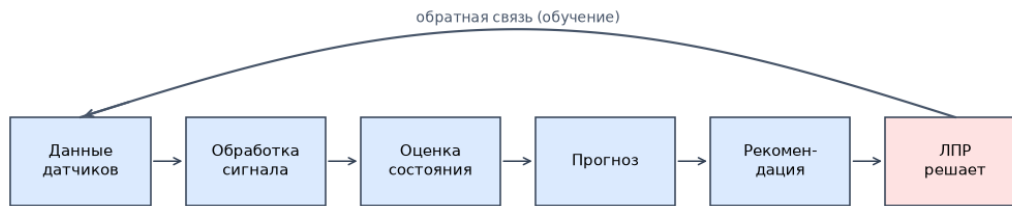
1.11 8. От сигнала к решению: место ЦОС в КСППР

Обработка сигналов — это **входной тракт** системы поддержки принятия решений (КСППР). Она превращает сырые данные датчиков в достоверные числа, на которых строится состояние, прогноз и рекомендация человеку.

```
[17]: fig, ax = plt.subplots(figsize=(12.5, 2.8))
ax.axis("off")
boxes = ["Данные\ndатчиков", "Обработка\нсигнала", "Оценка\нсостояния",
        "Прогноз", "Рекомен-\ndация", "ЛПР\nпрешает"]
x = np.arange(len(boxes)) * 2.0
for xi, label in zip(x, boxes):
    color = "#dbeeaf" if label != "ЛПР\nпрешает" else "#fee2e2"
    ax.add_patch(plt.Rectangle((xi, 0), 1.6, 1.0, facecolor=color,
                               edgecolor="#334155"))
    ax.text(xi + 0.8, 0.5, label, ha="center", va="center", fontsize=10)
for xi in x[:-1]:
    ax.annotate("", xy=(xi + 2.0, 0.5), xytext=(xi + 1.6, 0.5),
                arrowprops=dict(arrowstyle="->", color="#334155"))
xs = np.linspace(x[-1] + 0.8, x[0] + 0.8, 120)
ys = 1.0 + 1.0 * np.sin(np.linspace(0, np.pi, 120))
ax.plot(xs, ys, color="#475569", lw=1.4)
ax.annotate("", xy=(x[0] + 0.8, 1.0), xytext=(xs[-8], ys[-8]),
            arrowprops=dict(arrowstyle="->", color="#475569", lw=1.4))
ax.text((x[0] + x[-1]) / 2 + 0.8, 2.08, "обратная связь (обучение)",
        ha="center", fontsize=9, color="#475569")
ax.set_xlim(-0.3, x[-1] + 2.3)
ax.set_ylim(-0.3, 2.5)
ax.set_title("Контур КСППР: обработка сигнала — вход в систему принятия решений")
save(fig, "fig08_ksppr_loop.png")
plt.show()
```

сохранено: fig08_ksppr_loop.png

Контур КСППР: обработка сигнала — вход в систему принятия решений



1.12 9. Аппаратный стенд курса

Чтобы сигналы были настоящими, используем доступное оборудование:

- **ESP32-C3 (XIAO)** — узел сбора данных: АЦП, I2S, Wi-Fi; шум и нелинейность его АЦП — отличный натуральный пример к теме квантования.
- **LiDAR** — измерение дальности и углов, сканы пространства.
- **RFID 800 МГц (UHF)** — идентификация объектов по метке; RSSI и фаза сигнала.
- **Локальный GPU (AMD RX 6750 XT)** — обучение моделей и локальные LLM для AI-ассистента.

Студенты ведут **свой сквозной проект** — один из 12 индустриальных кейсов или свою тему (см. projects.md). «Умный микрофон» — пример преподавателя.

```
[18]: fig, ax = plt.subplots(figsize=(12.5, 3.4))
ax.axis("off")

# Сенсорный слой — отдельная область сверху; RFID входит в него
ax.add_patch(plt.Rectangle((0.2, 1.72), 9.9, 0.96, facecolor="#eef2f7",
                           edgecolor="#94a3b8", linewidth=1.2))
ax.text(0.35, 2.74, "Сенсорный слой", fontsize=10, color="#475569",
        ha="left", va="bottom")
sensors = ["Микрофон\n(I2S)", "Акселеро-\nметр", "LiDAR", "RFID\n800 МГц"]
for i, s in enumerate(sensors):
    x = 0.4 + i * 2.45
    ax.add_patch(plt.Rectangle((x, 1.84), 2.25, 0.72,
                              facecolor="#e2e8f0", edgecolor="#334155"))
    ax.text(x + 1.12, 2.2, s, ha="center", va="center", fontsize=9)

# Тракт обработки
ax.add_patch(plt.Rectangle((1.2, 0.35), 1.9, 0.9, facecolor="#dbeafe",
                           edgecolor="#334155"))
ax.text(2.15, 0.8, "ESP32-C3\nсбор и стриминг", ha="center", va="center",
        fontsize=9)
ax.add_patch(plt.Rectangle((4.0, 0.35), 1.9, 0.9, facecolor="#dcfce7",
                           edgecolor="#334155"))
```

```

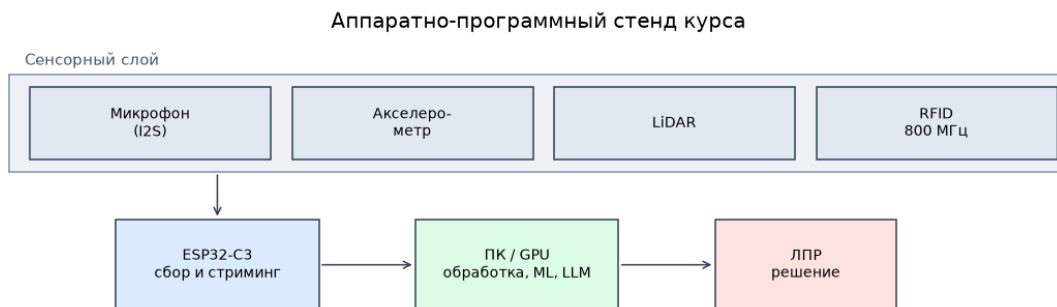
ax.text(4.95, 0.8, "ПК / GPU\нобработка, ML, LLM", ha="center", va="center",
↪ fontsize=9)
ax.add_patch(plt.Rectangle((6.8, 0.35), 1.7, 0.9, facecolor="#fee2e2",
↪ edgecolor="#334155"))
ax.text(7.65, 0.8, "ЛПР\нрешение", ha="center", va="center", fontsize=9)

ax.annotate("", xy=(2.15, 1.25), xytext=(2.15, 1.72),
            arrowprops=dict(arrowstyle="->", color="#334155"))
for x0, x1 in [(3.1, 4.0), (5.9, 6.8)]:
    ax.annotate("", xy=(x1, 0.8), xytext=(x0, 0.8),
                arrowprops=dict(arrowstyle="->", color="#334155"))

ax.set_xlim(0, 10.3)
ax.set_ylim(0.1, 3.05)
ax.set_title("Аппаратно-программный стенд курса")
save(fig, "fig09_stand.png")
plt.show()

```

сохранено: fig09_stand.png



1.13 10. Пример проекта преподавателя: «умный микрофон»

Покажу **свой проект** — систему подготовки лекций на базе умного микрофона. Он нужен как живой пример: на нём видно, как из сигнала получается решение. Ваши проекты (12 кейсов или своя тема) устроены так же. Система слушает аудио и помогает делать качественные записи и их разметку:

- следит за **ОСШ** сигнала с микрофона в реальном времени;
- **распознаёт речь и паузы** (детектор активности голоса, VAD);
- делает **разметку видеоряда** по событиям (речь / пауза / начало темы);
- автоматически **ставит запись видео на паузу** в тишине — экономия и чистая нарезка;
- далее (лекции 11–13): транскрибация и конспект с помощью ML и локальной LLM.

Идеальный конечный результат — **готовая лекция без этапа вёрстки**: система сама нарезает видео, расставляет главы и субтитры, готовит конспект — ручной монтаж не нужен. Каждая

лекция курса добавляет в систему свой блок — от спектра и фильтрации до Калмана и машинного обучения. Ниже — маленькое демо ровно того, с чего проект начинается.

```
[19]: fs = 16000
dur = 12
t = np.arange(0, dur, 1 / fs)
rng = np.random.default_rng(3)

def phrase(length, f0=140):
    tt = np.arange(0, length, 1 / fs)
    env = (0.55 + 0.45 * np.sin(2 * np.pi * 3 * tt)) * np.hanning(len(tt))
    voice = sum(np.sin(2 * np.pi * f0 * k * tt) for k in (1, 2, 3))
    return env * voice

sig = 0.02 * rng.standard_normal(len(t))
for start, length in [(0.5, 2.0), (3.2, 2.5), (6.5, 1.8), (9.0, 2.2)]:
    i0 = int(start * fs)
    seg = phrase(length)
    sig[i0:i0 + len(seg)] += seg

frame, hop = 512, 256
n_frames = (len(sig) - frame) // hop
rms = np.array([
    np.sqrt(np.mean(sig[i * hop:i * hop + frame] ** 2)) for i in range(n_frames)
])
fct = (np.arange(n_frames) * hop + frame / 2) / fs
db = 20 * np.log10(rms + 1e-9)
thr_db = np.percentile(db, 15) + 8
active = db > thr_db

speech, pause = rms[active], rms[~active]
print(f"ОСШ печи ≈ {20 * np.log10(speech.mean() / pause.mean()):.1f} дБ")
print(f"кадров: печь {active.sum()}, пауза {(~active).sum()}")

fig, ax = plt.subplots(3, 1, figsize=(12, 7), sharex=True,
                      gridspec_kw={"height_ratios": [2, 1.4, 0.6]})
ax[0].plot(t, sig, linewidth=0.5)
ax[0].set_ylabel("микрофон")
ax[0].set_title("Умный микрофон: сигнал, ОСШ и разметка пауз")
ax[1].plot(fct, db, color="C0")
ax[1].axhline(thr_db, color="C3", ls="--", label="порог паузы")
ax[1].fill_between(fct, db.min(), db, where=~active, color="C3", alpha=0.15)
ax[1].set_ylabel("ОСШ, дБ")
ax[1].legend(loc="upper right", fontsize=9)
for a in ax[2:]:
```

```

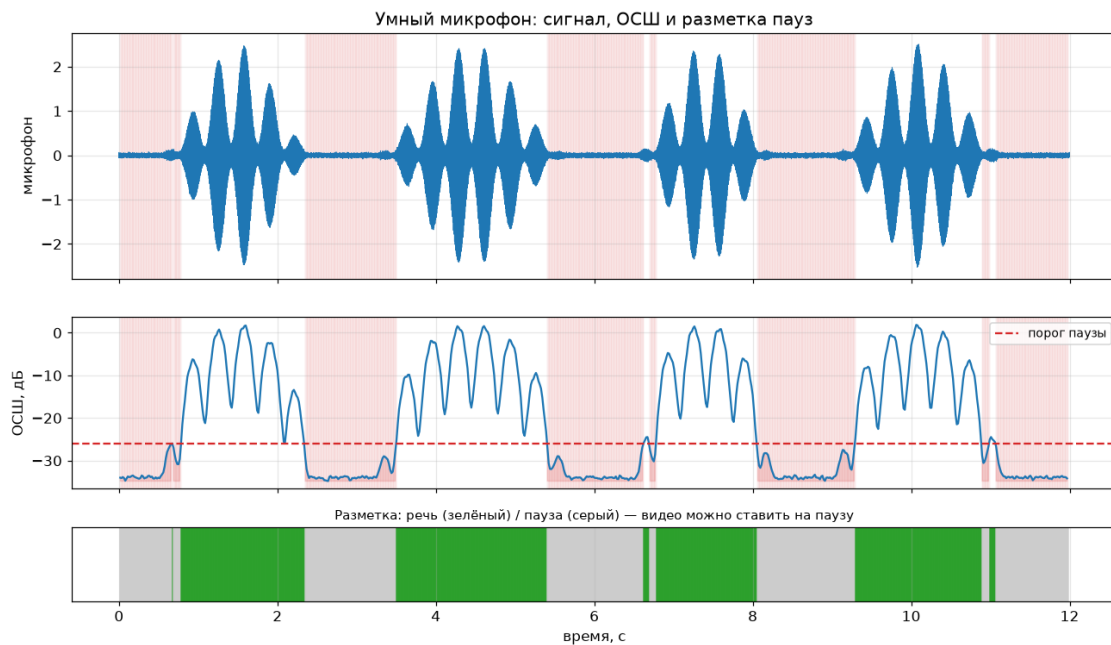
for i in range(n_frames):
    if not active[i]:
        a.axvspan(fct[i], fct[i] + hop / fs, color="C3", alpha=0.05)
for i in range(n_frames):
    ax[2].axvspan(fct[i], fct[i] + hop / fs,
                  color="C2" if active[i] else "0.8", alpha=0.9)
ax[2].set_yticks([])
ax[2].set_xlabel("время, с")
ax[2].set_title("Разметка: речь (зелёный) / пауза (серый) — видео можно ставить ↵
↵на паузу",
                fontsize=10)
fig.tight_layout()
save(fig, "fig10_smartmic.png")
plt.show()

```

ОСШ речи ≈ 25.4 дБ

кадров: речь 408, пауза 340

сохранено: fig10_smartmic.png



На графике видно всё, что понадобится проекту: где речь, где тишина, как меняется ОСШ и как из этого получается **разметка**. В реальной системе эта разметка управляет записью видео и превращается в структуру лекции.

1.14 11. Инструменты

- **Python:** NumPy, SciPy, Matplotlib — база для расчётов и визуализации.

- Далее: scikit-learn и PyTorch — для машинного обучения на сигналах.
- **Local LLM** (Ollama) + RAG по учебникам — AI-ассистент курса, который помогает, но не заменяет проверку человеком.

Принцип курса: **сначала решаем задачу сами, потом автоматизируем.**

1.15 Сквозной проект: задание №1

Каждый студент (команда) ведёт **свой проект** — один из 12 индустриальных кейсов или свою тему (см. projects.md). «Умный микрофон» из раздела 10 — пример преподавателя, на нём мы разбирали приёмы.

Задание после лекции 1 (для вашего проекта):

1. Зафиксировать тему и определить **сигнал**, который в ней обрабатывается (звук, вибрация, ток, RFID/LiDAR и т.п.).
2. Получить 10–20 с сигнала: синтетика или запись с датчика / ESP32-C3.
3. Построить сигнал во времени и **спектр**; оценить **ОСШ** по кадрам.
4. Выделить простой признак/разметку (для аудио — речь/пауза) и сохранить в JSON.
5. (Опционально) Снять реальный сигнал через ESP32-C3 (АЦП или I2S).

Сдавать: ноутбук с графиками + **один** слайд-вывод, к следующей лекции.

Пример выполнения всей цепочки — «умный микрофон» преподавателя (раздел 10): запись речи → ОСШ → паузы → разметка.

1.16 Итоги лекции 1

- Сигнал несёт информацию; его описывают во времени и по частоте.
- Дискретизация должна соблюдать $f_d \geq 2 \cdot f_{\text{макс}}$, иначе — алиасинг.
- Квантование ограничивает точность; каждый бит $\approx +6$ дБ качества.
- Шум неизбежен, ОСШ определяет трудность решения.
- Обработка сигналов — входной тракт КСППР: от данных к решению для человека.

Дальше: спектральный анализ подробнее, свёртка и фильтрация.

1.17 Литература

1. Оппенгейм А., Шафер Р. *Цифровая обработка сигналов*. — М.: Техносфера, 2012.
2. Сергиенко А. Б. *Цифровая обработка сигналов*. — СПб.: БХВ-Петербург, 2011.
3. Солонина А. И. и др. *Основы цифровой обработки сигналов*. — СПб.: БХВ-Петербург, 2005.
4. Умняшкин С. В. *Основы теории цифровой обработки сигналов*.
5. Материалы курса и RAG-ассистент по этим книгам — в папке курса.