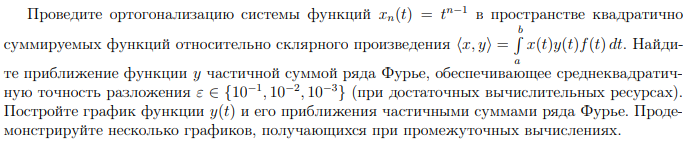
**Расчётно-графическая работа по функциональному анализу**

В приведённых ниже вариантах k ∈ {1, 3, 6, 7, 8, 9, 10, 12} — номер группы, l = 15, если номер студента в списке группы не больше 10, l = 10, если номер студента в списке группы от 11 до 20, l = 5, если номер студента не меньше 21. Вариант выбирается как остаток от деления номера по списку группы на 10.

**Задание II**



**Вариант 7**

 , k = 7, l = 10

**Реализация**

Будем строить ортонормированную систему из ЛНЗ системы функций xn(t) = tn. На каждой итерации, имея ортонормированную систему, будем сравнивать длину ошибки проектирования приближаемой функции y с заданной пользователем точностью eps.

Определим функции из варианта:

def f(t):

    return t \*\* 2 - 8 \* t + 16

def y(t):

    return np.sin(3 \* t)

Все вычисления будем проводить численно, не в общем виде. Разобьем отрезок [a, b] на точки, вычислим значения функций в этих точках:

ts = np.linspace(a, b, countPoints)

ys = np.array([y(t) for t in ts])

fs = np.array([f(t) for t in ts])

step = (b - a) / (countPoints - 1)

Определим вспомогательные методы. Вычисление определенного интеграла Римана по массиву значений:

def calcIntegral(values):

    integral = 0

    for i in range(1, countPoints):

        integral += step \* (values[i] + values[i - 1]) / 2

    return integral

Вычисление заданного скалярного произведения. Получаем массивы вычисленных в точках значений двух функций, домножаем эти значения на значение функции f из варианта, вычисляем интеграл:

def calcScalarProduct(xs, ys):

    values = xs \* ys \* fs

    return calcIntegral(values)

Норму вычисляем как корень из скалярного произведения:

def calcNorm(xs):

    return np.sqrt(calcScalarProduct(xs, xs))

Полезно определить метод, вычисляющий значение многочлена. Многочлен здесь и далее задается как массив коэффициентов при степенях. Например, многочлен t2 + 5 будет задан массивом [5, 0, 1].

def calcValuesPolynom(polynom):

    values = np.zeros(countPoints)

    for i in range(countPoints):

        curDegreeT = 1

        for coef in polynom:

            values[i] += coef \* curDegreeT

            curDegreeT \*= ts[i]

    return values

Вычисление длины ошибки проектирования. Ищем норму разности истинной функции и приближающего многочлена:

def calcLengthProjectionError(projection):

    # Инициализация истинным значением приближаемой функции

    values = np.copy(ys)

    # Вычитаем текущее приближение многочленом

    values -= calcValuesPolynom(projection)

    return calcNorm(values)

Вычисление коэффициента проекции для нового элемента в ортонормированной системе. Для этого ищем скалярное произведение истинной функции и нового элемента, затем обновляем текущую проекцию. Проекция, являясь многочленом, задается так же, как и другие многочлены, - одним массивом.

def calcProjection(projection: list, newElem, curDegree):

    projection.append(0)

    coef = calcScalarProduct(calcValuesPolynom(newElem), ys)

    for coord in range(curDegree + 1):

        projection[coord] += coef \* newElem[coord]

Общий итерационный процесс:

def solve(eps):

    system = []

    curDegree = -1

    projection = []

    calcProjection(projection, [1], curDegree)

    while (True):

        # Вычисляем текущую длину ошибки проектирования

        error = calcLengthProjectionError(projection)

        # Останавливаемся когда длина ошибки проектирования будет меньше заданной точности

        if error < eps:

            break

        # Увеличиваем степень многочлена

        # Добавляем еще один элемент в текущую ортонормированную систему функций

        # Ортогонализируем - для каждого элемента из системы вычисляем свой коэффициент

        curDegree += 1

        newElem = [0] \* (curDegree + 1)

        newElem[curDegree] = 1

        for elem in system:

            coef = calcScalarProduct(calcValuesPolynom(newElem), calcValuesPolynom(elem))

            for coord in range(len(elem)):

                newElem[coord] -= coef \* elem[coord]

        # Нормируем

        normNewElem = calcNorm(calcValuesPolynom(newElem))

        for coord in range(curDegree + 1):

            newElem[coord] /= normNewElem

        system.append(newElem)

        # Пересчитываем проекцию

        calcProjection(projection, newElem, curDegree)

        plot(calcValuesPolynom(projection), curDegree)

**Результаты**

