

lecture02

September 25, 2026

1 Лекция 2. Гармонический анализ сигналов

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

Направленность: Искусственный интеллект и системная аналитика

На лекции 1 мы увидели, что у сигнала есть два портрета: **во времени** и **по частоте**, и назвали спектр «рентгеном» сигнала. Но откуда берётся спектр и что означают его пики, мы не разбирали. Сегодня вскрываем этот «чёрный ящик»: любой сигнал можно собрать из **синусоид**, а спектр показывает, сколько какой синусоиды в нём содержится. Формул по-прежнему минимум — всё картинками на Python.

1.1 Цели лекции

К концу лекции вы сможете:

1. Объяснить, почему **синусоиду** называют «кирпичом» любого сигнала.
2. Собрать сложный периодический сигнал из **ряда Фурье** (суммы гармоник).
3. Читать **амплитудный** и **фазовый** спектр и понимать, что теряется без фазы.
4. Объяснить свойства преобразования Фурье, включая **перенос спектра** при умножении на несущую.
5. Увидеть **спектры модуляции** (АМ и ЧМ) как примеры и **спектры реальных сигналов**.
6. Понять, почему у реального спектра пики «расплываются» (**утечка**), — мостик к окнам.
7. Решить **минимальную задачу ЦОС**: по дискретным отсчётам восстановить параметры сигнала (амплитуду, частоту, фазу) методами **МНК** и **БПФ**.

1.2 1. Синусоида — «кирпич» любого сигнала

Простейшее колебание — **синусоида**. У неё всего три параметра:

- **амплитуда A** — «насколько высоко»;
- **частота f** — «как часто» (или период $T = 1/f$);
- **фаза φ** — «с какого места» начинается колебание.

Мысль лекции: **любой сигнал можно собрать из таких кирпичиков**, а спектр — это список деталей с указанием, сколько каждого кирпичика взять.

```
[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, 1000)

fig, ax = plt.subplots(1, 3, figsize=(13, 3.2))

ax[0].plot(t, 0.4 * np.sin(2 * np.pi * 3 * t), label="A = 0.4")
ax[0].plot(t, 1.0 * np.sin(2 * np.pi * 3 * t), label="A = 1.0")
ax[0].set_title("Амплитуда A")

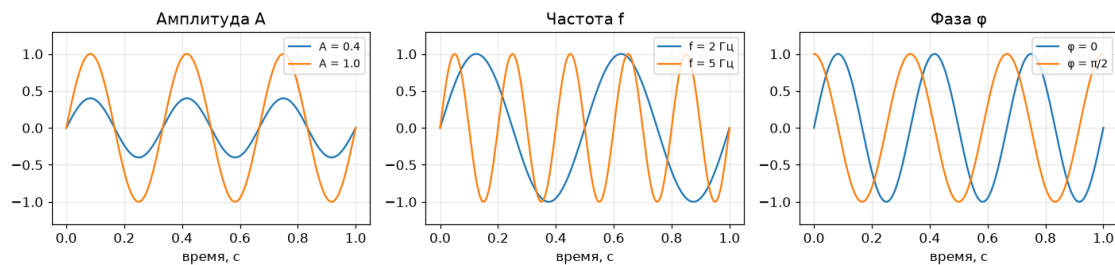
ax[1].plot(t, np.sin(2 * np.pi * 2 * t), label="f = 2 Гц")
ax[1].plot(t, np.sin(2 * np.pi * 5 * t), label="f = 5 Гц")
ax[1].set_title("Частота f")

ax[2].plot(t, np.sin(2 * np.pi * 3 * t), label="φ = 0")
ax[2].plot(t, np.sin(2 * np.pi * 3 * t + np.pi / 2), label="φ = π/2")
ax[2].set_title("Фаза φ")

for a in ax:
    a.set_xlabel("время, с")
    a.set_ylim(-1.3, 1.3)
    a.legend(fontsize=9, loc="upper right")
fig.tight_layout()
save(fig, "fig_sine_params.png")
plt.show()

```

сохранено: fig_sine_params.png



1.3 2. Ряд Фурье: сигнал как сумма гармоник

Идея Фурье. Любой периодический сигнал раскладывается в сумму синусоид, частоты которых кратны основной: f_0 , $2 \cdot f_0$, $3 \cdot f_0$, ... Эти составляющие называют **гармониками**. Чем больше гармоник возьмём, тем точнее приблизим форму.

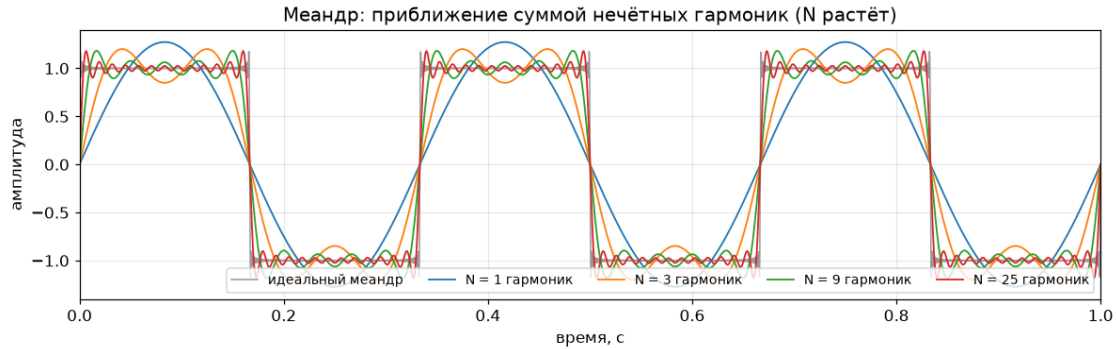
Проверим на **меандре** (прямоугольном сигнале): у него остаются только **нечётные** гармоники, а амплитуда каждой падает как $1/k$.

```
[3]: t = np.linspace(0, 1, 4000)

def square_partial(t, N, f0=3):
    s = np.zeros_like(t)
    for k in range(1, N + 1, 2):
        s += np.sin(2 * np.pi * f0 * k * t) / k
    return 4 / np.pi * s

fig, ax = plt.subplots(figsize=(11, 3.6))
ax.plot(t, square_partial(t, 201), color="0.6", lw=1.0, label="идеальный меандр")
for N, col in [(1, "C0"), (3, "C1"), (9, "C2"), (25, "C3")]:
    ax.plot(t, square_partial(t, N), lw=1.2, color=col, label=f"N = {N} ↗ гармоник")
ax.set_xlim(0, 1)
ax.set_ylim(-1.4, 1.4)
ax.set_xlabel("время, с")
ax.set_ylabel("амплитуда")
ax.set_title("Меандр: приближение суммой нечётных гармоник (N растёт)")
ax.legend(loc="lower right", ncol=5, fontsize=9)
fig.tight_layout()
save(fig, "fig_gibbs.png")
plt.show()
```

сохранено: fig_gibbs.png



Явление Гиббса. Около разрыва сумма всегда «звонит» — появляется выброс $\approx 9\%$ высоты. Добавляя гармоники, мы сужаем этот выброс, но не убираем. Это важное напоминание: сигнал с разрывами «дорого» описывать гармониками — понадобится много составляющих.

```
[4]: N = 2000
t = np.linspace(0, 1, N, endpoint=False)
sq = square_partial(t, 99)
freqs = np.fft.rfftfreq(N, 1 / N)
spec = np.abs(np.fft.rfft(sq)) / N * 2

fig, ax = plt.subplots(figsize=(11, 3.6))
ax.stem(freqs, spec, basefmt=" ", linefmt="C0", markerfmt="C0o")
ax.set_xlim(0, 40)
ax.set_xlabel("частота, Гц")
ax.set_ylabel("амплитуда гармоник")
ax.set_title("Спектр меандра: только нечётные гармоники 3, 9, 15, 21, ...")
fig.tight_layout()
save(fig, "fig_square_spectrum.png")
plt.show()
```

сохранено: fig_square_spectrum.png



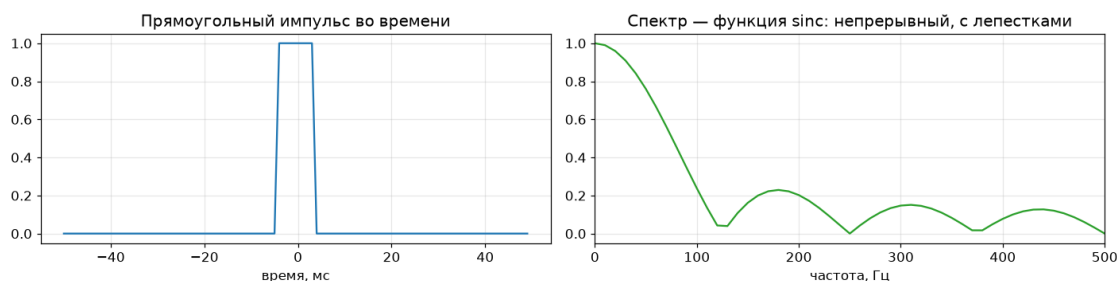
1.4 3. Непериодический сигнал: спектр становится непрерывным

Если сигнал не повторяется (одиночный импульс), сумма превращается в **интеграл Фурье**, а отдельные линии — в **непрерывную кривую** спектра. Для прямоугольного импульса эта кривая — функция $\text{sinc}(\sin(x)/x)$: чем уже импульс во времени, тем шире его спектр.

```
[5]: fs = 1000
t = np.arange(-0.05, 0.05, 1 / fs)
pulse = np.where(np.abs(t) < 0.004, 1.0, 0.0)
freqs = np.fft.rfftfreq(len(pulse), 1 / fs)
spec = np.abs(np.fft.rfft(pulse))

fig, ax = plt.subplots(1, 2, figsize=(13, 3.4))
ax[0].plot(t * 1e3, pulse, color="C0")
ax[0].set_title("Прямоугольный импульс во времени")
ax[0].set_xlabel("время, мс")
ax[1].plot(freqs, spec / spec.max(), color="C2")
ax[1].set_xlim(0, 500)
ax[1].set_title("Спектр — функция sinc: непрерывный, с лепестками")
ax[1].set_xlabel("частота, Гц")
fig.tight_layout()
save(fig, "fig_sinc.png")
plt.show()
```

сохранено: fig_sinc.png



1.5 4. Амплитудный и фазовый спектр

Полный спектр — **комплексный**: у каждой частоты есть амплитуда **и** фаза. Обычно мы смотрим только на амплитуды и теряем фазу, но именно фаза отвечает за **форму** сигнала и за расположение его во времени. Это критично, например, для RFID, где именно фаза несёт информацию о расстоянии.

```
[6]: fs = 200
t = np.arange(0, 1, 1 / fs)
x = 1.0 * np.sin(2 * np.pi * 7 * t) + 0.5 * np.sin(2 * np.pi * 20 * t + np.pi / 3)
↪3)
```

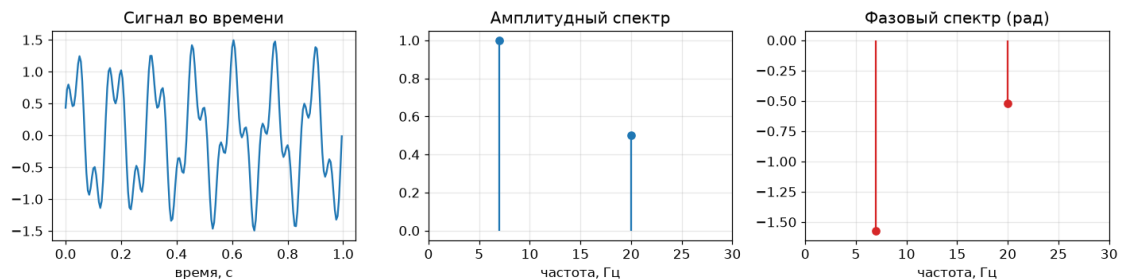
```

X = np.fft.rfft(x)
freqs = np.fft.rfftfreq(len(x), 1 / fs)
amp = np.abs(X) / len(x) * 2
phase = np.angle(X)
peaks = amp > 0.1

fig, ax = plt.subplots(1, 3, figsize=(13, 3.4))
ax[0].plot(t, x)
ax[0].set_title("Сигнал во времени")
ax[0].set_xlabel("время, с")
ax[1].stem(freqs[peaks], amp[peaks], basefmt=" ")
ax[1].set_xlim(0, 30)
ax[1].set_title("Амплитудный спектр")
ax[1].set_xlabel("частота, Гц")
ax[2].stem(freqs[peaks], phase[peaks], basefmt=" ", linefmt="C3",
            markerfmt="C3o")
ax[2].set_xlim(0, 30)
ax[2].set_title("Фазовый спектр (рад)")
ax[2].set_xlabel("частота, Гц")
fig.tight_layout()
save(fig, "fig_amp_phase.png")
plt.show()

```

сохранено: fig_amp_phase.png



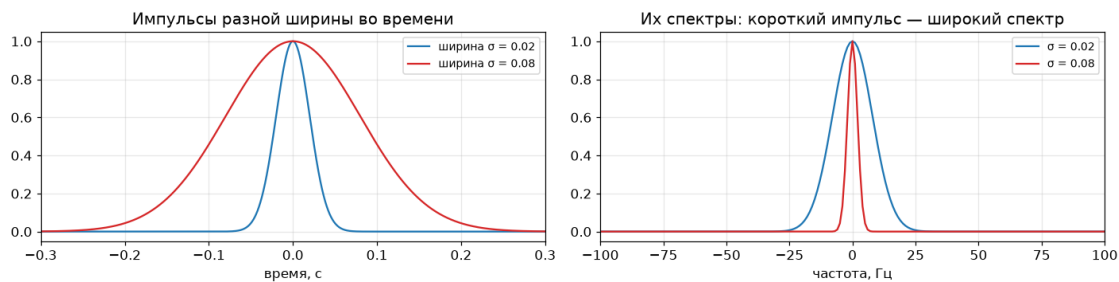
1.6 5. Свойства преобразования Фурье

Полезно помнить несколько свойств — без доказательств:

- **Линейность:** спектр суммы = сумма спектров.
- **Сдвиг во времени ↔ фаза:** сигнал сдвинули — амплитуды те же, фазы изменились.
- **Масштаб (подобие):** чем **короче** сигнал во времени, тем **шире** его спектр.
- **Умножение на несущую = перенос спектра** — это и есть модуляция (см. ниже).

```
[7]: t = np.linspace(-0.5, 0.5, 4000)
fig, ax = plt.subplots(1, 2, figsize=(13, 3.4))
for w, col in [(0.02, "C0"), (0.08, "C3")]:
    pulse = np.exp(-t**2 / (2 * w**2))
    spec = np.abs(np.fft.fft(pulse))
    spec /= spec.max()
    ax[0].plot(t, pulse, color=col, label=f"ширина  $\sigma = \{w\}$ ")
    freqs = np.fft.fftfreq(len(t), t[1] - t[0])
    ax[1].plot(np.fft.fftshift(freqs), np.fft.fftshift(spec), color=col,
               label=f" $\sigma = \{w\}$ ")
ax[0].set_xlim(-0.3, 0.3)
ax[0].set_title("Импульсы разной ширины во времени")
ax[0].set_xlabel("время, с")
ax[1].set_xlim(-100, 100)
ax[1].set_title("Их спектры: короткий импульс — широкий спектр")
ax[1].set_xlabel("частота, Гц")
for a in ax:
    a.legend(fontsize=9)
fig.tight_layout()
save(fig, "fig_time_bw.png")
plt.show()
```

сохранено: fig_time_bw.png



1.7 6. Спектры радиосигналов: АМ и ЧМ (примеры)

Амплитудная модуляция (АМ). Умножаем сообщение на несущую — спектр сообщения «переезжает» на частоту несущей и даёт **две боковые полосы** $f_0 \pm F$. В эфире это выглядит как несущая и два её «спутника».

Частотная модуляция (ЧМ). Меняем частоту несущей. Спектр уже не две линии, а **расходящийся набор** боковых: чем больше **девиация**, тем шире полоса. Похоже ведёт себя и фазовая модуляция (ФМ) — фаза и частота связаны.

```
[8]: fs = 200e3
T = 4e-3
t = np.arange(0, T, 1 / fs)
```

```

fm, fc = 1e3, 20e3
m = np.cos(2 * np.pi * fm * t)

am = (1 + 0.7 * m) * np.cos(2 * np.pi * fc * t)
beta = 5.0
fm_sig = np.cos(2 * np.pi * fc * t + beta * np.sin(2 * np.pi * fm * t))

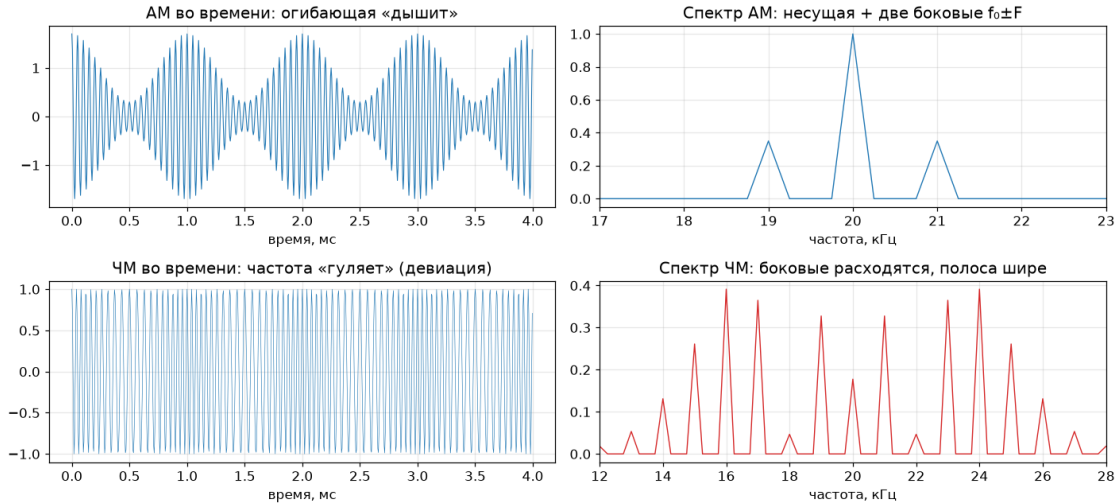
def spectrum(sig):
    f = np.fft.rfftfreq(len(sig), 1 / fs)
    s = np.abs(np.fft.rfft(sig)) / len(sig) * 2
    return f, s

fig, ax = plt.subplots(2, 2, figsize=(13, 6))
fa, sa = spectrum(am)
ax[0, 0].plot(t * 1e3, am, lw=0.7)
ax[0, 0].set_title("АМ во времени: огибающая «дышит»")
ax[0, 0].set_xlabel("время, мс")
ax[0, 1].plot(fa / 1e3, sa, lw=0.9)
ax[0, 1].set_xlim(17, 23)
ax[0, 1].set_title("Спектр АМ: несущая + две боковые  $f_0 \pm F$ ")
ax[0, 1].set_xlabel("частота, кГц")

ff, sf = spectrum(fm_sig)
ax[1, 0].plot(t * 1e3, fm_sig, lw=0.4)
ax[1, 0].set_title("ЧМ во времени: частота «гуляет» (девиация)")
ax[1, 0].set_xlabel("время, мс")
ax[1, 1].plot(ff / 1e3, sf, lw=0.9, color="C3")
ax[1, 1].set_xlim(12, 28)
ax[1, 1].set_title("Спектр ЧМ: боковые расходятся, полоса шире")
ax[1, 1].set_xlabel("частота, кГц")
fig.tight_layout()
save(fig, "fig_am_fm.png")
plt.show()

```

сохранено: fig_am_fm.png



Затравка: цифровая модуляция. В цифровой связи сигнал удобно рисовать **точкой на плоскости I/Q**: положение точки задаёт амплитуду и фазу символа. BPSK — две точки, QPSK — четыре, QAM — целая решётка. Подробно разберём это на лекции 5 (вместе с комплексной огибающей).

```
[9]: from scipy import signal as sps

fig, ax = plt.subplots(1, 3, figsize=(13, 3.6))

qpsk = np.array([[1, 1], [1, -1], [-1, 1], [-1, -1]])
ax[0].plot(qpsk[:, 0], qpsk[:, 1], "o", ms=10, color="C0")
ax[0].axhline(0, color="0.8")
ax[0].axvline(0, color="0.8")
ax[0].set_xlim(-1.6, 1.6)
ax[0].set_ylim(-1.6, 1.6)
ax[0].set_title("QPSK: 4 точки (2 бита)")
ax[0].set_xlabel("I")
ax[0].set_ylabel("Q")

levels = [-3, -1, 1, 3]
gx, gy = np.meshgrid(levels, levels)
ax[1].plot(gx.ravel(), gy.ravel(), "o", ms=7, color="C2")
ax[1].axhline(0, color="0.8")
ax[1].axvline(0, color="0.8")
ax[1].set_title("16-QAM: решётка 4×4 (4 бита)")
ax[1].set_xlabel("I")
ax[1].set_ylabel("Q")

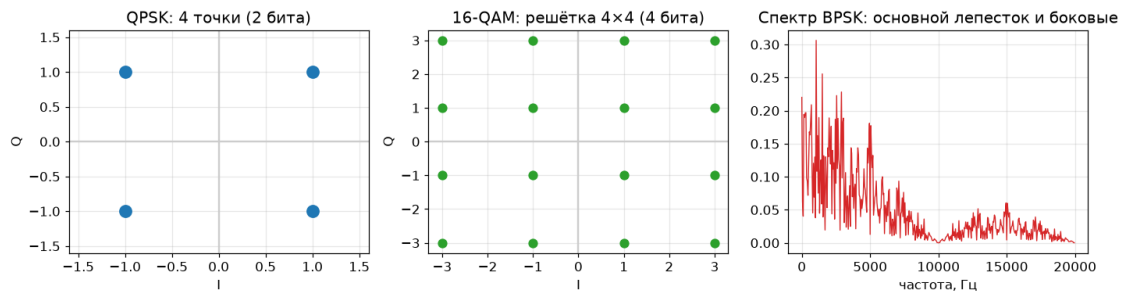
rng = np.random.default_rng(0)
sps, n = 20, 200
```

```

bits = rng.choice([-1.0, 1.0], n)
sym = np.repeat(bits, sps)
f2, s2 = spectrum(sym)
ax[2].plot(f2[:400], s2[:400], lw=0.9, color="C3")
ax[2].set_title("Спектр BPSK: основной лепесток и боковые")
ax[2].set_xlabel("частота, Гц")
fig.tight_layout()
save(fig, "fig_psk_qam.png")
plt.show()

```

сохранено: fig_psk_qam.png



1.8 7. Спектры реальных сигналов

Тот же аппарат работает на настоящих данных. Возьмём сигнал вибрации из нашего кейса (диагностика насосного агрегата): базовая частота, признак дефекта подшипника и шум. Во времени дефект почти не виден, а в спектре — это отчётливая линия.

```

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

freqs = np.fft.rfftfreq(len(vib), 1 / fs)
spec = np.abs(np.fft.rfft(vib)) / len(vib) * 2

fig, ax = plt.subplots(1, 2, figsize=(13, 3.6))
ax[0].plot(t, vib, lw=0.6)
ax[0].set_title("Вибрация во времени: дефект не заметен")
ax[0].set_xlabel("время, с")
ax[1].plot(freqs, spec, lw=0.9)
for f0, lab in [(50, "база 50 Гц"), (120, "дефект 120 Гц")]:
    ax[1].annotate(lab, xy=(f0, spec[np.argmax(spec - f0)]),
                  xytext=(f0 + 25, 0.35), color="C3",

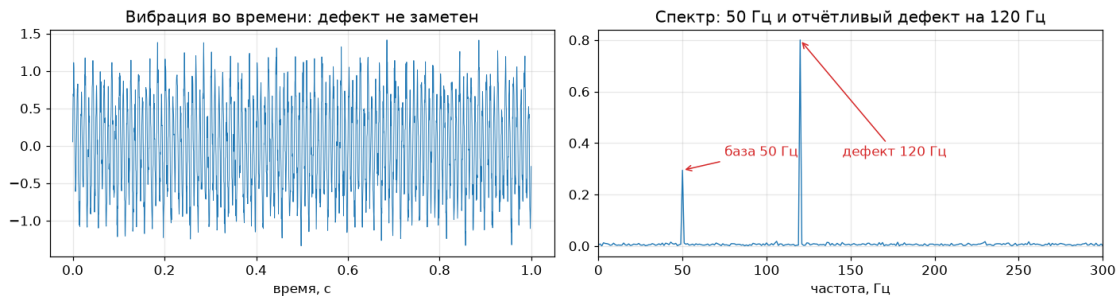
```

```

        arrowprops=dict(arrowstyle="->", color="C3"))
ax[1].set_xlim(0, 300)
ax[1].set_title("Спектр: 50 Гц и отчётливый дефект на 120 Гц")
ax[1].set_xlabel("частота, Гц")
fig.tight_layout()
save(fig, "fig_vib_spec.png")
plt.show()

```

сохранено: fig_vib_spec.png



1.9 8. Усечение сигнала: почему пики «расплываются»

На практике мы всегда берём **конечный** кусок сигнала. Это равносильно умножению на **прямоугольное окно**, и спектр «размазывается» — возникает **утечка**. Если частота не попадает точно в отсчёт спектра, вместо одной линии получается «холмик». Лечится сглаживающими **окнами** (например, Ханна) — это тема лекции 4.

```

[11]: fs, N = 100, 100
      t = np.arange(N) / fs
      f0 = 10.5

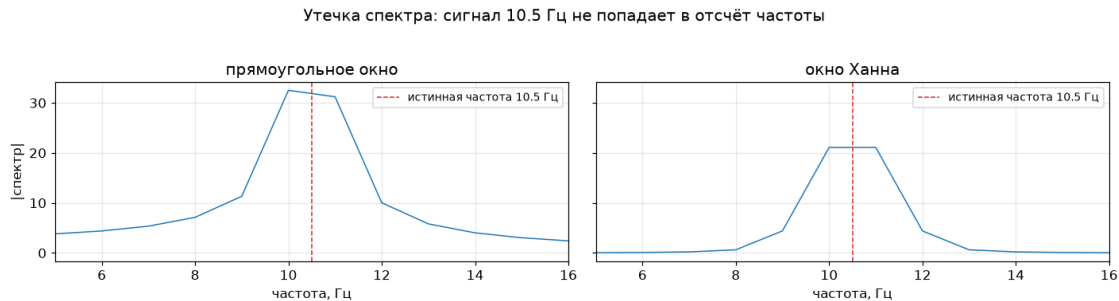
      rect = np.sin(2 * np.pi * f0 * t)
      hann = rect * np.hanning(N)
      freqs = np.fft.rfftfreq(N, 1 / fs)

      fig, ax = plt.subplots(1, 2, figsize=(13, 3.4), sharey=True)
      for a, sig, name in [(ax[0], rect, "прямоугольное окно"),
                           (ax[1], hann, "окно Ханна")]:
          a.plot(freqs, np.abs(np.fft.rfft(sig)), lw=1.0)
          a.axvline(f0, color="C3", ls="--", lw=1.0, label="истинная частота 10.5 Гц")
          a.set_xlim(5, 16)
          a.set_title(name)
          a.set_xlabel("частота, Гц")
          a.legend(fontsize=9)
      ax[0].set_ylabel("|спектр|")

```

```
fig.suptitle("Утечка спектра: сигнал 10.5 Гц не попадает в отсчёт частоты", y=1.03)
fig.tight_layout()
save(fig, "fig_window_leak.png")
plt.show()
```

сохранено: fig_window_leak.png



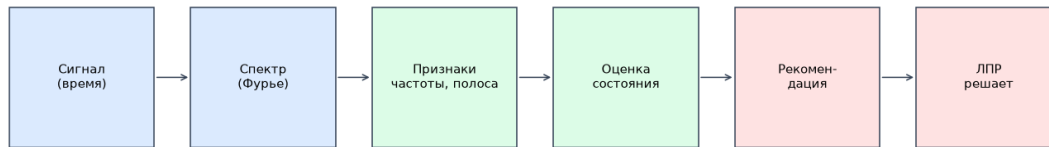
1.10 9. От спектра к решению: признаки для КСППР

Спектр — это ещё не решение. Чтобы он стал полезным в системе поддержки принятия решений, из него извлекают **признаки**: доминирующие частоты, гармоники, ширину полосы, энергию в полосах. Уже эти числа описывают **состояние** объекта и подсказывают **рекомендацию** человеку.

```
[12]: fig, ax = plt.subplots(figsize=(12.5, 3.0))
ax.axis("off")
boxes = ["Сигнал\n(время)", "Спектр\n(Фурье)", "Признаки\nчастоты, полоса",
        "Оценка\nсостояния", "Рекомен-\ndация", "ЛПР\nрешает"]
cols = ["#d8eafe", "#d8eafe", "#dcfce7", "#dcfce7", "#fee2e2", "#fee2e2"]
x = np.arange(len(boxes)) * 2.0
for xi, label, col in zip(x, boxes, cols):
    ax.add_patch(plt.Rectangle((xi, 0), 1.6, 1.0, facecolor=col,
                               edgecolor="#334155"))
    ax.text(xi + 0.8, 0.5, label, ha="center", va="center", fontsize=9.5)
for xi in x[:-1]:
    ax.annotate("", xy=(xi + 2.0, 0.5), xytext=(xi + 1.6, 0.5),
                arrowprops=dict(arrowstyle="->", color="#334155"))
ax.set_xlim(-0.3, x[-1] + 1.9)
ax.set_ylim(-0.3, 1.3)
ax.set_title("Спектр — источник признаков для КСППР")
fig.tight_layout()
save(fig, "fig_ksppr_feat.png")
plt.show()
```

сохранено: fig_ksppr_feat.png

Спектр — источник признаков для КСППР



11.11 11. Восстановление параметров сигнала по дискретным отсчётам

Минимальная задача цифровой обработки сигналов. Пусть у нас есть только N дискретных отсчётов $x[n]$ аналогового сигнала. Можно ли по ним восстановить сам сигнал и описать его несколькими числами? Да: если сигнал — одна синусоида, достаточно найти три её параметра — **амплитуду A , частоту f и фазу φ** — и мы знаем сигнал в любой момент времени. На этой «минимальной» задаче видна вся цепочка обработки: *дискретизация* → *оценка параметров* → *восстановление*.

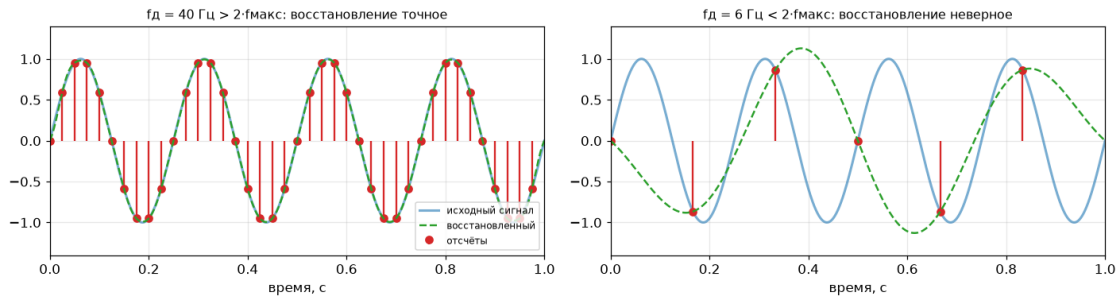
```
[13]: f0 = 4.0
t = np.linspace(0, 1, 2000)
x = np.sin(2 * np.pi * f0 * t)

def sinc_interp(t, ts, xs, fs):
    return sum(xs[k] * np.sinc(fs * (t - ts[k]))) for k in range(len(ts))

fig, ax = plt.subplots(1, 2, figsize=(13, 3.6))
for a, fs, title in [(ax[0], 40.0, "fd = 40 Гц > 2·fмакс: восстановление
↪точное"),
                    (ax[1], 6.0, "fd = 6 Гц < 2·fмакс: восстановление
↪неверное")]:
    ts = np.arange(0, 1, 1 / fs)
    xs = np.sin(2 * np.pi * f0 * ts)
    a.plot(t, x, color="C0", lw=2, alpha=0.6, label="исходный сигнал")
    a.stem(ts, xs, basefmt=" ", linefmt="C3", markerfmt="C3o", label="отсчёты")
    a.plot(t, sinc_interp(t, ts, xs, fs), "--", color="C2", lw=1.6,
           label="восстановленный")
    a.set_xlim(0, 1)
    a.set_ylim(-1.4, 1.4)
    a.set_title(title, fontsize=10.5)
    a.set_xlabel("время, с")
ax[0].legend(fontsize=8, loc="lower right")
fig.tight_layout()
save(fig, "fig_rec_sampling.png")
```

```
plt.show()
```

сохранено: fig_rec_sampling.png



Сколько отсчётов нужно? Теорема Котельникова требует $f_d \geq 2 \cdot f_{\text{макс}}$. Формально для одной синусоиды хватило бы трёх точек, но при шуме нужен запас. Если взять отсчёты слишком редко, восстановленная кривая уходит на **ложную (алиасную)** частоту — как на правой панели.

```
[14]: from scipy.optimize import curve_fit

A_t, f_t, phi_t = 1.3, 7.0, 0.6
fs, N = 100.0, 120
t = np.arange(N) / fs
rng = np.random.default_rng(2)
clean = A_t * np.sin(2 * np.pi * f_t * t + phi_t)
y = clean + 0.25 * rng.standard_normal(N)

def model(t, A, f, phi):
    return A * np.sin(2 * np.pi * f * t + phi)

popt, _ = curve_fit(model, t, y, p0=[1.0, 6.5, 0.5],
                    bounds=([0.0, 5.0, -np.pi], [3.0, 10.0, np.pi]))
tt = np.linspace(0, t[-1], 1000)

fig = plt.figure(figsize=(13, 3.8))
gs = fig.add_gridspec(1, 2, width_ratios=[2.1, 1], wspace=0.25)
a = fig.add_subplot(gs[0])
a.plot(tt, model(tt, *popt), color="C2", lw=1.8, label="подгонка МНК")
a.plot(tt, A_t * np.sin(2 * np.pi * f_t * tt + phi_t), color="C0", lw=1.2,
       alpha=0.6, label="истинный")
a.stem(t, y, basefmt=" ", linefmt="0.6", markerfmt="ko", label="шумные отсчёты")
a.set_title("МНК: подгоняем  $A \cdot \sin(2\pi f t + \phi)$  к отсчётам", fontsize=10.5)
a.set_xlabel("время, с")
```

```

a.legend(fontsize=9)

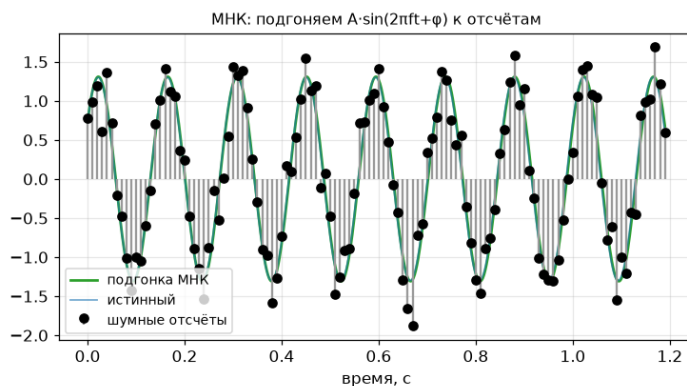
b = fig.add_subplot(gs[1])
b.axis("off")
rows = [("параметр", "истина", "оценка"),
        ("A", f"{A_t:.2f}", f"{popt[0]:.2f}"),
        ("f, Гц", f"{f_t:.2f}", f"{popt[1]:.3f}"),
        ("φ, рад", f"{phi_t:.2f}", f"{popt[2]:.3f}")]
for i, (c0, c1, c2) in enumerate(rows):
    yy = 0.9 - i * 0.22
    b.text(0.0, yy, c0, fontsize=11, weight="bold" if i == 0 else "normal")
    b.text(0.48, yy, c1, fontsize=11, weight="bold" if i == 0 else "normal")
    b.text(0.78, yy, c2, fontsize=11, weight="bold" if i == 0 else "normal")
b.set_xlim(-0.02, 1.05)
b.set_ylim(-0.05, 1.0)
b.set_title("Оценки параметров", fontsize=10.5)
fig.tight_layout()
save(fig, "fig_rec_lsfit.png")
plt.show()

```

/tmp/ipykernel_3929037/3887415665.py:44: UserWarning: This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.

fig.tight_layout()

сохранено: fig_rec_lsfit.png



Оценки параметров

параметр	истина	оценка
A	1.30	1.31
f, Гц	7.00	6.992
φ, рад	0.60	0.590

Метод наименьших квадратов. Задаём модель $A \cdot \sin(2\pi ft + \varphi)$ и подбираем параметры так, чтобы кривая шла как можно ближе ко всем отсчётам. Оценки почти совпадают с истиной (см. таблицу). Минус метода — нужна модель и стартовое приближение, а поиск дороже.

```

[15]: Nfft = N * 16
X = np.fft.rfft(y, Nfft)
freqs = np.fft.rfftfreq(Nfft, 1 / fs)
amp = np.abs(X) / N * 2

```

```

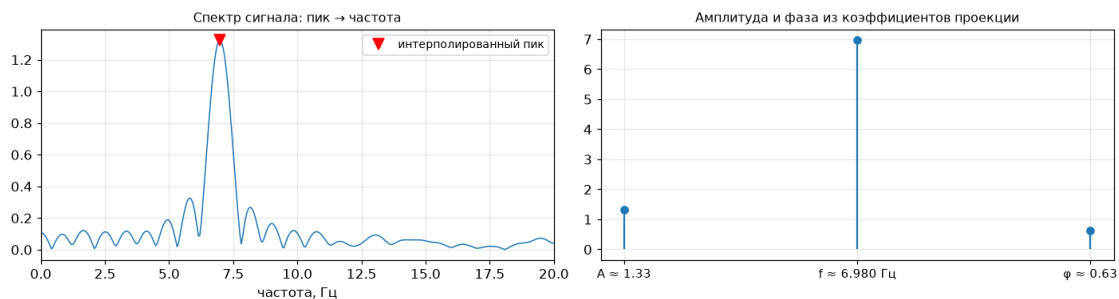
k = int(np.argmax(amp))
d = 0.5 * (amp[k - 1] - amp[k + 1]) / (amp[k - 1] - 2 * amp[k] + amp[k + 1])
f_est = (k + d) * fs / Nfft

theta = 2 * np.pi * f_est * t
a_c = 2 / N * np.sum(y * np.sin(theta))
b_c = 2 / N * np.sum(y * np.cos(theta))
A_est = np.hypot(a_c, b_c)
phi_est = np.arctan2(b_c, a_c)

fig, ax = plt.subplots(1, 2, figsize=(13, 3.6))
ax[0].plot(freqs, amp, color="C0", lw=1.0)
ax[0].plot(f_est, amp[k], "rv", ms=9, label="интерполированный пик")
ax[0].set_xlim(0, 20)
ax[0].set_title("Спектр сигнала: пик → частота", fontsize=10.5)
ax[0].set_xlabel("частота, Гц")
ax[0].legend(fontsize=9)
ax[1].stem([0, 1, 2], [A_est, f_est, phi_est], basefmt=" ")
ax[1].set_xticks([0, 1, 2])
ax[1].set_xticklabels([f"A ≈ {A_est:.2f}", f"f ≈ {f_est:.3f} Гц",
                      f"φ ≈ {phi_est:.2f}"], fontsize=10)
ax[1].set_title("Амплитуда и фаза из коэффициентов проекции", fontsize=10.5)
fig.tight_layout()
save(fig, "fig_rec_fft.png")
plt.show()

```

сохранено: fig_rec_fft.png



Спектральный метод. Считаем спектр, находим пик — это частота. Параболическая интерполяция соседних отсчётов уточняет частоту между бинами, а амплитуду и фазу даёт проекция сигнала на $\sin$ и $\cos$ найденной частоты. Метод быстрый и не требует модели, но чувствителен к утечке.

```

[16]: snr_grid = np.arange(0, 41, 4)
      trials = 40
      err_ls, err_fft = [], []

```

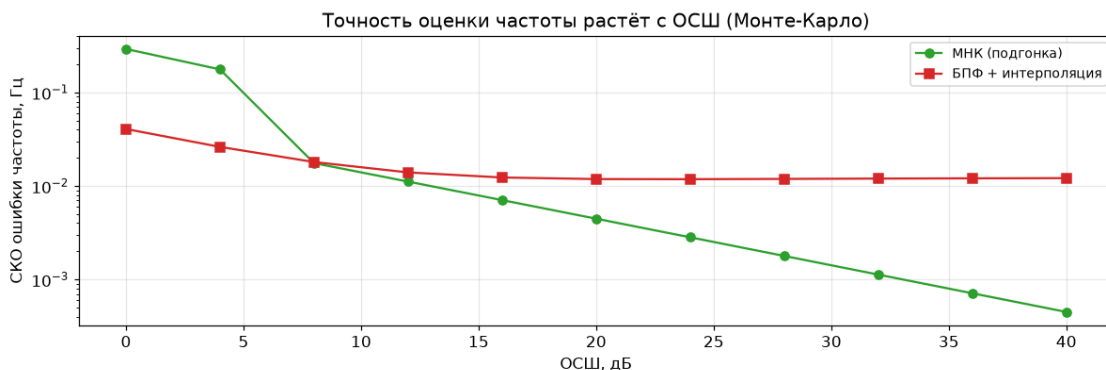
```

for snr_db in snr_grid:
    e_ls, e_fft = [], []
    for it in range(trials):
        rng = np.random.default_rng(1000 + it)
        c = A_t * np.sin(2 * np.pi * f_t * t + phi_t)
        noise = rng.standard_normal(N)
        noise *= np.std(c) / np.std(noise) / 10 ** (snr_db / 20)
        yy = c + noise
        try:
            p, _ = curve_fit(model, t, yy, p0=[1.0, 6.5, 0.5],
                             bounds=([0.0, 5.0, -np.pi], [3.0, 10.0, np.pi]))
            e_ls.append(abs(p[1] - f_t))
        except RuntimeError:
            pass
        Xx = np.fft.rfft(yy, N * 16)
        aa = np.abs(Xx) / N * 2
        kk = int(np.argmax(aa))
        dd = 0.5 * (aa[kk - 1] - aa[kk + 1]) / (aa[kk - 1] - 2 * aa[kk] + aa[kk +
↵ 1])
        e_fft.append(abs((kk + dd) * fs / (N * 16) - f_t))
    err_ls.append(np.sqrt(np.mean(np.square(e_ls))))
    err_fft.append(np.sqrt(np.mean(np.square(e_fft))))

fig, ax = plt.subplots(figsize=(11, 3.8))
ax.semilogy(snr_grid, err_ls, "o-", color="C2", label="МНК (подгонка)")
ax.semilogy(snr_grid, err_fft, "s-", color="C3", label="БПФ + интерполяция")
ax.set_xlabel("ОСШ, дБ")
ax.set_ylabel("СКО ошибки частоты, Гц")
ax.set_title("Точность оценки частоты растёт с ОСШ (Монте-Карло)")
ax.legend(fontsize=9)
fig.tight_layout()
save(fig, "fig_rec_error.png")
plt.show()

```

сохранено: fig_rec_error.png



Точность. Чем выше ОСШ, тем точнее оценка. МНК выходит на теоретический предел точности, а спектральный метод быстрее, но упирается в систематическую ошибку интерполяции. Выбор — компромисс точность/скорость.

Зачем это в курсе. Это «тест-драйв» тракта АЦП: проверка разрядности и частоты дискретизации. И прямой мост к проекту — частота вибрации, фаза RFID, задержка GPS. Все дальнейшие темы — обобщения этой задачи: много частот (ДПФ, БПФ), шум и оценивание, изменяющиеся параметры (фильтр Калмана).

1.12 Сквозной проект: задание №2

Задание после лекции 2 (для вашего проекта):

1. Для сигнала своего проекта найти **спектр** (амплитудный и фазовый).
2. Выделить **три доминирующие частоты** и дать им физическую интерпретацию.
3. Оценить **ширину полосы** сигнала.
4. Если в канале есть несущая — определить её частоту и оценить полосы боковых.
5. Проверить, что выбранная частота дискретизации не даёт **утечки** (или применить окно).

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

1.13 Итоги лекции 2

- Любой сигнал собирается из **синусоид**; спектр — «список деталей».
- **Ряд Фурье** даёт гармоники; у меандра — только нечётные, с амплитудой $1/k$.
- Полный спектр **комплексный**: амплитуда **и** фаза; без фазы форма теряется.
- Свойства: линейность, сдвиг $\leftrightarrow$ фаза, **короткий импульс $\leftrightarrow$ широкий спектр**.
- **Модуляция** = перенос спектра: АМ даёт две боковые, ЧМ — расходящиеся.
- Конечный кусок сигнала даёт **утечку**; спасают окна (лекция 4).
- **Минимальная задача ЦОС**: по дискретным отсчётам восстанавливаются параметры сигнала (A, f, φ) — методом **МНК** (точно) или **БПФ** (быстро).

Дальше: дискретное преобразование Фурье (ДПФ) — как посчитать спектр точно и быстро; затем БПФ и окна.

1.14 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.