Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного автономного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

ДОМАШНЯ РАБОТА 2

«ПРЕОБРАЗОВАНИЕ ИЗОБРАЖЕНИЙ, НАЛОЖЕНИЕ ШУМА НА ИЗОБРАЖЕНИЕ И ФИЛЬТРАЦИЯ ИЗОБРАЖЕНИЙ»

ДИСЦИПЛИНА: «Цифровая обработка сигналов»

Выполнил: студент гр. ИУК4-72Б _____ (____Губин Е.В._____)
(Подпись) (Ф.И.О.)

Проверил: _____ (____Чурилин О.И___)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2025

Цель: Получение практических навыков доступа к объектам графики в системе MatLab. Сравнительный анализ различных преобразований. Получение практических навыков использования двумерного преобразования Фурье при исследовании диапазона яркости изображения. Получение практических навыков наложения на изображение шума и фильтрации изображения.

Задачи:

1. Вывести изображение в графическое окно.
2. Осуществить прямое и обратное косинусное преобразование над изображением. Вывести в графическое окно результаты преобразований.
3. Осуществить прямое и обратное преобразование Фурье над изображением. Вывести в графическое окно результаты преобразований.
4. Определить коэффициент корреляции между исходным изображением и изображениями, полученными в результате обратных преобразований.
5. Выявить какое из полученных изображений менее всего отличается от оригинала.
6. Построить график зависимости спектра яркости от частоты.
7. На исходное изображение наложить различного рода шум, согласно варианту.
8. Произвести фильтрацию исходного изображения, согласно варианту.
9. Производить фильтрацию для трех различных масок фильтров.
10. С помощью коэффициента корреляции оценить действие каждого из фильтров с учетом размера маски. Построить графики зависимости коэффициента корреляции от размера маски.

Вариант №3

Ход выполнения работы:

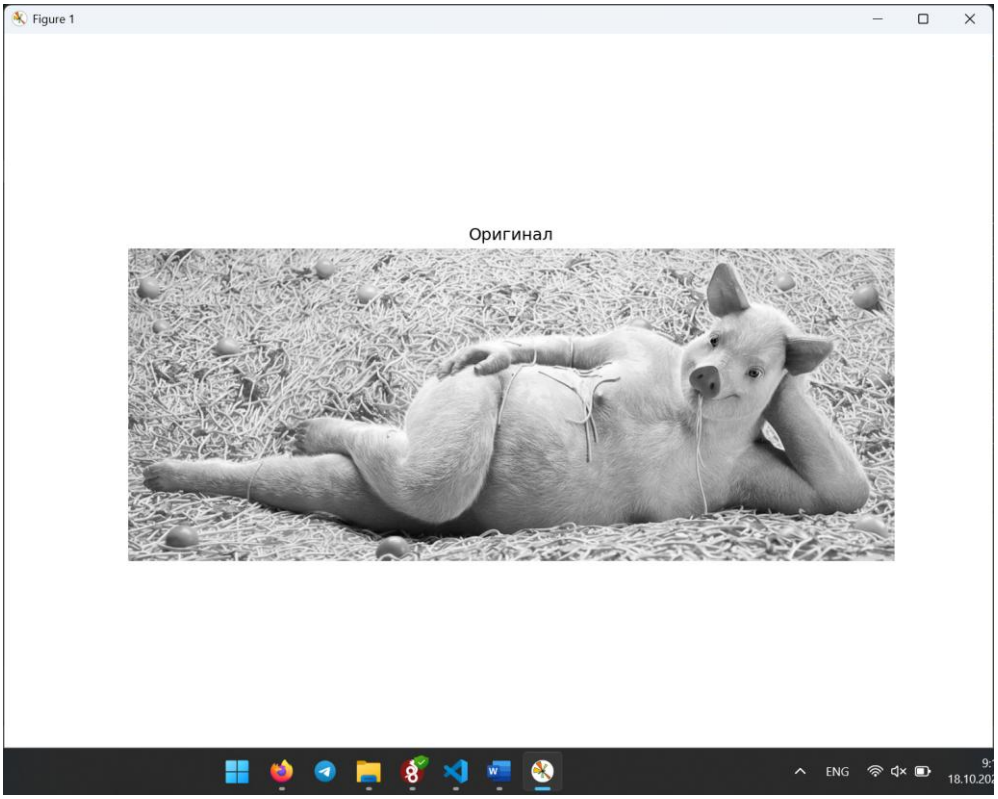

Рисунок 1 Оригинал

Рисунок 2 реобразование по DCT и его реконструкция

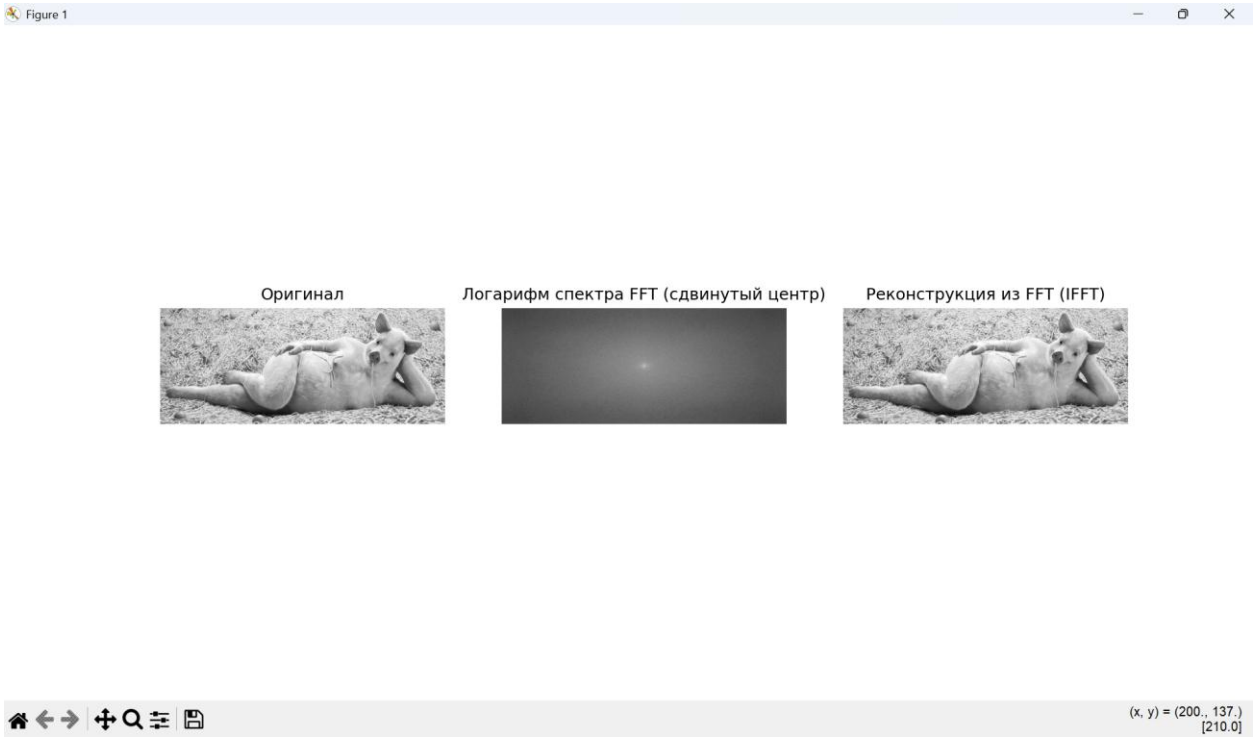

Рисунок 3 ДПФ и обратный ДПФ

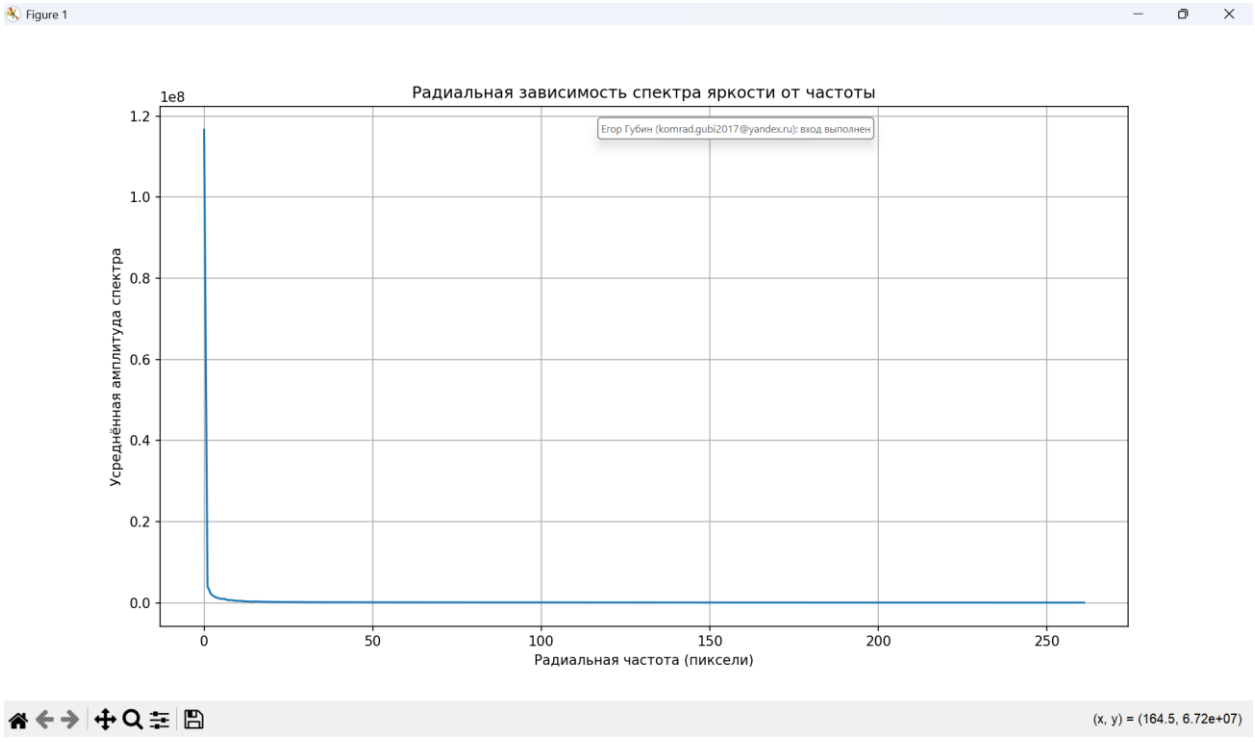

Рисунок 4 Радиальная зависимость спектра яркости от частоты

Рисунок 5 Наложение шума включённые пиксели

Рисунок 6 Медианная фильтрация 3x3

Рисунок 7 Медианная фильтрация 5x5

Рисунок 8 Медианная фильтрация 7x7

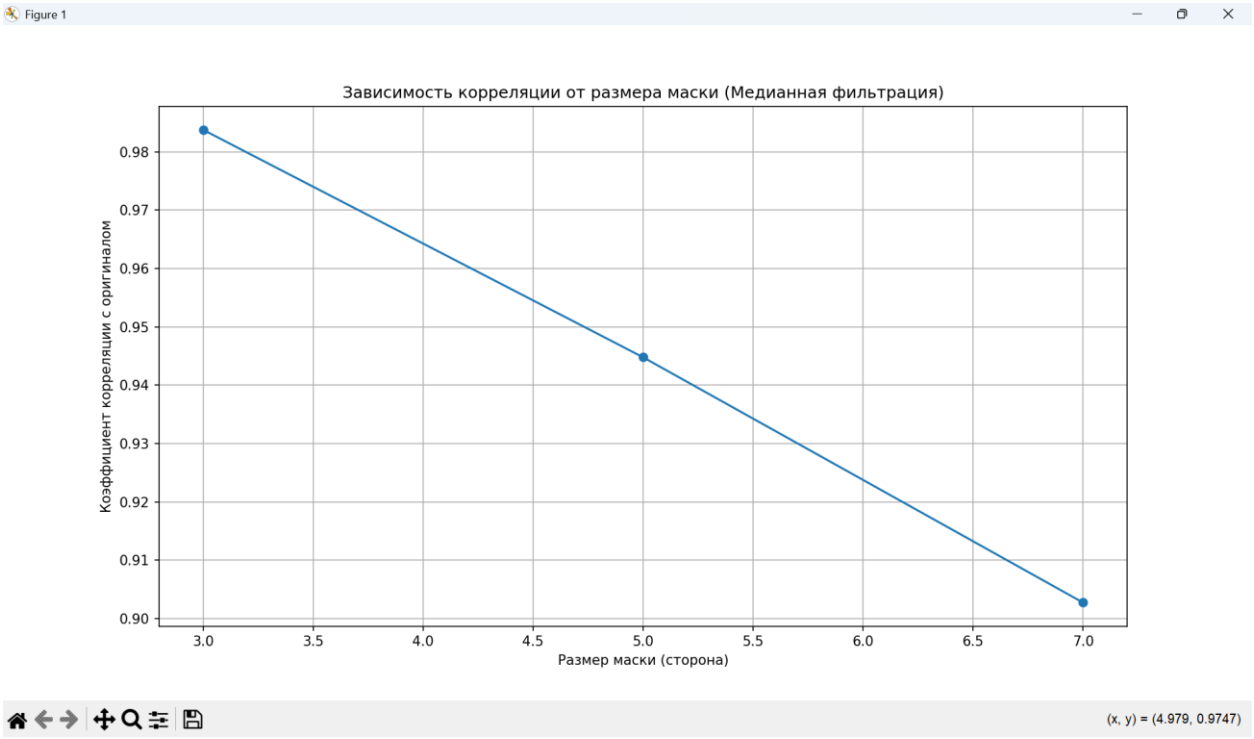

Рисунок 9 Зависимость корреляции при медианной фильтрации

Рисунок 10 Мультипликативный шум

Рисунок 11 Ранговая фильтрация 3x3

Рисунок 12 Ранговая фильтрация 5x5

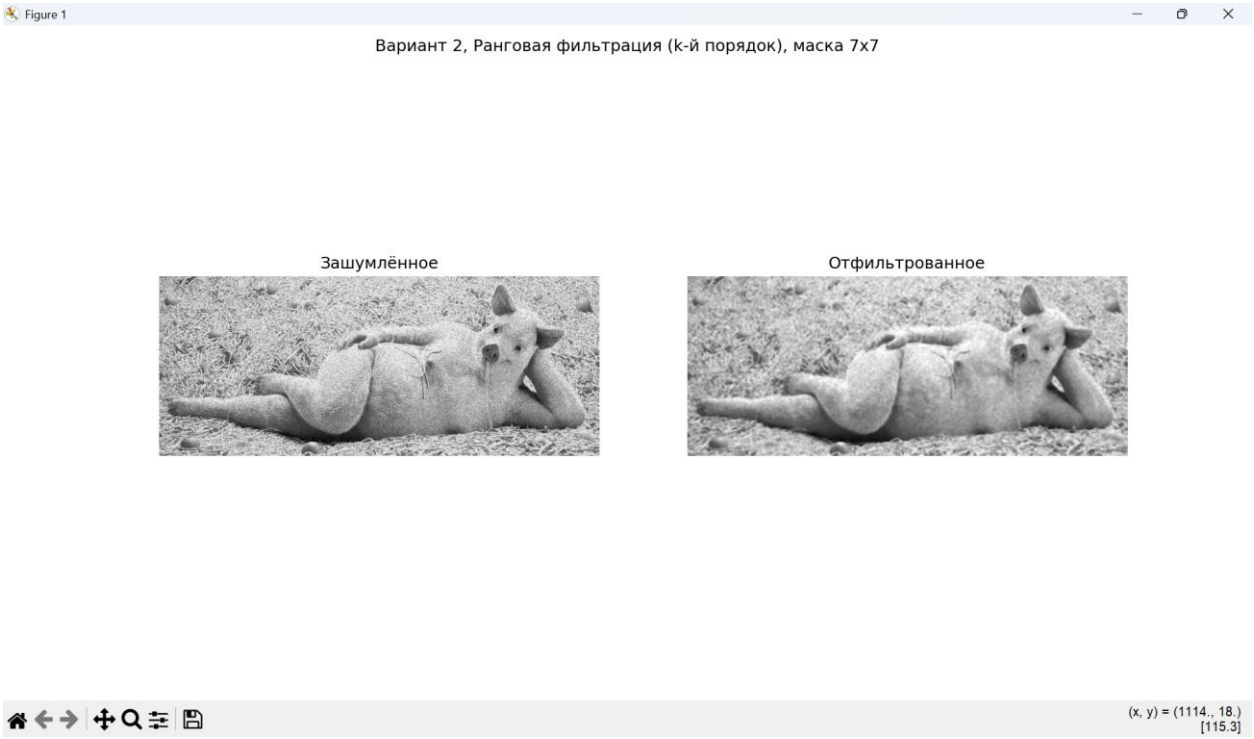

Рисунок 13 Ранговая фильтрация 7x7

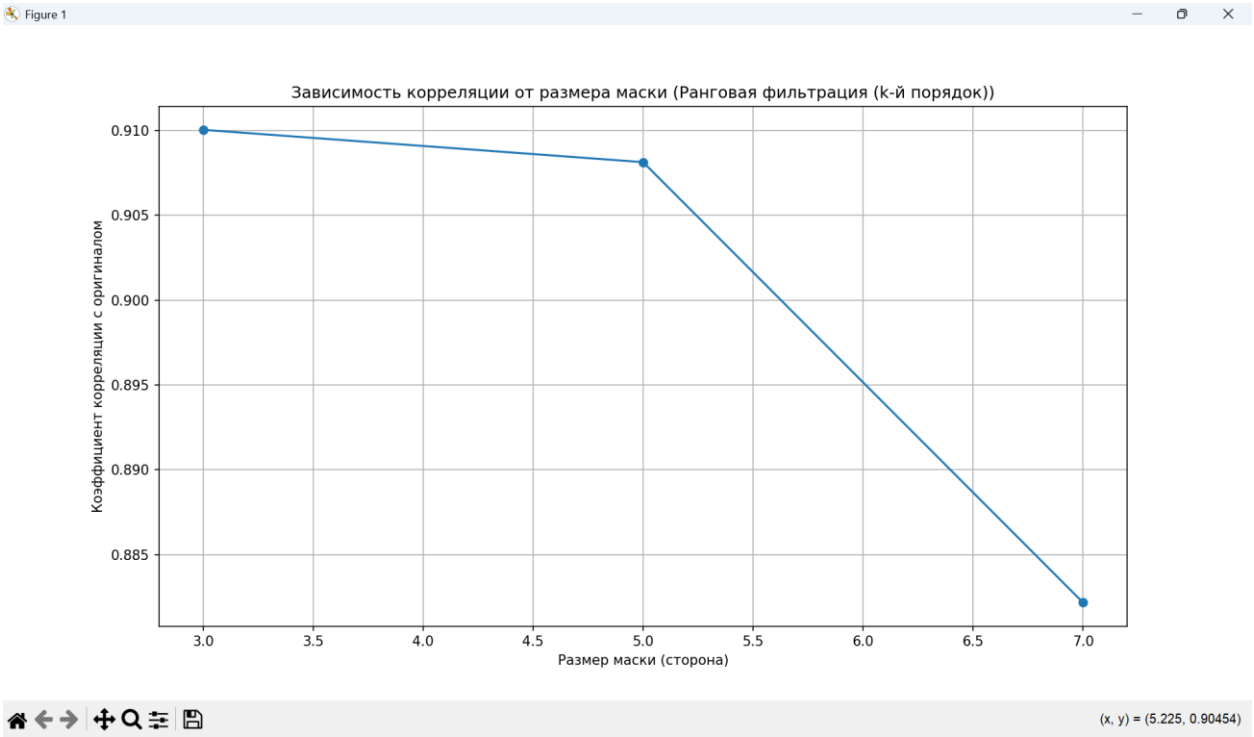

Рисунок 14 Зависимость корреляции от маски

Рисунок 15 Гауссовый шум

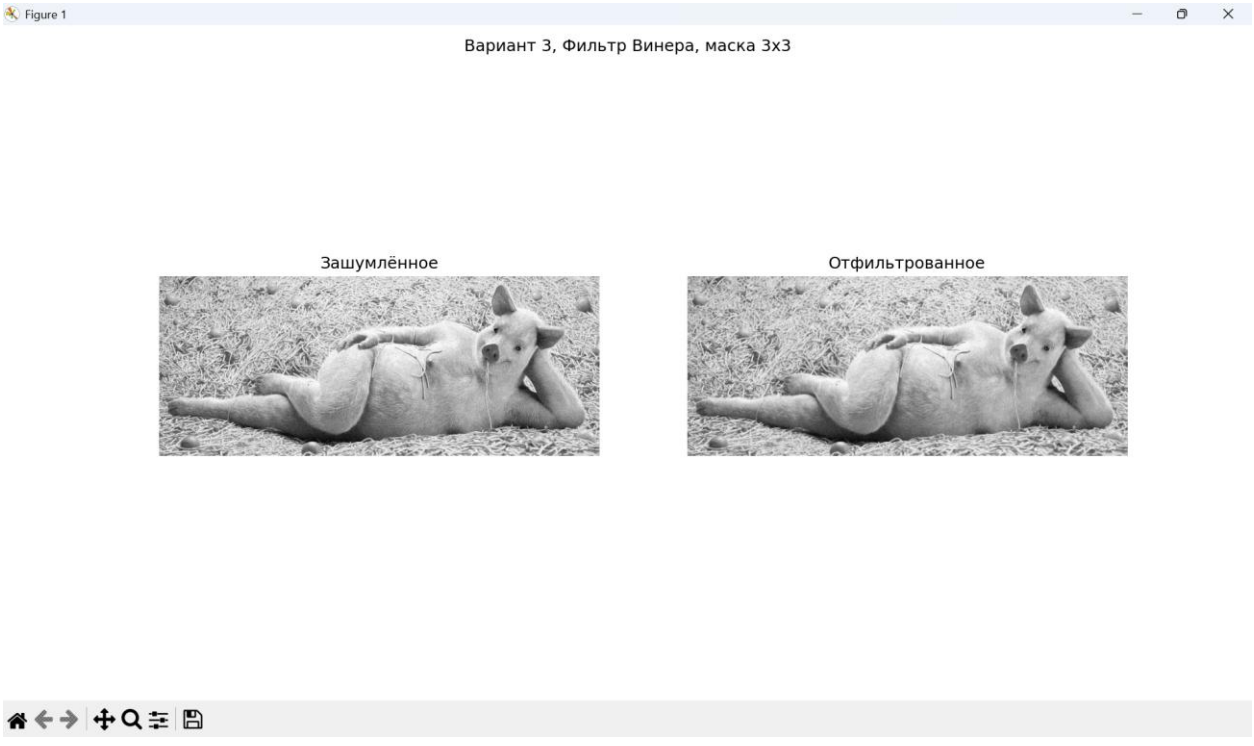

Рисунок 16 Фильтр Винера 3x3

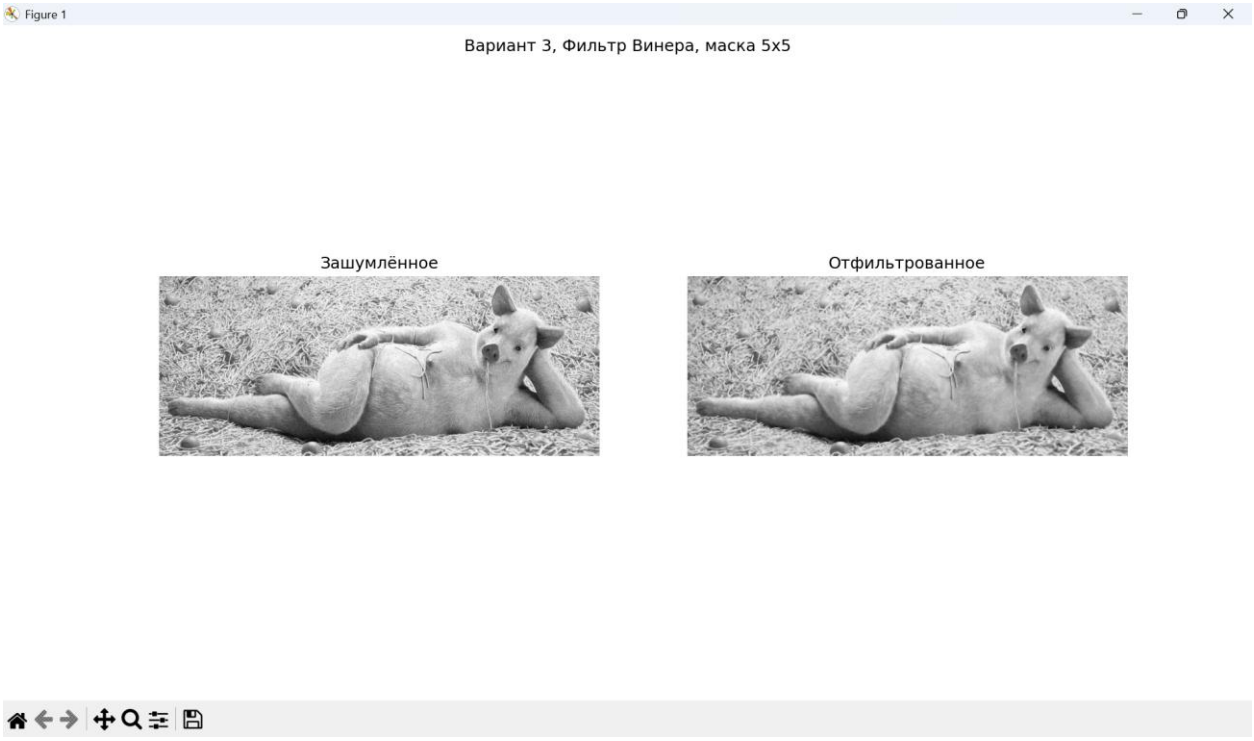

Рисунок 17 Фильтр Винера 5x5

Рисунок 18 Фильтра Винера 7x7

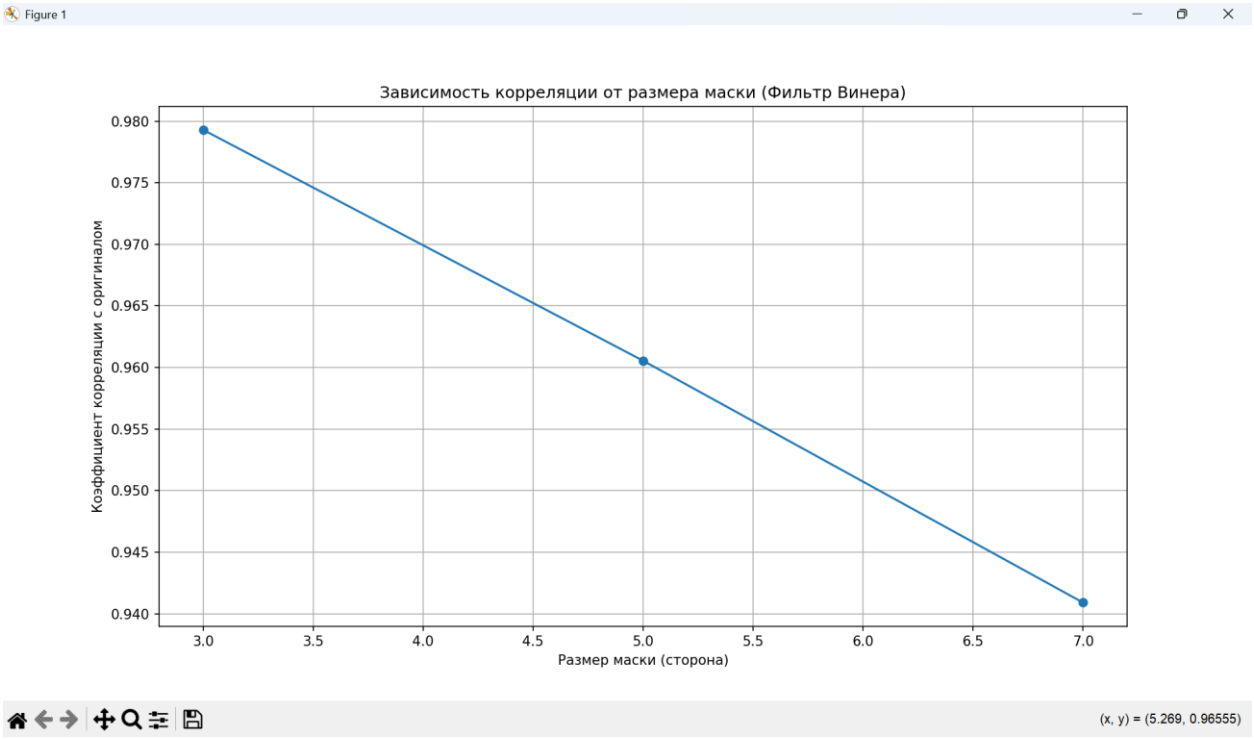

Рисунок 19 Зависимость корреляции от маски

Листинг программы:

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from scipy.fftpack import dct, idct
import scipy.fft as fft
from scipy import ndimage
from scipy.signal import wiener
import os

input_filename = "input.jpg"
variant = 3
mask_sizes = [3, 5, 7]

def load_image_gray(path):
    img = Image.open(path).convert('L')
    arr = np.asarray(img).astype(np.float64)
    return arr

def show_image(ax, image, title='', cmap='gray', vmin=None, vmax=None):
    ax.imshow(image, cmap=cmap, vmin=vmin, vmax=vmax)
    ax.set_title(title)
    ax.axis('off')

def dct2(a):
    return dct(dct(a.T, norm='ortho').T, norm='ortho')

def idct2(a):
    return idct(idct(a.T, norm='ortho').T, norm='ortho')

def fft2(a):
    return np.fft.fft2(a)

def ifft2(A):
```

```
    return np.fft.ifft2(A)

def corrcoef_image(a, b):
    a_flat = a.flatten()
    b_flat = b.flatten()
    if np.std(a_flat) == 0 or np.std(b_flat) == 0:
        return 0.0
    return np.corrcoef(a_flat, b_flat)[0,1]

def radial_spectrum_profile(fft_image):
    A = np.abs(np.fft.fftshift(fft_image))
    h, w = A.shape
    cy, cx = h//2, w//2
    y, x = np.indices((h, w))
    r = np.sqrt((x-cx)**2 + (y-cy)**2)
    r = r.astype(np.int32)
    maxr = min(cx, cy)
    radial_mean = np.zeros(maxr+1, dtype=np.float64)
    counts = np.zeros(maxr+1, dtype=np.int32)
    for rr in range(maxr+1):
        mask = (r == rr)
        counts[rr] = np.count_nonzero(mask)
        if counts[rr] > 0:
            radial_mean[rr] = A[mask].mean()
        else:
            radial_mean[rr] = 0.0
    freqs = np.arange(maxr+1)
    return freqs, radial_mean

def add_salt_noise(img, amount=0.01):
    noisy = img.copy()
    h, w = img.shape
    num_pixels = int(amount * h * w)
    coords = (np.random.randint(0, h, num_pixels), np.random.randint(0, w,
num_pixels))
    noisy[coords] = 255.0
    return noisy

def add_multiplicative_noise(img, sigma=0.2):
    noise = np.random.normal(loc=1.0, scale=sigma, size=img.shape)
    noisy = img * noise
    noisy = np.clip(noisy, 0, 255)
    return noisy

def add_gaussian_noise(img, mean=0.0, sigma=10.0):
    noisy = img + np.random.normal(mean, sigma, img.shape)
    noisy = np.clip(noisy, 0, 255)
    return noisy

def median_filter(img, size=3):
    return ndimage.median_filter(img, footprint=np.ones((size, size)))

def rank_filter(img, size=3, rank_frac=0.5):
    area = size * size
    k = int(np.clip(rank_frac * (area - 1), 0, area - 1))
    def kth(arr):
        return np.partition(arr, k)[k]
    footprint = np.ones((size, size))
    return ndimage.generic_filter(img, kth, footprint=footprint,
mode='mirror')

def wiener_filter(img, size=3):
    return wiener(img, mysize=(size, size))
```

```
def normalize_uint8(img):
    img = np.clip(img, 0, 255)
    return img.astype(np.uint8)

def main():
    if not os.path.exists(input_filename):
        raise FileNotFoundError(f"Файл {input_filename} не найден в текущей
директории.")
    orig = load_image_gray(input_filename)
    h, w = orig.shape
    print(f"Загружено изображение {input_filename} размером {w}x{h}")

    fig_main = plt.figure(figsize=(10,8))
    ax = fig_main.add_subplot(111)
    show_image(ax, orig, title='Оригинал')
    plt.show()

    A_dct = dct2(orig)
    recon_from_dct = idct2(A_dct)
    recon_from_dct_clipped = np.clip(recon_from_dct, 0, 255)

    fig, axes = plt.subplots(1,3, figsize=(15,5))
    show_image(axes[0], orig, 'Оригинал')
    show_image(axes[1], np.log(np.abs(A_dct)+1), 'Логарифм спектра DCT')
    show_image(axes[2], recon_from_dct_clipped, 'Реконструкция из DCT')
    plt.show()

    corr_dct = corrcoef_image(orig, recon_from_dct_clipped)
    print(f"Коэффициент корреляции оригинала и реконструкции из DCT:
{corr_dct:.6f}")

    A_fft = fft2(orig)
    spectrum_log = np.log(np.abs(np.fft.fftshift(A_fft)) + 1)
    recon_from_fft = np.real(ifft2(A_fft))
    recon_from_fft_clipped = np.clip(recon_from_fft, 0, 255)

    fig, axes = plt.subplots(1,3, figsize=(15,5))
    show_image(axes[0], orig, 'Оригинал')
    show_image(axes[1], spectrum_log, 'Логарифм спектра FFT (сдвинутый
центр)')
    show_image(axes[2], recon_from_fft_clipped, 'Реконструкция из FFT
(IFFT)')
    plt.show()

    corr_fft = corrcoef_image(orig, recon_from_fft_clipped)
    print(f"Коэффициент корреляции оригинала и реконструкции из FFT:
{corr_fft:.6f}")

    print("\nСравнение восстановленных изображений:")
    print(f" DCT-corr = {corr_dct:.6f}")
    print(f" FFT-corr = {corr_fft:.6f}")
    if corr_dct > corr_fft:
        print(" Реконструкция DCT ближе к оригиналу (по корреляции).")
    elif corr_fft > corr_dct:
        print(" Реконструкция FFT ближе к оригиналу (по корреляции).")
    else:
        print(" Корреляции равны (в пределах машинной точности).")

    freqs, radial = radial_spectrum_profile(A_fft)
    plt.figure(figsize=(8,4))
    plt.plot(freqs, radial)
    plt.xlabel('Радиальная частота (пиксели)')
    plt.ylabel('Усреднённая амплитуда спектра')
    plt.title('Радиальная зависимость спектра яркости от частоты')
```

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

    if variant not in (1,2,3):
        raise ValueError("variant должен быть 1, 2 или 3")

    print(f"\nРаботаем с вариантом {variant} ...")
    if variant == 1:
        noisy = add_salt_noise(orig, amount=0.02)
        filter_name = "Медианная фильтрация"
    elif variant == 2:
        noisy = add_multiplicative_noise(orig, sigma=0.25)
        filter_name = "Ранговая фильтрация (k-й порядок)"
    else:
        noisy = add_gaussian_noise(orig, mean=0.0, sigma=15.0)
        filter_name = "Фильтр Винера"

    fig, axes = plt.subplots(1,2, figsize=(12,5))
    show_image(axes[0], orig, 'Исходное изображение')
    show_image(axes[1], noisy, f'Зашумлённое ({variant})')
    plt.show()

    correlations = []
    filtered_images = []

    for size in mask_sizes:
        if variant == 1:
            filtered = median_filter(noisy, size=size)
        elif variant == 2:
            filtered = rank_filter(noisy, size=size, rank_frac=0.5)
        else:
            filtered = wiener_filter(noisy, size=size)
        filtered = np.clip(filtered, 0, 255)
        filtered_images.append(filtered)
        c = corrcoef_image(orig, filtered)
        correlations.append(c)
        print(f"Размер маски {size}x{size}: корреляция оригинал <->
отфильтровано = {c:.6f}")

        plt.figure(figsize=(6,4))
        plt.suptitle(f"Вариант {variant}, {filter_name}, маска
{size}x{size}")
        plt.subplot(1,2,1)
        plt.title("Зашумлённое")
        plt.imshow(noisy, cmap='gray', vmin=0, vmax=255)
        plt.axis('off')
        plt.subplot(1,2,2)
        plt.title("Отфильтрованное")
        plt.imshow(filtered, cmap='gray', vmin=0, vmax=255)
        plt.axis('off')
        plt.show()

    plt.figure(figsize=(7,4))
    plt.plot(mask_sizes, correlations, marker='o')
    plt.xlabel('Размер маски (сторона)')
    plt.ylabel('Коэффициент корреляции с оригиналом')
    plt.title(f'Зависимость корреляции от размера маски ({filter_name})')
    plt.grid(True)
    plt.show()

    best_idx = int(np.argmax(correlations))
    print(f"Лучший размер маски по корреляции: {mask_sizes[best_idx]} (corr =
{correlations[best_idx]:.6f})")
```

```
    out_dir = "results_dsp"
    os.makedirs(out_dir, exist_ok=True)
    Image.fromarray(normalize_uint8(orig)).save(os.path.join(out_dir,
"orig.png"))

Image.fromarray(normalize_uint8(recon_from_dct_clipped)).save(os.path.join(ou
t_dir, "recon_dct.png"))

Image.fromarray(normalize_uint8(recon_from_fft_clipped)).save(os.path.join(ou
t_dir, "recon_fft.png"))
    Image.fromarray(normalize_uint8(noisy)).save(os.path.join(out_dir,
"noisy.png"))
    for i, size in enumerate(mask_sizes):

Image.fromarray(normalize_uint8(filtered_images[i])).save(os.path.join(out_di
r, f"filtered_size_{size}.png"))
    print(f"Результаты сохранены в папке: {out_dir}")

if __name__ == "__main__":
    main()
```

Вывод: в ходе выполнения работы был наложен шум на изображение и применён фильтр.