Расчётно графическая Работа №1

Выполнил: Андрюшин Лев М80-312Б-22

Задание

Проведите ортогонализацию системы функций $x_n(t) = t^{n-1}$ в пространстве квадратично суммируемых функций относительно склярного произведения $\langle x, y\rangle = \int\limits_a^b x(t)y(t)f(t)\,dt$. Найдите приближение функции y частичной суммой ряда Фурье, обеспечивающее среднеквадратичную точность разложения $\varepsilon \in \{10^{-1}, 10^{-2}, 10^{-3}\}$ (при достаточных вычислительных ресурсах). Постройте график функции $y(t)$ и его приближения частичными суммами ряда Фурье. Продемонстрируйте несколько графиков, получающихся при промежуточных вычислениях.

Варианты задания II

1) $[a,b] = \left[0; 0{,}8 + \frac{k}{10}\right]$, $f(t) = \left(\frac{2l}{5} - t\right)t$, $y(t) = e^t$;
2) $[a,b] = \left[0; 0{,}8 + \frac{k}{10}\right]$, $f(t) = \left(\frac{2l}{5} - t\right)t$, $y(t) = \cos(2t)$;
3) $[a,b] = \left[-0{,}8 - \frac{k}{10}; 0{,}8 + \frac{k}{10}\right]$, $f(t) = \left(\frac{2l}{5} - t\right)^2$, $y(t) = \sin(2t)$;
4) $[a,b] = \left[-0{,}8 - \frac{k}{10}; 0{,}8 + \frac{k}{10}\right]$, $f(t) = \left(\frac{2l}{5} - t\right)^2$, $y(t) = \cos(3t)$;
5) $[a,b] = \left[0; 0{,}8 + \frac{k}{10}\right]$, $f(t) = \left(\frac{2l}{5} - t\right)^2$, $y(t) = e^t$;
6) $[a,b] = \left[0; 0{,}8 + \frac{k}{10}\right]$, $f(t) = \left(\frac{2l}{5} - t\right)^2$, $y(t) = \cos(2t)$;
7) $[a,b] = \left[0; 0{,}8 + \frac{k}{10}\right]$, $f(t) = \left(\frac{2l}{5} - t\right)^2$, $y(t) = \sin(3t)$;
8) $[a,b] = \left[-0{,}8 - \frac{k}{10}; 0{,}8 + \frac{k}{10}\right]$, $f(t) = \frac{4l}{5} - t^2$, $y(t) = \sin(2t)$;
9) $[a,b] = \left[-0{,}8 - \frac{k}{10}; 0{,}8 + \frac{k}{10}\right]$, $f(t) = \frac{4l}{5} - t^2$, $y(t) = \cos(3t)$;
10) $[a,b] = \left[-0{,}8 - \frac{k}{10}; 0{,}8 + \frac{k}{10}\right]$, $f(t) = \frac{4l}{5} - t^2$, $y(t) = e^{-t}$.

Вариант 2

Импортируем нужные нам библиотеки:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
```

- **numpy**: Для численных операций с массивами.
- **matplotlib.pyplot**: Для визуализации графиков.
- **scipy.integrate.quad**: Для точного вычисления интегралов.

Задаем параметры нашей задачи:

```
a, b = 0, 2.0
f = lambda t: (4 - t)*t  # Весовая функция
y = lambda t: np.cos(2*t)  # Аппроксимируемая функция
```

- **a, b**: Границы интервала $[0, 2]$.
- **f(t)**: Весовая функция, используемая при ортогонализации и вычислении ошибки.
- **y(t)**: Функция $\cos(2t)$, которую мы приближаем рядом Фурье.

Класс **Polynomial**

```
class Polynomial:
    def __init__(self, coeffs):
        self.coeffs = np.array(coeffs)

    def __call__(self, t):
        return sum(c * t**i for i, c in enumerate(self.coeffs))

    def __sub__(self, other):
        max_len = max(len(self.coeffs), len(other.coeffs))
        new_coeffs = np.zeros(max_len)
        new_coeffs[:len(self.coeffs)] = self.coeffs
        new_coeffs[:len(other.coeffs)] -= other.coeffs
        return Polynomial(new_coeffs)

    def __mul__(self, scalar):
        return Polynomial(self.coeffs * scalar)

    def __str__(self):
        return " + ".join(f"{c:.3f}t^{i}" for i, c in enumerate(self.coeffs) if c
!= 0)
```

- **__init__**: Инициализирует полином с заданными коэффициентами.
- **__call__**: Позволяет вычислять значение полинома в точке t.
- **__sub__**: Вычитает два полинома, выравнивая их степени.
- **__mul__**: Умножает полином на скаляр.
- **__str__**: Возвращает строковое представление полинома.

Процесс ортогонализации Грама-Шмидта

```
def gram_schmidt(max_degree):
    basis = []
    print("\nПроцесс ортогонализации:")
    for n in range(max_degree + 1):
        p = Polynomial([0]*n + [1])
        print(f"\nНачальный полином P_{n}: {p}")

        for m in range(n):
            numerator = quad(lambda t: p(t) * basis[m](t) * f(t), a, b)[0]
            denominator = quad(lambda t: basis[m](t)**2 * f(t), a, b)[0]
            alpha = numerator / denominator
            print(f"Проекция на P_{m}: α = {alpha:.4f}")
            p = p - basis[m] * alpha

        basis.append(p)
        norm = np.sqrt(quad(lambda t: p(t)**2 * f(t), a, b)[0])
        print(f"Ортогональный полином P_{n}: {p}")
        print(f"Норма P_{n}: {norm:.4f}")
    return basis
```

Преобразует систему $\{1, t, t^2, \ldots\}$ в ортогональные полиномы относительно весовой функции $f(t)$.

- **Шаги**:
  1. Начинаем с монома t^n.
  2. Вычитаем проекции на все предыдущие полиномы в базисе.
  3. Нормируем полученный полином.

Вычисление коффициентов Фурье

```
def compute_coefficients(func, basis):
    coeffs = []
    print("\nВычисление коэффициентов Фурье:")
    for i, p in enumerate(basis):
        numerator = quad(lambda t: func(t) * p(t) * f(t), a, b)[0]
        denominator = quad(lambda t: p(t)**2 * f(t), a, b)[0]
        coeff = numerator / denominator
        coeffs.append(coeff)
        print(f"c_{i} = {coeff:.6f} (∫={numerator:.4f},
норма={denominator:.4f})")
    return coeffs
```

Находим коэффициенты разложения функции $y(t)$ по ортогональным полиномам.

- **Формула**:

$$c_n = \frac{\langle y, P_n \rangle}{\langle P_n, P_n \rangle} = \frac{\int_a^b y(t) P_n(t) f(t)\, dt}{\int_a^b P_n^2(t) f(t)\, dt}$$

Вычисление ошибки

```
def exact_error(N):
    integrand = lambda t: (y(t) - sum(coeffs[n] * basis[n](t) for n in range(N +
1)))**2 * f(t)
    error_integral, _ = quad(integrand, a, b)
    return np.sqrt(error_integral)
```

- **Формула**:

$$\text{error} = \sqrt{\int_a^b \left(y(t) - \sum_{n=0}^{N} c_n P_n(t) \right)^2 f(t)\, dt}$$

Подбираем N

```
epsilon_list = [1e-1, 1e-2, 1e-3]
required_N = {}
print("\nПодбор N для заданных точностей:")

for eps in epsilon_list:
    print(f"\nЦелевая точность ε = {eps:.0e}:")
    for N in range(max_degree + 1):
        err = exact_error(N)
        print(f"N = {N}: ошибка = {err:.4e}", end="")
        if err < eps:
            required_N[eps] = N
            print(" < ε - достигнуто")
            break
        print()
```

Ищем минимальное N, при котором ошибка становится меньше заданного ε.

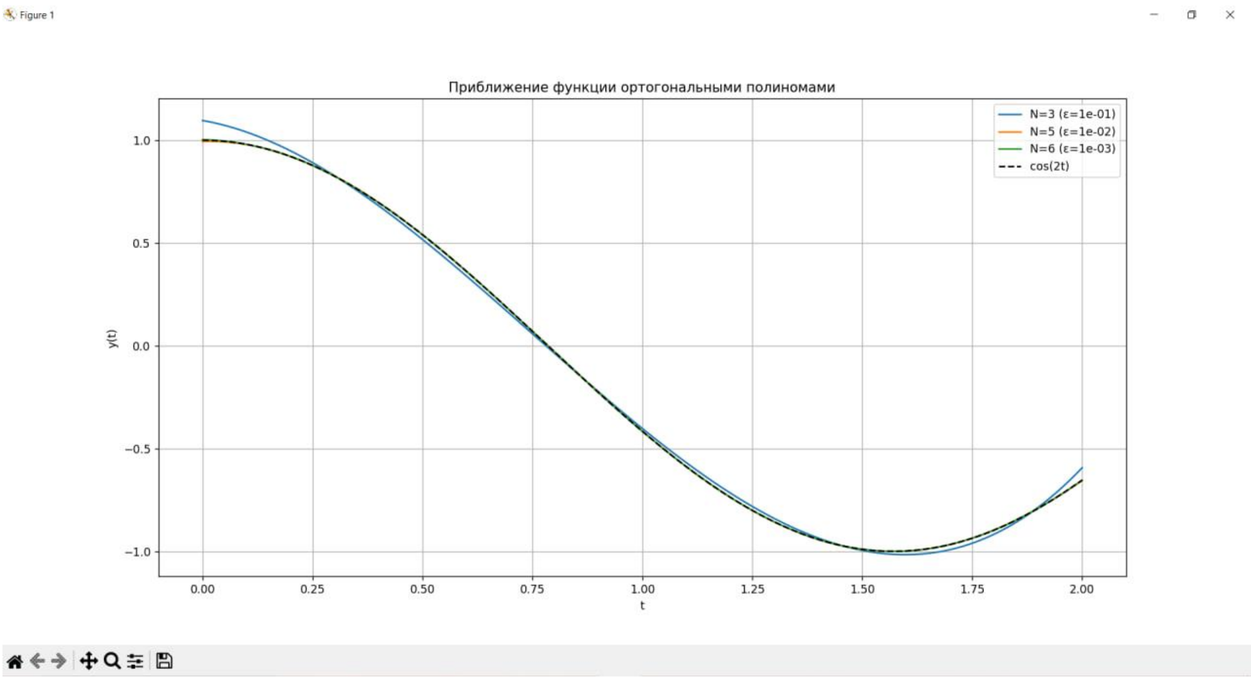
- Для каждого ε перебираем N от 0 до max_degree.
- Как только ошибка становится меньше ε, запоминаем N и переходим к следующему ε.

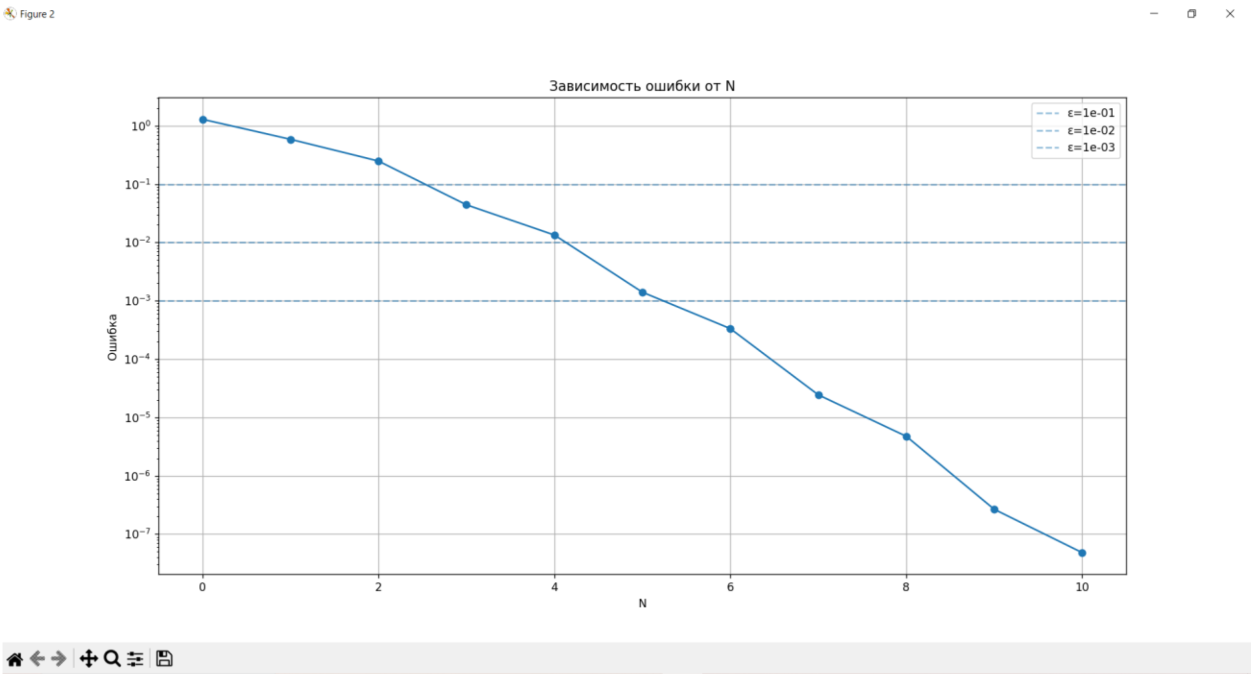
Вот полная версия кода:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

# Параметры задачи
a, b = 0, 2.0
f = lambda t: (4 - t)*t  # Весовая функция
y = lambda t: np.cos(2*t)  # Аппроксимируемая функция

class Polynomial:
    def __init__(self, coeffs):
        self.coeffs = np.array(coeffs)

    def __call__(self, t):
        return sum(c * t**i for i, c in enumerate(self.coeffs))

    def __sub__(self, other):
        max_len = max(len(self.coeffs), len(other.coeffs))
        new_coeffs = np.zeros(max_len)
        new_coeffs[:len(self.coeffs)] = self.coeffs
        new_coeffs[:len(other.coeffs)] -= other.coeffs
        return Polynomial(new_coeffs)
```

```
    def __mul__(self, scalar):
        return Polynomial(self.coeffs * scalar)

    def __str__(self):
        return " + ".join(f"{c:.3f}t^{i}" for i, c in enumerate(self.coeffs) if c
!= 0)

def gram_schmidt(max_degree):
    basis = []
    print("\nПроцесс ортогонализации:")
    for n in range(max_degree + 1):
        p = Polynomial([0]*n + [1])
        print(f"\nНачальный полином P_{n}: {p}")

        for m in range(n):
            numerator = quad(lambda t: p(t) * basis[m](t) * f(t), a, b)[0]
            denominator = quad(lambda t: basis[m](t)**2 * f(t), a, b)[0]
            alpha = numerator / denominator
            print(f"Проекция на P_{m}: α = {alpha:.4f}")
            p = p - basis[m] * alpha

        basis.append(p)
        norm = np.sqrt(quad(lambda t: p(t)**2 * f(t), a, b)[0])
        print(f"Ортогональный полином P_{n}: {p}")
        print(f"Норма P_{n}: {norm:.4f}")
    return basis

# Основные вычисления
max_degree = 15
basis = gram_schmidt(max_degree)

def compute_coefficients(func, basis):
    coeffs = []
    print("\nВычисление коэффициентов Фурье:")
    for i, p in enumerate(basis):
        numerator = quad(lambda t: func(t) * p(t) * f(t), a, b)[0]
        denominator = quad(lambda t: p(t)**2 * f(t), a, b)[0]
        coeff = numerator / denominator
        coeffs.append(coeff)
        print(f"c_{i} = {coeff:.6f} (∫={numerator:.4f},
норма={denominator:.4f})")
    return coeffs

coeffs = compute_coefficients(y, basis)

# Точное вычисление ошибки
def exact_error(N):
    integrand = lambda t: (y(t) - sum(coeffs[n] * basis[n](t) for n in range(N +
1)))**2 * f(t)
    error_integral, _ = quad(integrand, a, b)
    return np.sqrt(error_integral)
```

```python
# Подбор N
epsilon_list = [1e-1, 1e-2, 1e-3]
required_N = {}
print("\nПодбор N для заданных точностей:")

for eps in epsilon_list:
    print(f"\nЦелевая точность ε = {eps:.0e}:")
    for N in range(max_degree + 1):
        err = exact_error(N)
        print(f"N = {N}: ошибка = {err:.4e}", end="")
        if err < eps:
            required_N[eps] = N
            print(" < ε - достигнуто")
            break
        print()

# Визуализация
t_plot = np.linspace(a, b, 500)
plt.figure(figsize=(12, 8))

for eps, N in required_N.items():
    y_approx = [sum(c * p(t) for c, p in zip(coeffs[:N+1], basis[:N+1])) for t in
t_plot]
    plt.plot(t_plot, y_approx, label=f'N={N} (ε={eps:.0e})')

plt.plot(t_plot, [y(t) for t in t_plot], 'k--', label='cos(2t)')
plt.xlabel('t'); plt.ylabel('y(t)')
plt.title('Приближение функции ортогональными полиномами')
plt.legend(); plt.grid()

# График ошибок
errors = [exact_error(N) for N in range(max_degree + 1)]
plt.figure(figsize=(10, 5))
plt.semilogy(range(max_degree + 1), errors, 'o-')
for eps in epsilon_list:
    plt.axhline(y=eps, linestyle='--', alpha=0.5, label=f'ε={eps:.0e}')
plt.xlabel('N'); plt.ylabel('Ошибка')
plt.title('Зависимость ошибки от N')
plt.legend(); plt.grid()
plt.show()
```





Полный вывод программы:

PS D:\python> & C:/Users/Vingael/AppData/Local/Programs/Python/Python313/python.exe d:/python/Articles/RGR2.2.py

Процесс ортогонализации:

Начальный полином P_0: 1.000t^0

Ортогональный полином P_0: 1.000t^0

Норма P_0: 2.3094

Начальный полином P_1: 1.000t^1

Проекция на P_0: α = 1.2500

Ортогональный полином P_1: -1.250t^0 + 1.000t^1

Норма P_1: 1.1255

Начальный полином P_2: 1.000t^2

Проекция на P_0: α = 1.8000

Проекция на P_1: α = 2.3158

Ортогональный полином P_2: 1.095t^0 + -2.316t^1 + 1.000t^2

Норма P_2: 0.5550

Начальный полином P_3: 1.000t^3

Проекция на P_0: α = 2.8000

Проекция на P_1: α = 4.5113

Проекция на P_2: α = 3.3438

Ортогональный полином P_3: -0.821t^0 + 3.232t^1 + -3.344t^2 + 1.000t^3

Норма P_3: 0.2752

Начальный полином P_4: 1.000t^4

Проекция на P_0: α = 4.5714

Проекция на P_1: α = 8.4211

Проекция на P_2: α = 8.1944

Проекция на P_3: α = 4.3593

Ортогональный полином P_4: 0.565t^0 + -3.534t^1 + 6.382t^2 + -4.359t^3 + 1.000t^4

Норма P_4: 0.1369

Начальный полином P_5: 1.000t^5

Проекция на P_0: α = 7.7143

Проекция на P_1: α = 15.5388

Проекция на P_2: α = 17.9167

Проекция на P_3: α = 12.8688

Проекция на P_4: α = 5.3692

Ортогональный полином P_5: -0.367t^0 + 3.335t^1 + -9.152t^2 + 10.537t^3 + -5.369t^4 + 1.000t^5

Норма P_5: 0.0682

Начальный полином P_6: 1.000t^6

Проекция на P_0: α = 13.3333

Проекция на P_1: α = 28.6316

Проекция на P_2: α = 37.1212

Проекция на P_3: α = 32.3136

Проекция на P_4: α = 18.5394

Проекция на P_5: α = 6.3761

Ортогональный полином P_6: 0.230t^0 + -2.848t^1 + 10.966t^2 + -18.680t^3 + 15.695t^4 + -6.376t^5 + 1.000t^6

Норма P_6: 0.0340

Начальный полином P_7: 1.000t^7

Проекция на P_0: α = 23.4667

Проекция на P_1: α = 52.8740

Проекция на P_2: α = 74.7374

Проекция на P_3: α = 74.5228

Проекция на P_4: α = 52.6191

Проекция на P_5: α = 25.2080

Проекция на P_6: α = 7.3811

Ортогональный полином P_7: -0.139t^0 + 2.263t^1 + -11.593t^2 + 27.120t^3 + -33.119t^4 + 21.855t^5 + -7.381t^6 + 1.000t^7

Норма P_7: 0.0170

Начальный полином P_8: 1.000t^8

Проекция на P_0: α = 41.8909

Проекция на P_1: α = 97.9904

Проекция на P_2: α = 148.0280

Проекция на P_3: α = 163.4344

Проекция на P_4: α = 133.9195

Проекция на P_5: α = 79.8359

Проекция на P_6: α = 32.8754

Проекция на P_7: α = 8.3850

Ортогональный полином P_8: 0.083t^0 + -1.702t^1 + 11.171t^2 + -34.167t^3 + 56.459t^4 + -53.470t^5 + 29.015t^6 + -8.385t^7 + 1.000t^8

Норма P_8: 0.0085

Начальный полином P_9: 1.000t^9

Проекция на P_0: α = 75.6364

Проекция на P_1: α = 182.3187

Проекция на P_2: α = 290.3497

Проекция на P_3: α = 347.1292

Проекция на P_4: α = 318.4948

Проекция на P_5: α = 222.4795

Проекция на P_6: α = 114.9651

Проекция на P_7: α = 41.5420

Проекция на P_8: α = 9.3880

Ортогональный полином P_9: -0.048t^0 + 1.226t^1 + -10.017t^2 + 38.684t^3 + -82.546t^4 + 104.647t^5 + -80.735t^6 + 37.177t^7 + -9.388t^8 + 1.000t^9

Норма P_9: 0.0042

Начальный полином P_10: 1.000t^10

Проекция на P_0: α = 137.8462

Проекция на P_1: α = 340.5437

Проекция на P_2: α = 566.1538

Проекция на P_3: α = 721.6582

Проекция на P_4: α = 723.9955

Проекция на P_5: α = 571.0926

Проекция на P_6: $\alpha = 348.3689$

Проекция на P_7: $\alpha = 159.0074$

Проекция на P_8: $\alpha = 51.2081$

Проекция на P_9: $\alpha = 10.3905$

Ортогональный полином P_{10}: 0.028t^0 + -0.853t^1 + 8.483t^2 + -40.278t^3 + 107.356t^4 + -174.119t^5 + 178.347t^6 + -115.913t^7 + 46.339t^8 + -10.391t^9 + 1.000t^10

Норма P_{10}: 0.0021

Вычисление коэффициентов Фурье:

$c_0 = -0.506776$ (∫=-2.7028, норма=5.3333)

$c_1 = -1.026998$ (∫=-1.3009, норма=1.2667)

$c_2 = 0.960223$ (∫=0.2957, норма=0.3080)

$c_3 = 0.891401$ (∫=0.0675, норма=0.0757)

$c_4 = -0.308607$ (∫=-0.0058, норма=0.0187)

$c_5 = -0.196223$ (∫=-0.0009, норма=0.0046)

$c_6 = 0.040253$ (∫=0.0000, норма=0.0012)

$c_7 = 0.019647$ (∫=0.0000, норма=0.0003)

$c_8 = -0.002834$ (∫=-0.0000, норма=0.0001)

$c_9 = -0.001124$ (∫=-0.0000, норма=0.0000)

$c_{10} = 0.000116$ (∫=0.0000, норма=0.0000)

Подбор N для заданных точностей:

Целевая точность ε = 1e-01:

N = 0: ошибка = 1.2970e+00

N = 1: ошибка = 5.8830e-01

N = 2: ошибка = 2.4929e-01

N = 3: ошибка = 4.4332e-02 < ε - достигнуто

Целевая точность ε = 1e-02:

N = 0: ошибка = 1.2970e+00

N = 1: ошибка = 5.8830e-01

N = 2: ошибка = 2.4929e-01

N = 3: ошибка = 4.4332e-02

N = 4: ошибка = 1.3454e-02

N = 5: ошибка = 1.4090e-03 < ε - достигнуто

Целевая точность ε = 1e-03:

N = 0: ошибка = 1.2970e+00

N = 1: ошибка = 5.8830e-01

N = 2: ошибка = 2.4929e-01

N = 3: ошибка = 4.4332e-02

N = 4: ошибка = 1.3454e-02

N = 5: ошибка = 1.4090e-03

N = 6: ошибка = 3.3426e-04 < ε - достигнуто