**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра компьютерных технологий и систем**

Контролируемая самостоятельная работа

по курсу **«**Функциональный анализ и Интегральные уравнения**»**

**«Принцип сжимающих отображений»**

Вариант 13

Студентки 2 курса 10 группы

Крагель Алины Олеговны

Научный руководитель:

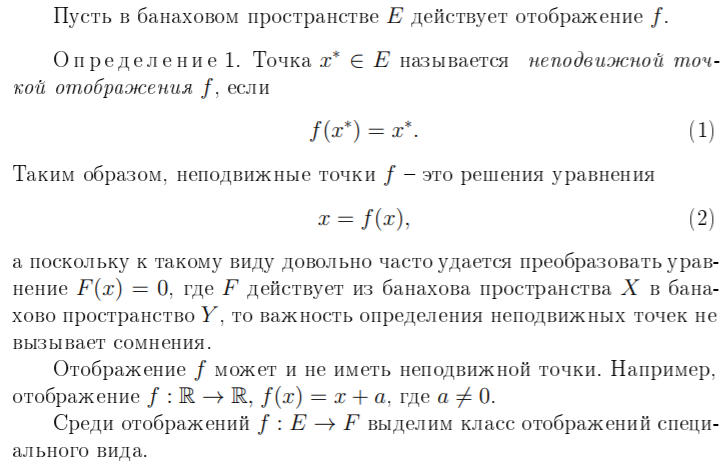
Чеб Елена Сергеевна,

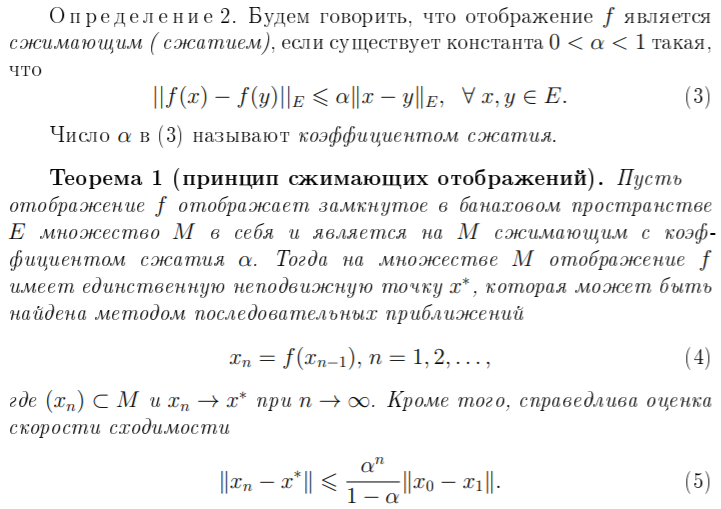
Доцент кафедры

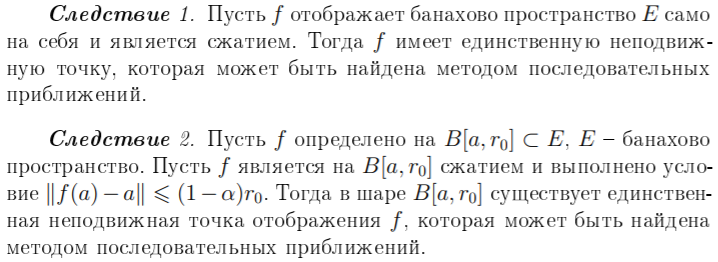
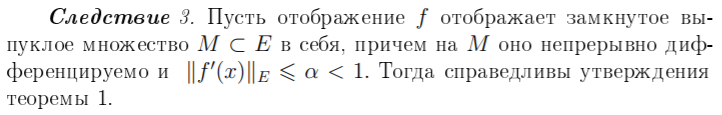
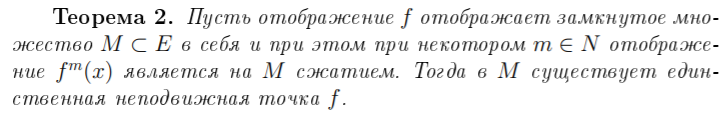
компьютерных технологий и систем

Минск 2021

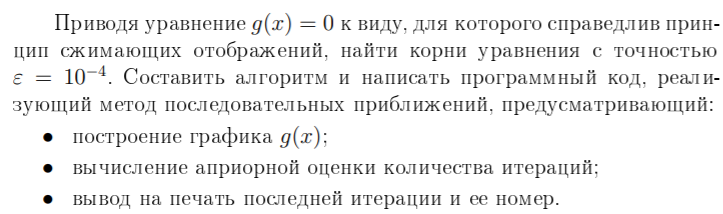
# **Задание №1**

**Краткая теория:** **



*Постановка задачи:*





**Решение:**

Определим через построение графика количество корней, коих у нас получается 3.

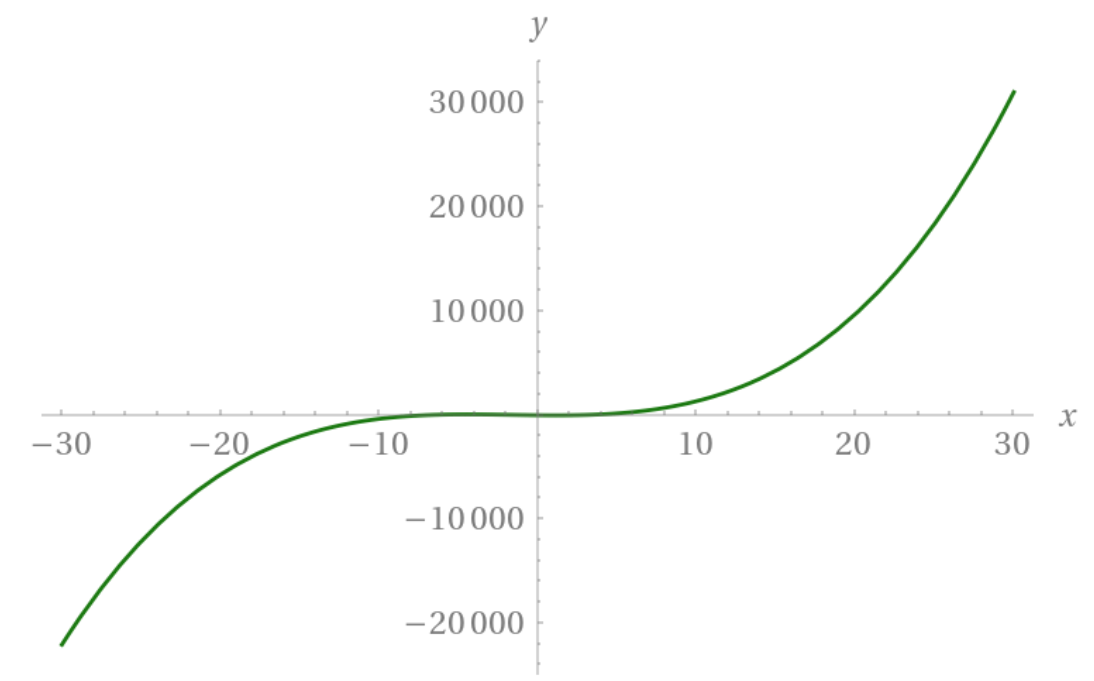
Найдем шар, инвариантен относительно на котором отображение будет сжимающим, дабы привести уравнение вида к виду . Применим локальный принцип сжимающих отображений.

Перепишем уравнение в виде:

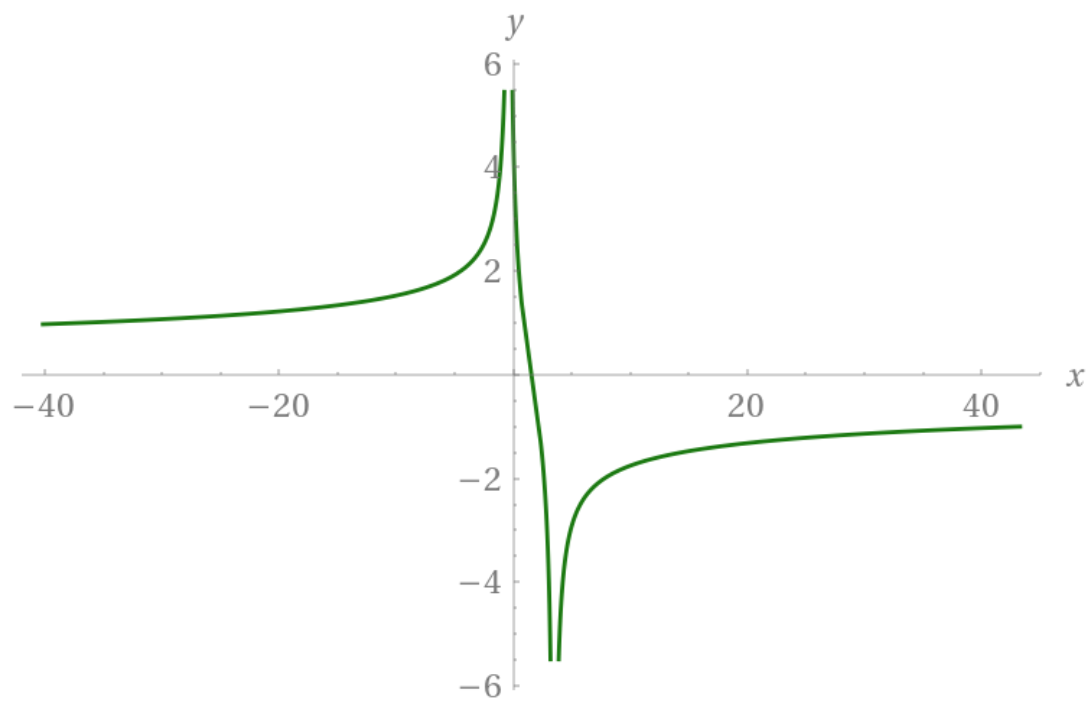
Следовательно:

Функция дифференцируема, следовательно в качестве константы Липшица можно взять: , в нашем случае — это .

Изобразим графики (рис. 1) и (рис. 2):



*Рис. 1*



*Рис. 2*

В окрестности корня, находящегося на промежутке [-1;0], первая производная имеет неограниченные значения. Следовательно, для этого корня указанное *f(x)* не подходит.

Строим шар *B*[]. Проверим выполнение локального принципа сжимающих отображений:  
 – верно.

Следовательно, в шаре *B*[] существует неподвижная точка. Априорную оценка числа итераций:

Построим шар *B*[]. Проверим выполнение локального принципа сжимающих отображений:

– верно.

Следовательно, в шаре B[] существует неподвижная точка. Априорная оценку числа итераций:

Перепишем уравнение в виде . Тогда f(. В силу дифференцируемости : .

В окрестности оставшегося из корней первая производная ограничена, потому строим *B*[]. Проверим выполнение локального принципа сжимающих отображений:

Условие выполняется. Следовательно, в шаре B[] существует неподвижная точка. Априорную оценку числа итераций:

**Код программы:**

def first\_phi\_function(first\_x):  
 return (-5 \* first\_x \*\* 2 + 15 \* first\_x + 7) \*\* (1 / 3)  
  
  
def second\_phi\_function(second\_x):  
 return (second\_x \*\* 3 + 5 \* second\_x \*\* 2 - 7) / 15  
  
  
epsilon = 10\*\*(-4)  
  
counter = 0  
x = 2.425  
print("B[2.425; 0.025]")  
while True:  
 counter += 1  
 y = first\_phi\_function(x)  
 if abs((x - y)) < epsilon:  
 break  
 else:  
 x = y  
print("Количество итераций: " + str(counter) + "\nКорень: " + str(y))  
  
counter = 0  
x = -6.9  
print("\nB[-6.9; 0.2]")  
while True:  
 counter += 1  
 y = first\_phi\_function(x)  
 if abs((x - y)) < epsilon:  
 break  
 else:  
 x = y  
print("Количество итераций: " + str(counter) + "\nКорень: " + str(y))  
  
counter = 0  
x = -0.4  
print("\nB[-0.4; 0.1]")  
while True:  
 counter += 1  
 y = second\_phi\_function(x)  
 if abs((x - y)) < epsilon:  
 break  
 else:  
 x = y  
print("Количество итераций: " + str(counter) + "\nКорень: " + str(y))

**Ответ:**

B[2.425; 0.025]

Количество итераций: 9

Корень: 2.4141

B[-6.9; 0.2]

Количество итераций: 13

Корень: -6.9999

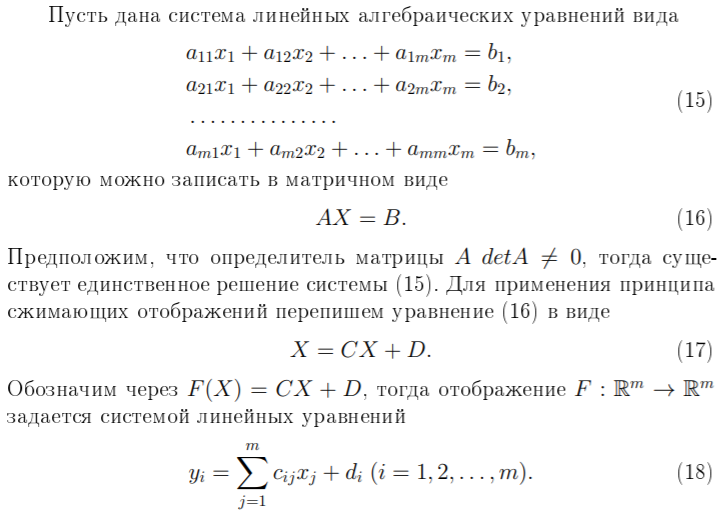
B[-0.4; 0.1]

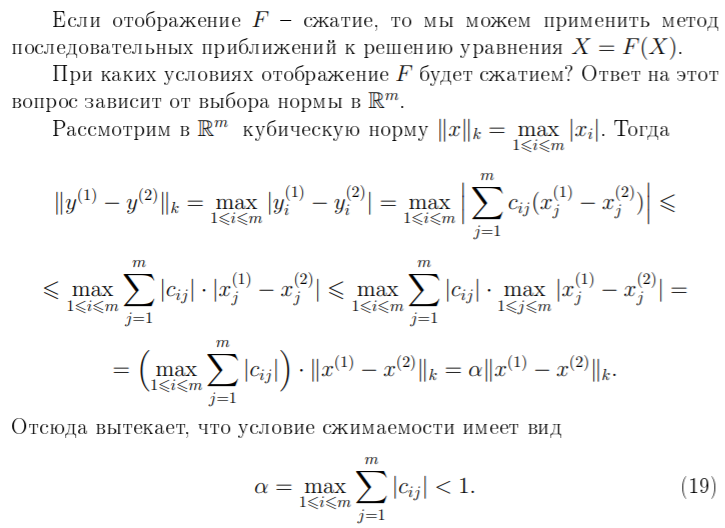
Количество итераций: 5

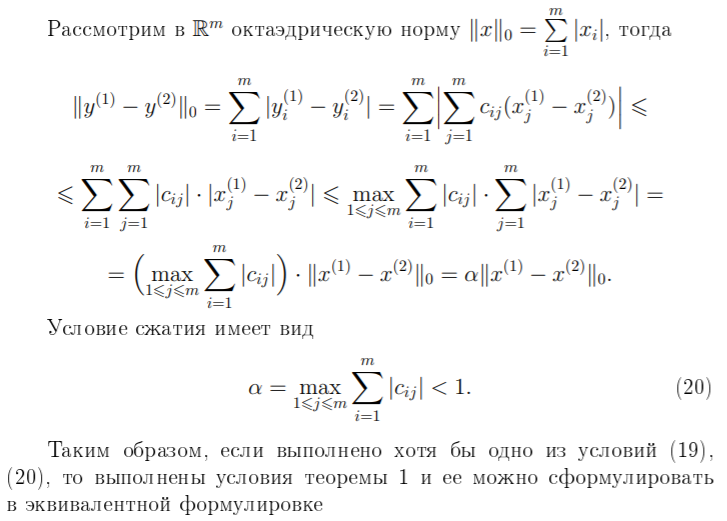
Корень: -0.4142

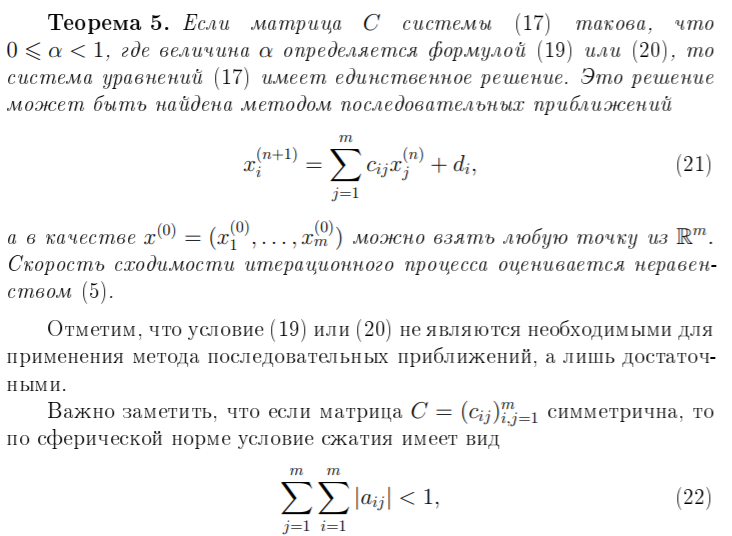
# **Задание 2**

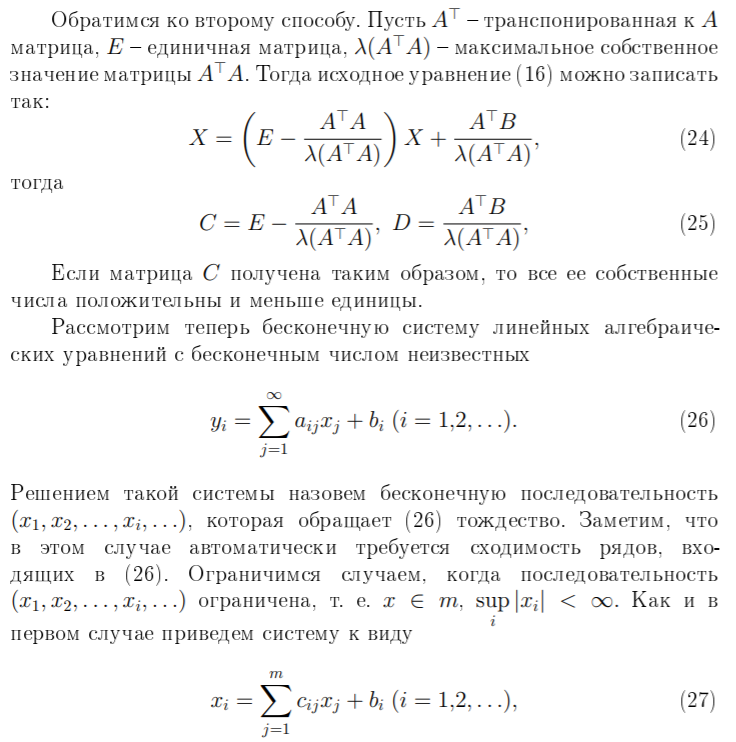
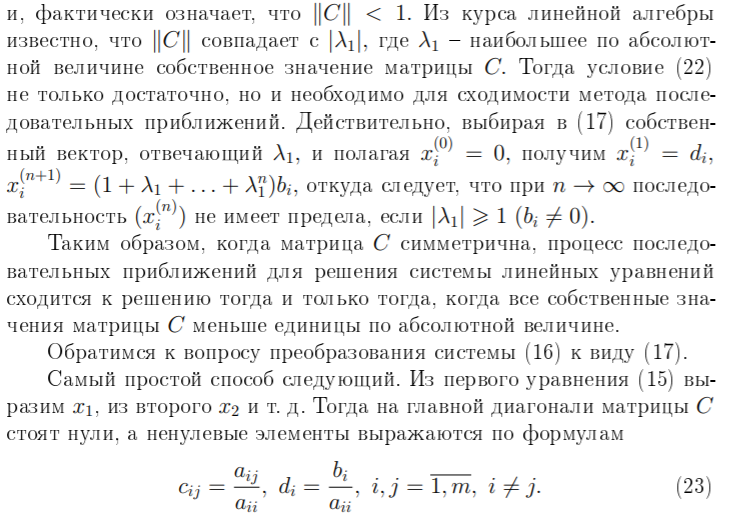
**Краткая теория:**

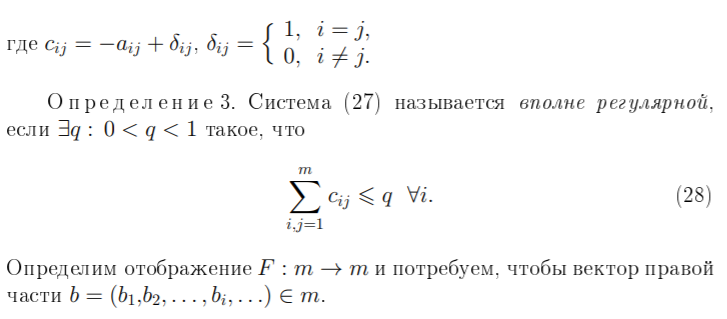
****

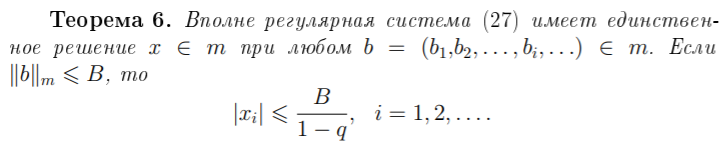
****

****

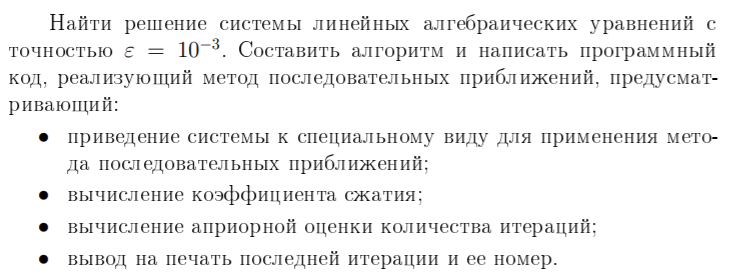
****

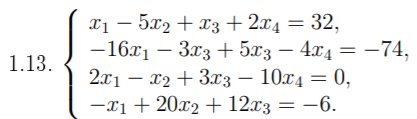
****





**Постановка задачи:**





**Решение:**

Имеем матрицу: A = ,

Приведем матрицу к положительно определенной и симметричной через умножение на саму себя транспонированную. Используем методом Якоби для построения столбца g и матрицы B:

– к-ое и к+1-ое итерационные приближения точного решения , а матрица и вектор :

Имеем матрицу B: B =

Ее октаэдрическая норма *||B||* равна 0,9798 < 1, следовательно метод простых итераций применим и устойчив. Априорная оценка числа итераций:

**Код программы:**

I = map()  
e = 10\*\*(-3)  
  
  
def new\_f(matrix\_a, matrix\_x):  
 matrix\_f = [[]]  
 for i in range(4):  
 matrix\_f[0][i] = 0  
 for j in range(4):  
 matrix\_f[0][j] = matrix\_f[0][i] + matrix\_a[i][j] \* matrix\_x[0][j]  
 return matrix\_f  
  
  
def first\_norm(matrix\_a):  
 maximum = 0  
 for j in range(4):  
 norma = 0  
 for i in range(4):  
 norma += abs(matrix\_a[i][j])  
 maximum = max(maximum, norma)  
 return maximum  
  
  
def e\_testing(matrix\_x):  
 summary = 0  
 for i in range(4):  
 summary += summary + matrix\_x[0][i] \* matrix\_x[0][i]  
 return summary\*\*(1 / 2)  
  
  
def matrix\_multiplication(matrix\_a, matrix\_b):  
 matrix\_c = [[]]  
 for i in range(4):  
 for j in range(4):  
 matrix\_c[i][j] = 0  
 for k in range(4):  
 matrix\_c[i][j] = matrix\_c[i][j] + matrix\_a[i][k] \* matrix\_b[k][j]  
 return matrix\_c  
  
  
def matrix\_tranposing(matrix\_a):  
 matrix\_b = [[]]  
 for i in range(4):  
 for j in range(4):  
 matrix\_b[i][j] = matrix\_a[j][i]  
 return matrix\_b  
  
  
a = [[1, -5, 1, 2], [-16, -3, 5, -4], [-1, 20, 12, 0], [2, -1, 3, -10]]  
f = [[32, -74, 0, -6]]  
at = matrix\_tranposing(a)  
x = [[]]  
aa = matrix\_multiplication(at, a)  
ff = new\_f(at, f)  
n = first\_norm(aa)  
b = []  
for i in range(4):  
 for j in range(4):  
 if i == j:  
 b[i][j] = 1 - aa[i][j] / n  
 else:  
 b[i][j] = -aa[i][j] / n  
g = []  
for i in range(4):  
 for j in range(4):  
 g[i][j] = ff[i][j] / n  
counter = 0  
x\_it = g  
eps = 0  
while True:  
 temp = new\_f(b, x\_it)  
 counter += 1  
 tmp = [temp[0][0], temp[0][1], temp[0][2], temp[0][3]]  
 for i in range(1):  
 for j in range(4):  
 temp[i][j] = temp[i][j] = x\_it[i][j]  
 x\_it = tmp  
 eps = e\_testing(temp)  
 if eps <= e:  
 break  
x = x\_it  
print("Количество итераций: " + str(counter))  
print("Коэффицент сжатия: " + str(eps))  
print("Полученные корни: "+ x\_it)

**Ответ:**

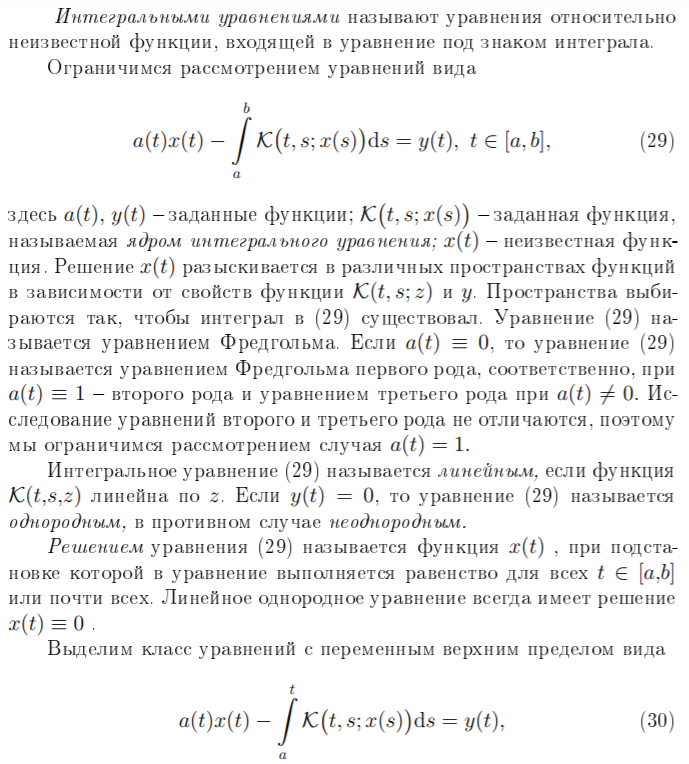
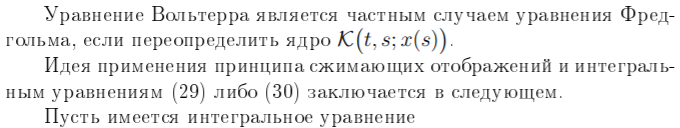
Количество итераций: 197

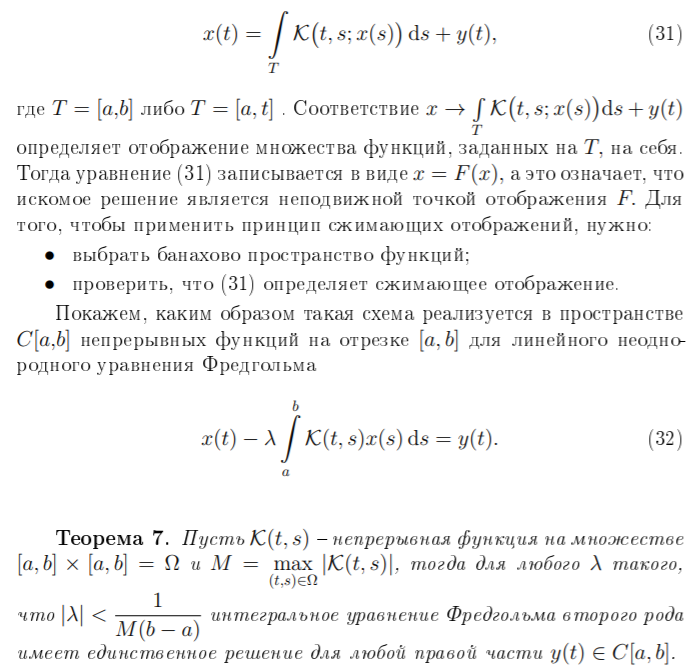
Коэффицент сжатия: 0.00040004428909412850847

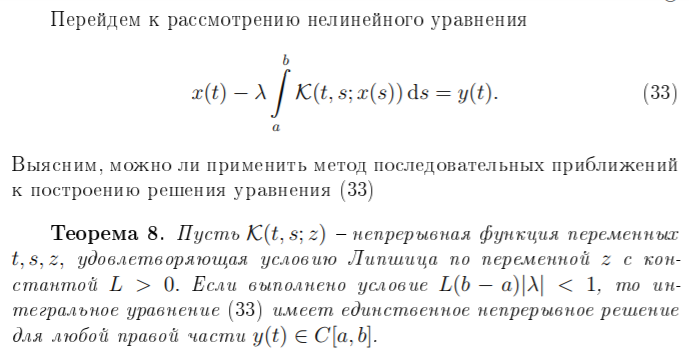
Полученные корни:

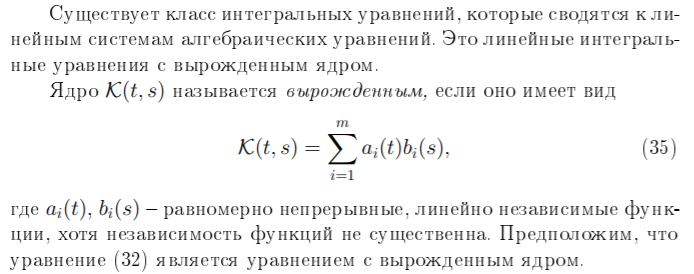
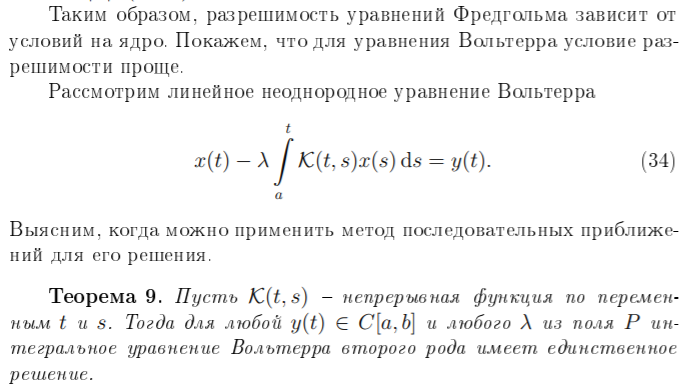
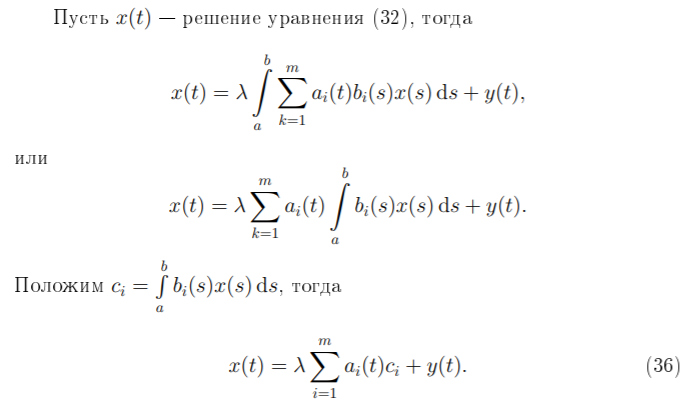
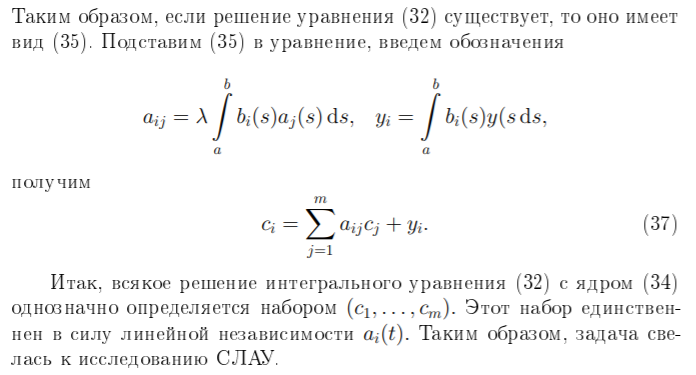
# **Задание 3**

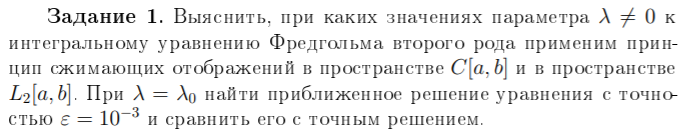
**Краткая теория (к заданиям 3 и 4):**





  **Постановка задачи:**



****

**Решение:**

Приведем уравнение к виду: Пусть . Рассмотрим пространство . Отображение *F* - сумма двух функций (причём непрерывных)*,* следовательно отображение *F* задает отображение . Докажем, что отображение на является сжимающим:

– коэффициент сжатия. Тогда при к уравнению (исходному) в пространстве можно применить принцип сжимающих отображений*.* Допустим, . Оценим число приближений по следующей формуле:

Пусть , ,

Нужная точность достигается при .

Найдем точное решение уравнения. Пусть Тогда . Подставляя его в исходное уравнение, получаем . Значит, точное решение имеет вид:

Рассмотрим пространство . Оценим ядро

Отображение F отображает пространство на себя. Оно сжимающееся, если . При можно применить принцип сжимающих отображений в пространстве . Вычислим количество необходимых итераций:

Тогда n = 4.

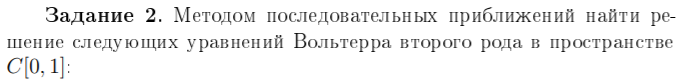
**Ответ:**

– искомое количество итераций,вид функции: на итерации, точное решение уравнения: .

# **Задание 4**

**Краткая теория (см. Теорию к Заданию 3):**

**Постановка задачи:**



****

**Решение:**

Интегральное уравнение Вольтерра разрешимо в пространстве непрерывных функций при любой правой части. Построим последовательные приближения по правилу:

Пусть - начальное приближение. Тогда:

*; ;*

Тогда:

Исследуем ряд на сходимость:

сходится (бесконечно убывающая геометрическая прогрессия), следовательно для него существует сходящаяся числовая мажоранта, значит, по признаку Вейерштрасса он сходится к некой функции, которая в сумме c t и есть решение.

**Ответ:**