БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Цуранов Никита Васильевич

**Контрольная самостоятельная работа**

**Принцип сжимающих отображений**

Студента 2 курса 6 группы

|  |
| --- |
| Преподаватель  Чеб Елена Сергеевна |

**Содержание**

[**Задание 1 (Тема 2, Задание 2, Вариант 21(7))** 2](#_Toc7433116)

[**Задание 2 (Тема 3, Задание 1, Вариант 21(7))** 6](#_Toc7433117)

[**Задание 3 (Тема 4, Задание 1, Вариант 21(7))** 8](#_Toc7433118)

[**Задание 4 (Тема 4, Задание 2, Вариант 21(7))** 10](#_Toc7433119)

# **Задание 1 (Тема 2, Задание 2, Вариант 21(7))**

2.7. ;

* *Теоретическая часть*

Решение уравнения основывается на поиске неподвижной точки x такой, что , которая является корнем уравнения .

*Принцип сжимающих отображений:*

Пусть отображение *f* отображает замкнутое в банаховом пространстве *Е* множество *М* в себя и является на *М* сжимающим с коэффициентом сжатия α. Тогда на множестве *М* отображение *f* имеет единственную неподвижную точку которая может быть найдена методом последовательных приближений

где , , и справедлива оценка:

Для решения задачи нужно привести уравнение к и применим принцип сжимающих отображений. Для этого найдем шар , инвариантный относительно отображения *f,* на котором отображение будет сжимающим. В качестве константы Липшица мы можем взять . Радиус шара выберем из условий

а дальше оценим расстояние и найдем решение.

* *Код*

xSymbol = sympy.Symbol(**'x'**)  
eps = 1e-4  
g = **lambda** x: 3\*x\*\*2 - 10\*x - 141  
print(**"g(x) ="**, g(xSymbol))  
xRange = np.arange(-10, 10, 0.1)  
plt.plot(xRange, g(xRange))  
plt.plot(xRange, 0 \* xRange)  
plt.show()  
  
  
**def** calculate(g, f):  
 absdf = **lambda** x: np.abs(derivative(f, x, dx=eps))  
 print(**"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"**)  
 print(**"f(x) ="**, f(xSymbol))  
 print(**"f'(x) ="**, sympy.diff(f(xSymbol), xSymbol))  
  
 xRange = np.arange(-10, 10.1, 0.1)  
 xRange = xRange[np.where(np.abs(absdf(xRange)) < 1)]  
 **if** len(xRange) > 1:  
 print(**"Отображение сжимающее для x из (%.4f, %.4f)"**

% (xRange[0], xRange[len(xRange) - 1]))  
  
 found = **False  
 for** a **in** xRange:  
 **for** r0 **in** np.arange(0.1, 0.5, 0.1):  
 L = max(absdf(np.arange(a - r0, a + r0, eps)))  
 **if** np.abs(a - f(a)) < (1 - L) \* r0 **and** L < 1:  
 print(**"Сжимающий коэффициент: %.4f"** % L)  
 print(**"%.4f < %.4f"** % (np.abs(a - f(a)), (1 - L) \* r0))  
 print(**"%.4f < 1"** % L)  
 print(**"Условия выполняются"**)  
 print(**"a = %.4f, r0 = %.4f"** % (a, r0))  
 estimate =

int(np.ceil(np.log((1 - L) \* eps / np.abs(a - f(a))) / np.log(L)))  
 found = **True** print(**"Примерная оценка: %d"** % estimate)  
 **for** i **in** range(estimate):  
 a = f(a)  
 print(**"Значение g(%.5f) = %.4e"** % (a, g(a)))  
 **break  
 if** found:  
 **break  
 if not** found:  
 print(**"Условия не выполняются"**)  
  
  
f = **lambda** x: 0.3\*x\*\*2 - 14.1  
calculate(g, f)  
  
f = **lambda** x: ((10\*x + 141) / 3)\*\*0.5  
calculate(g, f)  
  
f = **lambda** x: 141 / (3\*x - 10)  
calculate(g, f)

* *Решение*

g(x) = 3\*x\*\*2 - 10\*x - 141

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

f(x) = 0.3\*x\*\*2 - 14.1

f'(x) = 0.6\*x

Отображение сжимающее для x из (-1.6000, 1.6000)

Условия не выполняются

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

f(x) = (10\*x/3 + 47)\*\*0.5

f'(x) = 1.66666666666667\*(10\*x/3 + 47)\*\*(-0.5)

Отображение сжимающее для x из (-10.0000, 10.0000)

Сжимающий коэффициент: 0.1942

0.2603 < 0.3223

0.1942 < 1

Условия выполняются

a = 8.4000, r0 = 0.4000

Примерная оценка: 5

Значение g(8.72192) = -3.4881e-03

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

f(x) = 141/(3\*x - 10)

f'(x) = -423/(3\*x - 10)\*\*2

Отображение сжимающее для x из (-10.0000, -3.6000)

Сжимающий коэффициент: 0.6306

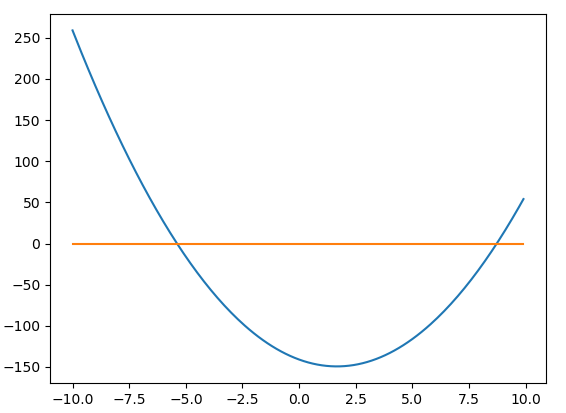
0.0183 < 0.0369

0.6306 < 1

Условия выполняются

a = -5.4000, r0 = 0.1000

Примерная оценка: 14

Значение g(-5.38868) = 5.6580e-04

# **Задание 2 (Тема 3, Задание 1, Вариант 21(7))**

1.7.

* *Теоретическая часть*

Пусть задана система вида , тогда перепишем систему в виде . Таким образом решение системы равносильно нахождению неподвижной точки отображения , а , тогда выполняется оценка:

* *Код*

xSymbol = sympy.Symbol(**'x'**)  
eps = 1e-3  
A = np.array([  
 [0.9, 0.1, 0.1],  
 [0.2, 1.1, 0.2],  
 [0.1, 0.2, 1.2]  
])  
b = np.array([1, 1, 0])  
  
B = -A + np.identity(len(A))  
y = -b  
x = np.zeros(len(A))  
  
print(**"A:\n"**, A)  
print(**"b: "**, b)  
print(**"B:\n"**, B)  
print(**"y: "**, y)  
normB = np.linalg.norm(B)  
print(**"||B|| ="**, normB)  
nextApproximation = **lambda** B, x, y: np.dot(B, x) - y  
estimate = int(np.log(eps \* (1 - normB) / np.linalg.norm(x - nextApproximation(B, x, y))) / np.log(normB)) + 1  
print(**"Примерная оценка:"**, estimate)  
  
**for** i **in** range(estimate):  
 print((**" %+.4f"** \* len(x)) % tuple(value **for** value **in** x))  
 x = nextApproximation(B, x, y)  
  
print(**"Решение:"**, x)  
print(**"F(x) ="**, np.dot(A, x) - b)

* *Решение*

A:

[[0.9 0.1 0.1]

[0.2 1.1 0.2]

[0.1 0.2 1.2]]

b: [1 1 0]

B:

[[ 0.1 -0.1 -0.1]

[-0.2 -0.1 -0.2]

[-0.1 -0.2 -0.2]]

y: [-1 -1 0]

||B|| = 0.45825756949558405

Примерная оценка: 11

0.0000 0.0000 0.0000

1.0000 1.0000 0.0000

1.0000 0.7000 -0.3000

1.0600 0.7900 -0.1800

1.0450 0.7450 -0.2280

1.0528 0.7621 -0.2079

1.0499 0.7548 -0.2161

1.0511 0.7578 -0.2127

1.0506 0.7565 -0.2141

1.0508 0.7570 -0.2135

1.0507 0.7568 -0.2138

Решение: [ 1.05076734 0.75692613 -0.21368477]

F(x) = [1.474386e-05 3.525417e-05 4.023066e-05]

# **Задание 3 (Тема 4, Задание 1, Вариант 21(7))**

1.21. Выяснить при каких значениях параметра λ к интегральному уравнению Фридгольма второго рода применим принцип сжимающих отображений в пространстве . Найти решение уравнения с точностью и сравнить с точным решением.

* *Теоретическая часть*

Отображение является сжимающим, если существует константа : . Проведя оценку найдем константу Липшица , тогда справедлива оценка

* *Код*

xSymbol = sympy.Symbol(**'x'**)  
sSymbol = sympy.Symbol(**'s'**)  
tSymbol = sympy.Symbol(**'t'**)  
a = 0  
b = 1  
down = 0  
up = 1  
eps = 1e-3  
h = 1.11  
  
kernel = **lambda** s, t: s\*\*2 \* (t\*\*2 + 1)  
**def** F(x):  
 f = sympy.integrate(kernel(sSymbol, tSymbol) \* x, sSymbol)  
 **return** h \* (f.subs(sSymbol, up) - f.subs(sSymbol, down)).subs(tSymbol, sSymbol) + 1  
norm = **lambda** x: max(np.abs(x.subs(sSymbol, value)) **for** value **in** np.linspace(a, b, 100))  
  
intKernel = sympy.integrate(kernel(sSymbol, tSymbol), sSymbol)  
alfa = max(  
 np.abs((intKernel.subs(sSymbol, up) - intKernel.subs(sSymbol, down))  
 .subs(tSymbol, value)) **for** value **in** np.linspace(a, b, 100))  
  
print(**"Ядро интеграла:"**, kernel(sSymbol, tSymbol))  
print(**"Коэффициент сжатия: %.3f"** % alfa)  
print(**" => для |h| < %.3f можно применить метод сжимающих отображений"** % (1/alfa))  
  
x = sympy.S(0)  
print(**"||x0 - x1|| = %.2f"** % norm(x - F(x)))  
estimate = int(np.log(float(eps \* (1 - alfa) / norm(x - F(x)))) / np.log(float(alfa))) + 1  
print(**"Примерная оценка: %d"** % estimate)

**while** norm(x - F(x)) > eps:  
 print(**""**, x.evalf(5))  
 x = F(x)  
  
print(**"Решение уравнения при h = %.2f"** % h)  
print(**" x(t) ="**, x.subs(sSymbol, tSymbol).evalf(5))  
print(**" x(t) - %.2f Ss^2(t^2 + 1)x(s)ds ="** % h,(  
 x.subs(sSymbol, tSymbol)  
 - h \* sympy.integrate(kernel(sSymbol, tSymbol) \* x, sSymbol).subs(sSymbol, up)  
 + h \* sympy.integrate(kernel(sSymbol, tSymbol) \* x, sSymbol).subs(sSymbol, down)  
 ).evalf(5)  
)

* *Решение*

Ядро интеграла: s\*\*2\*(t\*\*2 + 1)

Коэффициент сжатия: 0.667

=> для |h| < 1.500 можно применить метод сжимающих отображений

||x0 - x1|| = 1.00

Примерная оценка: 20

0

1.0000

0.37\*s\*\*2 + 1.37

0.58904\*s\*\*2 + 1.589

0.71871\*s\*\*2 + 1.7187

0.79548\*s\*\*2 + 1.7955

0.84092\*s\*\*2 + 1.8409

0.86783\*s\*\*2 + 1.8678

0.88375\*s\*\*2 + 1.8838

0.89318\*s\*\*2 + 1.8932

0.89876\*s\*\*2 + 1.8988

0.90207\*s\*\*2 + 1.9021

0.90402\*s\*\*2 + 1.904

0.90518\*s\*\*2 + 1.9052

Решение уравнения при h = 1.11

x(t) = 0.90587\*t\*\*2 + 1.9059

x(t) - 1.11 Ss^2(t^2 + 1)x(s)ds = -0.00040587\*t\*\*2 + 0.99959

# **Задание 4 (Тема 4, Задание 2, Вариант 21(7))**

1.21. Методом последовательных приближений найти решение следующих уравнений Вольтера второго рода в пространстве .

* *Теоретическая часть*

Подход к уравнениям Вольтера отличается тем, что известно о разрешимости уравнения в пространстве. Построим последовательность приближений . Останавливаем итерации, когда следующий член будет незначим, т.е. по модулю меньше точности.

* *Код*

xSymbol = sympy.Symbol(**'x'**)  
sSymbol = sympy.Symbol(**'s'**)  
tSymbol = sympy.Symbol(**'t'**)  
a = 0  
b = 1  
down = 0  
up = tSymbol  
eps = 1e-3  
  
kernel = **lambda** s, t: 1  
**def** F(x):  
 f = sympy.integrate(kernel(sSymbol, tSymbol) \* x, sSymbol)  
 **return** (f.subs(sSymbol, up) - f.subs(sSymbol, down)).subs(tSymbol, sSymbol) + sSymbol\*\*3/3 - 2 \* sSymbol  
norm = **lambda** x: max(np.abs(x.subs(sSymbol, value)) **for** value **in** np.linspace(a, b, 100))  
  
intKernel = sympy.integrate(kernel(sSymbol, tSymbol), sSymbol)  
alfa = max(  
 np.abs((intKernel.subs(sSymbol, up) - intKernel.subs(sSymbol, down))  
 .subs(tSymbol, value)) **for** value **in** np.linspace(a + eps, b - eps, 100))  
  
print(**"Ядро интеграла:"**, kernel(sSymbol, tSymbol))  
  
x = 0  
**while** max(np.abs((F(x) - x).subs(sSymbol, value)) **for** value **in** np.linspace(a, b, 100)) > eps:  
 print(**""**, x)  
 x = F(x)  
  
print(**"Решение уравнения: "**)  
print(**" x(t) ="**, x.subs(sSymbol, tSymbol).evalf(5))  
print(**" x(t) - Sx(s)ds ="**,(  
 x.subs(sSymbol, tSymbol)  
 - sympy.integrate(kernel(sSymbol, tSymbol) \* x, sSymbol).subs(sSymbol, up)  
 + sympy.integrate(kernel(sSymbol, tSymbol) \* x, sSymbol).subs(sSymbol, down)  
 ).evalf(5)  
)

* *Решение*

Ядро интеграла: 1

0

s\*\*3/3 - 2\*s

s\*\*4/12 + s\*\*3/3 - s\*\*2 - 2\*s

s\*\*5/60 + s\*\*4/12 - s\*\*2 - 2\*s

s\*\*6/360 + s\*\*5/60 - s\*\*2 - 2\*s

s\*\*7/2520 + s\*\*6/360 - s\*\*2 - 2\*s

Решение уравнения:

x(t) = 4.9603e-5\*t\*\*8 + 0.00039683\*t\*\*7 - t\*\*2 - 2.0\*t

x(t) - Sx(s)ds =

-5.5115e-6\*t\*\*9 + 0.00039683\*t\*\*7 + 0.33333\*t\*\*3 - 2.0\*t