Лабораторная работа № 5.
связь непрерывного и дискретного преобразования Фурье

Выполнил: Смирнов Алексей Владимирович — R3242
409578

г. Санкт-Петербург, 2025

Содержание

1 Задание 1

Рассмотрим прямоугольную функцию $\Pi : \mathbb{R} \to \mathbb{R}$:

$$\Pi(t) = \begin{cases} 1, & |t| \le \frac{1}{2} \\ 0, & |t| > \frac{1}{2} \end{cases} \tag{1}$$

Ее истинным фурье-образов является интеграл:

$$\hat{\Pi}(\nu) = \int_{-\infty}^{\infty} \Pi(t) e^{-2\pi i \nu t} \, \mathrm{d}t \tag{2}$$

Вычислим значение интеграла:

$$\begin{gathered} \int_{-\infty}^{\infty} \Pi(t) e^{-2\pi i \nu t} \, \mathrm{d}t = \int_{-\frac{1}{2}}^{\frac{1}{2}} 1 \cdot e^{-2\pi i \nu t} \, \mathrm{d}t = \\ = \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{-2\pi i \nu t} \frac{1}{-2\pi i \nu} \, \mathrm{d}(-2\pi i \nu t) = \frac{1}{-2\pi i \nu} e^{-2\pi i \nu t} \Big|_{t=-\frac{1}{2}}^{t=\frac{1}{2}} = \\ = \frac{e^{-\pi i \nu} - e^{\pi i \nu}}{-2\pi i \nu} = \frac{\cos(\pi\nu) - i \sin(\pi\nu) - \cos(\pi\nu) - i \sin(\pi\nu)}{-2\pi i \nu} = \\ = \frac{-2i \sin(\pi\nu)}{-2\pi i \nu} = \frac{\sin(\pi\nu)}{\pi\nu} = \boxed{\operatorname{sinc}(\pi\nu)} \end{gathered} \tag{3}$$

Построим графики функции и ее образа, полученного аналитическим путем:

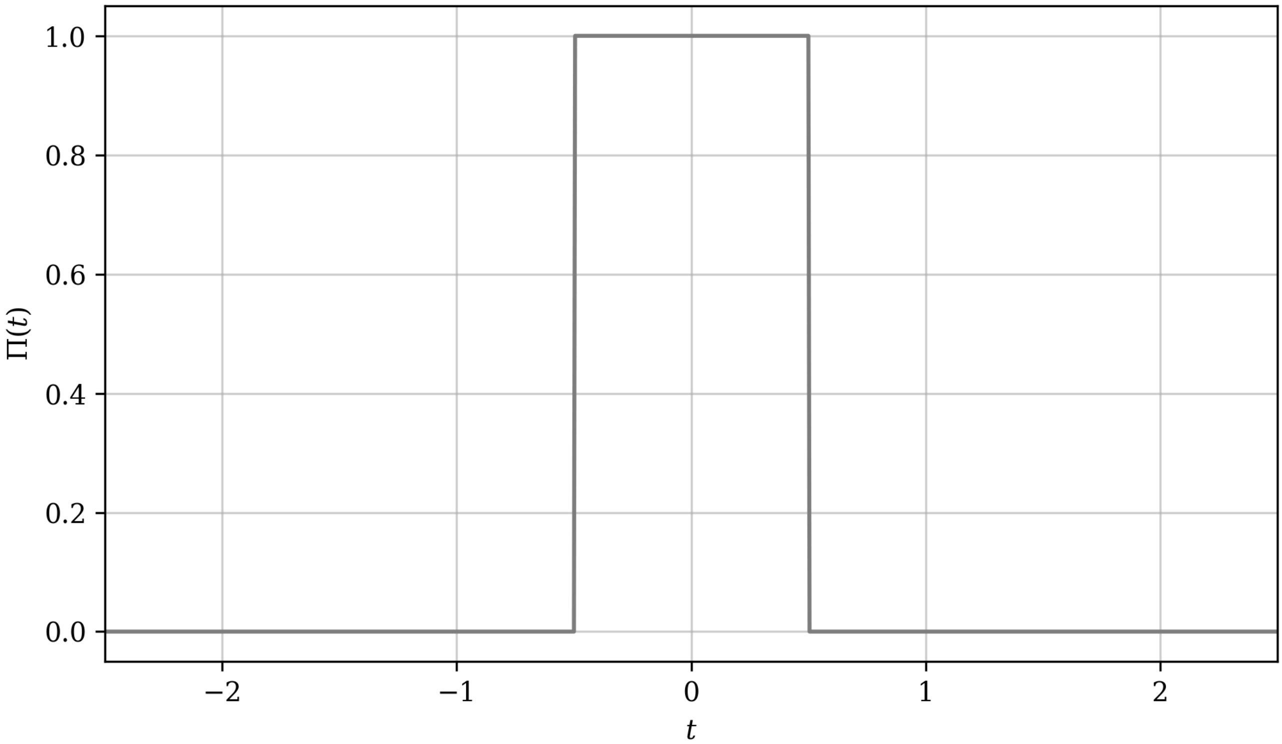

Рисунок 1 — График функции $\Pi(t)$

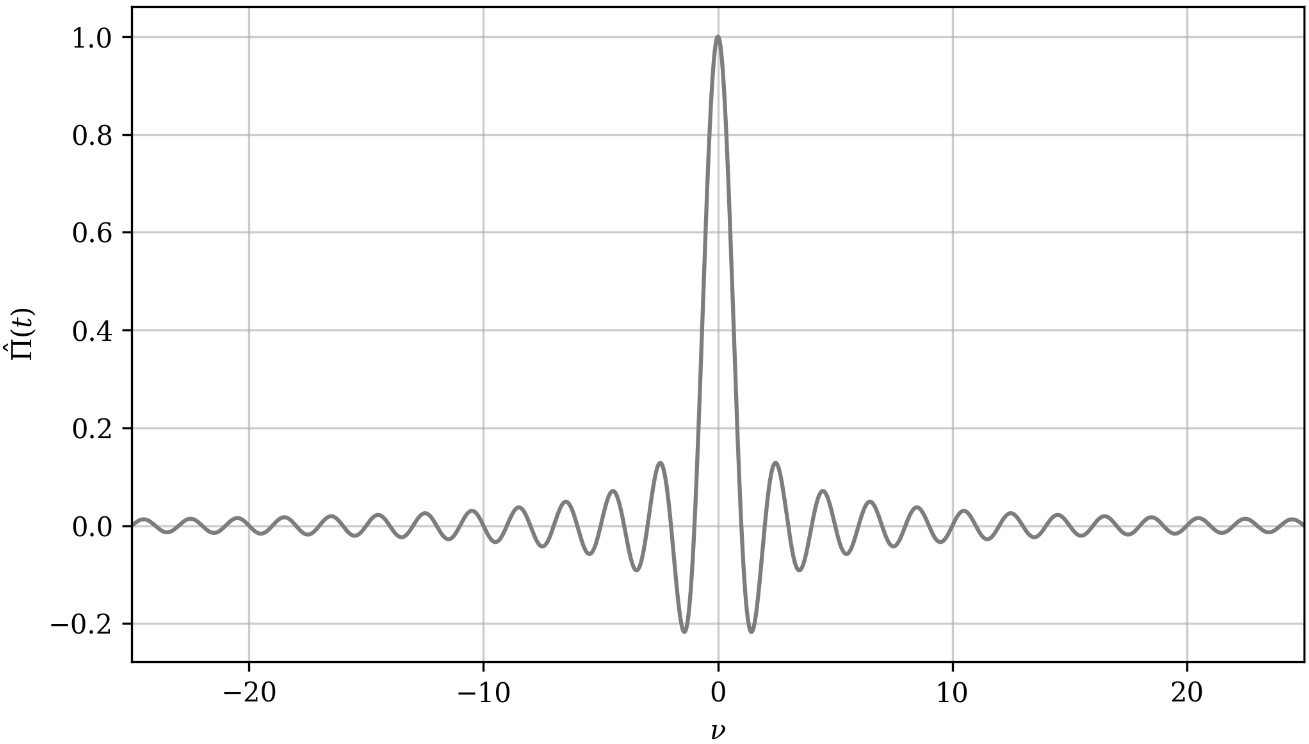

Рисунок 2 — График аналитического образа функции $\hat{\Pi}(t)$

1.1 Численное интегрирование

Найдем спектр функции $\Pi(t)$ и затем восстановим функцию из него, используя численный метод интегрирования `nympy.trapezoid`:

Построим графики результатов при разных парах значений T и $\mathrm{d}t$:

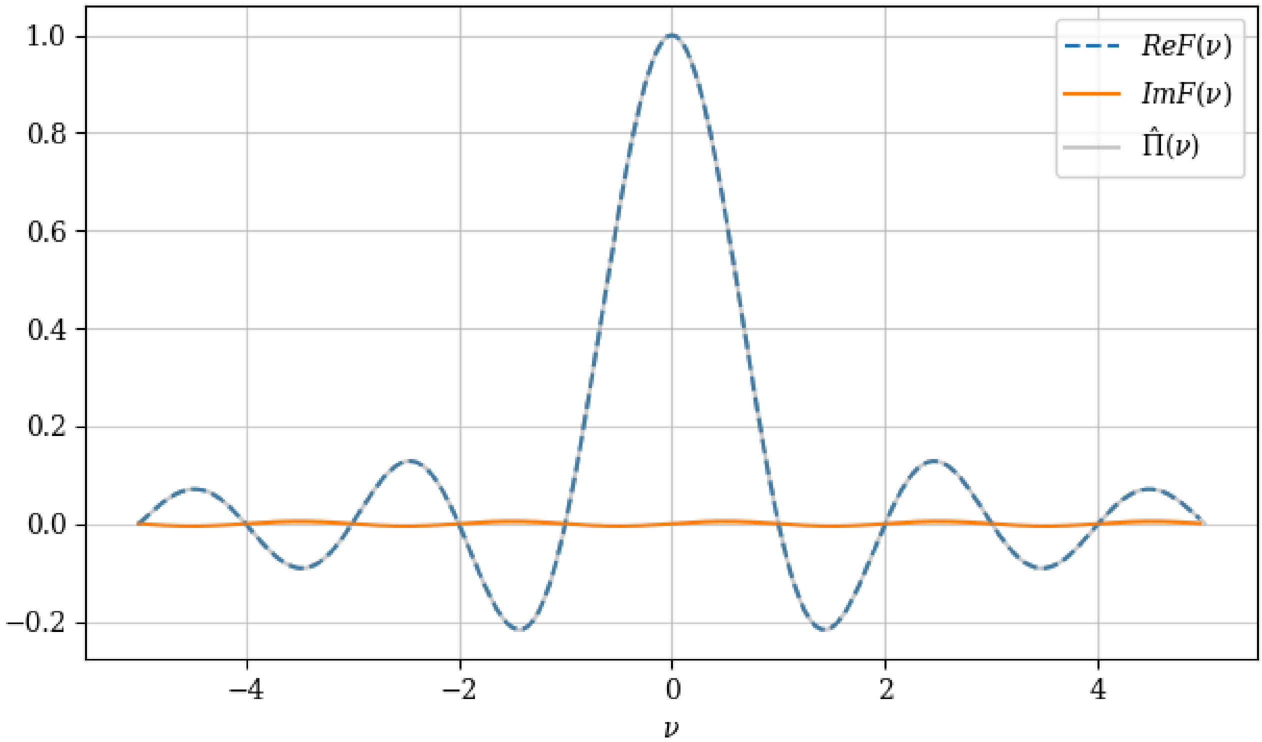

Рисунок 3 — Образ $\hat{\Pi}(t)$ полученный численным интегрированием при $T = 2$ $\mathrm{d}t = 0.005$ $V = 10$ $\mathrm{d}\nu = 0.05$

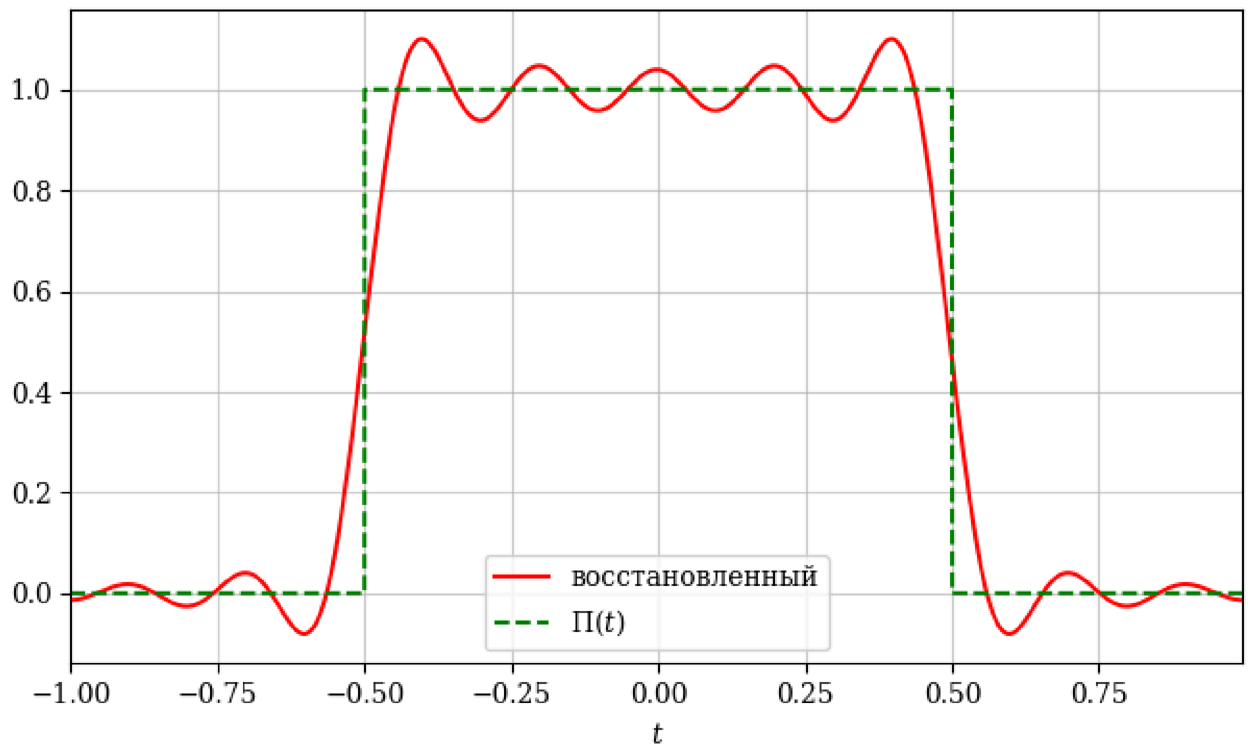

Рисунок 4 — Восстановленный из спектра сигнал методом числ. интегрирования при $T = 2$ $\mathrm{d}t = 0.005$ $V = 10$ $\mathrm{d}\nu = 0.05$

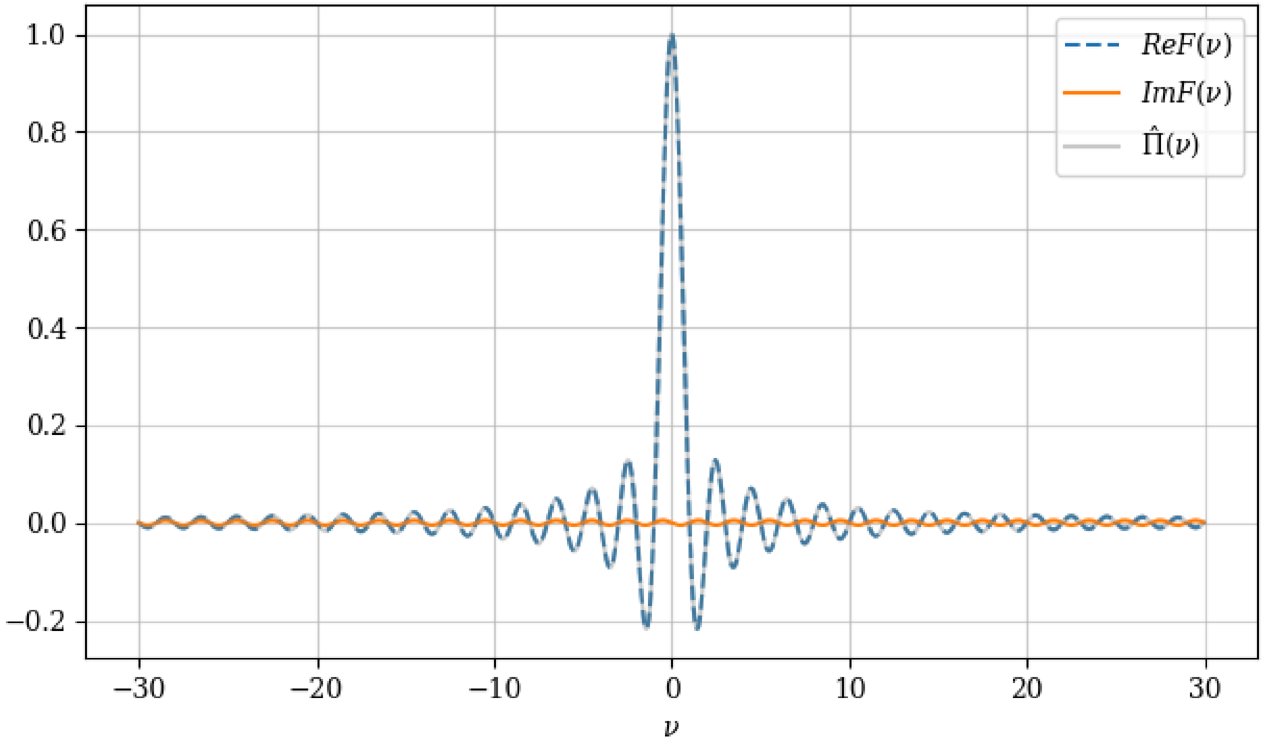

Рисунок 5 — Образ $\hat{\Pi}(t)$ полученный численным интегрированием при $T = 5$ $\mathrm{d}t = 0.005$ $V = 60$ $\mathrm{d}\nu = 0.05$

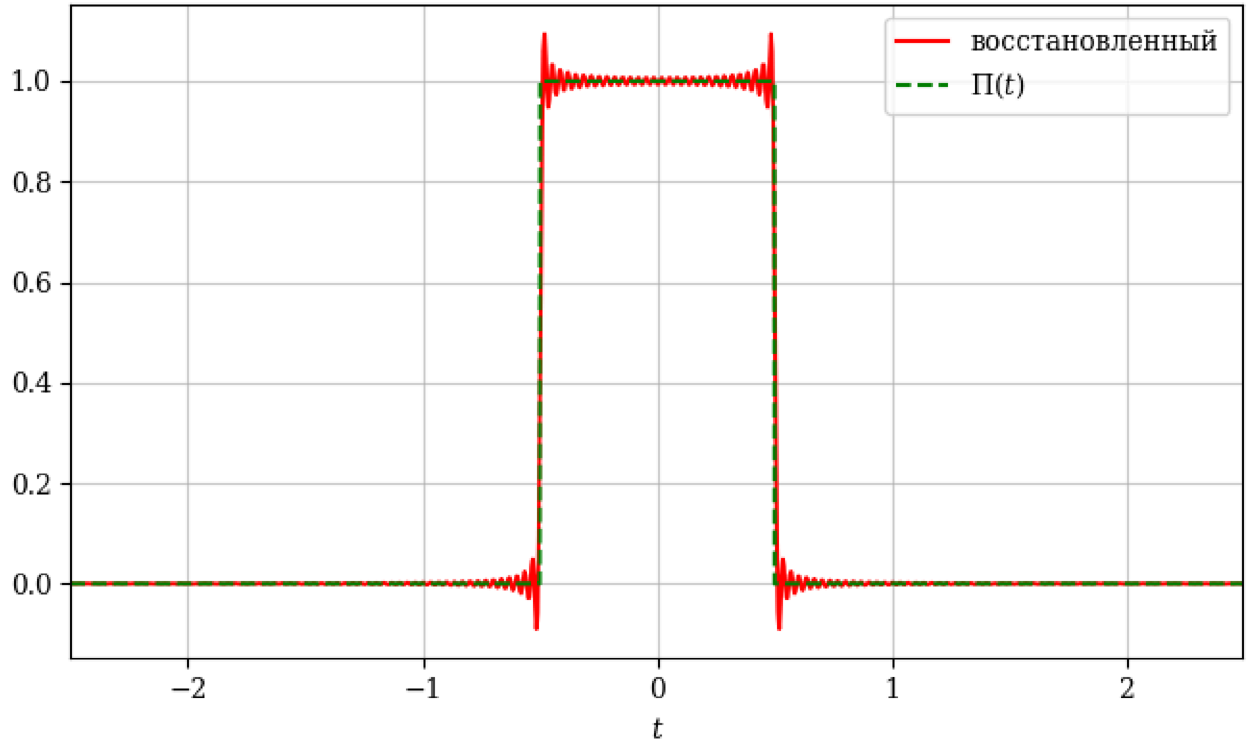

Рисунок 6 — Восстановленный из спектра сигнал методом числ. интегрирования при $T = 5$ d$t = 0.005$ $V = 60$ d$\nu = 0.05$

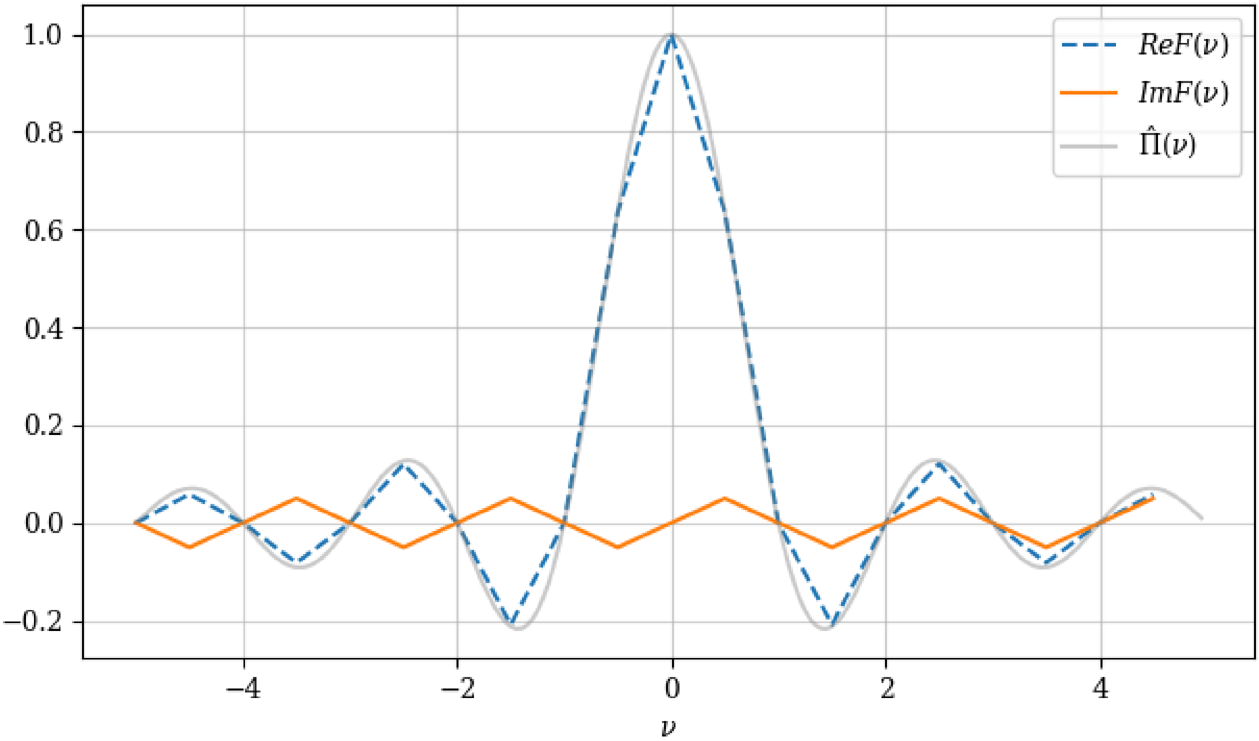

Рисунок 7 — Образ $\hat{\Pi}(t)$ полученный численным интегрированием при $T = 2$ $\mathrm{d}t = 0.05$ $V = 10$ $\mathrm{d}\nu = 0.5$

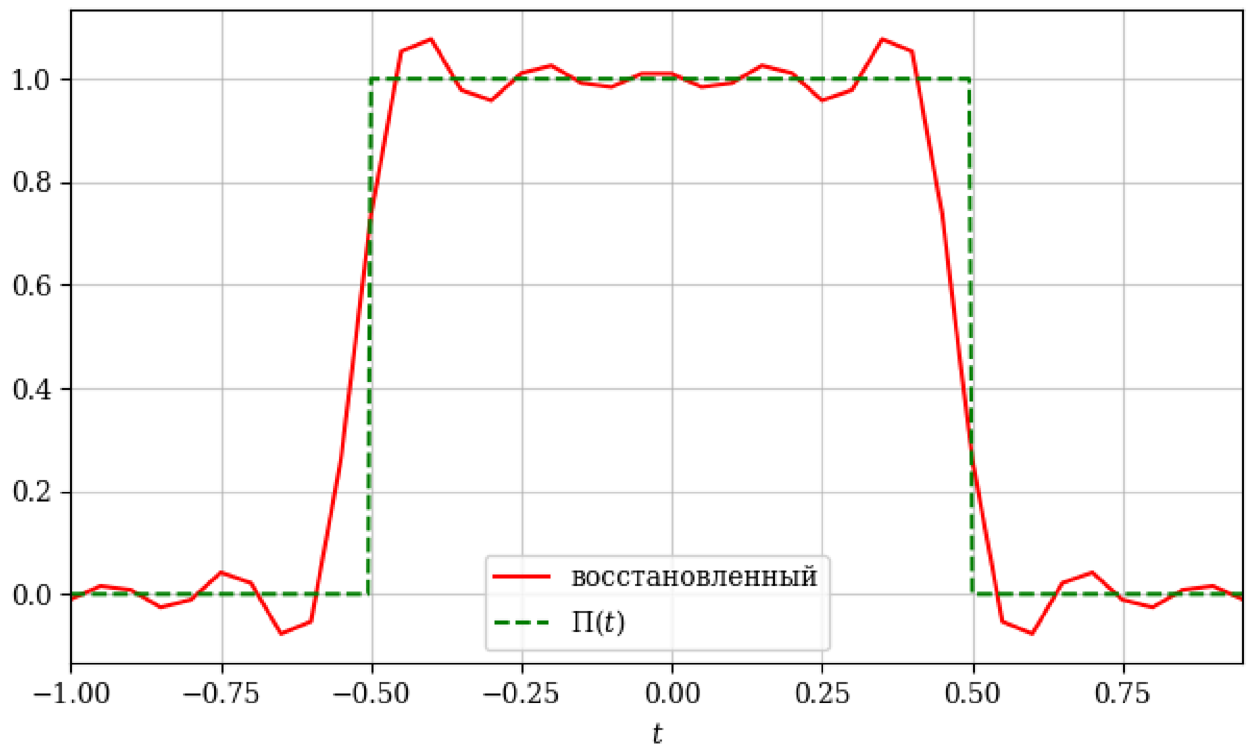

Рисунок 8 — Восстановленный из спектра сигнал методом числ. интегрирования при $T = 2$ $\mathrm{d}t = 0.05$ $V = 10$ $\mathrm{d}\nu = 0.5$

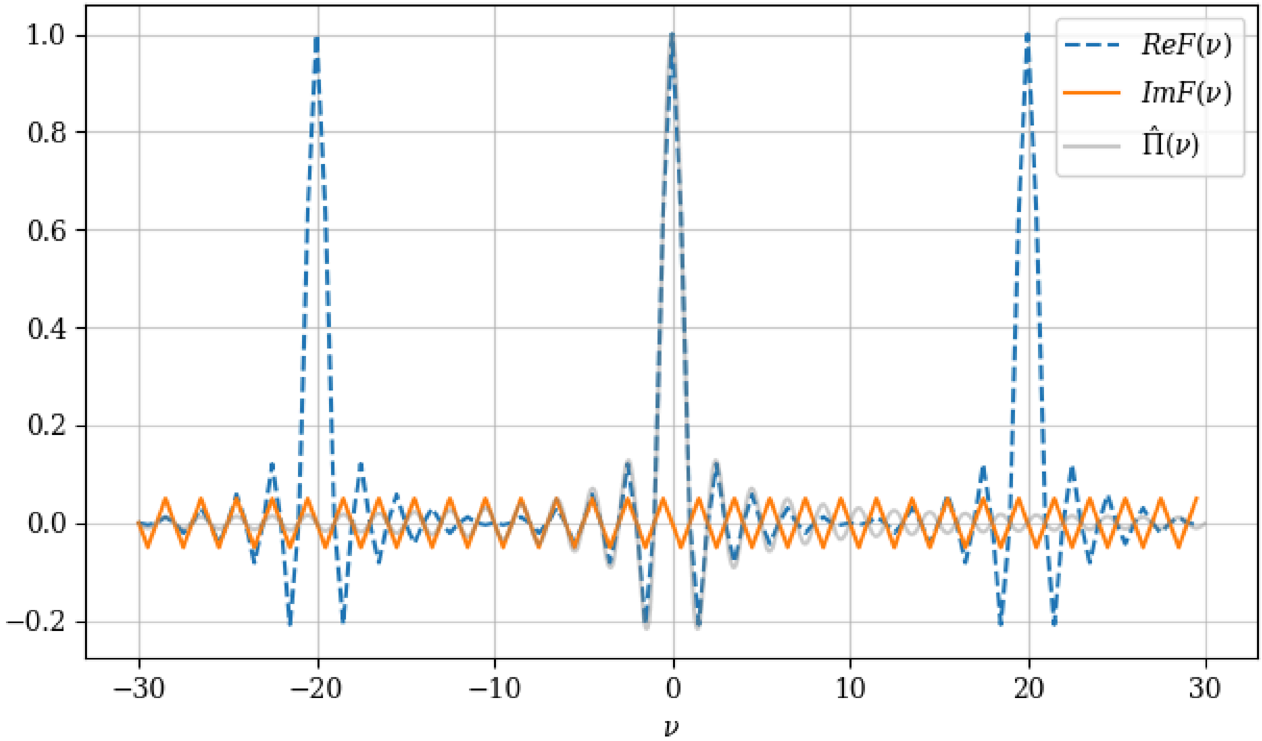

Рисунок 9 — Образ $\hat{\Pi}(t)$ полученный численным интегрированием при $T = 5$ $\mathrm{d}t = 0.05$ $V = 60$ $\mathrm{d}\nu = 0.5$

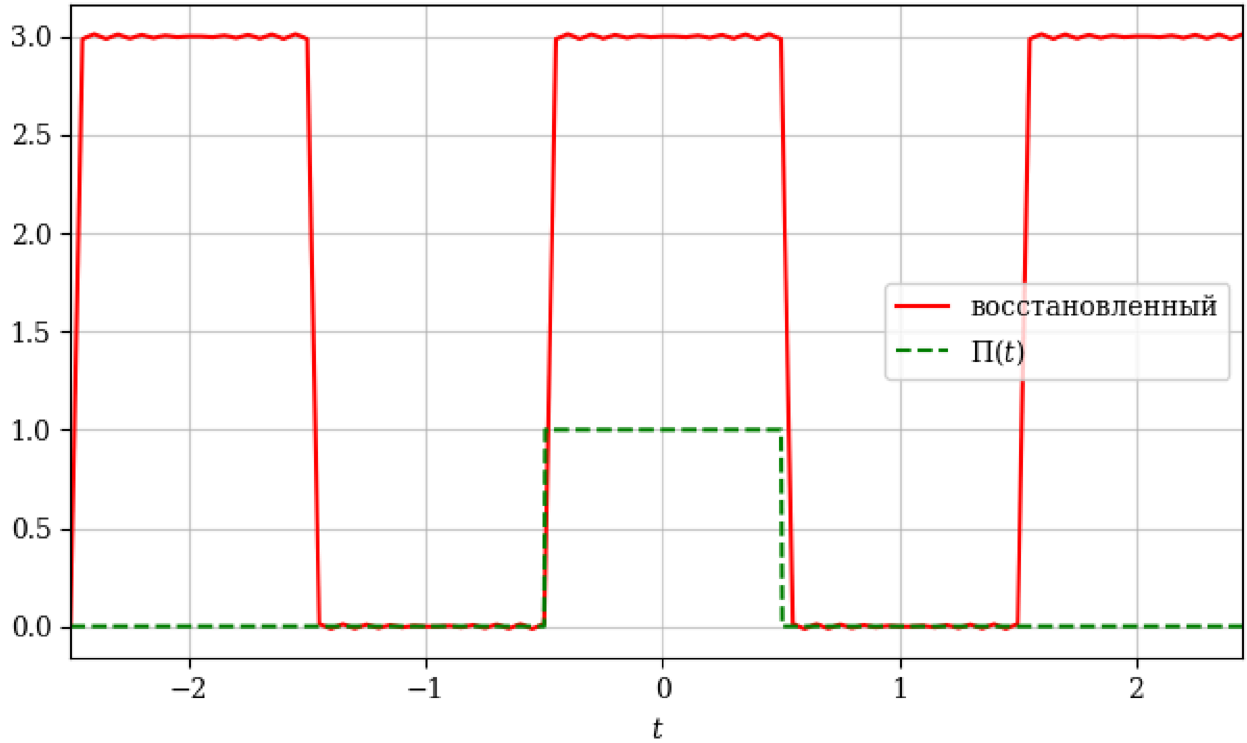

Рисунок 10 — Восстановленный из спектра сигнал методом числ. интегрирования
при $T = 5$ d$t = 0.05$ $V = 60$ d$\nu = 0.5$

1.1.1 Выводы

- Метод достаточно точно приближает аналитический спектр функции.
- При восстановлении сигнала появляются искажения, связанные с разрывами в сигнале.
- При слишком маленьком dν в спектре сигнала появляется периодичность.

1.2 Использование DFT

Классическое унитарное DFT дискретного сигнала x_k вычисляется следующим образом:

$$X[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] e^{-2\pi i n k / N} \tag{4}$$

Обратное:

$$x[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} X[n] e^{2\pi i n k / N} \tag{5}$$

Воспользуемся реализацией, которая есть в пакете numpy.

В случае с DFT параметрами которые мы можем менять являются только T и $\mathrm{d}t$, а величины V и $\mathrm{d}\nu$ зависят от параметров T и $\mathrm{d}t$ следующим образом:

$$\mathrm{d}\nu = \frac{1}{\mathrm{d}t} \cdot \frac{1}{N} = \frac{1}{\mathrm{d}tN} = \frac{1}{\mathrm{d}t \cdot \left(\frac{T}{\mathrm{d}t}\right)} = \frac{1}{T} \tag{6}$$

$$V = \frac{1}{\mathrm{d}t} \tag{7}$$

1.2.1 Графики

Построим графики DFT сигнала $\Pi(t)$ с различными парами параметров T и $\mathrm{d}t$ и соответствующим им V и $\mathrm{d}\nu$.

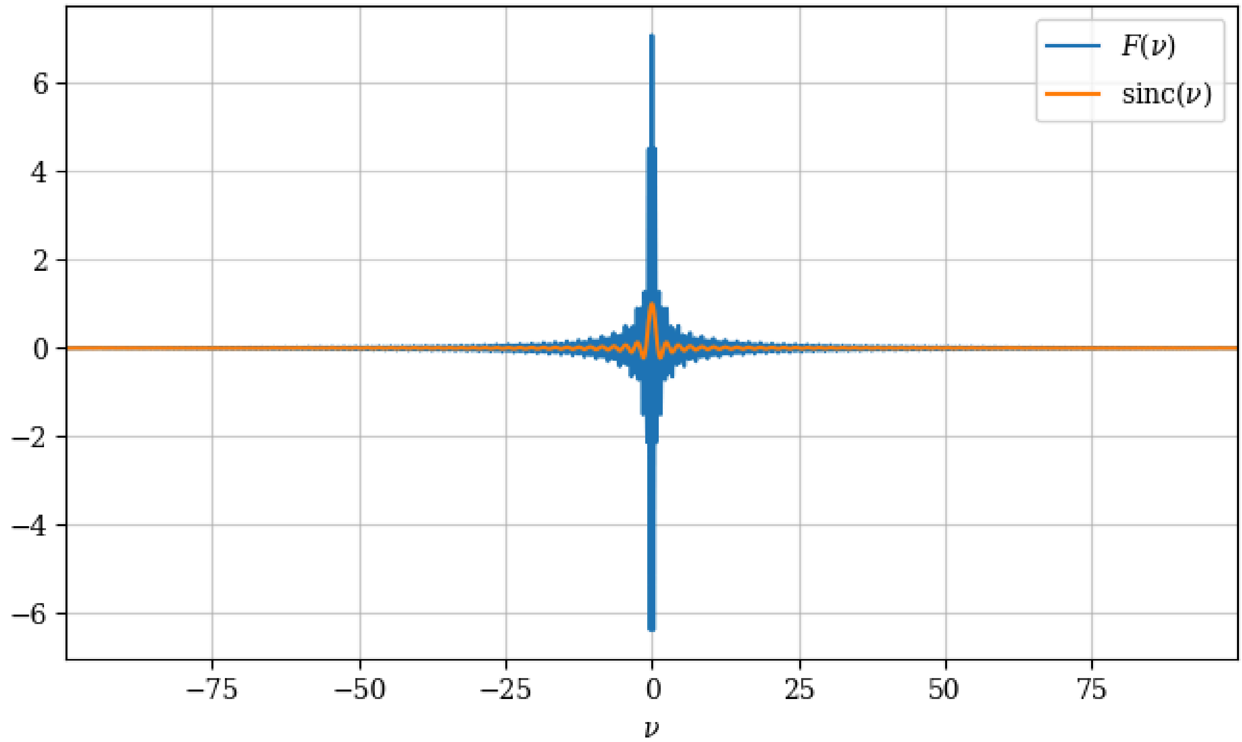

Рисунок 11 — DFT сигнала $\Pi(t)$ и $\mathrm{sinc}(\nu)$
при $T = 4$ $\mathrm{d}t = 0.005$ $V = 200$ $\mathrm{d}\nu = 0.25$

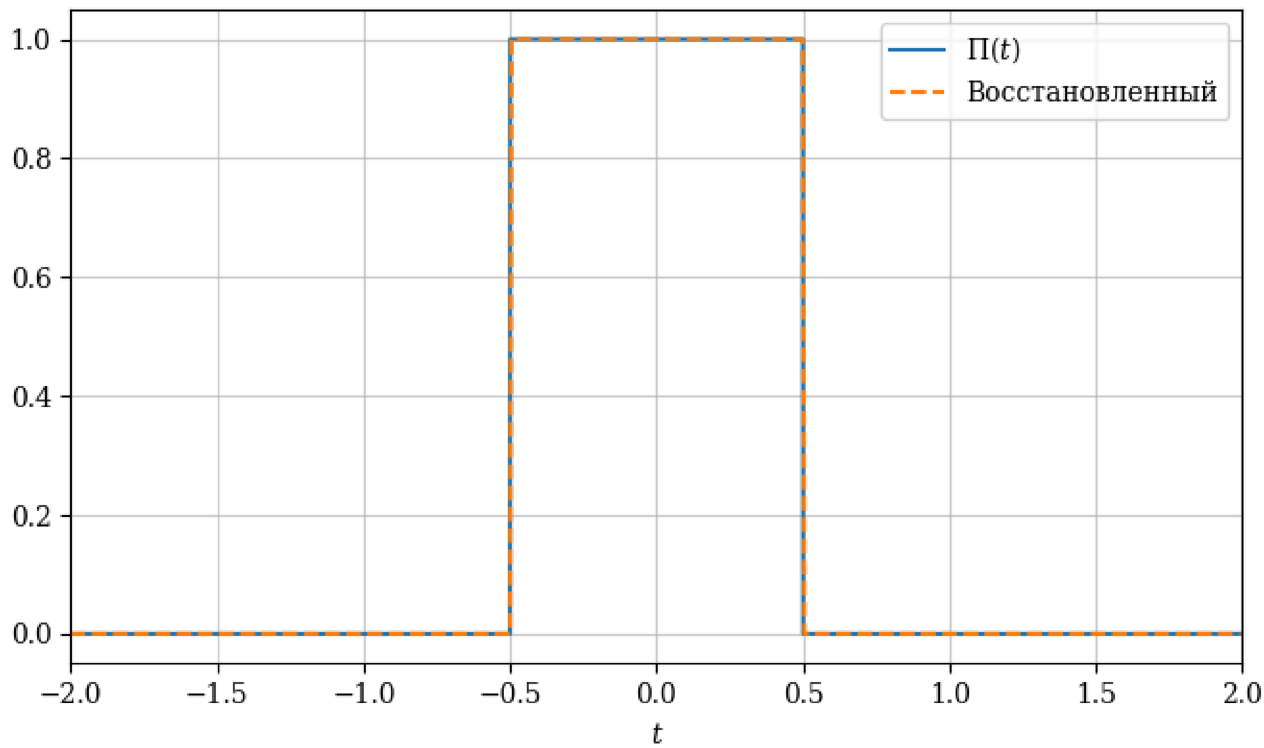

Рисунок 12 — Обратное DFT сигнала $\hat{\Pi}(\nu)$ при $T = 4$ $\mathrm{d}t = 0.005$ $V = 200$ $\mathrm{d}\nu = 0.25$

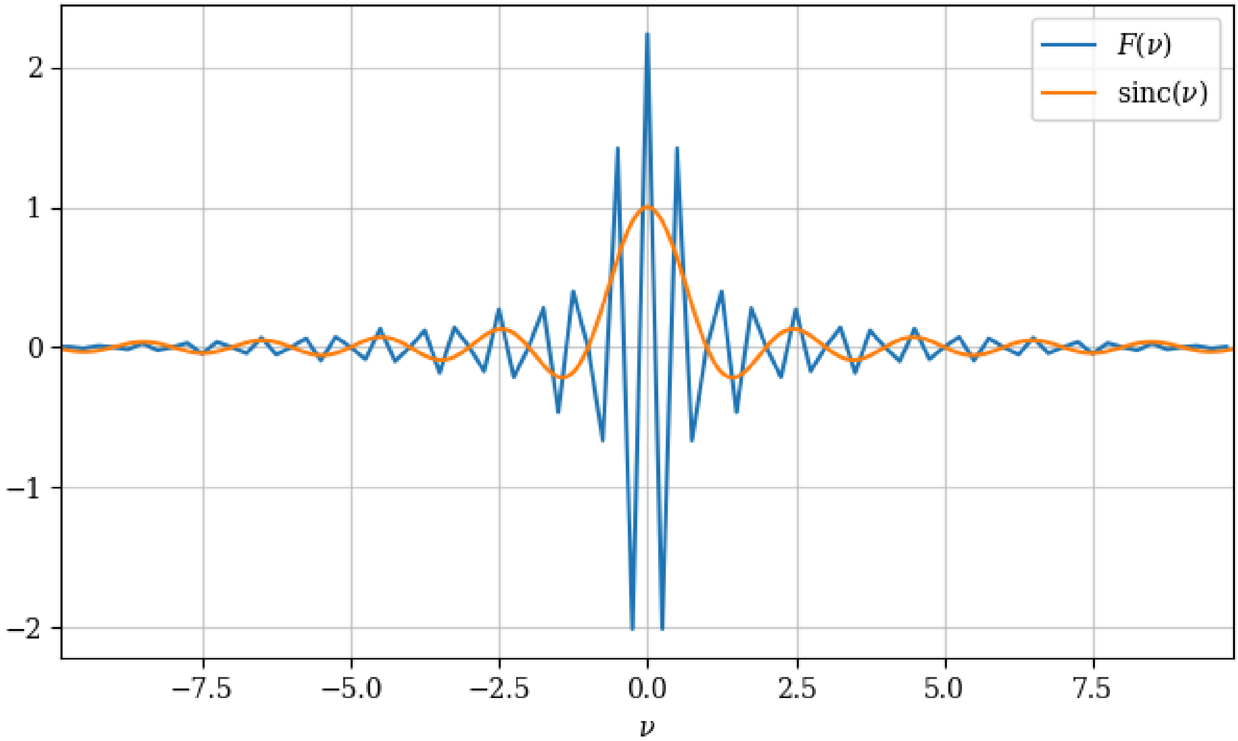

Рисунок 13 — DFT сигнала $\Pi(t)$ и $\mathrm{sinc}(\nu)$
при $T = 4$ $\mathrm{d}t = 0.05$ $V = 20$ $\mathrm{d}\nu = 0.25$

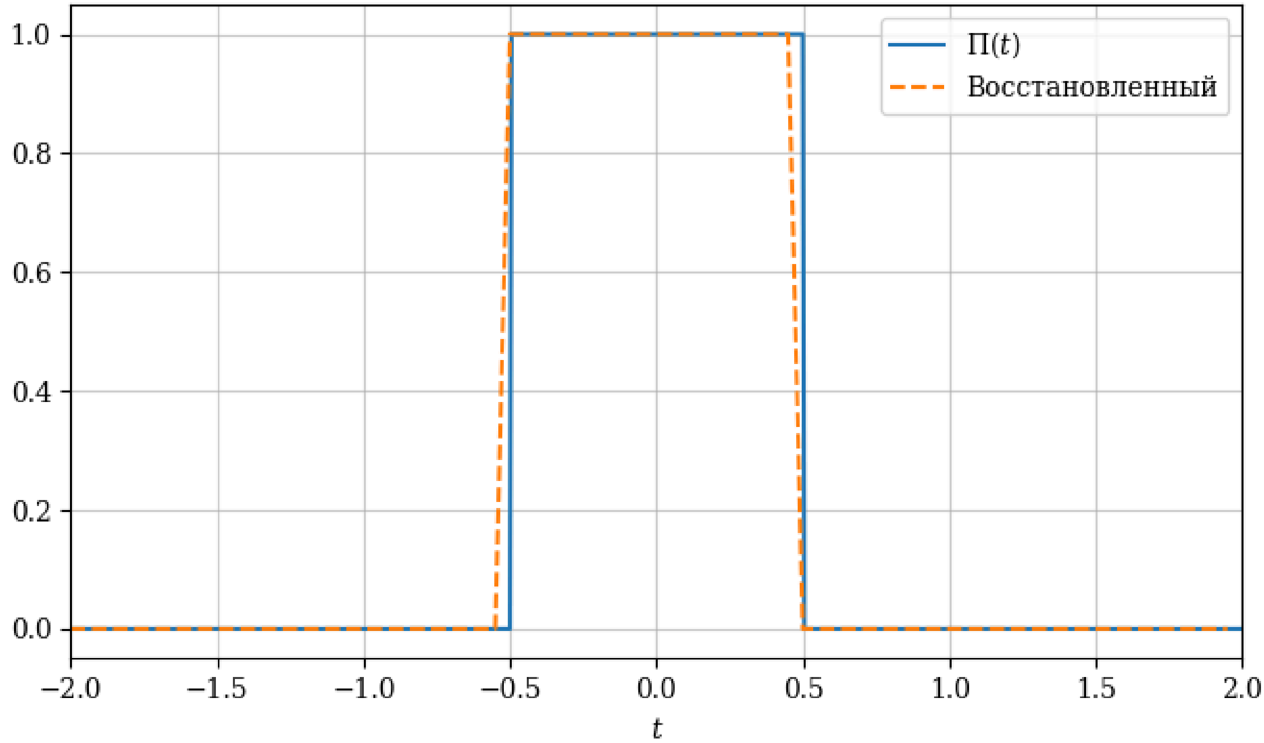

Рисунок 14 — Обратное DFT сигнала $\hat{\Pi}(\nu)$
при $T = 4$ $\mathrm{d}t = 0.05$ $V = 20$ $\mathrm{d}\nu = 0.25$

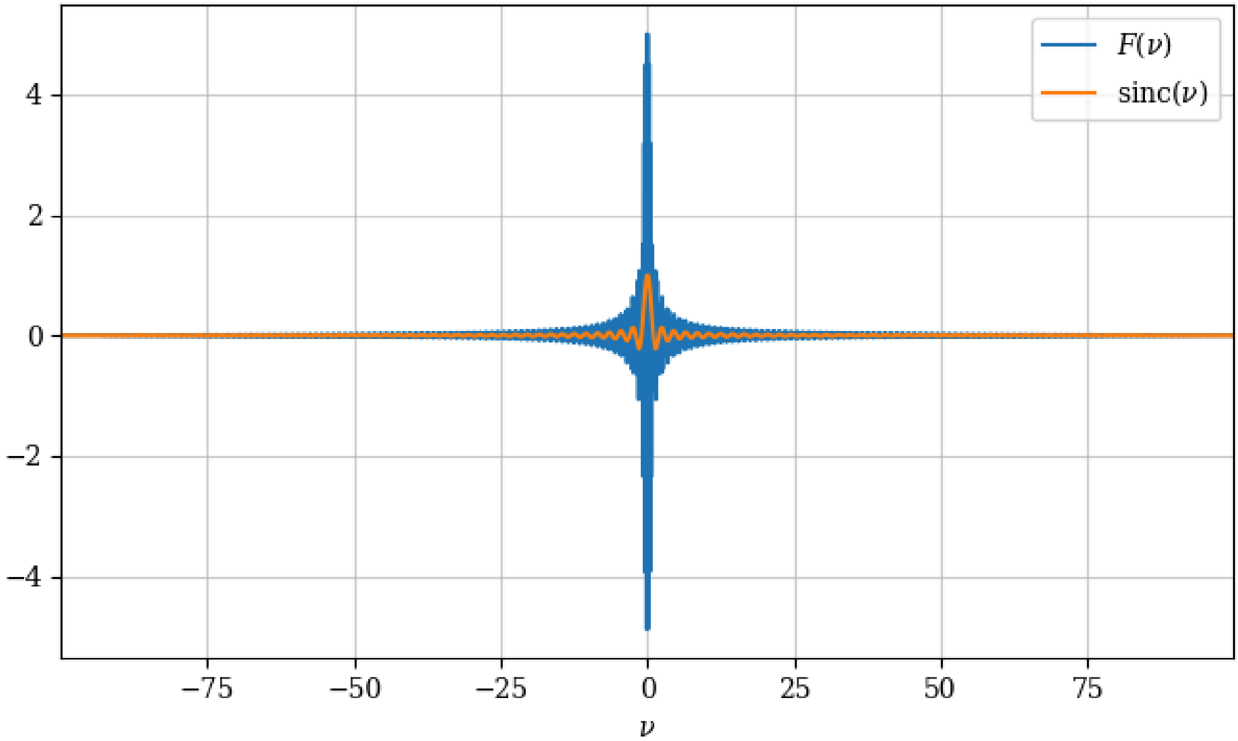

Рисунок 15 — DFT сигнала $\Pi(t)$ и $\mathrm{sinc}(\nu)$
при $T = 8$ $\mathrm{d}t = 0.005$ $V = 200$ $\mathrm{d}\nu = 0.125$

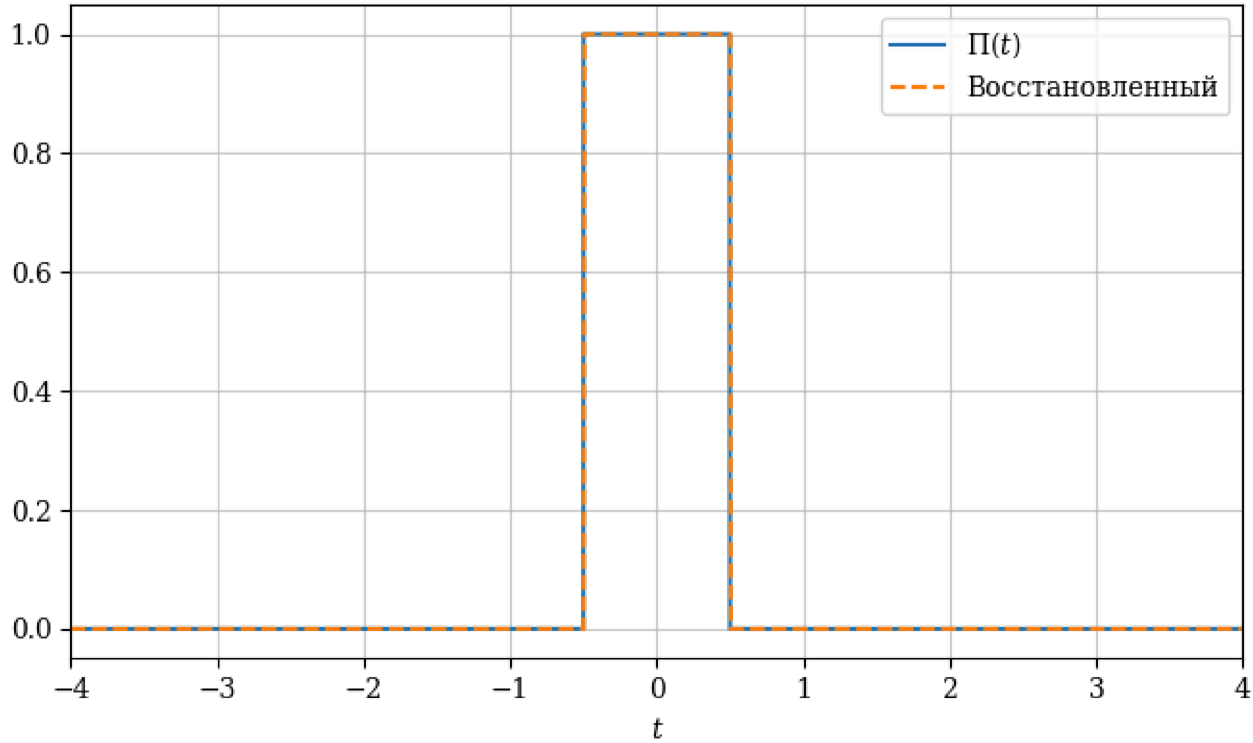

Рисунок 16 — Обратное DFT сигнала $\hat{\Pi}(\nu)$
при $T = 8$ $\mathrm{d}t = 0.005$ $V = 200$ $\mathrm{d}\nu = 0.125$

1.2.2 Выводы

- FFT выполняется на несколько порядков быстрее, чем метод численного интегрирования.
- Сигнал восстанавливается из спектра точнее, чем при использовании численного интегрирования.
- При увеличении временного окна T разрешение спектра $\mathrm{d}\nu$ увеличивается имея зависимость $\mathrm{d}\nu = 1/T$.
- При уменьшении dt ширина спектра увеличивается имея зависимость $V = 1/\,\mathrm{d}t$

1.3 Сравнение методов

Численное интегрирование приблизительно вычисляет значения интегралов преобразования Фурье:

$$\hat{f}(\nu) = \int_{-T/2}^{T/2} f(t)e^{-2\pi i\nu t}\,\mathrm{d}t \approx \sum_{n=0}^{N-1} f(t_n)e^{-2\pi i\nu t_n}\Delta t \tag{8}$$

Метод DFT вычисляет следующую сумму:

$$\begin{aligned} \hat{f}_k &= \frac{1}{\sqrt{N}} \sum_{n-0}^{N-1} f_n e^{2\pi i \frac{kn}{N}} \\ f_n &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \hat{f}_k e^{2\pi i \frac{kn}{N}} \end{aligned} \tag{9}$$

«Честное» интегрирование по Формуле 8 работает заметно дольше алгоритма FFT и имеет Сложность $O(MN) \sim O(N^2)$, в то время как FFT применяет несколько оптимизаций и выполняется за $O(N \log N)$.

Если сравнить скорость работы алгоритмов которые использовались для получения данных для рисунков выше, видно, что FFT работает на порядок быстрее:

Таблица 1 — Сравнение времени исполнения разных алгоритмов преобразования Фурье

N	Приближение интегралов	FFT
1000	26.8528 мс	0.1009 мс
2000	91.3551 мс	0.1142 мс
10000	2675.1659 мс	0.3968 мс

В это время метод трапеций производит относительно точный спектр, но нужно учитывать что при численном интегрировании накапливается ошибка, а при обратном преобразовании эта ошибка будет ещё больше из-за двойного интегрирования.

Метод FFT пользуется дискретностью сигнала и приминение пары обратных операций DFT/IDFT возвращают исходный сигнал. Но спектр, полученный при помощи FFT не совпадает с истинным спектром, так как оно привязяно к дискретности исходного сигнала.

Справедливо разграничить области применения численного интегрирования и FFT: первый стоит использовать для оценки непрервыного преобразования, а FFT для обработки сигнала — его фильтрации или анализа.

1.4 Приближение непрерывного преобразования с помощью FFT

Попробуем приблизить результат DFT к истинному спектру сигнала:

Пусть $f(t) : [-T/2, T/2] \to \mathbb{R}$.

$$\hat{f}(\nu_m) = \int_{-T/2}^{T/2} f(t) e^{-2\pi i \nu_m t} \, \mathrm{d}t, \nu_m = \frac{m}{T}, m = 0, 1..N-1 \tag{10}$$

$$\hat{f}(\nu_m) \approx \sum_{n=0}^{N-1} f(t_n) e^{-2\pi i \nu_m t_n} \Delta t, \text{где} \quad t_n = -\frac{T}{2} + n\Delta t \tag{11}$$

так как $v_m = \frac{m}{T}$:

$$e^{-2\pi i \nu_m t_n} = e^{-2\pi i \frac{m}{T}\left(-\frac{T}{2}+n\Delta t\right)} = e^{i\pi m} e^{-2\pi i \frac{mn}{N}} = (-1)^m e^{-2\pi i \frac{mn}{N}} \tag{12}$$

Получили:

$$\hat{f}(\nu_m) \approx \Delta t(-1)^m \underbrace{\sum_{n=0}^{N-1} f_n e^{-2\pi i \frac{mn}{N}}}_{\mathrm{FFT}(f)} \tag{13}$$

Обозначим $\Delta t e^{2\pi i m} = \Delta t(-1)^m = c_m$

В коэффициенте c_m Δt отвечает за масштабирование сигнала, а $e^{2\pi i m} = e^{-2\pi i \frac{T}{2}}$ за фазу, учитая начальное время сигнала $t_0 = -\frac{T}{2}$

1.4.1 Релизация метода

1. Имеем дискретизированную функцию f_n, отсчеты по которой взяты по временному окну шириной T с шагом dt.
2. Вычисляем DFT сигнала при помощи FFT и сдвигаем нуль в центр:

$$F = \mathrm{fftshift}(\mathrm{FFT}\{f_n\}) \tag{14}$$

3. Домножаем спектр на коэффициент масштабирования:

$$F_{\text{умное}} = c_m \cdot F \tag{15}$$

4. Восстановление сигнала в обратном порядке:

$$f_{\text{восст}} = \mathrm{ifft}\Big(\mathrm{ifftshift}\Big\{F_{\text{умное}}/c_m\Big\}\Big) \tag{16}$$

Сравним спектры и восстановленный сигнала тремя известными методами при различный парах параметров T и dt и соответствующих им V dν:

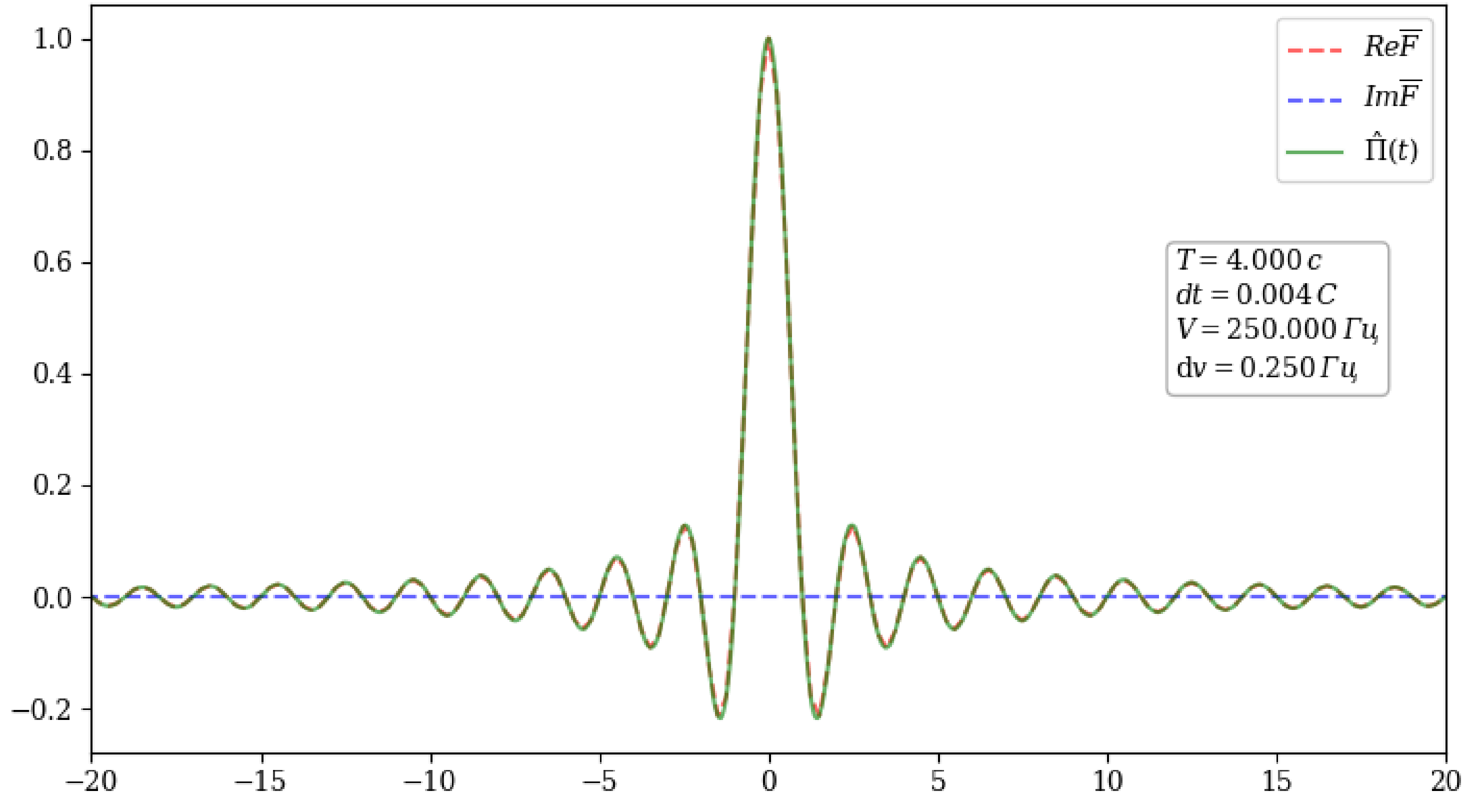

Рисунок 17 — «умное» DFT сигнала $\Pi(t)$
при $T = 4$ $\mathrm{d}t = 0.004$ $V = 250$ $\mathrm{d}\nu = 0.25$

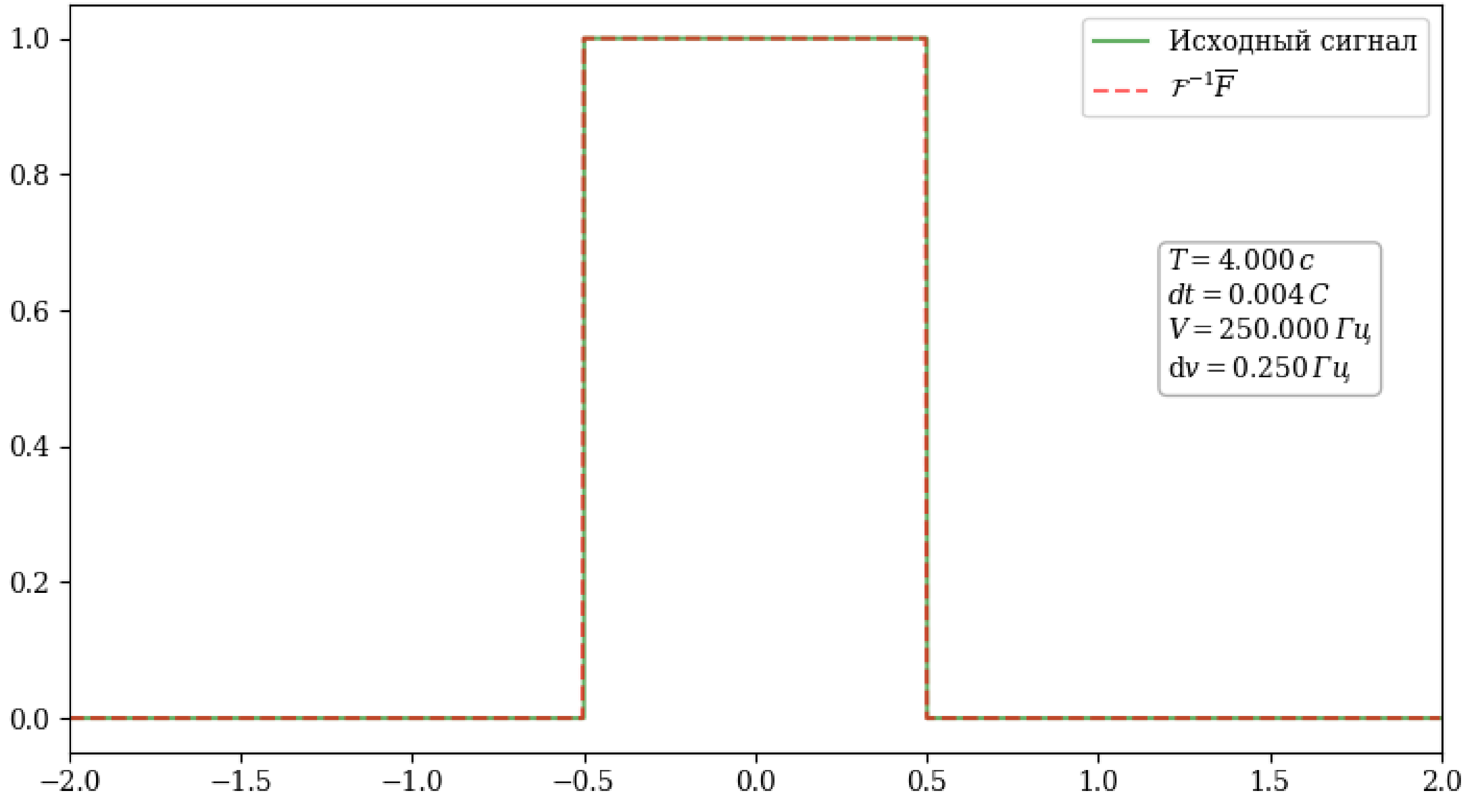

Рисунок 18 — Обратное «умное» DFT сигнала $\hat{\Pi}(\nu)$ при $T = 4$ $\mathrm{d}t = 0.004$ $V = 250$ $\mathrm{d}\nu = 0.25$

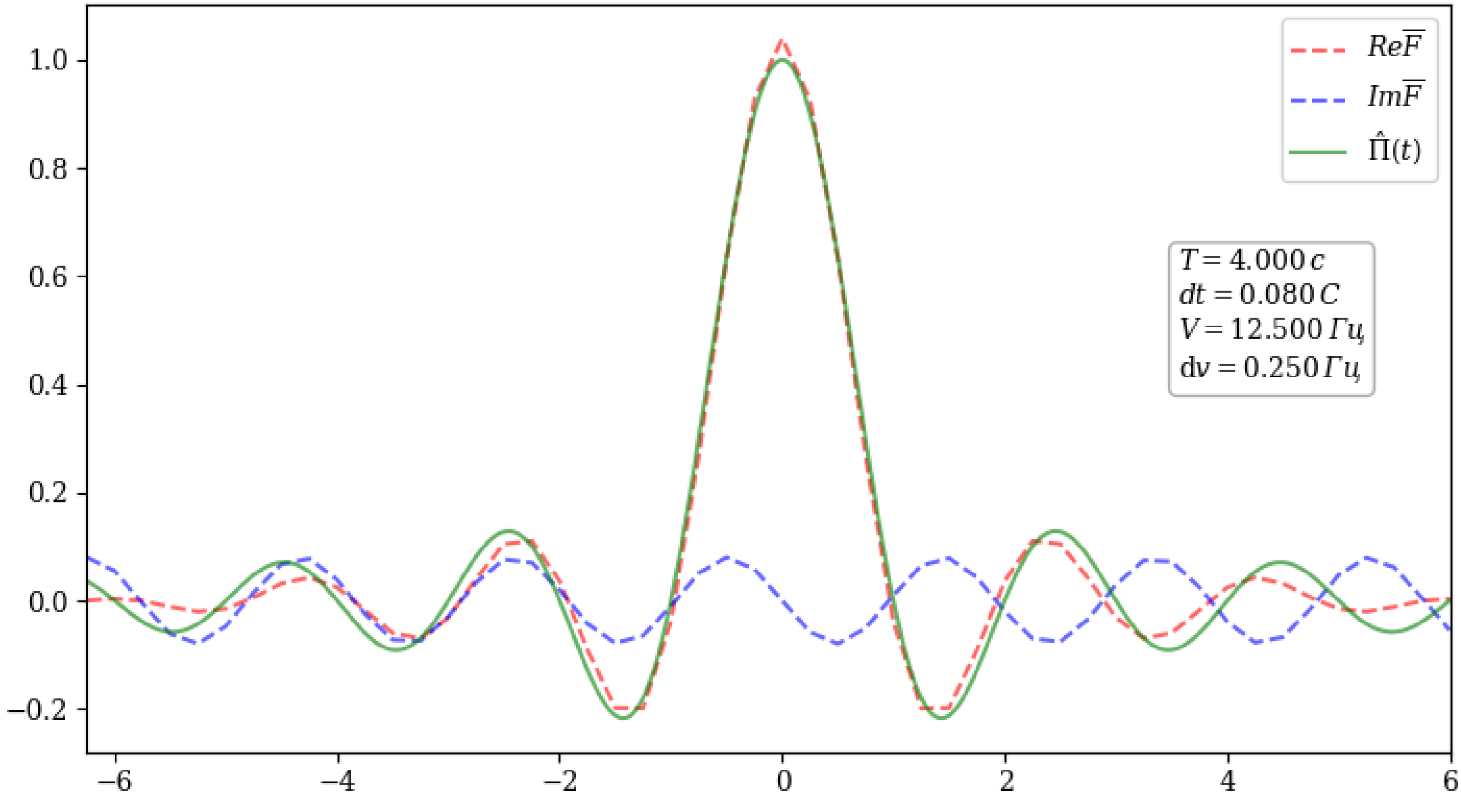

Рисунок 19 — «умное» DFT сигнала $\Pi(t)$
при $T = 4$ $\mathrm{d}t = 0.08$ $V = 125$ $\mathrm{d}\nu = 0.25$

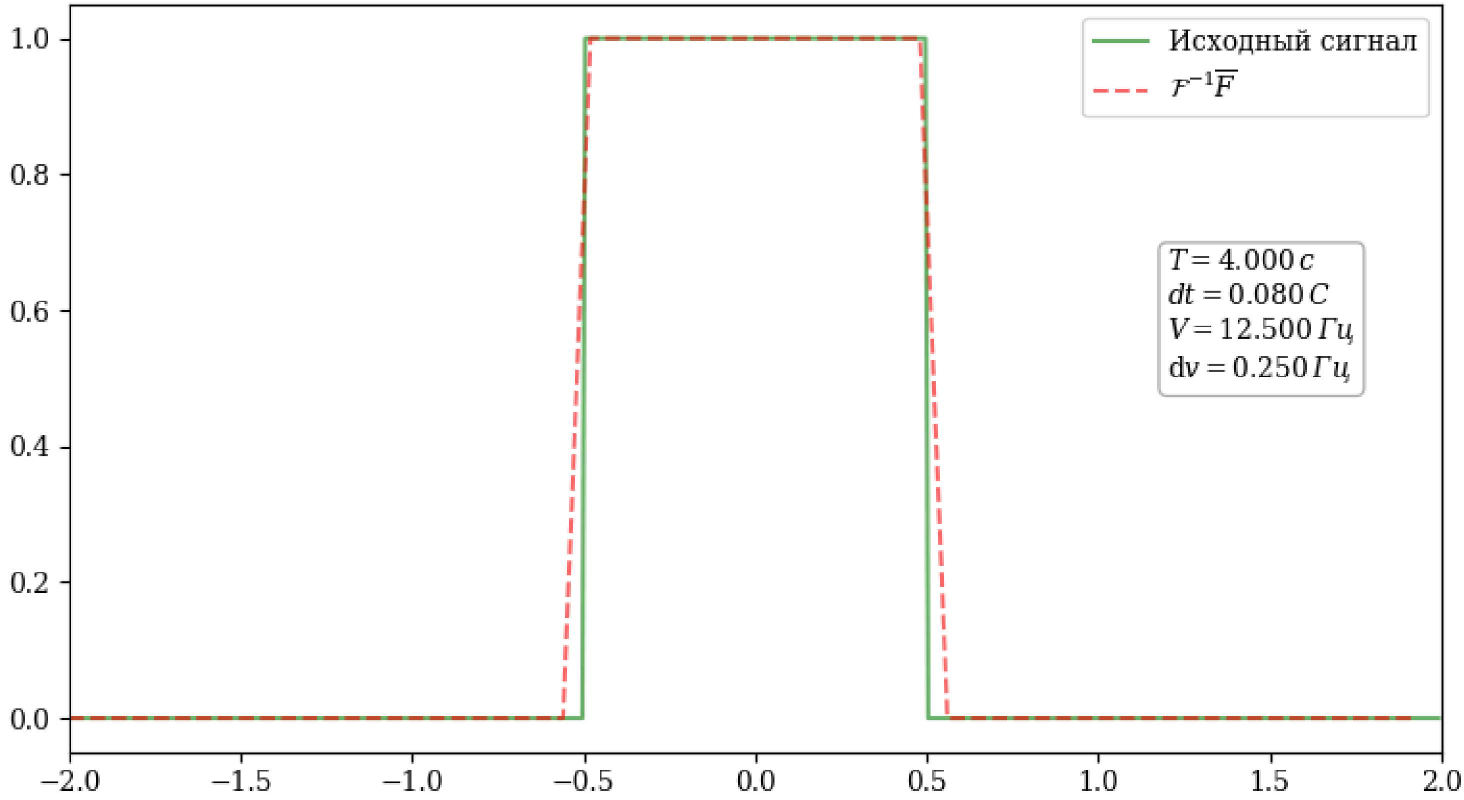

Рисунок 20 — Обратное «умное» DFT сигнала $\hat{\Pi}(\nu)$ при $T = 4$ $\mathrm{d}t = 0.08$ $V = 125$ $\mathrm{d}\nu = 0.25$

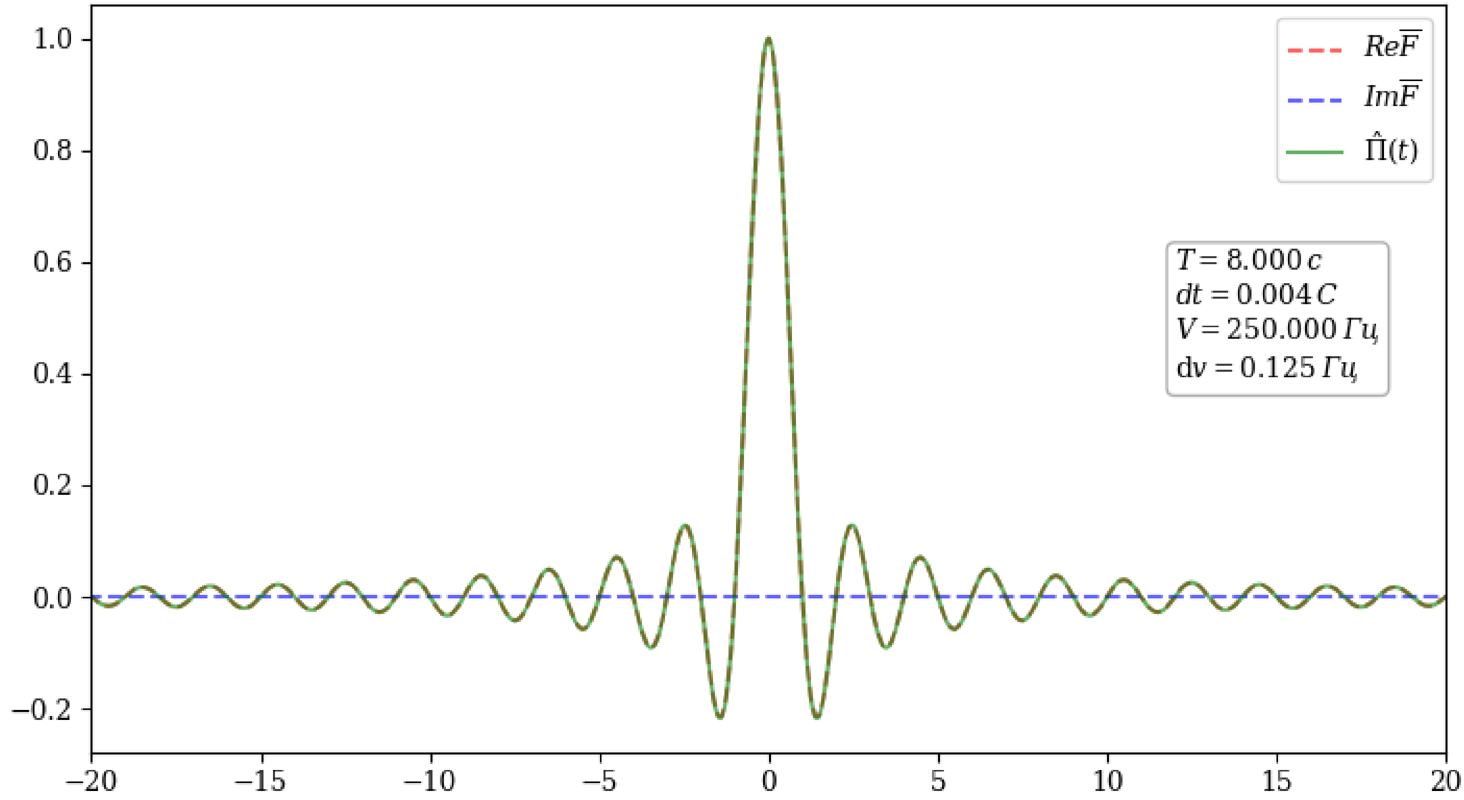

Рисунок 21 — «умное» DFT сигнала $\Pi(t)$
при $T = 8$ $\mathrm{d}t = 0.004$ $V = 250$ $\mathrm{d}\nu = 0.125$

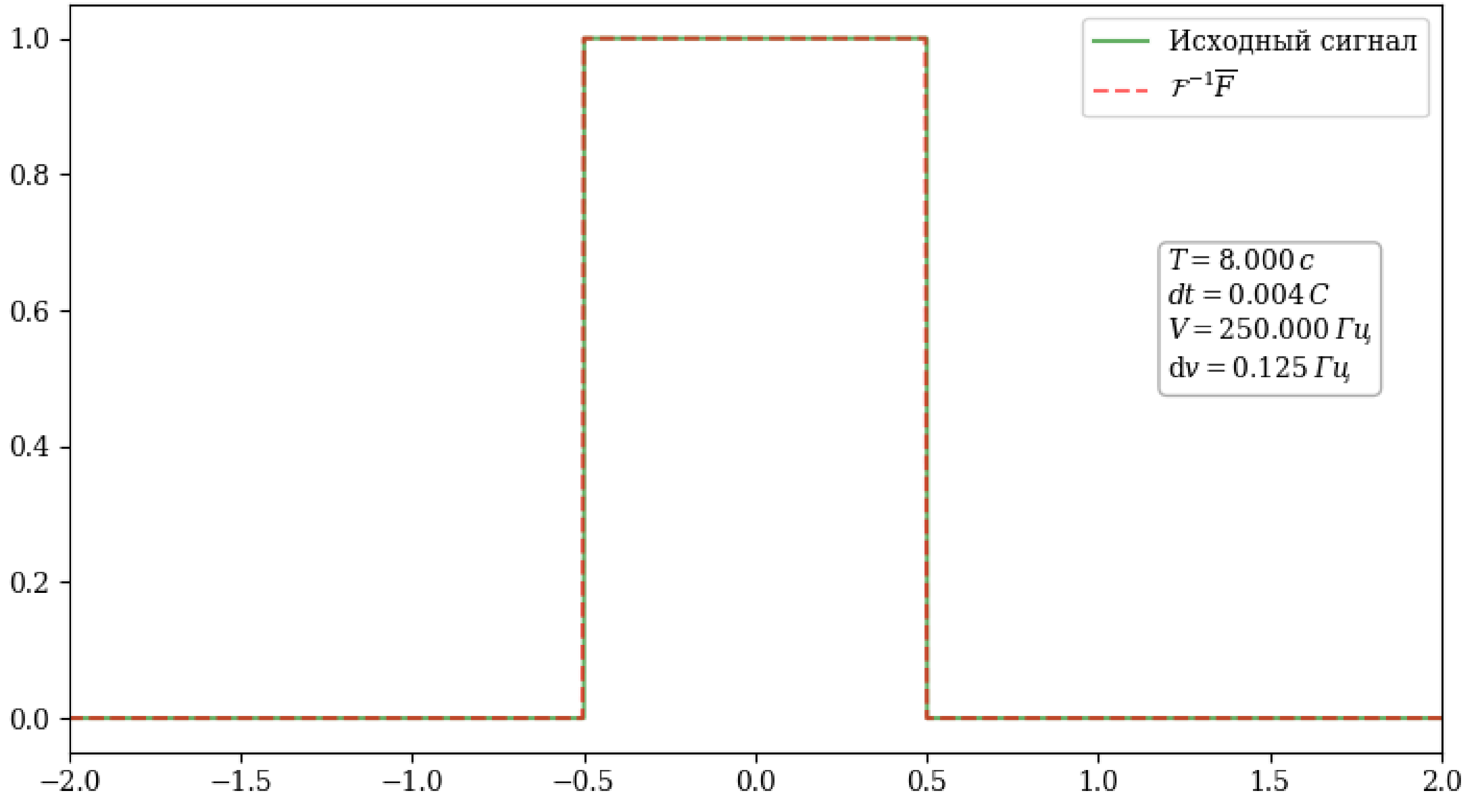

Рисунок 22 — Обратное «умное» DFT сигнала $\hat{\Pi}(\nu)$ при $T = 8$ d$t = 0.004$ $V = 250$ d$\nu = 0.125$

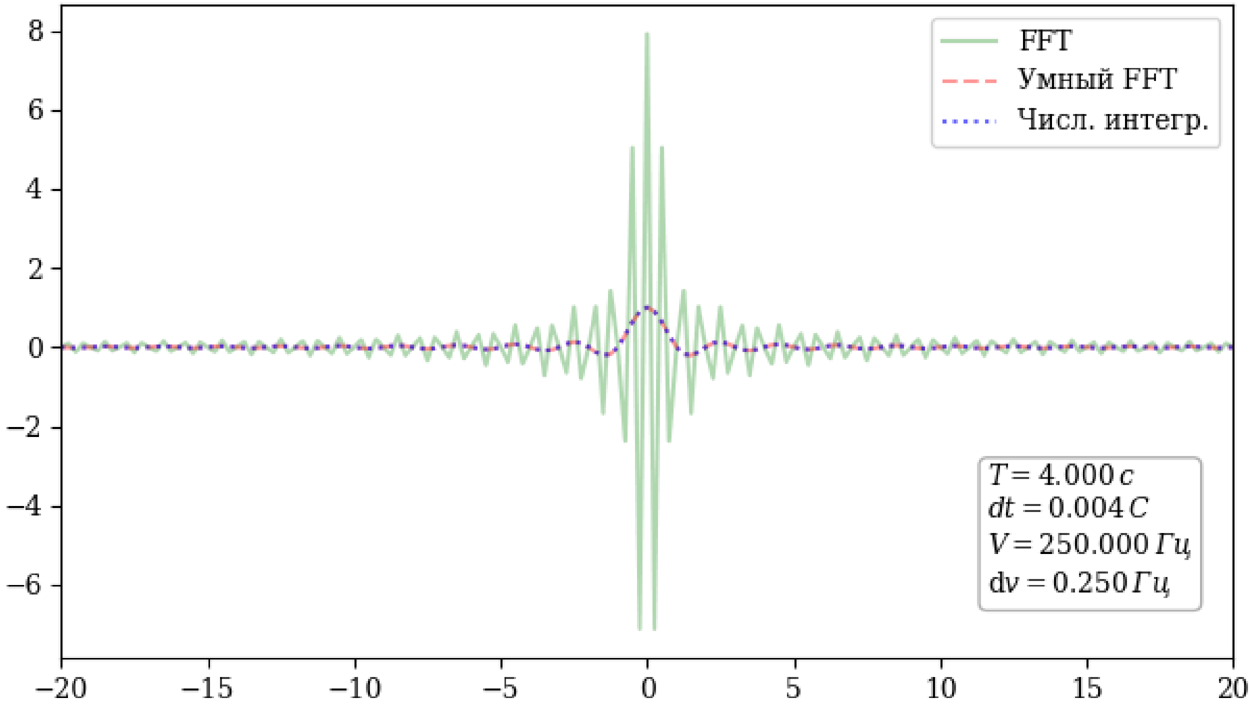

Рисунок 23 — Сранение спектров «умного» DFT, DFT и численного интегрирования

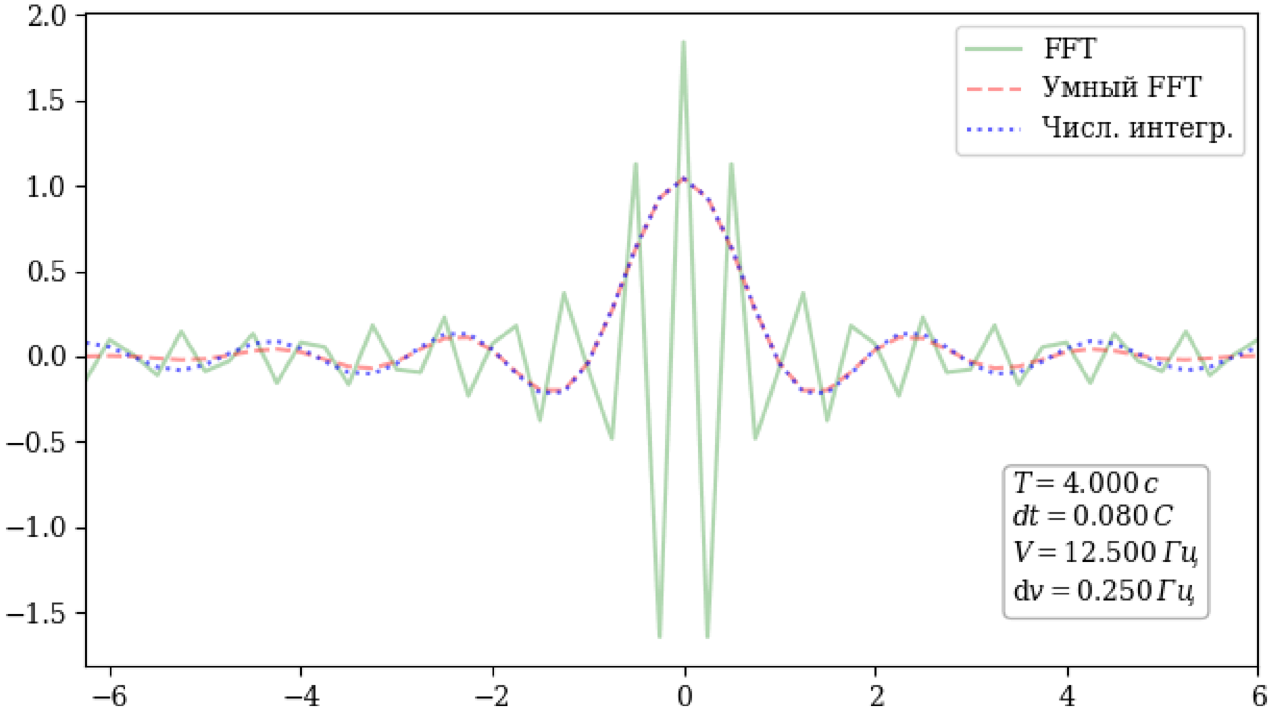

Рисунок 24 — Сранение спектров «умного» DFT, DFT и численного интегрирования

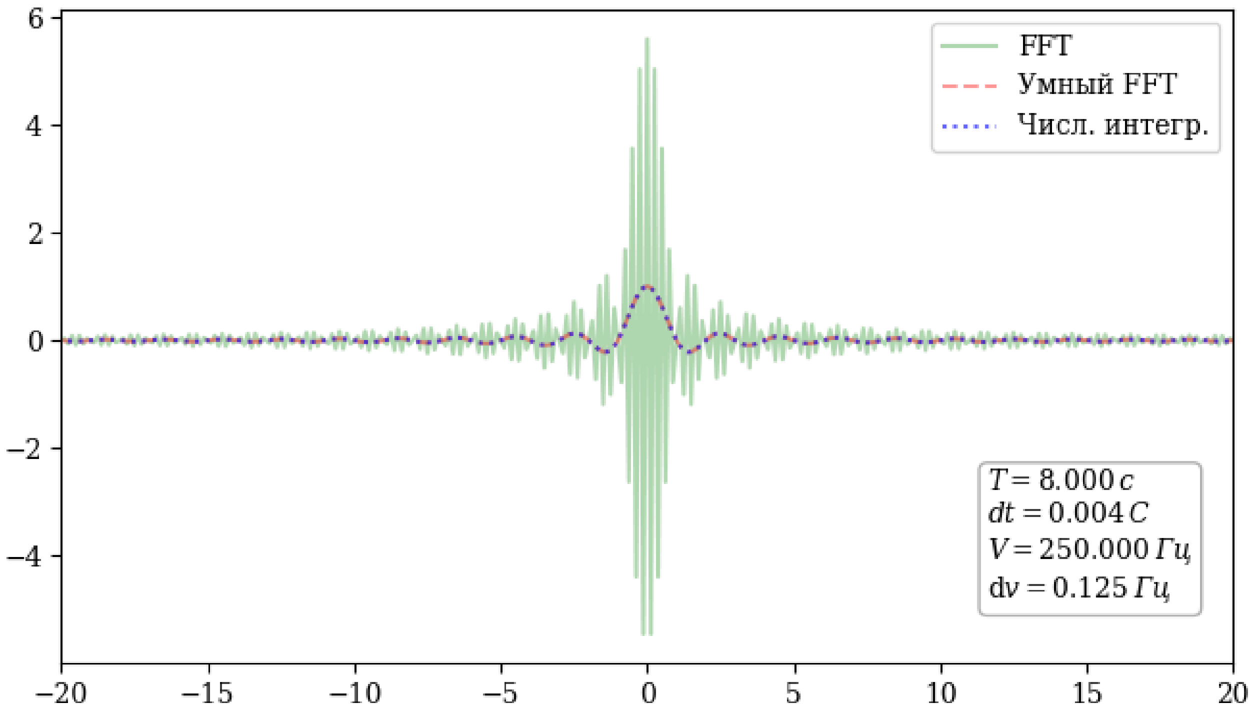

Рисунок 25 — Сранение спектров «умного» DFT, DFT и численного интегрирования

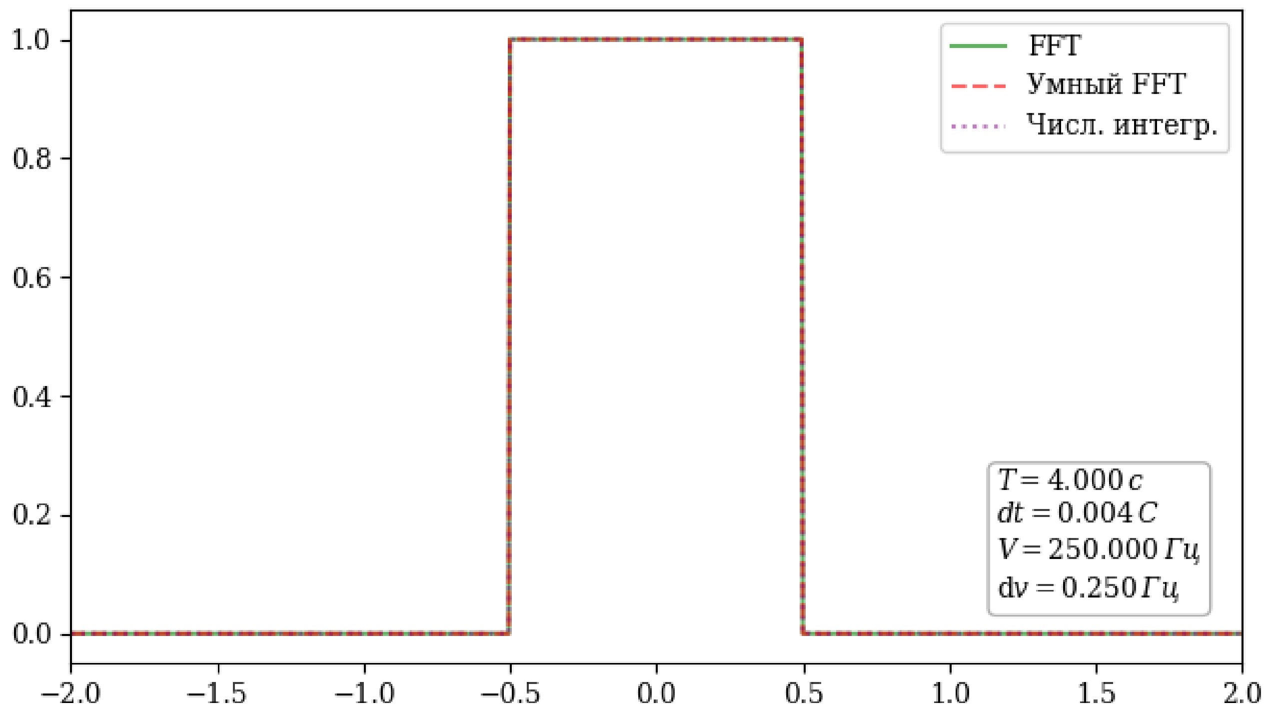

Рисунок 26 — Сранение восстановленного сигнала через «умного» DFT, DFT и численного интегрирования

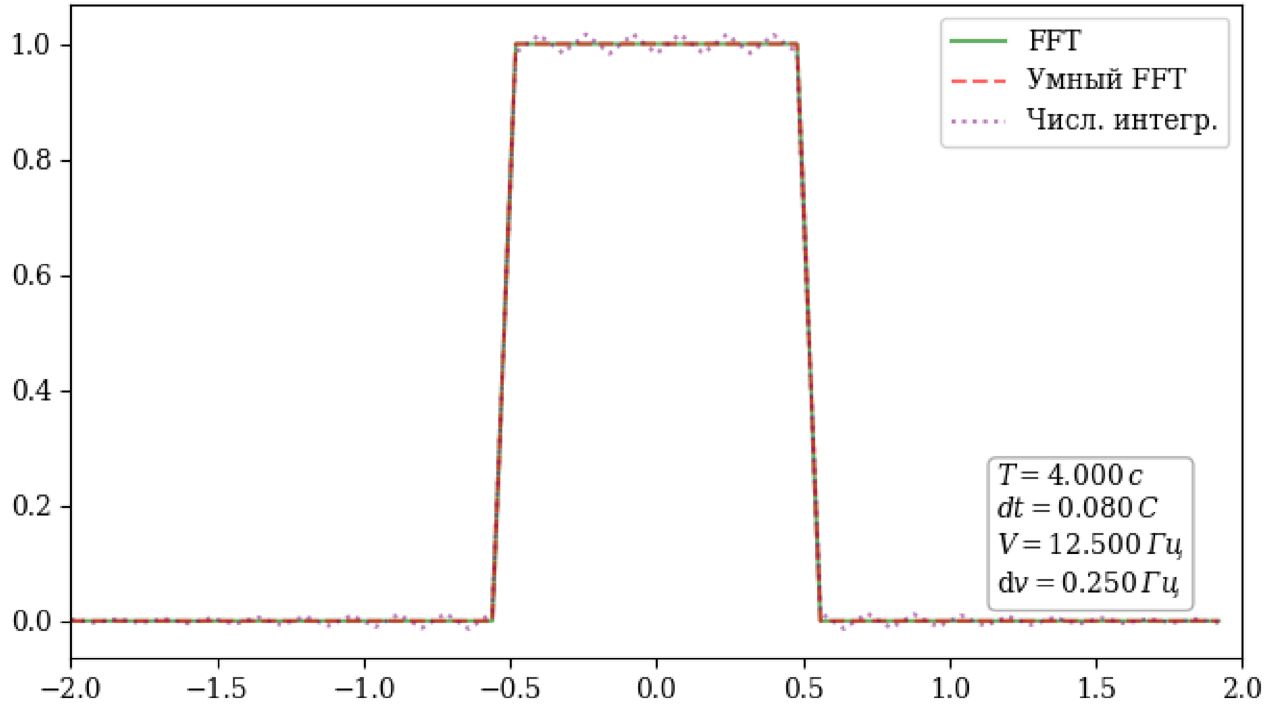

Рисунок 27 — Сранение восстановленного сигнала через «умного» DFT, DFT и численного интегрирования

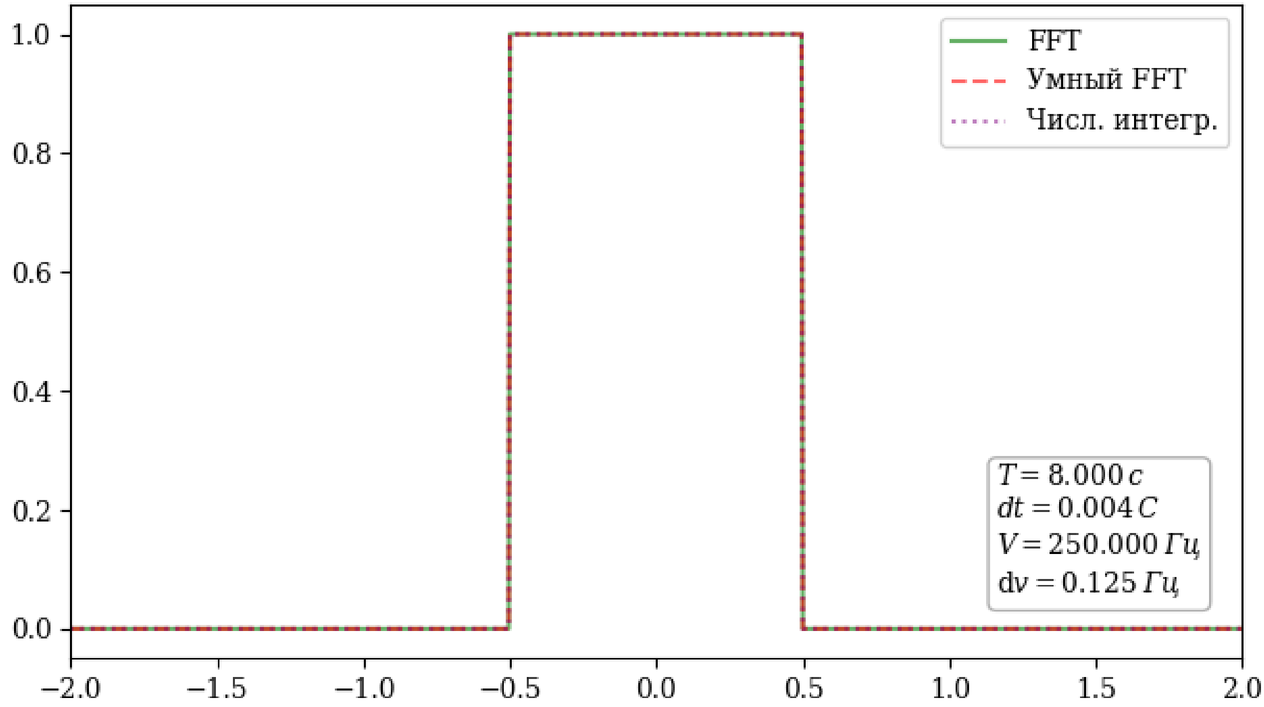

Рисунок 28 — Сранение восстановленного сигнала через «умного» DFT, DFT и численного интегрирования

1.4.2 Выводы

- Умножение на c_m компенсирует сдвиг по фазе, а Δt масштабирование спектра сигнала.
- Полученный метод сочетает скорость алгоритма FFT и точность численного интегрирвования.
- Спектр, полученный новым методом совпадает со спектром, полученным аналитическим образом.
- Восстановление сигнала полученным метод даёт возвращает исходный сигнал, в отличие от метода численного интегрирования, в котором накапливается ошибка при интегрировании.
- Величины при использовании нового метода V и $\mathrm{d}v$ зависят от параметров T и $\mathrm{d}t$ таким же образом, как и при использовании классического DFT.

2 Задание 2. Семплирование

В этом задании проверим теорему Найквиста-Шеннона-Котельникова о восстановлении семлированного сигнала. Формулировка теоремы:

Ограниченно-полосовой сигнал с максимальной частотой в спектре f_m можно восстановить из семплированного сигнала, если частота дискретизации $f_d \geq 2f_m = 2B$.

Рассмотрим 2 функции

$$y_1(t) = a_1 \sin(\omega_1 t + \varphi_1) + a_1 \sin(\omega_1 t + \varphi_1) \tag{17}$$

$$y_2(t) = \operatorname{sinc}(bt) \tag{18}$$

Определим знаечение параметра B для функций y_1 и y_2:

$$\hat{y}_1 = \frac{a_1}{2i}\left[\delta\left(\nu - \frac{\omega_1}{2\pi}\right) - \delta\left(\nu + \frac{\omega_1}{2\pi}\right)\right] + \frac{a_2}{2i}\left[\delta\left(\nu - \frac{\omega_2}{2\pi}\right) - \delta\left(\nu + \frac{\omega_2}{2\pi}\right)\right] \tag{19}$$

Получаем 2 пика на значениях частот $\pm f_i = \pm\omega_i/2\pi$. Тогда $B = \max(f_1, f_2)$.

$$\begin{gathered}\hat{y}_2 = \int_{-\infty}^{\infty} \frac{\sin(\pi bt)}{\pi b} = \frac{1}{|b|}\Pi\left(\frac{\nu}{b}\right)\\ \Pi(\nu) = \begin{cases} 1, & |\nu| \leq \frac{1}{2} \\ 0, & |\nu| > \frac{1}{2} \end{cases}\end{gathered} \tag{20}$$

Прямоугольная фукнция шириной b, тогда $B = \frac{1}{2}b$.

Зададим параметры функий

$$a_1 = 1, a_2 = 3, \omega_1 = 20\pi, \omega_2 = 30\pi, \varphi_1 = 2, \varphi_2 = 5, b = 10 \tag{21}$$

Рассмотрим различные пары параметров T и $\mathrm{d}t$ и как при них восстанавливается исходный сигнал.

2.1 Графики

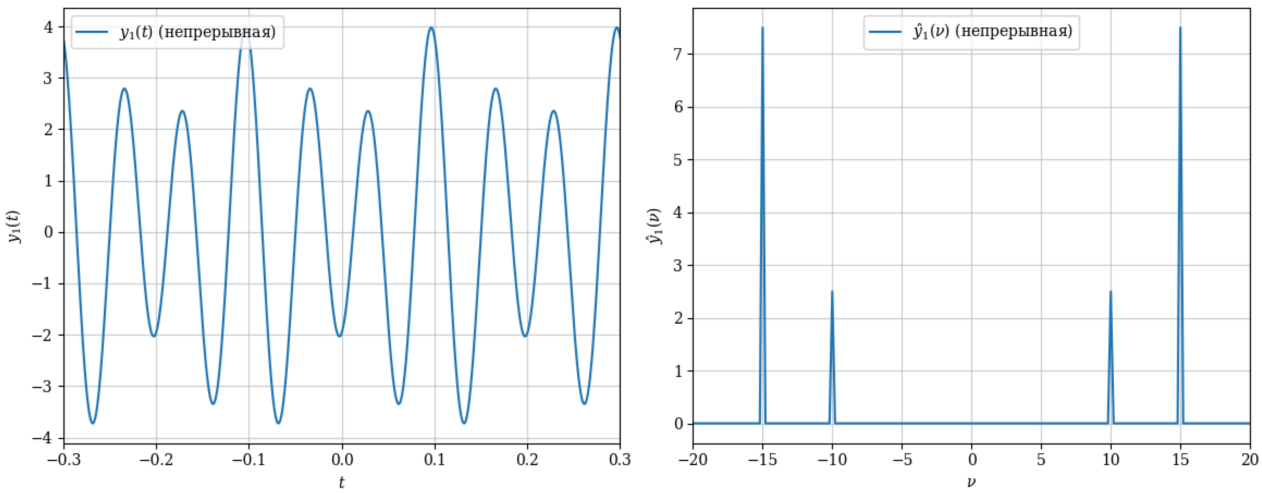

Рисунок 29 — График исходной непрерывной функции $y_1(t)$ и ее образа

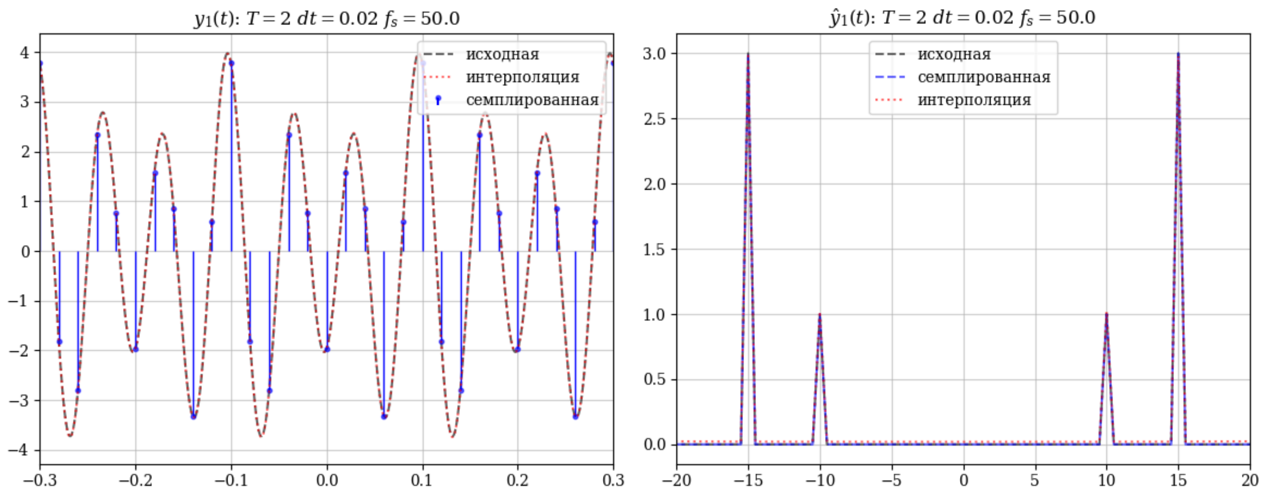

Рисунок 30 — Графики семплированной функции и интерполяции $y_1(t)$, а также ее спектра при $\mathrm{d}t = 0.02$ $T = 2$

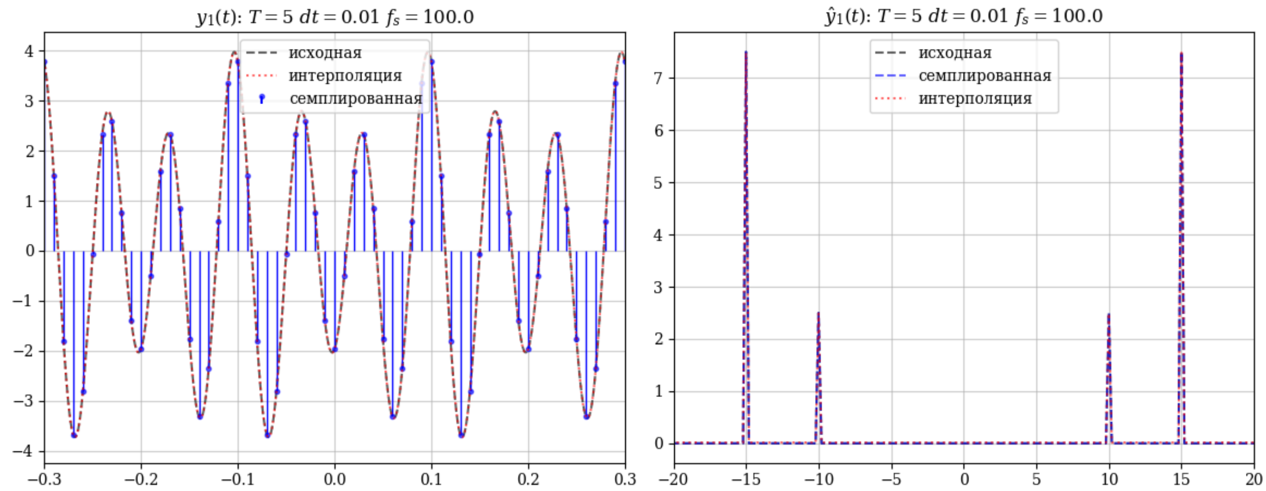

Рисунок 31 — Графики семплированной функции и интерполяции $y_1(t)$, а также ее спектра при $\mathrm{d}t = 0.01$ $T = 5$

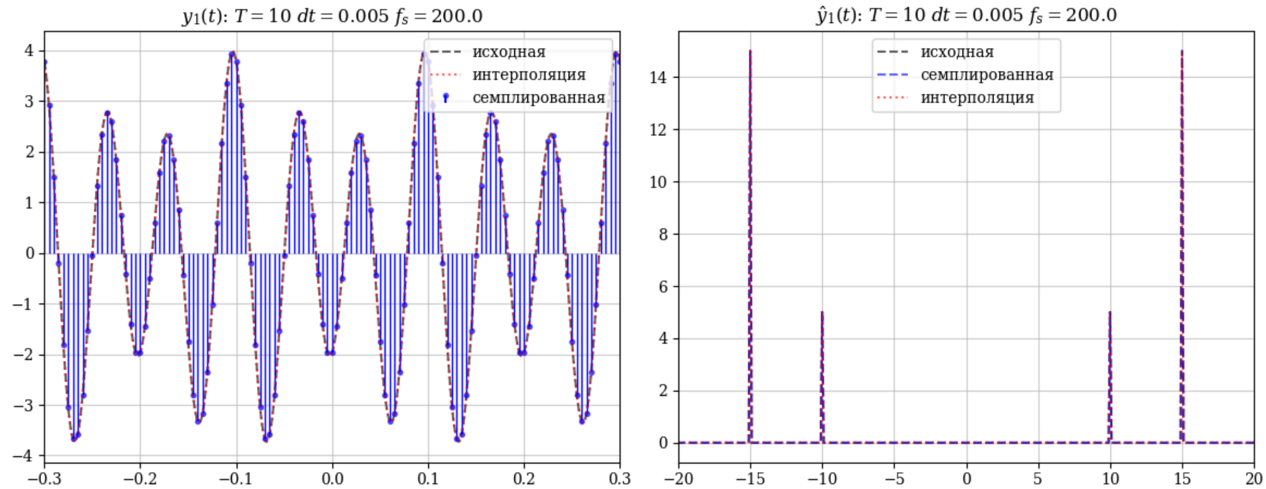

Рисунок 32 — Графики семплированной функции и интерполяции $y_1(t)$, а также ее спектра при $\mathrm{d}t = 0.005$ $T = 10$

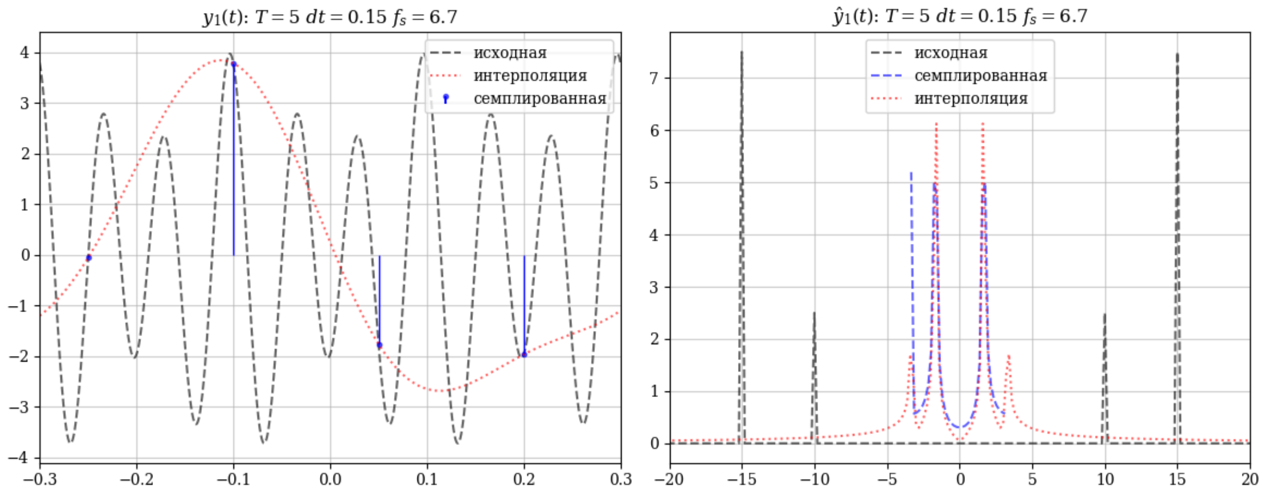

Рисунок 33 — Графики семплированной функции и интерполяции $y_1(t)$, а также ее спектра при $\mathrm{d}t = 0.15$ $T = 5$

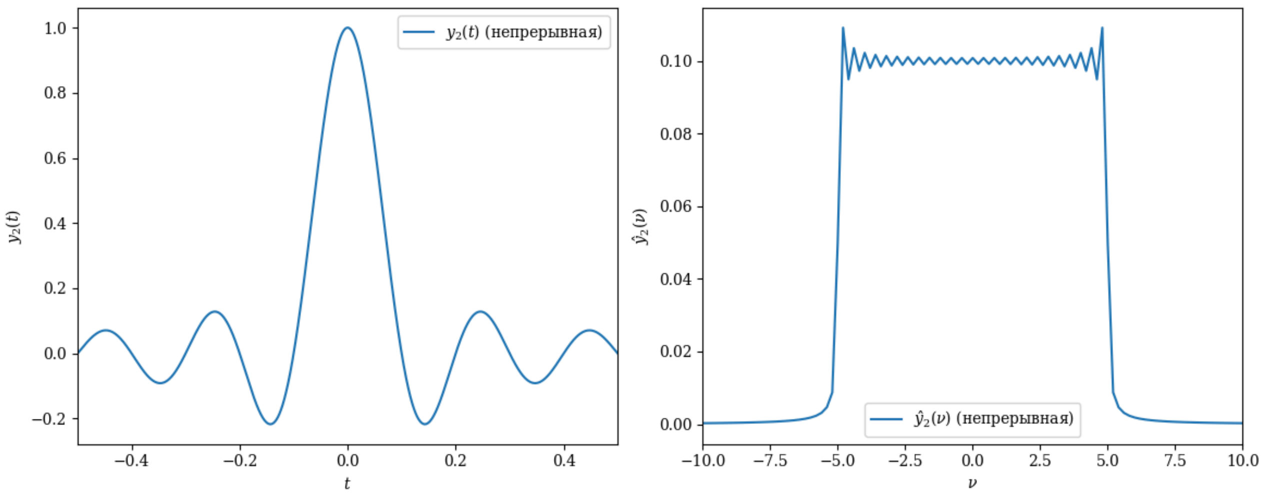

Рисунок 34 — График исходной непрерывной функции $y_2(t)$ и ее спектра

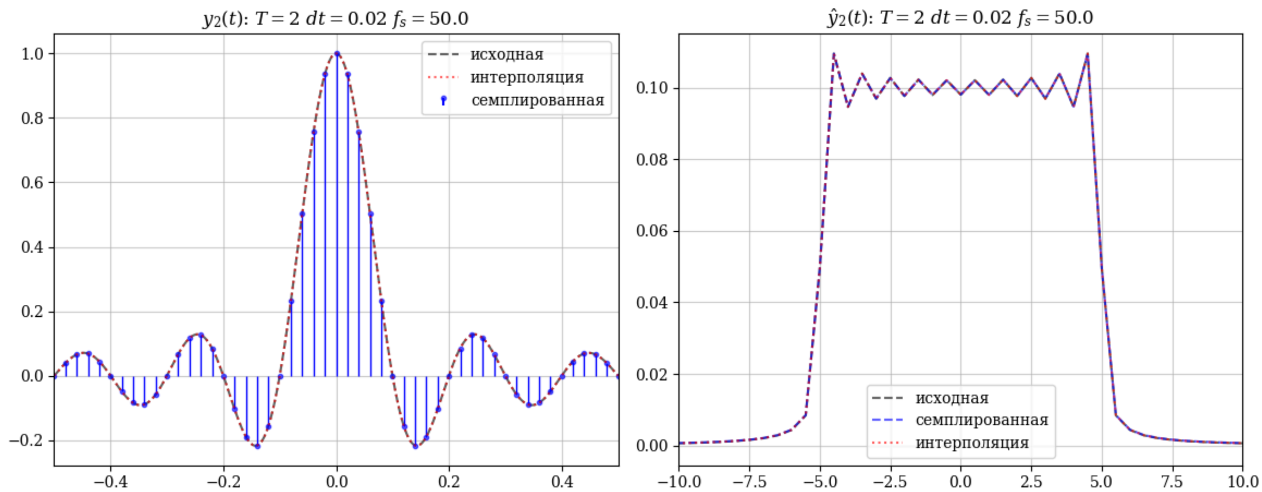

Рисунок 35 — Графики семплированной функции и интерполяции $y_2(t)$, а также ее спектра при $\mathrm{d}t = 0.02$ $T = 2$

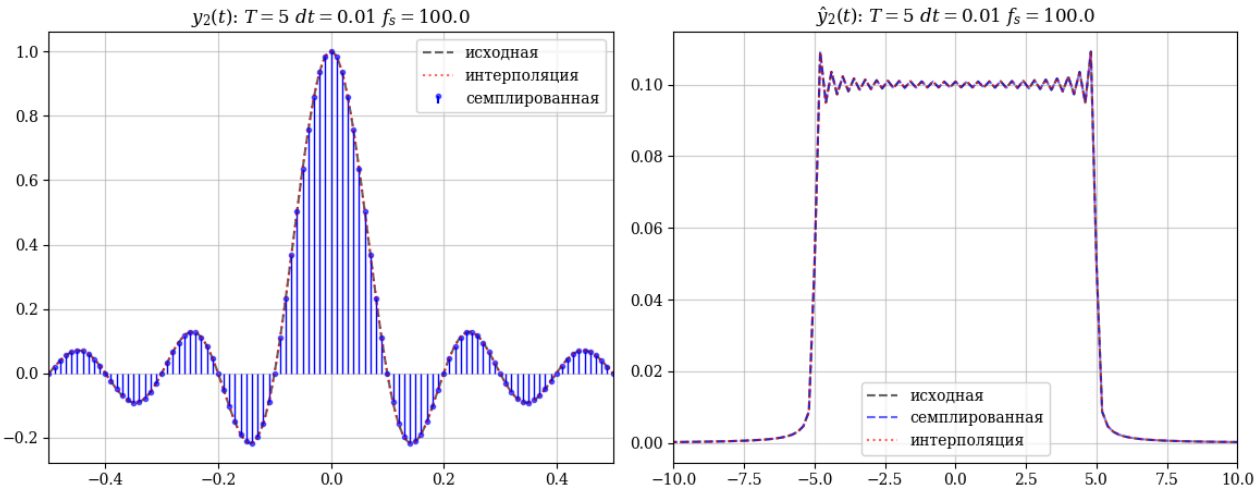

Рисунок 36 — Графики семплированной функции и интерполяции $y_2(t)$, а также ее спектра при $\mathrm{d}t = 0.01$ $T = 5$

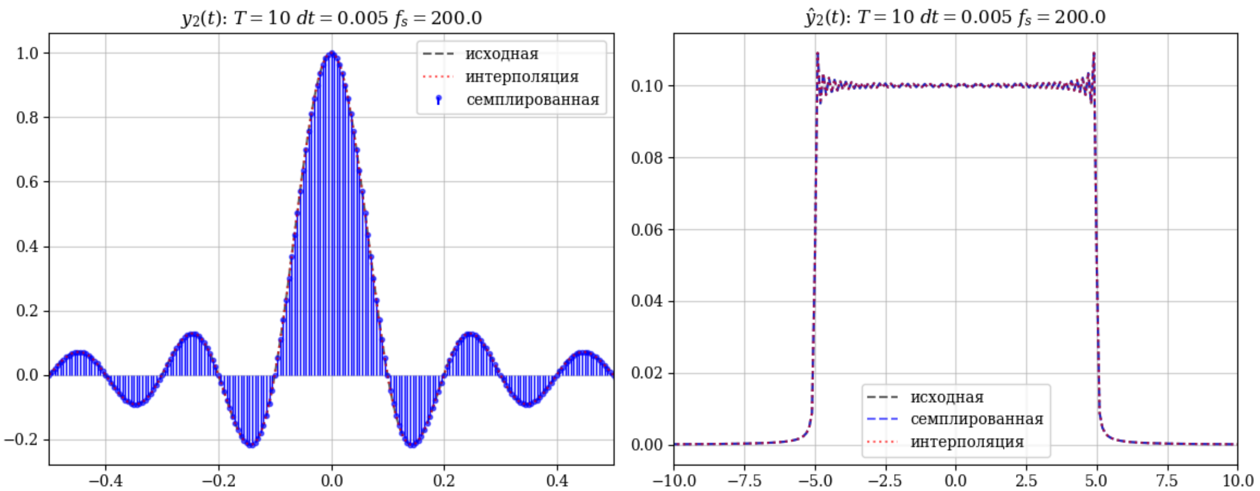

Рисунок 37 — Графики семплированной функции и интерполяции $y_2(t)$, а также ее спектра при $\mathrm{d}t = 0.005$ $T = 10$

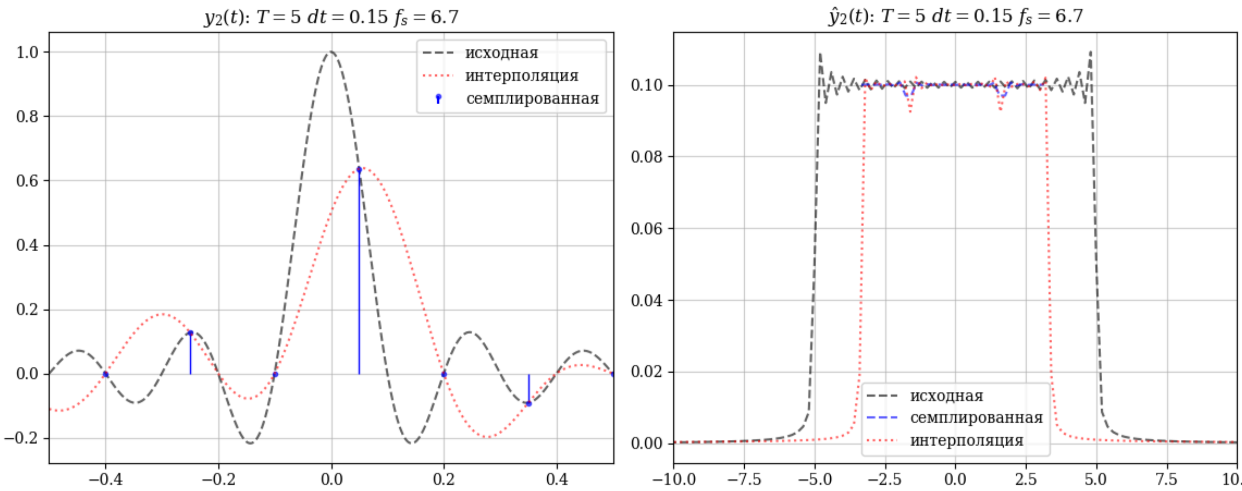

Рисунок 38 — Графики семплированной функции и интерполяции $y_2(t)$, а также ее спектра при $\mathrm{d}t = 0.15$ $T = 5$

2.2 Выводы

- Уменьшение Δt увеличивает ширину окна спектра V.
- Сигнал корректно восстанавливается интерполяцией если выполняется условие $\Delta t \leq 1/2B$, в обратном случае попытка восстаносить сигнал даёт плохой результат.
- Уменьшение Δt увеличивает ширину частотного окна V.
- Уменьшение Δt после $\Delta t = 1/2B$ не имеет смысла, так как непрерывный сигнал уже можно получить при помощи интерполяции.
- Увеличение временного окна T так же увеличивает разрешение спектра $\mathrm{d}\nu$.

3 Выводы

- В работе приблизили истинное преобразование Фурье при помощи численных методов, сравнили его с дискретным преобразованием Фурье. Выявили зависимость величин $V = \frac{1}{\mathrm{d}t}$ и $d\nu = \frac{1}{T}$ в случае с дискретным преобразованием.
- Сравнили быстродействие быстрого алгоритма DFT с методом численного интегрирования — Первый выполняется на несколько порядков быстрее при тех же входных данных.
- Добились приближения истинного фурье-образа Дискретным преобразованием при помощи коэффициента фазового сдвига $c_m = (-1)^m$
- Проверили теорему Котельникова о восстановлении семплированного сигнала — восстановили семплированный сигнал с частотой дискретизации $\Delta t \leq \frac{1}{2B}$, где B — максимальная частота в спектре сигнала; а также убедились в невозможности это сделать, если условие не выполняется.

4 Приложение

```python
import os
import functools
import time

import matplotlib.pyplot as plt
from matplotlib.axes import Axes
import numpy as np
from numpy.fft import fft, ifft, fftshift, ifftshift, fftfreq


def timeit(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Function {func.__name__} took {(end_time -
start_time)*1000:.4f} ms")
        return result

    return wrapper


plt.rcParams["figure.figsize"] = FIGSIZE = (
    200 / 25.4,
    200 / (16 / 9) / 25.4,
)  # 16 by 9 aspect & 200mm width

FIGSIZE_WIDE = (
    300 / 25.4,
    300 / 2.5 / 25.4,
)  # 5 by 2 aspect & 200mm width


plt.rcParams["mathtext.fontset"] = "dejavuserif"
plt.rcParams["font.family"] = "serif"
plt.rcParams["figure.dpi"] = 100

PI = np.pi
TAU = 2 * np.pi


def add_grid(ax: Axes):
    ax.grid(which="major", linestyle="-", alpha=0.6)
    ax.grid(which="minor", linestyle=":", alpha=0.4)
```

```python
# === Task 1 ===

t = np.linspace(-2, 2, 1000)

rect = lambda t: np.where(np.abs(t) < 0.5, 1, 0)
Pi = rect(t)

Pi_hat_true = np.sinc(t * PI)

os.makedirs("../fig/task1/", exist_ok=True)

plt.figure()
plt.plot(t, Pi)
plt.xlabel("t")
plt.ylabel(r"$\Pi(t)$")
plt.savefig(r"../fig/task1/Pi.png")
plt.close()

plt.figure()
plt.plot(t, Pi_hat_true)
plt.savefig(r"../fig/task1/Pi_hat_true.png")
plt.close()


# === Task 1.1
def trapz_ft(signal, t, v):
    dt = t[1] - t[0]
    y = signal * np.exp(-1j * TAU * v[..., np.newaxis] * t)
    integral = np.trapezoid(y, dx=dt, axis=1)

    assert integral.shape == v.shape
    return integral


def trapz_ift(ft, t, v):
    dv = v[1] - v[0]
    y = ft * np.exp(1j * TAU * t[..., np.newaxis] * v)
    integral = np.trapezoid(y, dx=dv, axis=1)

    assert integral.shape == t.shape
    return np.real(integral)


def task11(T, dt, V, dv):
    t = np.arange(-T / 2, T / 2, dt)
```

```
    v = np.arange(-V / 2, V / 2, dv)

    t_fine = np.arange(-T / 2, T / 2, dt / 10)
    v_fine = np.arange(-V / 2, V / 2, dv / 10)

    signal = rect(t)

    global i
    os.makedirs(f"../fig/task1/1.1/{i}", exist_ok=True)

    @timeit
    def perform_ft():
        ft = trapz_ft(signal, t, v)
        recon = trapz_ift(ft, t, v)
        return ft, recon

    ft_, recon_ = perform_ft()

    true_ft = np.sinc(v_fine)
    true_rect = rect(t_fine)

    # Ft
    plt.figure()
    plt.margins(0.05, tight=True)
    plt.xlabel(r"$\nu$")

    plt.plot(v, np.real(ft_), "--", label=r"$Re F(\nu)$")
    plt.plot(v, np.imag(ft_), label=r"$Im F(\nu)$")
    plt.plot(v_fine, true_ft, "-", color="grey", alpha=0.4, label=r"$
\hat{\Pi}(\nu)$")
    plt.legend()
    add_grid(plt.gca())
    plt.savefig(f"../fig/task1/1.1/{i}/F.png")
    plt.close()

    # Recon
    plt.figure()
    plt.xlim(t[0], t[-1])
    # plt.margins(0.05, tight=True)
    plt.xlabel(r"$t$")

    plt.plot(t, np.real(recon_), "r-", label=r"восстановленный")
    plt.plot(t_fine, true_rect, "g--", label=r"$\Pi(t)$")
    plt.legend()
    add_grid(plt.gca())

    plt.savefig(f"../fig/task1/1.1/{i}/f_recon.png")
```

```
    plt.close()


# === Task 1.2 ===
def task12(T, dt):
    global i
    os.makedirs(f"../fig/task1/1.2/{i}", exist_ok=True)

    t = np.arange(-T / 2, T / 2, dt)
    v = fftshift(fftfreq(t.size, dt))
    V = v[-1] - v[0]

    v_min = -20 if v[0] < -20 else v[0]
    v_max = 20 if v[-1] > 20 else v[-1]

    t_fine = np.arange(-T / 2, T / 2, dt / 10)
    signal = rect(t)
    signal_fine = rect(t_fine)

    v_fine = np.arange(-V / 2, V / 2, 1 / T / 10)
    sinc_fine = np.sinc(v_fine)

    @timeit
    def perform_fft():
        ft = fftshift(fft(signal, norm="ortho"))
        recon = ifft(ifftshift(ft), norm="ortho")
        return ft, recon

    ft_, recon_ = perform_fft()

    # FT
    plt.figure()
    plt.xlim(-V / 2, V / 2)
    plt.xlabel(r"$\nu$")

    plt.plot(v, np.real(ft_), label=r"$F(\nu)$")
    plt.plot(v_fine, sinc_fine, label=r"$\text{sinc}(\nu)$")
    plt.legend()
    add_grid(plt.gca())
    plt.savefig(f"../fig/task1/1.2/{i}/F.png")
    plt.close()

    # Recon
    plt.figure()
    plt.xlim((-T / 2, T / 2))
    plt.margins(0.05, tight=True)
    plt.xlabel(r"$t$")
```

```
    plt.plot(t_fine, signal_fine, label=r"$\Pi(t)$")
    plt.plot(t, np.real(recon_), "--", label="Восстановленный")
    add_grid(plt.gca())
    plt.legend()
    plt.savefig(f"../fig/task1/1.2/{i}/f_recon.png")
    plt.close()


# === Task 1.4 ===


def ft_fft_normalized(signal, t, v, c=None, return_c=False):
    T = t[-1] - t[0]
    dt = t[1] - t[0]

    c = c if c is not None else dt * np.exp(-1j * TAU * v * -T / 2)
    # c = c or dt * np.pow(-1, np.arange(0, v.size))

    ft = c * fftshift(fft(signal))

    if return_c:
        return ft, c
    return ft


def ift_fft_normalized(ft, t, v, c=None):
    T = t[-1] - t[0]
    dt = t[1] - t[0]

    c = c if c is not None else dt * np.exp(-1j * TAU * v * -T / 2)

    recon = ifft(ifftshift(ft / c))
    return np.real(recon)


def task141(T, dt):
    """Графики умного fft"""

    t = np.arange(-T / 2, T / 2, dt)
    t_fine = np.arange(-T / 2, T / 2, dt / 10)
    v = fftshift(fftfreq(t.size, dt))
    v_fine = np.arange(-1 / dt / 2, 1 / dt / 2, 1 / T / 10)
    sinc_fine = np.sinc(v_fine)

    signal = rect(t)
    signal_fine = rect(t_fine)
```

```
    ft_fft, c = ft_fft_normalized(signal, t, v, return_c=True)
    recon_fft = ift_fft_normalized(ft_fft, t, v, c)

    ft_raw = fftshift(fft(signal, norm="ortho"))
    recon_raw = ifft(ifftshift(ft_raw), norm="ortho")
    recon_raw = np.real(recon_raw)

    ft_trapz = trapz_ft(signal, t, v)
    recon_trapz = trapz_ift(ft_trapz, t, v)

    global i
    os.makedirs(f"../fig/task1/1.4/new_method/{i}", exist_ok=True)

    params_text = """\
$T = {T:.3f}\\:c$
$dt = {dt:.3f}\\:C$
$V = {V:.3f}\\:Гц$
$\\mathrm{{d}} v = {dv:.3f}\\:Гц$"""
    params_text = params_text.format(T=T, dt=dt, V=1 / (dt), dv=1 / T)

    v_min = -20 if v.min() < -20 else v.min()
    v_max = 20 if v.max() > 20 else v.max()

    # F
    plt.figure()
    plt.xlim(v_min, v_max)
    plt.plot(v, np.real(ft_fft), "r--", alpha=0.6, label=r"$Re
\overline{F}$")
    plt.plot(v, np.imag(ft_fft), "b--", alpha=0.6, label=r"$Im
\overline{F}$")
    plt.plot(v_fine, sinc_fine, "g-", alpha=0.6, label=r"$\hat{\Pi}
(t)$")
    plt.text(
        0.80,
        0.50,
        params_text,
        transform=plt.gca().transAxes,
        bbox=dict(
            boxstyle="round,pad=.3", edgecolor="grey",
facecolor="white", alpha=0.6
        ),
    )
    plt.legend()
    plt.tight_layout()
    plt.savefig(f"../fig/task1/1.4/new_method/{i}/F.png")
    plt.close()
```

```
    # Recon
    plt.figure()
    plt.xlim((-2, 2))
    plt.plot(t_fine, signal_fine, "g-", alpha=0.6, label=r"Исходный
сигнал")
    plt.plot(
        t, np.real(recon_fft), "r--", alpha=0.6, label=r"$\mathcal{F}
^{-1}\overline{F}$"
    )
    plt.text(
        0.80,
        0.50,
        params_text,
        transform=plt.gca().transAxes,
        bbox=dict(
            boxstyle="round,pad=.3", edgecolor="grey",
facecolor="white", alpha=0.6
        ),
    )
    plt.legend()
    plt.tight_layout()
    plt.savefig(f"../fig/task1/1.4/new_method/{i}/recon.png")
    plt.close()

    os.makedirs("../fig/task1/1.4/F_comp/", exist_ok=True)

    v_min = -20 if v.min() < -20 else v.min()
    v_max = 20 if v.max() > 20 else v.max()

    # Compare new FFT, raw FFT, trapz
    plt.figure()
    plt.xlim(v_min, v_max)
    plt.plot(v, np.real(ft_raw), "g-", alpha=0.3, label="FFT")
    plt.plot(v, np.real(ft_fft), "r--", alpha=0.4, label="Умный FFT")
    plt.plot(v, np.real(ft_trapz), "b:", alpha=0.6, label="Числ.
интегр.")
    plt.text(
        0.79,
        0.10,
        params_text,
        transform=plt.gca().transAxes,
        bbox=dict(
            boxstyle="round,pad=.3", edgecolor="grey",
facecolor="white", alpha=0.6
        ),
    )
```

```
    plt.legend()
    plt.savefig(f"../fig/task1/1.4/F_comp/{i}.png")
    plt.close()

    os.makedirs(f"../fig/task1/1.4/recon_comp/", exist_ok=True)

    # Compare Recons
    plt.figure()
    plt.xlim(-2, 2)
    plt.plot(t, recon_raw, "g-", alpha=0.6, label="FFT")
    plt.plot(t, recon_fft, "r--", alpha=0.6, label="Умный FFT")
    plt.plot(t, recon_trapz, ":", color="purple", alpha=0.5,
label="Числ. интегр.")
    plt.text(
        0.79,
        0.10,
        params_text,
        transform=plt.gca().transAxes,
        bbox=dict(
            boxstyle="round,pad=.3", edgecolor="grey",
facecolor="white", alpha=0.6
        ),
    )
    plt.legend()
    plt.savefig(f"../fig/task1/1.4/recon_comp/{i}.png")
    plt.close()


# === Task 2 ===


def y1_func(t, a1, a2, phi1, phi2, w1, w2):
    return a1 * np.sin(w1 * t + phi1) + a2 * np.sin(w2 * t + phi2)


def y2_func(t, b):
    return np.sinc(b * t)


def task21(T=5, dt=1e-4):
    """Построить графики непрерывных функций"""

    os.makedirs("../fig/task2", exist_ok=True)

    global a1, a2, phi1, phi2, w1, w2
    global b
    params = [a1, a2, phi1, phi2, w1, w2]
```

```
t = np.arange(-T / 2, T / 2, dt)
y1_cont = y1_func(t, *params)
y2_cont = y2_func(t, b)

v = fftshift(fftfreq(t.size, dt))

y1_ft = ft_fft_normalized(y1_cont, t, v)
y2_ft = ft_fft_normalized(y2_cont, t, v)

# y1
plt.figure(figsize=FIGSIZE_WIDE)
plt.subplot(1, 2, 1)
plt.plot(t, y1_cont, label=r"$y_1(t)$ (непрерывная)")
plt.xlabel(r"$t$")
plt.ylabel(r"$y_1(t)$")
plt.xlim([-0.3, 0.3])
plt.legend()
add_grid(plt.gca())

plt.subplot(1, 2, 2)
plt.plot(v, np.abs(y1_ft), label=r"$\hat{y}_1(\nu)$ (непрерывная)")
plt.xlabel(r"$\nu$")
plt.ylabel(r"$\hat{y}_1(\nu)$")
plt.xlim([-20, 20])
plt.legend()
add_grid(plt.gca())

plt.tight_layout()
plt.savefig("../fig/task2/y1_cont.png")
plt.close()

# y2
plt.figure(figsize=FIGSIZE_WIDE)
plt.subplot(1, 2, 1)
plt.plot(t, y2_cont, label=r"$y_2(t)$ (непрерывная)")
plt.xlabel(r"$t$")
plt.ylabel(r"$y_2(t)$")
plt.xlim([-0.5, 0.5])
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(v, np.abs(y2_ft), label=r"$\hat{y}_2(\nu)$ (непрерывная)")
plt.xlabel(r"$\nu$")
plt.ylabel(r"$\hat{y}_2(\nu)$")
plt.xlim([-10, 10])
plt.legend()
```

```
    plt.tight_layout()
    plt.savefig("../fig/task2/y2_cont.png")
    plt.close()


def interp(ts, ys, t, dt):
    """Интерполировать семплированный сигнал `y`
    в моментах `ts` до времени `t`"""
    assert ts.size == ys.size
    x = (t[None, :] - ts[:, None]) / dt  # (N, M)
    return ys @ np.sinc(x)  # N @ (N, M) -> M


def test_inter():
    dt = 0.05
    t = np.arange(-1, 1, 0.001)
    ts = np.arange(-1, 1, dt)

    y = np.sin(TAU * 5 * t)
    ys = np.sin(TAU * 5 * ts)

    plt.figure()
    plt.plot(t, y, alpha=0.3, color="black")
    plt.stem(ts, ys)
    y_rec = interp(ts, ys, t, dt)

    assert t.size == y_rec.size
    plt.plot(t, y_rec, color="red", linestyle="--")

    plt.show()
    plt.close()


def task22(T=5, dt=1e-4):
    """Построить графики сеплированных функций"""
    global i

    global a1, a2, phi1, phi2, w1, w2
    global b
    params = [a1, a2, phi1, phi2, w1, w2]

    fig_root = f"../fig/task2/{i}"
    print(fig_root)
    os.makedirs(fig_root, exist_ok=True)

    t_cont = np.arange(-T / 2, T / 2, 1e-4)
```

```
y1_cont = y1_func(t_cont, *params)
y2_cont = y2_func(t_cont, b)
v_cont = fftshift(fftfreq(t_cont.size, 1e-4))
y1_cont_ft = ft_fft_normalized(y1_cont, t_cont, v_cont)
y2_cont_ft = ft_fft_normalized(y2_cont, t_cont, v_cont)

t_samp = np.arange(-T / 2, T / 2, dt)
y1_samp = y1_func(t_samp, *params)
y2_samp = y2_func(t_samp, b)
v_samp = fftshift(fftfreq(t_samp.size, dt))
y1_samp_ft = ft_fft_normalized(y1_samp, t_samp, v_samp)
y2_samp_ft = ft_fft_normalized(y2_samp, t_samp, v_samp)

y1_int = interp(t_samp, y1_samp, t_cont, dt)
y2_int = interp(t_samp, y2_samp, t_cont, dt)
y1_int_ft = ft_fft_normalized(y1_int, t_cont, v_cont)
y2_int_ft = ft_fft_normalized(y2_int, t_cont, v_cont)

y1_labels = ["исходная", "семплированная", "интерполяция"]
y2_labels = y1_labels

for j, (
    y_samp,
    y_cont,
    y_int,
    labels,
    limits,
    y_ft_cont,
    y_ft_samp,
    y_ft_int,
    limits2,
) in enumerate(
    [
        (
            y1_samp,
            y1_cont,
            y1_int,
            y1_labels,
            [-0.3, 0.3],
            y1_cont_ft,
            y1_samp_ft,
            y1_int_ft,
            [-20, 20],
        ),
        (
            y2_samp,
            y2_cont,
```

```
                y2_int,
                y2_labels,
                [-0.5, 0.5],
                y2_cont_ft,
                y2_samp_ft,
                y2_int_ft,
                [-10, 10],
            ),
        ],
        start=1,
    ):

        title_text1 = f"$y_{{{j}}}(t)$: $T={T}$ $dt={dt}$ $f_s={1/
dt:.1f}$"
        title_text2 = f"$\\hat{{y}}_{{{j}}}(t)$: $T={T}$ $dt={dt}$
$f_s={1/dt:.1f}$"

        plt.figure(figsize=FIGSIZE_WIDE)
        plt.subplot(1, 2, 1)
        plt.plot(
            t_cont,
            y_cont,
            label=labels[0],
            linestyle="--",
            color="black",
            alpha=0.6,
        )
        ml, sl, bl = plt.stem(
            t_samp,
            y_samp,
            "b-",
            markerfmt="bo",
            basefmt=" ",
            label=labels[1],
        )
        plt.setp(ml, markersize=3, alpha=0.6)
        plt.setp(bl, alpha=0)
        plt.setp(sl, linewidth=1)

        plt.plot(t_cont, y_int, "r:", label=labels[2], alpha=0.6)

        add_grid(plt.gca())
        plt.xlim(limits)
        plt.title(title_text1)
        plt.legend()

        plt.subplot(1, 2, 2)
```

```
        plt.plot(
            v_cont,
            np.abs(y_ft_cont),
            label=labels[0],
            color="black",
            linestyle="--",
            alpha=0.6,
        )
        plt.plot(
            v_samp,
            np.abs(y_ft_samp),
            label=labels[1],
            color="blue",
            linestyle="--",
            alpha=0.6,
        )
        plt.plot(
            v_cont,
            np.abs(y_ft_int),
            label=labels[2],
            color="red",
            linestyle=":",
            alpha=0.6,
        )

        add_grid(plt.gca())
        plt.xlim(limits2)
        plt.legend()

        plt.title(title_text2)
        plt.tight_layout()
        plt.savefig(f"../fig/task2/{i}/y{j}_comp.png")
        plt.close()


if __name__ == "__main__":
    # === Task 1.1 ===
    for i, (T, dt, V, dv) in enumerate(
        (
            (2, 0.005, 10, 0.05),
            (5, 0.005, 60, 0.05),  # small T
            (2, 0.05, 10, 0.5),  # small V  -> recon smoother
            (5, 0.05, 60, 0.5),  # large dt -> ft repeants & recon wrong
        )
    ):
        print(f"task 1.1 iteration {i}")
        task11(T, dt, V, dv)
```

```python
    # === Task 1.2 ===
    for i, (T, dt) in enumerate(
        (
            (4, 0.005),
            (4, 0.05),  # big dt -> small V
            (8, 0.005),  # big T -> small dv
        )
    ):
        print(f"task 1.2 iteration {i}")
        task12(T, dt)

    # === Task 1.4.1 ===
    for i, (T, dt) in enumerate(
        (
            (4, 0.004),
            (4, 0.080),  # small dt -> small V   NOTE: V = 1/(dt)
            (8, 0.004),  # big T -> small dv     NOTE: dv = 1/T
        )
    ):
        task141(T, dt)

    # === Task 2 ===
    a1, a2, phi1, phi2, w1, w2 = params = 1, 3, 2, 5, TAU * 10, TAU * 15
    b = 10

    # == original functions
    task21()

    # comparison
    for i, (T, dt) in enumerate(
        (
            (2, 0.02),
            (5, 0.01),
            (10, 0.005),
            (5, 0.15),
        )
    ):
        task22(T, dt)
```

Листинг 1 — Исходный код лабораторной работы

```python
import os
import functools
import time

import matplotlib.pyplot as plt
from matplotlib.axes import Axes
```

```
import numpy as np
from numpy.fft import fft, ifft, fftshift, ifftshift, fftfreq

TAU = np.pi * 2


def rect(t):
    return np.where(np.abs(t) <= 0.5, 1, 0)


def timeit(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Function {func.__name__} took {(end_time -
start_time)*1000:.4f} ms")
        return result

    return wrapper


def trapz_ft(signal, t, v):
    dt = t[1] - t[0]
    y = signal * np.exp(-1j * TAU * v[..., np.newaxis] * t)
    integral = np.trapezoid(y, dx=dt, axis=1)

    assert integral.shape == v.shape
    return integral


@timeit
def perform_ft(signal, t):
    v = fftshift(fftfreq(t.size, t[1] - t[0]))
    _ = trapz_ft(signal, t, v)


@timeit
def perform_fft(signal, t):
    _ = fftshift(fft(signal, norm="ortho"))


for T, dt in (
    (5, 0.005),
    (10, 0.005),
    (5, 0.0005),
```

```
):
    t = np.arange(-T / 2, T / 2, dt)
    print(f"N = {t.size}")
    signal = rect(t)
    perform_ft(signal, t)
    perform_fft(signal, t)
```

Листинг 2 — Сравнение быстродействия алгоритмов FFT и численного интегрирования