

lecture04

October 2, 2026

1 Лекция 4. БПФ и практический спектральный анализ

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

Направленность: искусственный интеллект и системная аналитика

На лекции 3 мы получили **ДПФ** — точный спектр из N отсчётов, но за $O(N^2)$ операций. Сегодня закрываем два практических вопроса:

1. **Как считать быстро** — алгоритм **БПФ** (быстрое преобразование Фурье): «разделяй и властвуй», бабочка, бит-реверс, $O(N \log N)$.
2. **Как считать правильно** — конечная запись и шум искажают спектр: **окна, разрешение, утечка, усреднение (метод Уэлча), спектрограмма**.

Итог лекции — рабочий рецепт: как из сырых отсчётов получить спектр, которому можно верить.

1.1 Цели лекции

К концу лекции вы сможете:

1. Объяснить идею БПФ «разделяй и властвуй» и записать «бабочку» radix-2.
2. Понять роль **бит-реверса** и почему БПФ даёт $O(N \log N)$.
3. Выбирать **окно** под задачу (утечка против разрешения).
4. Объяснить $\Delta f = 1/T$ и почему **zero-padding не улучшает разрешение**.
5. Усреднять периодограммы (**метод Уэлча**) и читать **PSD**.
6. Строить **спектрограмму** и понимать компромисс время–частота.
7. Считать всё это в Python (`scipy.signal`).

1.2 1. Зачем БПФ: ещё раз о цене

Прямое ДПФ — это умножение матрицы $N \times N$ на вектор:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \implies O(N^2) \text{ операций.}$$

Для $N = 10^6$ это 10^{12} — неприемлемо. **БПФ** (Cooley–Tukey, 1965) считает **тот же** результат за $O(N \log_2 N)$: для 10^6 это около $2 \cdot 10^7$, то есть в **десятки тысяч раз** быстрее. Посмотрим на числа и на измерение.

```
[1]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from time import perf_counter
from scipy import signal
from scipy.fft import fft, ifft

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)

def db(x):
    return 20 * np.log10(np.maximum(np.abs(x), 1e-12))

print("numpy", np.__version__, "| scipy", __import__("scipy").__version__)
```

numpy 2.5.3 | scipy 1.18.1

```
[2]: def dft_slow(x):
    N = len(x)
    out = np.zeros(N, dtype=complex)
    for k in range(N):
        s = 0.0
        for n in range(N):
            s += x[n] * np.exp(-2j * np.pi * k * n / N)
        out[k] = s
    return out

rng = np.random.default_rng(0)
Ns = 2 ** np.arange(3, 12)          # 8 ... 2048
t_dft, t_fft = [], []
for N in Ns:
```

```

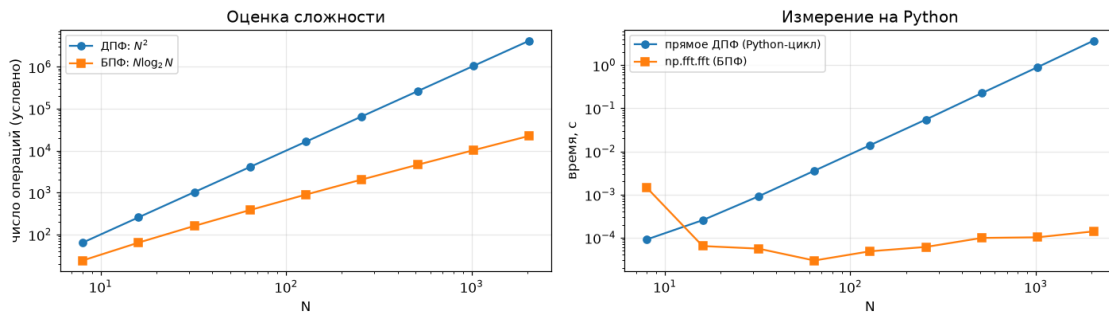
x = rng.standard_normal(N) + 1j * rng.standard_normal(N)
t0 = perf_counter(); dft_slow(x); t_dft.append(perf_counter() - t0)
t0 = perf_counter(); fft(x); t_fft.append(perf_counter() - t0)

ops_dft = Ns.astype(float) ** 2
ops_fft = Ns.astype(float) * np.log2(Ns)

fig, ax = plt.subplots(1, 2, figsize=(13, 3.8))
ax[0].loglog(Ns, ops_dft, "o-", label=r"ДПФ: $N^2$")
ax[0].loglog(Ns, ops_fft, "s-", label=r"БПФ: $N \log_2 N$")
ax[0].set_xlabel("N")
ax[0].set_ylabel("число операций (условно)")
ax[0].set_title("Оценка сложности")
ax[0].legend(fontsize=9)
ax[1].loglog(Ns, t_dft, "o-", label="прямое ДПФ (Python-цикл)")
ax[1].loglog(Ns, t_fft, "s-", label="np.fft.fft (БПФ)")
ax[1].set_xlabel("N")
ax[1].set_ylabel("время, с")
ax[1].set_title("Измерение на Python")
ax[1].legend(fontsize=9)
fig.tight_layout()
save(fig, "fig_fft_cost.png")
plt.show()

```

сохранено: fig_fft_cost.png



1.3 2. Идея БПФ: разделяй и властвуй

Пусть N чётно. Разделим сумму на **чётные** и **нечётные** отсчёты и учтём, что $w_N^{2kn} = w_{N/2}^{kn}$:

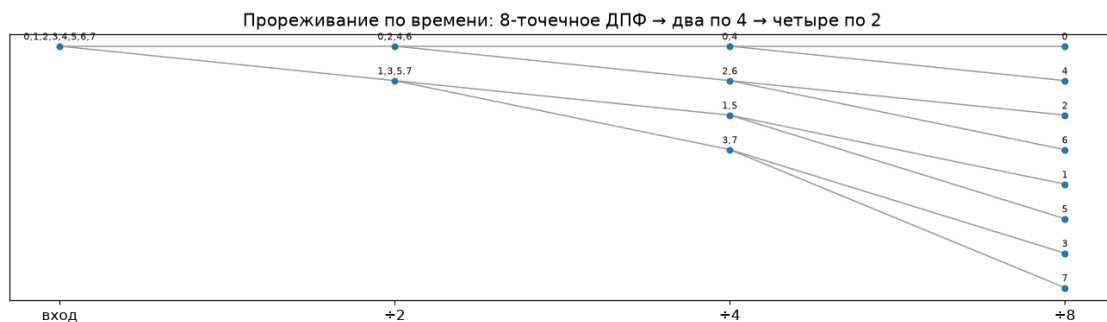
$$X[k] = \underbrace{\sum_{r=0}^{N/2-1} x[2r] W_{N/2}^{kr}}_{E[k]} + W_N^k \underbrace{\sum_{r=0}^{N/2-1} x[2r+1] W_{N/2}^{kr}}_{O[k]}.$$

То есть **ДПФ длины N свелось к двум ДПФ длины N/2**, а потом рекурсивно — к $\log_2 N$ уровням. Это и есть **прореживание по времени** (decimation in time, DIT).

```
[3]: # Дерево разбиения 8 отсчётов на чётные/нечётные
N = 8
levels = [[list(range(N))]]
cur = [list(range(N))]
while len(cur[0]) > 1:
    nxt = []
    for blk in cur:
        nxt += [blk[0::2], blk[1::2]]
    cur = nxt
    levels.append([b[:] for b in cur])

fig, ax = plt.subplots(figsize=(12, 3.6))
for li, blks in enumerate(levels):
    for bi, blk in enumerate(blks):
        y = -bi
        ax.scatter([li], [y], s=18, color="C0", zorder=3)
        ax.text(li, y + 0.12, ",".join(map(str, blk)), ha="center",
                va="bottom", fontsize=7)
        if li + 1 < len(levels):
            for child in (blk[0::2], blk[1::2]):
                cbi = levels[li + 1].index(child)
                ax.plot([li, li + 1], [y, -cbi], color="0.6", lw=1)
ax.set_xticks(range(len(levels)))
ax.set_xticklabels(["вход", "÷2", "÷4", "÷8"])
ax.set_yticks([])
ax.set_title("Прореживание по времени: 8-точечное ДПФ → два по 4 → четыре по 2")
ax.grid(False)
fig.tight_layout()
save(fig, "fig_fft_dit.png")
plt.show()
```

сохранено: fig_fft_dit.png



1.3.1 Бабочка radix-2

Два ДПФ половинной длины объединяются одной операцией — **бабочкой**:

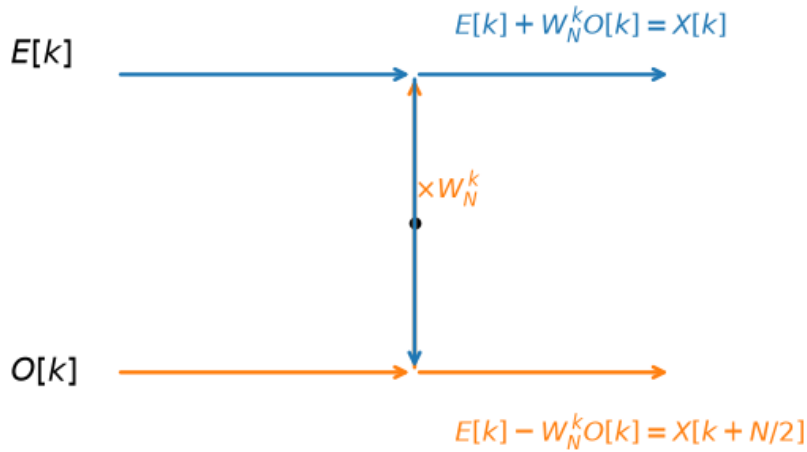
$$X[k] = E[k] + W_N^k O[k], \quad X[k + N/2] = E[k] - W_N^k O[k], \quad k = 0, \dots, N/2 - 1.$$

Одно комплексное умножение и два сложения на пару, вместо «в лоб». Именно эти бабочки, повторённые $\log_2 N$ раз, дают выигрыш.

```
[4]: fig, ax = plt.subplots(figsize=(6.6, 3.4))
ax.axis("off")
ax.text(0.05, 0.82, r"$E[k]$", fontsize=13, ha="left")
ax.text(0.05, 0.18, r"$O[k]$", fontsize=13, ha="left")
ax.annotate("", xy=(0.55, 0.80), xytext=(0.18, 0.80),
            arrowprops=dict(arrowstyle="->", lw=1.6, color="C0"))
ax.annotate("", xy=(0.55, 0.20), xytext=(0.18, 0.20),
            arrowprops=dict(arrowstyle="->", lw=1.6, color="C1"))
ax.annotate("", xy=(0.87, 0.80), xytext=(0.55, 0.80),
            arrowprops=dict(arrowstyle="->", lw=1.6, color="C0"))
ax.annotate("", xy=(0.87, 0.20), xytext=(0.55, 0.20),
            arrowprops=dict(arrowstyle="->", lw=1.6, color="C1"))
ax.annotate("", xy=(0.55, 0.80), xytext=(0.55, 0.20),
            arrowprops=dict(arrowstyle="->", lw=1.6, color="C1"))
ax.annotate("", xy=(0.55, 0.20), xytext=(0.55, 0.80),
            arrowprops=dict(arrowstyle="->", lw=1.6, color="C0"))
ax.plot(0.55, 0.5, "ko", ms=4)
ax.text(0.55, 0.55, r"$\times W_N^{\setminus,k}$", color="C1", ha="left")
ax.text(0.60, 0.88, r"$E[k] + W_N^{\setminus,k} O[k] = X[k]$", color="C0")
ax.text(0.60, 0.06, r"$E[k] - W_N^{\setminus,k} O[k] = X[k+N/2]$", color="C1")
ax.set_xlim(0, 1.35); ax.set_ylim(0, 1)
ax.set_title("Бабочка radix-2 (ДПФ по основанию 2)")
fig.tight_layout()
save(fig, "fig_butterfly.png")
plt.show()
```

сохранено: fig_butterfly.png

Бабочка radix-2 (ДПФ по основанию 2)



1.3.2 Полный граф 8-точечного БПФ и бит-реверс

Если прореживать по времени до конца, на **входе** отсчёты оказываются в **бит-реверсивном** порядке (индекс читается справа налево): 0, 4, 2, 6, 1, 5, 3, 7. На выходе — обычный порядок $X[0..7]$. Ниже — граф и таблица перестановки.

```
[5]: N = 8
nst = int(np.log2(N))

def bitrev(i, bits):
    return int(f"{i:0{bits}b}"[::-1], 2)

rev = [bitrev(i, nst) for i in range(N)]

# стадии бабочек DIT (вход в бит-реверсном порядке)
stages = []
m = 2
while m <= N:
    st = []
    for k0 in range(0, N, m):
        for j in range(m // 2):
            st.append((k0 + j, k0 + j + m // 2, np.exp(-2j * np.pi * j / m)))
    stages.append(st)
    m *= 2

fig, ax = plt.subplots(figsize=(12, 4.4))
```

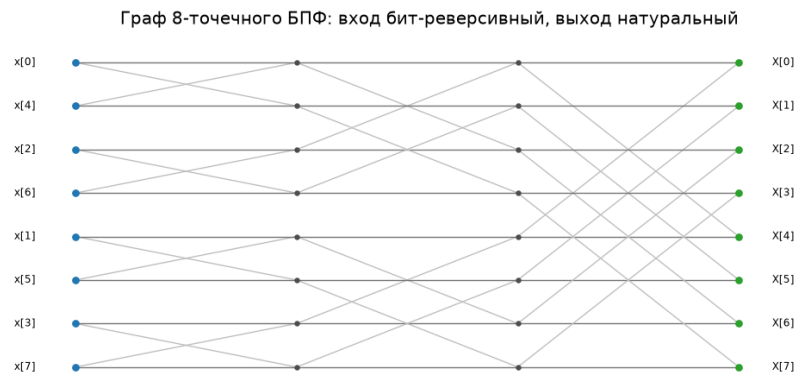
```

ax.axis("off")
for p in range(N):
    y = -p
    ax.scatter([0], [y], s=22, color="C0", zorder=3)
    ax.text(-0.18, y, f"x[{rev[p]}]", ha="right", va="center", fontsize=8)
    ax.scatter([nst], [y], s=22, color="C2", zorder=3)
    ax.text(nst + 0.15, y, f"x[{p}]", ha="left", va="center", fontsize=8)
for ci, st in enumerate(stages):
    for (p, q, w) in st:
        ax.plot([ci, ci + 1], [-p, -q], color="0.75", lw=0.9)
        ax.plot([ci, ci + 1], [-q, -p], color="0.75", lw=0.9)
        ax.plot([ci, ci + 1], [-p, -p], color="0.45", lw=0.9)
        ax.plot([ci, ci + 1], [-q, -q], color="0.45", lw=0.9)
for ci in range(1, nst):
    for p in range(N):
        ax.scatter([ci], [-p], s=10, color="0.3", zorder=3)
ax.set_xlim(-0.9, nst + 1.1)
ax.set_ylim(-N + 0.3, 0.7)
ax.set_title("Граф 8-точечного БПФ: вход бит-реверсивный, выход натуральный")
fig.tight_layout()
save(fig, "fig_fft_flow.png")
plt.show()

print("бит-реверс:", rev)
table = np.array([[i, f"{i:03b}", f"{bitrev(i,3):03b}", bitrev(i, 3)]
                  for i in range(N)])
print(table)

```

сохранено: fig_fft_flow.png



```

бит-реверс: [0, 4, 2, 6, 1, 5, 3, 7]
[ ['0' '000' '000' '0']
  ['1' '001' '100' '4']
  ['2' '010' '010' '2']

```

```
['3' '011' '110' '6']
['4' '100' '001' '1']
['5' '101' '101' '5']
['6' '110' '011' '3']
['7' '111' '111' '7']]
```

```
[6]: fig, ax = plt.subplots(figsize=(11, 3.0))
ax.axis("off")
cells_txt = [[f"{i}", f"{i:03b}", f"{bitrev(i,3):03b}", f"{rev[i]}"]
              for i in range(N)]
col_labels = ["n", "двоичный", "реверс", "новый индекс"]
tbl = ax.table(cellText=cells_txt, colLabels=col_labels, loc="center",
               cellLoc="center")
tbl.auto_set_font_size(False)
tbl.set_fontsize(10)
tbl.scale(1, 1.5)
ax.set_title("Бит-реверс индексов для N=8: 0,4,2,6,1,5,3,7")
fig.tight_layout()
save(fig, "fig_bitrev.png")
plt.show()
```

сохранено: fig_bitrev.png

Бит-реверс индексов для N=8: 0,4,2,6,1,5,3,7			
n	двоичный	реверс	новый индекс
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

1.3.3 Варианты и практика

- **radix-2** — деление пополам; проще всего, $N = 2^m$.
- **radix-4** — деление на 4; меньше умножений.
- **split-radix / mixed-radix** — ещё быстрее, работают для составных N .
- **Коварный случай:** простое большое N (например, 1000003) в лоб radix-2 не ляжет — на практике N округляют до степени двойки или используют алгоритмы Блустейна/prime-factor. `numpy.fft/scipy.fft` (`pocketfft`) прячут это внутри.
- **Реальный сигнал:** хватает $N/2+1$ точек (`rfft`) — вдвое дешевле.
- Зрелые библиотеки: **FFTW**, **pocketfft** (NumPy/SciPy), **MKL**, **cuFFT** (GPU).

1.4 3. Конечная запись: почему спектр «размазывается»

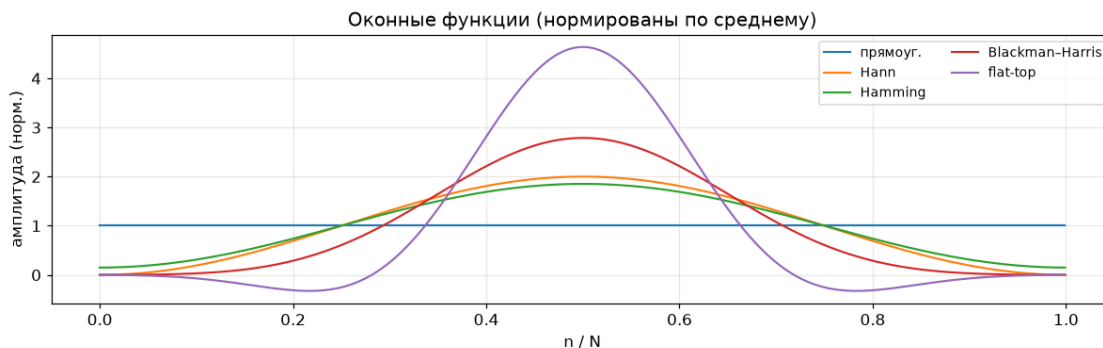
Всё, что мы считаем, — это ДПФ **конечного** куска сигнала. Формально мы дополняем сигнал нулями до бесконечности, то есть умножаем на **прямоугольное окно** шириной T . Умножение во времени = **свёртка в спектре** с sinc окна. Отсюда **утечка**: даже чистый тон превращается в «холмик» с боковыми лепестками.

Окно — плавное обрезание записи. Оно снижает боковые лепестки, но **расширяет главный** — это главный компромисс.

```
[7]: N = 1024
n = np.arange(N)
names = ["boxcar", "hann", "hamming", "blackmanharris", "flattop"]
labels = ["прямоуг.", "Hann", "Hamming", "Blackman-Harris", "flat-top"]

fig, ax = plt.subplots(figsize=(11, 3.6))
for name, lab in zip(names, labels):
    w = signal.get_window(name, N)
    ax.plot(n / N, w / w.mean(), lw=1.4, label=lab)
ax.set_xlabel("n / N")
ax.set_ylabel("амплитуда (норм.)")
ax.set_title("Оконные функции (нормированы по среднему)")
ax.legend(fontsize=9, ncol=2)
fig.tight_layout()
save(fig, "fig_windows_time.png")
plt.show()
```

сохранено: fig_windows_time.png



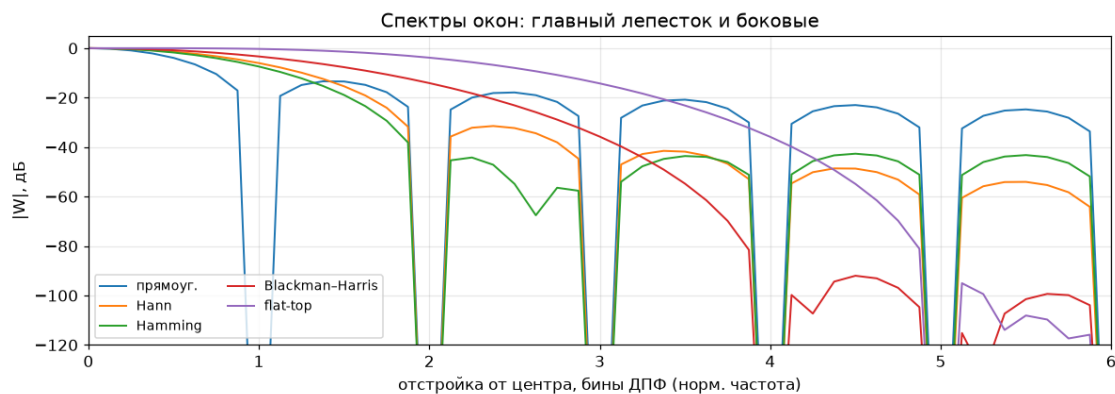
```
[8]: N = 1024
nfft = 8192
fig, ax = plt.subplots(figsize=(11, 4.0))
for name, lab in zip(names, labels):
    w = signal.get_window(name, N)
    W = np.fft.fft(w / w.sum(), nfft)
```

```

f = np.fft.fftfreq(nfft)
ax.plot(f[: nfft // 2] * N, db(W)[: nfft // 2], lw=1.3, label=lab)
ax.set_xlim(0, 6)
ax.set_ylim(-120, 5)
ax.set_xlabel("отстройка от центра, бины ДПФ (норм. частота)")
ax.set_ylabel("|W|, дБ")
ax.set_title("Спектры окон: главный лепесток и боковые")
ax.legend(fontsize=9, ncol=2)
fig.tight_layout()
save(fig, "fig_windows_spectra.png")
plt.show()

```

сохранено: fig_windows_spectra.png



```

[9]: N = 4096
NF = 1 << 16
rec = []
for name, lab in zip(names, labels):
    w = signal.get_window(name, N)
    W = np.abs(np.fft.rfft(w, NF))
    W /= W.max()
    d = np.fft.rfftfreq(NF)
    # первый ноль после главного лепестка
    i3 = np.where(W > 10 ** (-3 / 20))[0]
    j = int(i3.max())
    while j + 1 < len(W) and W[j + 1] < W[j]:
        j += 1
    # ширина главного лепестка «между нулями» (в бинах)
    main_bins = 2 * d[j] * N
    # максимум боковых лепестков правее первого нуля
    seg = W[j:]
    loc = np.where((seg[1:-1] > seg[:-2]) & (seg[1:-1] >= seg[2:]))[0] + 1
    sl = 20 * np.log10(seg[loc].max() + 1e-12) if len(loc) else -300.0

```

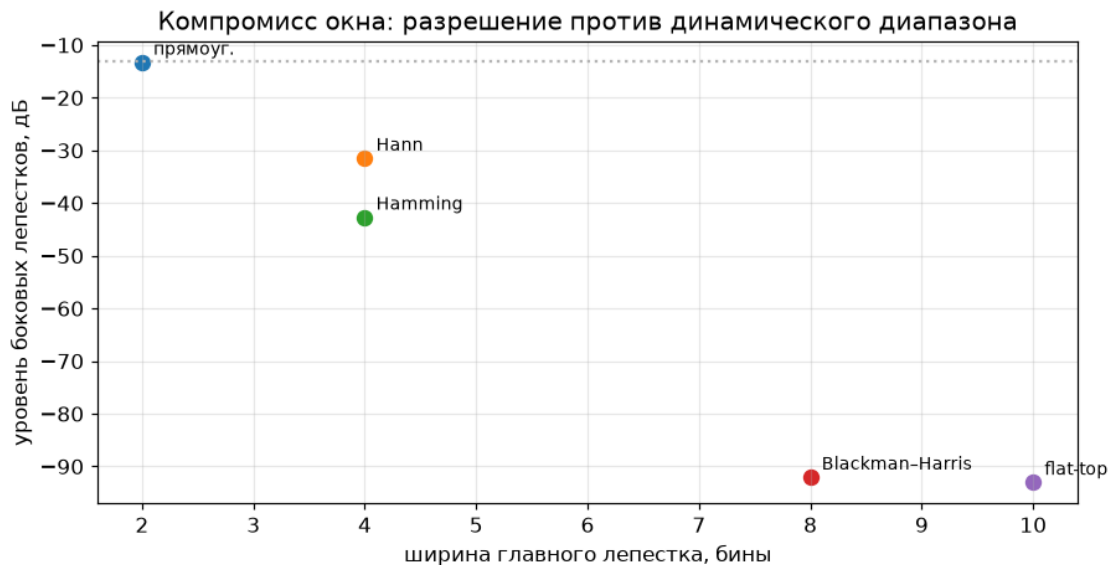
```

rec.append((lab, main_bins, sl))

fig, ax = plt.subplots(figsize=(8.4, 4.4))
for lab, ml, sl in rec:
    ax.scatter(ml, sl, s=60)
    ax.annotate(lab, (ml, sl), textcoords="offset points", xytext=(6, 4),
                fontsize=9)
ax.set_xlabel("ширина главного лепестка, бины")
ax.set_ylabel("уровень боковых лепестков, дБ")
ax.set_title("Компромисс окна: разрешение против динамического диапазона")
ax.axhline(-13, color="0.7", ls=":")
fig.tight_layout()
save(fig, "fig_window_tradeoff.png")
plt.show()
for lab, ml, sl in rec:
    print(f"{lab:18s} главный лепесток ≈ {ml:4.1f} бина, боковые ≈ {sl:5.1f} дБ")

```

сохранено: fig_window_tradeoff.png



прямоуг.	главный лепесток ≈ 2.0 бина, боковые ≈ -13.3 дБ
Hann	главный лепесток ≈ 4.0 бина, боковые ≈ -31.5 дБ
Hamming	главный лепесток ≈ 4.0 бина, боковые ≈ -42.7 дБ
Blackman-Harris	главный лепесток ≈ 8.0 бина, боковые ≈ -92.0 дБ
flat-top	главный лепесток ≈ 10.0 бина, боковые ≈ -93.0 дБ

1.4.1 Паспорт окна (запомнить)

- **Прямоугольное** (boxcar): максимальное разрешение, но боковые всего -13 дБ.
- **Hann**: боковые -31 дБ, главный лепесток 4 бина — «рабочая лошадка».

- **Hamming**: боковые -43 дБ.
- **Blackman-Harris**: боковые -74 дБ — для слабых компонент рядом с сильной.
- **flat-top**: очень широкая вершина — точная **амплитуда** (калибровка), плохое разрешение.

Кроме формы важны коэффициенты: **coherent gain** (нормировка амплитуды) и **ENBW** (эквивалентная шумовая полоса — нормировка мощности/PSD). Их считает `scipy.signal.windows` / учитывает `scipy.signal.welch`.

```
[10]: N = 1024
print(f"{'окно':16s} {'coherent gain':>14s} {'ENBW, бины':>12s}")
for name, lab in zip(names, labels):
    w = signal.get_window(name, N)
    cg = w.mean()
    enbw = N * np.sum(w ** 2) / np.sum(w) ** 2
    print(f"{'lab':16s} {'cg':14.4f} {'enbw':12.3f}")
```

окно	coherent gain	ENBW, бины
прямоуг.	1.0000	1.000
Hann	0.5000	1.500
Hamming	0.5400	1.363
Blackman-Harris	0.3588	2.004
flat-top	0.2156	3.770

1.5 4. Утечка и как окно её лечит

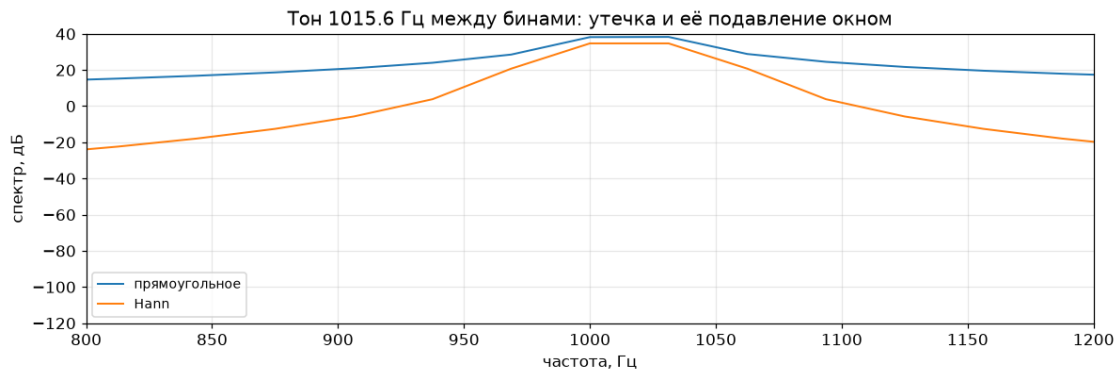
Возьмём тон **между бинами** и посмотрим спектр с прямоугольным окном и с Ханн: линия «размазывается», но у Ханн боковые лепестки падают гораздо быстрее.

```
[11]: fs = 8000.0
N = 256
n = np.arange(N)
f0 = 1015.625 # ровно между бинами (Δf = 31.25 Гц)
x = np.cos(2 * np.pi * f0 * n / fs)
f = np.fft.rfftfreq(N, 1 / fs)

fig, ax = plt.subplots(figsize=(11, 3.8))
for name, lab, c in [("boxcar", "прямоугольное", "C0"),
                    ("hann", "Hann", "C1")]:
    X = np.fft.rfft(x * signal.get_window(name, N))
    ax.plot(f, db(X), lw=1.3, color=c, label=lab)
ax.set_xlim(800, 1200)
ax.set_ylim(-120, 40)
ax.set_xlabel("частота, Гц")
ax.set_ylabel("спектр, дБ")
ax.set_title(f"Тон {f0:.1f} Гц между бинами: утечка и её подавление окном")
ax.legend(fontsize=9)
fig.tight_layout()
```

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

сохранено: fig_window_leak.png



1.6 5. Разрешение: сколько записей нужно

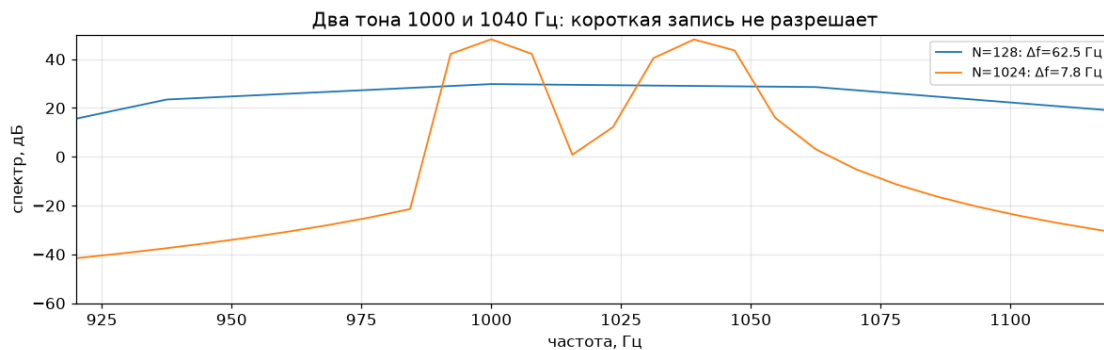
Разрешение по частоте задаёт длительность записи, а не частота дискретизации:

$$\Delta f = \frac{f_d}{N} = \frac{1}{T}.$$

Два тона различимы (критерий Рэлея), если расстояние между ними $> \Delta f$ (для окна Hann практического разрешения нужно $\approx 2 \Delta f$).

```
[12]: fs = 8000.0
f1, f2 = 1000.0, 1040.0
for N in (128, 1024):
    n = np.arange(N)
    x = np.cos(2 * np.pi * f1 * n / fs) + np.cos(2 * np.pi * f2 * n / fs)
    f = np.fft.rfftfreq(N, 1 / fs)
    X = np.fft.rfft(x * signal.get_window("hann", N))
    plt.plot(f, db(X), lw=1.2, label=f"N={N}: Δf={fs/N:.1f} Гц")
plt.xlim(920, 1120)
plt.ylim(-60, 50)
plt.xlabel("частота, Гц")
plt.ylabel("спектр, дБ")
plt.title("Два тона 1000 и 1040 Гц: короткая запись не разрешает")
plt.legend(fontsize=9)
plt.tight_layout()
plt.savefig(FIG / "fig_resolution.png", bbox_inches="tight")
print("сохранено: fig_resolution.png")
plt.show()
```

сохранено: fig_resolution.png



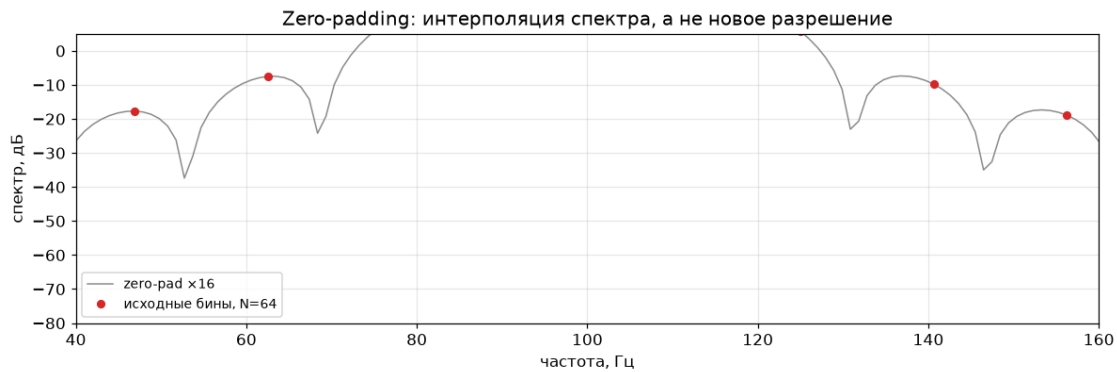
1.6.1 Zero-padding не создаёт разрешение

Дополнение нулями даёт **плотную** сетку частот — мы просто видим ту же самую спектральную функцию **между** старыми бинами. Разрешить близкие частоты это не помогает; для интерполяции — помогает.

```
[13]: fs, N = 1000.0, 64
n = np.arange(N)
f0 = 100.0
x = np.cos(2 * np.pi * f0 * n / fs)
f = np.fft.rfftfreq(N, 1 / fs)
X = np.fft.rfft(x * signal.get_window("hann", N))
fz = np.fft.rfftfreq(16 * N, 1 / fs)
Xz = np.fft.rfft(x * signal.get_window("hann", N), 16 * N)

fig, ax = plt.subplots(figsize=(11, 3.8))
ax.plot(fz, db(Xz), "-", color="0.55", lw=1.0, label="zero-pad ×16")
ax.plot(f, db(X), "o", color="C3", ms=5, label="исходные бины, N=64")
ax.set_xlim(40, 160)
ax.set_ylim(-80, 5)
ax.set_xlabel("частота, Гц")
ax.set_ylabel("спектр, дБ")
ax.set_title("Zero-padding: интерполяция спектра, а не новое разрешение")
ax.legend(fontsize=9)
fig.tight_layout()
save(fig, "fig_zeropad.png")
plt.show()
```

сохранено: fig_zeropad.png



1.7 6. Усреднение: метод Уэлча (Welch)

Одна периодограмма

$$\hat{P}(f) = |X(f)|^2$$

сильно **шумит**: дисперсия оценки не падает с ростом N . Идея Уэлча:

1. разрезать запись на **сегменты** (можно с **перекрытием**);
2. к каждому применить **окно**;
3. посчитать $|\text{БПФ}|^2$;
4. **усреднить** по сегментам.

Это **несмещённая** оценка с меньшей дисперсией: шум «сглаживается», а реальные линии остаются. Ценой — разрешение по длине сегмента.

```
[14]: fs = 1000.0
N = 8192
rng = np.random.default_rng(2)
n = np.arange(N)
x = (1.0 * np.sin(2 * np.pi * 50 * n / fs)
      + 0.5 * np.sin(2 * np.pi * 120 * n / fs)
      + rng.standard_normal(N))

fp, Pp = signal.periodogram(x, fs, window="boxcar", scaling="density")
fw, Pw = signal.welch(x, fs, window="hann", nperseg=256, noverlap=128,
                      scaling="density")

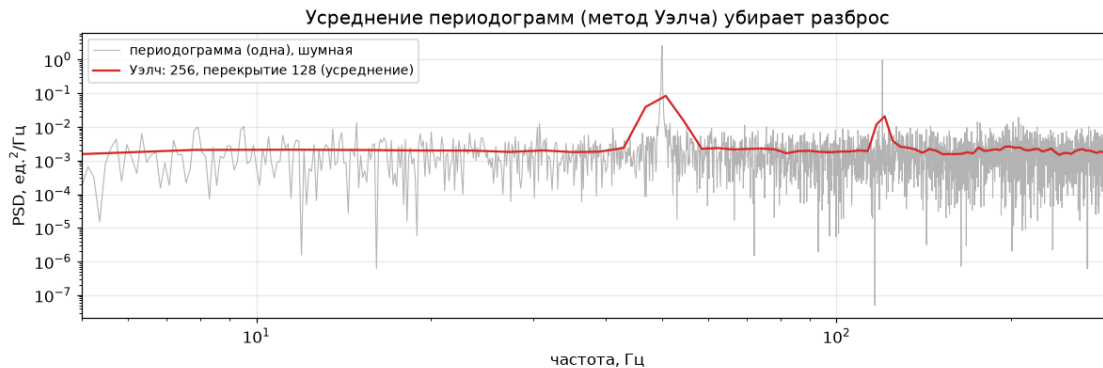
fig, ax = plt.subplots(figsize=(11, 3.8))
ax.loglog(fp[1:], Pp[1:], color="0.7", lw=0.7,
          label="периодограмма (одна), шумная")
ax.loglog(fw[1:], Pw[1:], color="C3", lw=1.5,
          label="Уэлч: 256, перекрытие 128 (усреднение)")
ax.set_xlim(5, 300)
```

```

ax.set_xlabel("частота, Гц")
ax.set_ylabel(r"PSD, ед.$^2$/Гц")
ax.set_title("Усреднение периодограмм (метод Уэлча) убирает разброс")
ax.legend(fontsize=9)
fig.tight_layout()
save(fig, "fig_welch.png")
plt.show()

```

сохранено: fig_welch.png



1.7.1 Что такое PSD

Спектральная плотность мощности (power spectral density, PSD) — мощность на **герц**. Она полезна, когда сигнал шумовой: «уровень шума» — это полка PSD. Площадь под кривой = полная мощность. При построении спектра шумового сигнала важно, с какой scaling считается оценка:

- `scaling='density'` — ед.²/Гц (PSD);
- `scaling='spectrum'` — ед.² (мощность в бине).

Окно **нормирует** PSD через ENBW: $PSD_{\text{пик}} = |W|^2 / (fs \cdot ENBW)$. Именно это делает `scipy.signal.welch` автоматически.

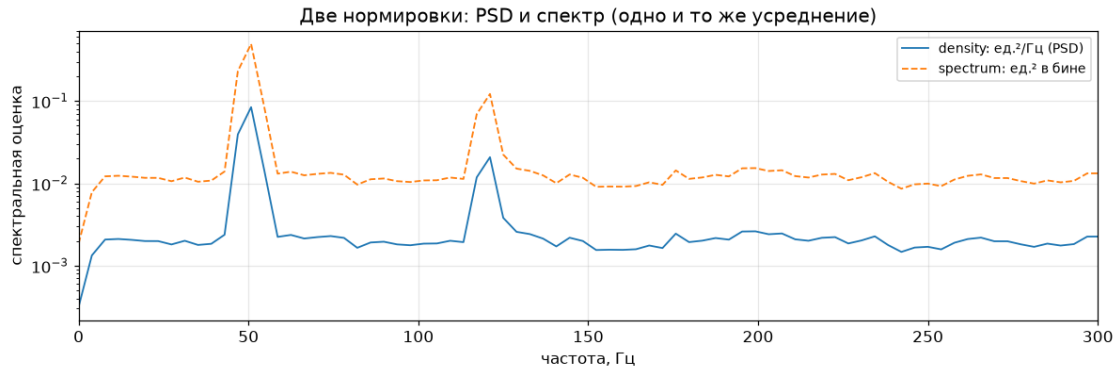
```

[15]: fig, ax = plt.subplots(figsize=(11, 3.8))
      for sc, ls, lab in [("density", "-", "density: ед.²/Гц (PSD)",
                           ("spectrum", "--", "spectrum: ед.² в бине"))]:
          f, P = signal.welch(x, fs, window="hann", nperseg=256, noverlap=128,
                              scaling=sc)
          ax.semilogy(f, P, ls, lw=1.2, label=lab)
      ax.set_xlim(0, 300)
      ax.set_xlabel("частота, Гц")
      ax.set_ylabel("спектральная оценка")
      ax.set_title("Две нормировки: PSD и спектр (одно и то же усреднение)")
      ax.legend(fontsize=9)
      fig.tight_layout()

```

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

сохранено: fig_psd.png



1.8 7. Время—частота: спектрограмма (STFT)

Если сигнал **меняется во времени**, одного спектра мало. **Короткое преобразование Фурье** (STFT) считает спектр в **скользящем окне**; результат — **спектрограмма**: по оси X — время, по Y — частота, цвет — мощность.

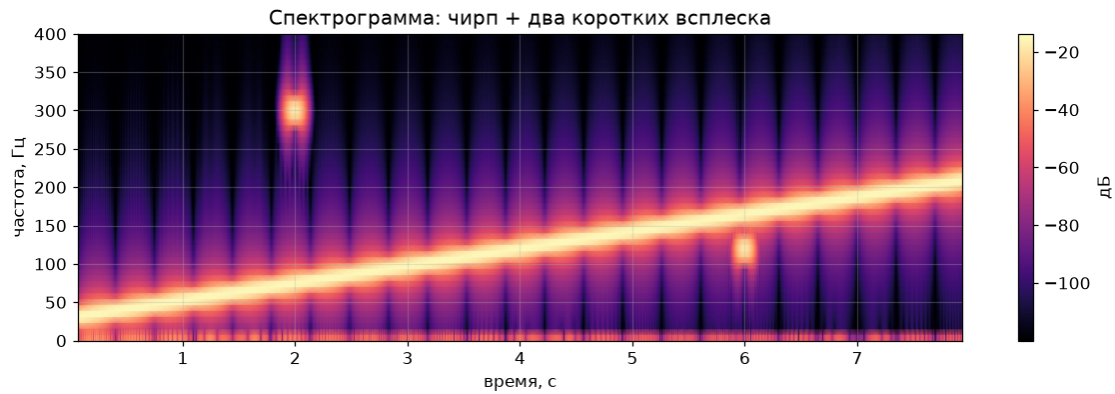
Здесь работает **принцип неопределённости**: короткое окно даёт хорошее **время** и плохую **частоту**, длинное — наоборот.

```
[16]: fs = 1000.0
T = 8.0
t = np.arange(0, T, 1 / fs)
chirp = np.cos(2 * np.pi * (30 * t + 0.5 * (180 / T) * t ** 2))
x = chirp.copy()
x += 0.8 * np.exp(-((t - 2) ** 2) / 0.002) * np.cos(2 * np.pi * 300 * t)
x += 0.8 * np.exp(-((t - 6) ** 2) / 0.002) * np.cos(2 * np.pi * 120 * t)

f, tt, S = signal.spectrogram(x, fs, window="hann", nperseg=128,
                              noverlap=120, scaling="density")
fig, ax = plt.subplots(figsize=(11, 3.8))
pcm = ax.pcolormesh(tt, f, 10 * np.log10(S + 1e-12), shading="gouraud",
                    cmap="magma")
ax.set_ylim(0, 400)
ax.set_xlabel("время, с")
ax.set_ylabel("частота, Гц")
ax.set_title("Спектрограмма: чирп + два коротких всплеска")
fig.colorbar(pcm, ax=ax, label="дБ")
fig.tight_layout()
save(fig, "fig_spectrogram.png")
```

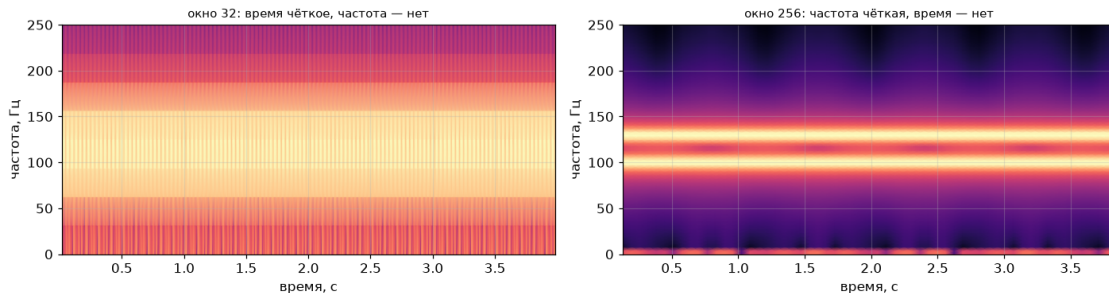
```
plt.show()
```

сохранено: fig_spectrogram.png



```
[17]: fs = 1000.0
T = 4.0
t = np.arange(0, T, 1 / fs)
x = np.cos(2 * np.pi * 100 * t) + np.cos(2 * np.pi * 130 * t)
fig, ax = plt.subplots(1, 2, figsize=(13, 3.6))
for a, nperseg, lab in [(ax[0], 32, "окно 32: время чёткое, частота — нет"),
                        (ax[1], 256, "окно 256: частота чёткая, время — нет")]:
    f, tt, S = signal.spectrogram(x, fs, window="hann", nperseg=nperseg,
                                   noverlap=nperseg * 3 // 4)
    pcm = a.pcolormesh(tt, f, 10 * np.log10(S + 1e-12), shading="gouraud",
                       cmap="magma")
    a.set_ylim(0, 250)
    a.set_title(lab, fontsize=10)
    a.set_xlabel("время, с")
    a.set_ylabel("частота, Гц")
fig.tight_layout()
save(fig, "fig_stft_tradeoff.png")
plt.show()
```

сохранено: fig_stft_tradeoff.png



1.9 8. Всё вместе: спектр вибрации, которому можно верить

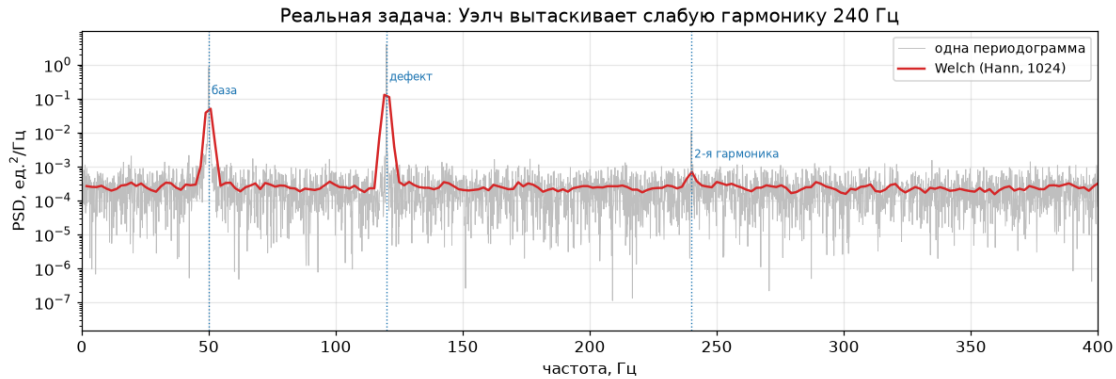
Соберём практический рецепт на сигнале из проекта: база 50 Гц, дефект 120 Гц, слабая вторая гармоника 240 Гц и шум. Одна периодограмма «утопает» в шуме, а **Уэлч** вытаскивает слабую линию 240 Гц.

```
[18]: fs = 2000.0
N = 16384
rng = np.random.default_rng(7)
n = np.arange(N)
vib = (0.6 * np.sin(2 * np.pi * 50 * n / fs)
      + 1.0 * np.sin(2 * np.pi * 120 * n / fs)
      + 0.05 * np.sin(2 * np.pi * 240 * n / fs)
      + 0.5 * rng.standard_normal(N))

f1, P1 = signal.periodogram(vib, fs, window="boxcar", scaling="density")
f2, P2 = signal.welch(vib, fs, window="hann", nperseg=1024, noverlap=512,
                      scaling="density")

fig, ax = plt.subplots(figsize=(11, 3.9))
ax.semilogy(f1[1:], P1[1:], color="0.75", lw=0.6, label="одна периодограмма")
ax.semilogy(f2[1:], P2[1:], color="C3", lw=1.6, label="Welch (Hann, 1024)")
for f0, lab in [(50, "база"), (120, "дефект"), (240, "2-я гармоника")]:
    ax.axvline(f0, color="C0", ls=":", lw=0.9)
    ax.annotate(lab, (f0, P2[np.argmax(abs(f2 - f0))]),
               textcoords="offset points", xytext=(2, 10), fontsize=8,
               color="C0")
ax.set_xlim(0, 400)
ax.set_xlabel("частота, Гц")
ax.set_ylabel(r"PSD, ед.$^2$/Гц")
ax.set_title("Реальная задача: Уэлч вытаскивает слабую гармонику 240 Гц")
ax.legend(fontsize=9)
fig.tight_layout()
save(fig, "fig_vib_welch.png")
plt.show()
```

сохранено: fig_vib_welch.png

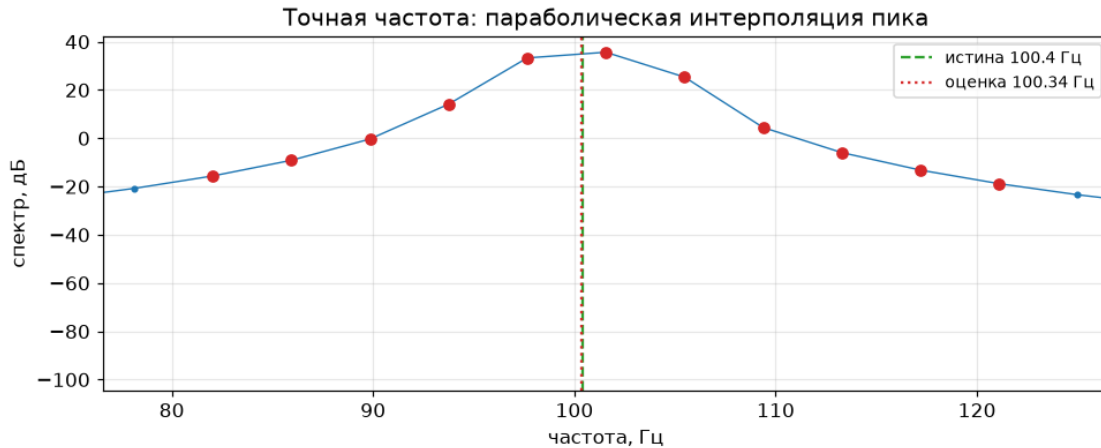


```
[19]: fs, N = 1000.0, 256
n = np.arange(N)
f0 = 100.4
x = np.cos(2 * np.pi * f0 * n / fs) * signal.get_window("hann", N)
f = np.fft.rfftfreq(N, 1 / fs)
X = np.abs(np.fft.rfft(x))
k = int(np.argmax(X))
fk = f[k]

# параболическая интерполяция по log|X|
y0, y1, y2 = np.log(X[k - 1]), np.log(X[k]), np.log(X[k + 1])
d = 0.5 * (y0 - y2) / (y0 - 2 * y1 + y2)
f_est = fk + d * (f[1] - f[0])

fig, ax = plt.subplots(figsize=(9, 3.8))
ax.plot(f, 20 * np.log10(X + 1e-12), "o-", ms=3, lw=0.9)
mm = slice(max(0, k - 5), min(N // 2, k + 6))
ax.plot(f[mm], 20 * np.log10(X[mm] + 1e-12), "o", color="C3")
ax.axvline(f0, color="C2", ls="--", label=f"истина {f0} Гц")
ax.axvline(f_est, color="C3", ls=":", label=f"оценка {f_est:.2f} Гц")
ax.set_xlim(fk - 25, fk + 25)
ax.set_xlabel("частота, Гц")
ax.set_ylabel("спектр, дБ")
ax.set_title("Точная частота: параболическая интерполяция пика")
ax.legend(fontsize=9)
fig.tight_layout()
save(fig, "fig_freq_estimate.png")
plt.show()
print(f"бин {fk} Гц, оценка {f_est:.3f} Гц, истина {f0} Гц, "
      f"ошибка {abs(f_est - f0):.3f} Гц")
```

сохранено: fig_freq_estimate.png



бин 101.5625 Гц, оценка 100.338 Гц, истина 100.4 Гц, ошибка 0.062 Гц

Оценка частоты по фазе двух окон даёт ещё точнее, но параболической интерполяции обычно достаточно для вибрации и радиосигналов.

1.10 9. Рецепт на Python (scipy.signal)

```
from scipy import signal
```

окна

```
w = signal.get_window("hann", N) # 'hamming', 'blackmanharris', ...
```

БПФ / реальный сигнал

```
X = np.fft.rfft(x * w)
f = np.fft.rfftfreq(N, 1 / fs)
```

оценка PSD с окном и усреднением

```
f, Pxx = signal.welch(x, fs, window="hann", nperseg=1024,
                      noverlap=512, scaling="density")
```

изменяющийся во времени сигнал

```
f, t, S = signal.spectrogram(x, fs, window="hann", nperseg=128,
                             noverlap=96, scaling="density")
```

Порядок действий: выбрать T под нужное Δf → выбрать окно → взять N как степень двойки (быстрее) → посчитать rfft/welch → перевести в дБ для глаз.

1.11 Сквозной проект: задание №4

Задание после лекции 4 (для вашего проекта):

1. Реализовать **свою** функцию БПФ radix-2 (рекурсивную) и сверить с `numpy.fft`.
2. Показать **утечку** и её подавление окном (прямоуг. против Hann/Hamming).

3. Сравнить **разрешение** при двух длительностях записи; проверить, что zero-padding разрешение не улучшает.
4. Оценить **PSD** методом **Уэлча** и объяснить выбор `nperseg/noverlap`.
5. Построить **спектрограмму** нестационарного участка своего сигнала.

Сдавать: ноутбук с графиками + **один** слайд-вывод (какое окно выбрали и почему).

1.12 Итоги лекции 4

- **БПФ** — не новое преобразование, а быстрый алгоритм ДПФ: $O(N^2) \rightarrow O(N \log N)$.
- Идея: прореживание по времени, **бабочки** radix-2, **бит-реверс** на входе.
- Конечная запись = прямоугольное окно → **утечка**; **окна** лечат, но расширяют главный лепесток.
- **Разрешение** $\Delta f = 1/T$; **zero-padding** только интерполирует.
- **Уэлч**: сегменты + окно + усреднение $|БПФ|^2 \rightarrow$ устойчивая **PSD**.
- **Спектрограмма** (STFT) показывает эволюцию спектра; компромисс время–частота.

Дальше (лекция 5): спектры радио и модуляция — АМ/ЧМ/ФМ, цифровые созвездия, комплексная огибающая (I/Q).

1.13 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. Harris F. J. *On the Use of Windows for Harmonic Analysis with the DFT*.
6. Welch P. D. *The Use of the Fast Fourier Transform for Estimation of Power Spectra*.
7. Материалы курса и RAG-ассистент по этим книгам — в папке курса.