МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра компьютерных технологий и систем

Отчёт по контролируемой самостоятельной работе

«Применение принципа сжимающих отображений в нормированных векторных пространствах»

Вариант 10

Выполнила:

Лавринович Анна Павловна
студентка 3 курса 7 группы

Преподаватель:

Чеб Елена Сергеевна

Минск, 2024 г.

ПРИНЦИП СЖИМАЮЩИХ ОТОБРАЖЕНИЙ

ПРИНЦИП СЖИМАЮЩИХ ОТОБРАЖЕНИЙ В БАНАХОВЫХ ПРОСТРАНСТВАХ

Пусть в банаховом пространстве E действует отображение f.

• *Точка $x^* \in E$ называется* ***неподвижной точкой*** *отображения f, если*

$$f(x^*) = x^*.$$

Таким образом, неподвижные точки f — это решения уравнения

$$x = f(x),$$

а поскольку к такому виду довольно часто удается преобразовать уравнение $F(x) = 0$, где $F : X \to Y$, причем X, Y являются банаховыми пространствами, то важность определения неподвижных точек не вызывает сомнения.

• *Отображение f называется* ***сжимающим (сжатием)****, если* $\exists \alpha \in \mathbb{R},\ 0 < \alpha < 1$:

$$\|f(x) - f(y)\|_E \leqslant \alpha \|x - y\|_E, \quad \forall x, y \in E.$$

Число α называется ***коэффициентом сжатия****.*

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

$$x_n = f(x_{n-1}),\ n = 1, 2 \ldots$$

где $(x_n) \subset M$ и $x_n \xrightarrow[n\to\infty]{} x^*$. Кроме того, справедлива оценка сходимости

$$\|x_n - x^*\| \leqslant \frac{\alpha^n}{1 - \alpha} \|x_0 - x_1\|.$$

Следствие. *Пусть f отображает банахово пространство E само на себя и является сжатием. Тогда f имеет единственную неподвижную точку, которая может быть найдена методом последовательных приближений.*

Метод последовательных приближений позволяет построить приближенное значение уравнения $x = f(x)$. Поскольку точное решение уравнения, как правило, неизвестно, то для организации итерационного процесса используют следующие оценки точности:

- априорная оценка
$$\|x_n - x^*\| \leqslant \frac{\alpha^n}{1-\alpha} \|x_0 - x_1\|;$$
- апостериорная оценка
$$\|x_n - x^*\| \leqslant \frac{\alpha}{1-\alpha} \|x_n - x_{n+1}\|.$$

С помощью априорной оценки можно предварительно оценить достаточное число итераций для нахождения приближенного значения с заданной точностью из неравенства

$$\frac{\alpha^n}{1-\alpha} \|x_0 - x_1\| \leqslant \varepsilon.$$

Откуда

$$n_{apr} = \left[\log_\alpha \frac{\varepsilon(1-\alpha)}{\|x_0 - x_1\|} \right] + 1.$$

Апостериорная оценка используется в процессе организации итерационного процесса, где на каждом шаге сравнивают значения x_n и x_{n-1} по формуле

$$\frac{\alpha}{1-\alpha} \|x_n - x_{n+1}\| \leqslant \varepsilon.$$

Фактическое число итераций всегда не превышает n_{apr}.

Теорема. *Пусть отображение f отображает замкнутое множество $M \subset E$ в себя и при этом при некотором $m \in \mathbb{N}$ отоюражение $f^m(x)$ является на M сжатием. Тогда в M существует единственная точка f.*

Следствие. *Пусть отображение f отображает замкнутое выпуклое множество $M \subset E$ в себя, причем на M оно непрерывно дифференцируемо и*

$$\|f'(x)\|_E \leqslant \alpha < 1.$$

Тогда справедливы утверждения первой теоремы.

ПРИМЕНЕНИЕ ПРИНЦИПА СЖИМАЮЩИХ ОТОБРАЖЕНИЙ ДЛЯ РЕШЕНИЯ УРАВНЕНИЙ

Одним из подходов для приближенного решения уравнений можно отнести метод последовательных приближений. Остановимся на его рассмотрении.

Пусть задано уравнение

$$x = f(x),$$

где $f : [a, b] \to [a, b]$. Сформулируем для него принцип сжимающих отображений.

Теорема. *Пусть f удовлетворяет условию Липшица с константой $L < 1$. Тогда уравнение $x = f(x)$ имеет единственное решение $x^* \in [a, b]$, которое может быть найдено методом последовательных приближений*

$$x_n = f(x_{n-1}),\ n = 1, 2, \ldots$$

Применим теорему к решению уравнения, заданного в общем виде

$$g(x) = 0.$$

Предположим, что $g(x) \in C^{(1)}[a, b]$. Пусть выполнены на $[a, b]$ следующие ограничения

$$0 < k_1 \leqslant g'(x) \leqslant k_2.$$

Перепишем исходное уравнение в виде

$$x = x - \lambda g(x) \quad \text{или} \quad x = f(x),$$

где $f(x) = x - \lambda g(x)$. С помощью ограничений выберем параметр λ таким образом, чтобы отображение f переводило $[a, b]$ в себя и при этом было сжимающим. Тогда

$$1 - \lambda k_2 \leqslant f'(x) = 1 - \lambda g'(x) \leqslant 1 - \lambda k_2.$$

В качестве λ можно взять точку минимума функции

$$h(\lambda) = \max\{|1 - \lambda k_1|, |1 - \lambda k_2|\},$$

то есть

$$\lambda^* = \frac{2}{k_1 + k_2}.$$

В этом случае

$$|f'(x)| \leqslant \frac{k_2 - k_1}{k_2 + k_1} < 1.$$

Так как уравнение $g(x) = 0$ имеет решение, то $a < f(a)$, $b > f(b)$, а это означает, что $f : [a, b] \to [a, b]$. Следовательно к полученным уравнениям применим принцип сжимающих отображений. Для вычисления коэффициента сжатия можно воспользоваться оценкой на производную.

ЗАДАЧА 1.

Приводя уравнение

$$x^{13} + x^7 + x - 1 = 0 \tag{1}$$

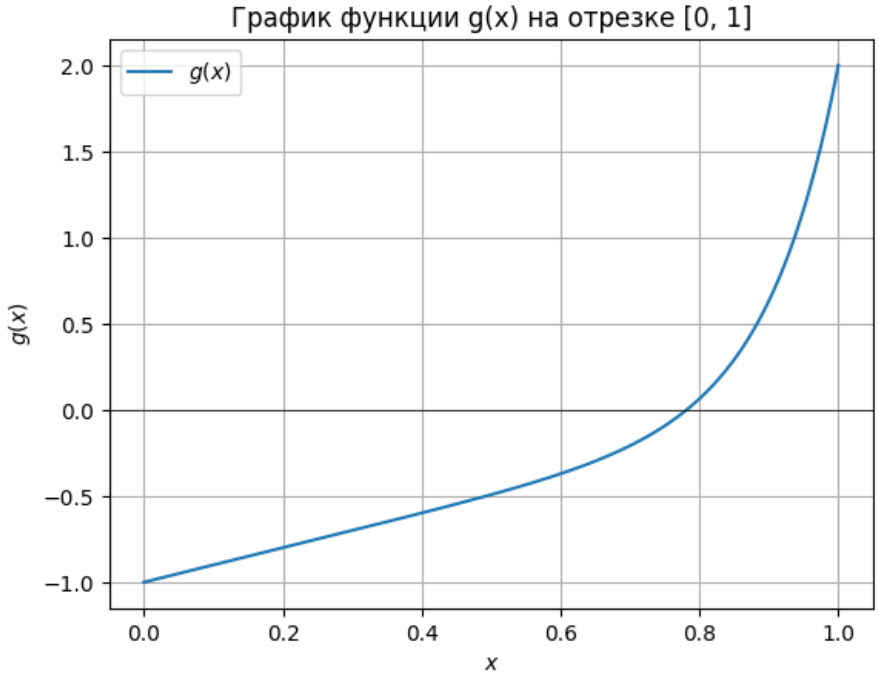
к виду, для которого справедлив принцип сжимающих отображений, найти корни уравнения с точностью $\varepsilon = 10^{-4}$. Составить алгоритм и написать программный код, реализующий метод последовательных приближений, предусматривающий:

- построение графика $g(x)$;
- вычисление априорной оценки количества операций;
- вывод на печать последней итерации и ее номера.

Рассматриваем уравнение (1). Обозначим его как $g(x) = 0$. Возьмем отрезок $[0, 1]$. Это отрезок выбран таким образом, что

$$g(0) < 0, \quad g(1) > 0.$$

Построим график данной функции на выбранном отрезке.



Функция $g(x) \in C^{(1)}[0,1]$, то есть она непрерывно дифференцируема на рассматриваемом отрезке. Следовательно, возьмем производную

$$g'(x) = 13x^{12} + 7x^6 + 1$$

Сделаем грубую оценку этой производной. Функция $g'(x)$ возрастает на отрезке $[0,1]$. Поэтому оценим производную по значениям на крайних точках отрезка следующим образом: $g'(0) = 1$, $g'(1) = 21$, тогда

$$1 \leqslant g'(x) \leqslant 21$$

Таким образом, мы получили значения

$$k_1 = 1, \quad k_2 = 21.$$

Перепишем исходное уравнение в виде $x = x - \lambda g(x) = f(x)$. Причем в качестве λ возьмем значение

$$\lambda = \frac{2}{k_1 + k_2} = \frac{2}{1 + 21} \approx 0.09.$$

То есть с помощью ограничений мы выбрали параметр λ таким образом, чтобы полученное отображение было сжимающим:

$$x = \underbrace{x - 0.09 \cdot (x^{13} + x^7 + x - 1)}_{f(x)}.$$

Для производной можно воспользоваться оценкой

$$|f'(x)| \leqslant \frac{k_2 - k_1}{k_2 + k_1} = \frac{21 - 1}{21 + 1} \approx 0.9091.$$

Это значение мы можем использовать в качестве коэффициента сжатия, т.е.

$$\alpha = 0.9091.$$

Мы получили отображение f, которое будет являться сжимающим, и вычислили для него коэффициент сжатия. Тогда по теореме полученное уравнение $x = f(x)$ имеет единственное решение $x^* \in [a, b]$, которое может быть найдено методом последовательных приближений

$$x_n = x_{n-1} - 0.09 \cdot \left(x_{n-1}^{13} + x_{n-1}^{7} + x_{n-1} - 1 \right), \; n = 1, 2, \ldots$$

Итерационный процесс метода последовательных приближений мы будем заканчивать при выполнении неравенства (из апостериорной оценки точности):

$$\frac{\alpha}{1-\alpha} \|x_n - x_{n+1}\| \leqslant \varepsilon.$$

Априорное число итераций мы вычислим по формуле (из априорной оценки точности)

$$n_{apr} = \left[\log_\alpha \frac{\varepsilon(1-\alpha)}{\|x_0 - x_1\|} \right] + 1.$$

Выберем в качестве начального приближения

$$x_0 = 0.5.$$

Листинг программы на Python:

```
import numpy as np
  import math

  def f(x_n):
      return x_n - 0.09 * (x_n**13 + x_n**7 + x_n - 1)

  def apriori(x_0, x_1, epsilon, alpha):
      return math.floor(math.log(epsilon * (1 - alpha) / abs(x_0 - x_1)) /
                                              math.log(alpha)) + 1

  def aposteriori(x_n, x_n1, alpha):
      return alpha / (1 - alpha) * abs(x_n - x_n1)

  def msa(x_0, x_1, alpha, epsilon):
      iterations = 0
      while True:
      iterations += 1
          x_0 = x_1
          x_1 = f(x_0)
          if aposteriori(x_1, x_0, alpha) <= epsilon:
              break
      return x_1, iterations

  x_0 = 0.50
  alpha = 0.9091
  epsilon = 1e-4

  n_apr = apriori(x_0, f(x_0), epsilon, alpha)
  print('apriori number of iterations:', n_apr)

  last_iteration, iterations = msa(x_0, f(x_0), alpha, epsilon)
  print('aposteriori number of iterations:', iterations)
  print('solution:', last_iteration)
```

Результат вывода:

apriori number of iterations: 90
aposteriori number of iterations: 30
solution: 0.7814519062459794

Таким образом, мы получили следующие значения:

- априорное (предполагаемое) число итераций равно 90;
- апостериорное (реальное) число итераций равно 30;
- приближенное решение исходного уравнения равно

$$x^* \approx 0.7815$$

ПРИМЕНЕНИЕ ПРИНЦИПА СЖИМАЮЩИХ ОТОБРАЖЕНИЙ ДЛЯ РЕШЕНИЯ СЛАУ

Пусть дана система линейных алгебраических уравнений вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1m}x_m = b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2m}x_m = b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mm}x_m = b_m, \end{cases} \tag{1}$$

которую можно записать в матричном виде

$$AX = B \tag{2}$$

Предположим, что определитель $\det A \neq 0$, тогда существует единственное решение системы (1). Для применения принципа сжимающих отображений перепишем уравнение (2) в виде

$$X = CX + D \tag{3}$$

Обозначим через $F(X) = CX + D$, тогда отображение $F : \mathbb{R}^m \to \mathbb{R}^m$ задается системой линейных уравнений

$$y_i = \sum_{j=1}^{m} c_{ij}x_j + d_i (i = 1, 2, \ldots, m)$$

Если отображение F — сжатие, то мы можем применить метод последовательных приближений к решению уравнения $X = F(X)$.

Теорема. *Если матрица C системы* (3) *такова, что $0 \leqslant \alpha < 1$, где величина α определяется формулой*

$$\alpha = \max_{1 \leqslant i \leqslant m} \sum_{j=1}^{m} |c_{ij}| < 1$$

или

$$\alpha = \max_{1 \leqslant j \leqslant m} \sum_{i=1}^{m} |c_{ij}| < 1,$$

то система уравнений (3) *имеет единственное решение. Это решение может быть найдено методом последовательных приближений*

$$x_i^{(n+1)} = \sum_{j=1}^{m} c_{ij}x_j^{(n)} + d_i$$

а в качестве $x^{(0)} = \left(x_1^{(0)}, \ldots, x_m^{(0)}\right)$ *можно взять любую точку из* $\mathbb{R}^m$. *Скорость сходимости итерационного процесса оценивается неравенством*

$$\|x_n - x^*\| \leqslant \frac{\alpha^n}{1-\alpha} \|x_0 - x_1\|.$$

Важно заметить, что если матрица $C = (c_{ij})_{i,j=1}^m$ симметрична, то по сферической норме условие сжатия имеет вид

$$\sum_{j=1}^{m} \sum_{i=1}^{m} |a_{ij}| < 1$$

и, фактически означает, что $\|C\| < 1$. Из курса линейной алгебры известно, что $\|C\|$ совпадает с $|\lambda_1|$, где λ_1 — наибольшее по абсолютной величине собственное значение матрицы C. Тогда условие сжатия не только достаточно, но и необходимо для сходимости метода последовательных приближений.

Таким образом, когда матрица C симметрична, процесс последовательных приближений для решения системы линейных уравнений сходится к решению тогда и только тогда, когда все собственные значения матрицы C меньше единицы по абсолютной величине.

Обратимся к вопросу преобразования системы (2) к виду (3). Самый простой способ следующий. Из первого уравнения (1) выразим x_1, из второго x_2 и т. д. Тогда на главной диагонали матрицы C стоят нули, а ненулевые элементы выражаются по формулам

$$c_{ij} = \frac{a_{ij}}{a_{ii}}, d_i = \frac{b_i}{a_{ii}}, i, j = \overline{1, m}, \quad i \neq j$$

Обратимся ко второму способу. Пусть $A^\top$ — транспонированная к A матрица, E — единичная матрица, $\lambda\left(A^\top A\right)$ — максимальное собственное значение матрицы $A^\top A$. Тогда исходное уравнение (2) можно записать так:

$$X = \left(E - \frac{A^\top A}{\lambda\left(A^\top A\right)}\right) X + \frac{A^\top B}{\lambda\left(A^\top A\right)}$$

тогда

$$C = E - \frac{A^\top A}{\lambda\left(A^\top A\right)}, D = \frac{A^\top B}{\lambda\left(A^\top A\right)}$$

Если матрица C получена таким образом, то все ее собственные числа положительны и меньше единицы.

ЗАДАЧА 2

Найти решение системы линейных алгебраических уравнений

$$\begin{cases} -1.4x_1 - 0.1x_3 = 1, \\ 0.1x_1 + 1.1x_2 - 0.1x_3 = 0, \\ 0.1x_2 - 1.2x_3 = -1 \end{cases}$$

с точностью $\varepsilon = 10^{-4}$. Составить алгоритм и написать программный код, реализующий метод последовательных приближений, предусматривающий:

- приведение системы к специальному виду для применения метода последовательных приближений;

- вычисление коэффициента сжатия;
- вычисление априорной оценки количества итераций;
- вывод на печать последней итерации и ее номера.

Данную нам систему перепишем в матричном виде

$$AX = B,$$

где

$$A = \begin{pmatrix} -1.4 & 0 & -0.1 \\ 0.1 & 1.1 & -0.1 \\ 0 & 0.1 & -1.2 \end{pmatrix}, \quad B = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}.$$

Можно легко проверить, что $\det A = 1.833 \neq 0$, то есть матрица A невырождена. Значит для данной системы существует единственное решение. Чтобы применить принцип сжимающих отображений, перепишем уравнение в виде

$$X = \underbrace{CX + D}_{F(X)}.$$

Следующим шагом будем построение матрицы C и вектора D таких, чтобы мы могли применить принцип сжимающих отображений. Для этого воспользуемся формулой

$$C = E - \frac{A^\top A}{\lambda\,(A^\top A)}, D = \frac{A^\top B}{\lambda\,(A^\top A)}$$

Так как полученная таким образом матрица имеет все собственные значения меньше единицы, то мы сможем воспользоваться методом последовательных приближений для отыскания решения системы.

Мы не будем считать вручную матрицу C и D, а сразу реализуем алгоритм для вычисления по указанной выше формуле. Сперва вычислим матрицу C:

```
import numpy as np
  import math

  A = np.array([
      [-1.4, 0, -0.1],
      [0.1, 1.1, -0.1],
      [0, 0.1, -1.2]])
  B = np.array([[1], [0], [-1]])

  max_eigenvalue = np.max(np.linalg.eigvals(np.dot(A.T, A)))
  C = np.eye(A.shape[0]) - (np.dot(A.T, A)) / max_eigenvalue

  print(*C, sep='\n')
```

Результат вывода:

$$[0.01765659 \quad -0.05485166 \quad -0.06482469]$$
$$[-0.05485166 \quad 0.3916452 \quad 0.11468984]$$
$$[-0.06482469 \quad 0.11468984 \quad 0.27196884]$$

Теперь вычислим вектор D:

```
  D = np.dot(A.T, B) / max_eigenvalue
print(*D, sep='\n')
```

Результат вывода:

$$\begin{bmatrix}-0.69811207\end{bmatrix}$$
$$\begin{bmatrix}-0.04986515\end{bmatrix}$$
$$\begin{bmatrix}0.54851662\end{bmatrix}$$

Таким образом, мы получили матрицу

$$C = \begin{pmatrix} 0.01765659 & -0.05485166 & -0.06482469 \\ -0.05485166 & 0.3916452 & 0.11468984 \\ -0.06482469 & 0.11468984 & 0.27196884 \end{pmatrix}, \quad D = \begin{pmatrix} -0.69811207 \\ -0.04986515 \\ 0.54851662 \end{pmatrix}$$

Матрица C симметрическая. Следовательно, процесс последовательных приближений для решения системы линейных уравнений сходится к решению тогда и только тогда, когда все собственные значения матрицы C меньше единицы по абсолютной величине. Вычислим модули собственных чисел матрицы C:

```
    print(*abs(np.linalg.eigvals(C)))
```

Результат вывода:

$$4.996003610813204e-16, \quad 0.4754116881086019, \quad 0.20585894525424797$$

То есть мы можем применить процесс последовательных приближений для отыскания приближенного решения исходной системы в виде

$$X_n = F(X_{n-1}) = CX_{n-1} + D.$$

Итерационный процесс метода последовательных приближений мы будем заканчивать при выполнении неравенства (из апостериорной оценки точности):

$$\frac{\alpha}{1-\alpha} \|X_n - X_{n+1}\| \leqslant \varepsilon.$$

Априорное число итераций мы вычислим по формуле (из априорной оценки точности)

$$n_{apr} = \left[\log_\alpha \frac{\varepsilon(1-\alpha)}{\|X_0 - X_1\|} \right] + 1.$$

Выберем в качестве начального приближения нулевой вектор

$$X_0 = (0, 0, 0)^T.$$

В качестве оценки для коэффициента сжатия возьмем

$$\alpha = \|C\|_2.$$

Листинг программы на Python:

```
def F(X_n):
      return np.dot(C, X_n) + D

  def aposteriori(X_n, X_n1, alpha):
      return alpha / (1 - alpha) * np.linalg.norm(X_n - X_n1, 2)

  def apriori(X_0, X_1, epsilon, alpha):
      return math.floor(math.log(epsilon * (1 - alpha) / (np.linalg.norm(
                                          X_0 - X_1, 2)), alpha)) + 1
```

```
    def msa(X_0, X_1, alpha, epsilon):
        iterations = 0
        while True:
            iterations += 1
            X_0 = X_1
            X_1 = F(X_0)
            if aposteriori(X_1, X_0, alpha) <= epsilon:
                break
        return X_1, iterations

    X_0 = np.zeros([3, 1])
    alpha = np.linalg.norm(C, 2)
    epsilon = 1e-4

    print('compression coefficient: ' + str(alpha))

    n_apr = apriori(X_0, F(X_0), epsilon, alpha)
    print('apriori number of iterations: ' + str(n_apr))

    last_iteration, iterations = msa(X_0, F(X_0), alpha, epsilon)
    print('aposteriori number of iterations: ' + str(iterations))
    print('solution: \n' + str(last_iteration))
```

Результат вывода:

compression coefficient: 0.4754116881086013
apriori number of iterations: 14
aposteriori number of iterations: 11
solution:

$$\begin{matrix}[[-0.77467001] \\ [0.14722159] \\ [0.84555917]]\end{matrix}$$

Таким образом, мы получили следующие значения:

- коэффициент сжатия равен ≈ 0.475
- априорное число итераций равно 14;
- апостериорное число итераций равно 11;
- приближенное решение исходной системы равно

$$X^* \approx \begin{pmatrix} -0.7747 \\ 0.1472 \\ 0.8456 \end{pmatrix}.$$

ПРИМЕНЕНИЕ ПРИНЦИПА СЖИМАЮЩИХ ОТОБРАЖЕНИЙ ДЛЯ РЕШЕНИЯ ИНТЕГРАЛЬНЫХ УРАВНЕНИЙ

Интегральными уравнениями называют уравнения относительно неизвестной функции, входящей в уравнение под знаком интеграла.

Ограничимся рассмотрением уравнений вида

$$a(t)x(t) - \int_a^b \mathcal{K}(t,s;x(s))\mathrm{d}s = y(t), \quad t \in [a,b] \tag{1}$$

здесь $a(t), y(t)$ — заданные функции; $\mathcal{K}(t,s;x(s))$ — заданная функция, называемая *ядром интегрального уравнения*; $x(t)$ — неизвестная функция. Решение $x(t)$ разыскивается в различных пространствах функций в зависимости от свойств функции $\mathcal{K}(t,s;z)$ и y. Пространства выбираются так, чтобы интеграл в (1) существовал. Уравнение (1) называется уравнением Фредгольма. Если $a(t) \equiv 0$, то уравнение (1) называется уравнением Фредгольма первого рода, соответственно, при $a(t) \equiv 1$ — второго рода и уравнением третьего рода при $a(t) \neq 0$. Исследование уравнений второго и третьего рода не отличаются, поэтому мы ограничимся рассмотрением случая $a(t) = 1$.

Интегральное уравнение (1) называется линейным, если функция $\mathcal{K}(t,s,z)$ линейна по z. Если $y(t) = 0$, то уравнение (1) называется однородным, в противном случае неоднородным.

Решением уравнения (1) называется функция $x(t)$, при подстановке которой в уравнение выполняется равенство для всех $t \in [a,b]$ или почти всех. Линейное однородное уравнение всегда имеет решение $x(t) \equiv 0$.

Выделим класс уравнений с переменным верхним пределом вида

$$a(t)x(t) - \int_a^t \mathcal{K}(t,s;x(s))\mathrm{d}s = y(t)$$

называемые интегральными уравнениями Вольтерра.

Уравнение Вольтерра является частным случаем уравнения Фредгольма, если переопределить ядро $\mathcal{K}(t,s;x(s))$.

Идея применения принципа сжимающих отображений и интегральным уравнениям (29) либо (30) заключается в следующем.

Пусть имеется интегральное уравнение

$$x(t) = \int_T \mathcal{K}(t,s;x(s))\mathrm{d}s + y(t)$$

где $T = [a,b]$ либо $T = [a,t]$. Соответствие $x \to \int_T \mathcal{K}(t,s;x(s))\mathrm{d}s + y(t)$ определяет отображение множества функций, заданных на T, на себя. Тогда уравнение (31) записывается в виде $x = F(x)$, а это означает, что искомое решение является неподвижной точкой отображения F. Для того, чтобы применить принцип сжимающих отображений, нужно

- выбрать банахово пространство функций;
- проверить, что (31) определяет сжимающее отображение.

Покажем, каким образом такая схема реализуется в пространстве $C[a,b]$ непрерывных функций на отрезке $[a,b]$ для линейного неоднородного уравнения Фредгольма

$$x(t) - \lambda \int_a^b \mathcal{K}(t,s)x(s)\mathrm{d}s = y(t)$$

Теорема. *Пусть $\mathcal{K}(t,s)$ - непрерывная функиия на множестве $[a,b]\times[a,b]=\Omega$ и $M=\max_{(t,s)\in\Omega}|\mathcal{K}(t,s)|$, тогда для любого λ такого, что $|\lambda|<\frac{1}{M(b-a)}$ интегральное уравнение Фредгольма второго рода имеет единственное решение для любой правой части $y(t)\in C[a,b]$.*

На практике при численной реализации метода последовательных приближений необходимо приближенно вычислять интегралы по методу квадратур, что вносит дополнительную погрешность и довольно большую при большом числе итераций. С этой целью интегрирование нужно выполнять с большей точностью, чем погрешность метода последовательных приближений.

Так, для приближенного вычисления интеграла от гладкой функции хорошо подходит метод Симпсона или метод парабол.

$$\int_a^b f(t)\mathrm{d}t\approx\frac{b-a}{m}\left[f_0+f_m+2\left(f_2+f_4+\ldots+f_{m-2}\right)+4\left(f_1+f_3+\ldots+f_{m-1}\right)\right],$$

где $f_m=f\left(t_m\right), t_m=t_0+\frac{b-a}{m}$.

Обозначим через $t_i, i=0,1,\ldots,m$ узлы сетки, расположенной на отрезке a,b. Тогда соотношение (33) перепишется в виде

$$x_n\left(t_i\right)=\lambda\int_a^b\mathcal{K}\left(t_i,s\right)x_{n-1}\left(s_i\right)\mathrm{d}s+y\left(t_i\right)$$

Если воспользоваться квадратурной формулой трапеций на равномерной сетке с шагом $h=\frac{b-a}{m}$, то расчетные формулы метода последовательных приближений примут вид

$$x_n\left(t_i\right)=\lambda\frac{h}{2}\left[k_{i,0}x_{n-1,0}+2\left(k_{i,1}x_{n-1,1}+\ldots k_{i,m-1}x_{n-1,m-1}\right)+k_{i,m}x_{n-1,m}\right]+$$
$$+y\left(t_i\right), i=0,1,\ldots,m.$$

Здесь $k_{i,j}=\mathcal{K}\left(t_i,s_j\right), x_{n,j}=x_n\left(s_j\right)$.

Отметим, что при решении линейных интегральных уравнений сходимость метода последовательных приближений не зависит от вида правой части и начального приближения, которые влияют на скорость сходимости итерационного процесса.

Разрешимость уравнений Фредгольма зависит от условий на ядро. Покажем, что для уравнения Вольтерра условие разрешимости проще.

Рассмотрим линейное неоднородное уравнение Вольтерра

$$x(t)-\lambda\int_a^t\mathcal{K}(t,s)x(s)\mathrm{d}s=y(t).$$

Выясним, когда можно применить метод последовательных приближений для его решения.

Теорема. *Пусть $\mathcal{K}(t,s)$ — непрерывная функция по переменным t и s. Тогда для любой $y(t)\in C[a,b]$ и любого λ из поля P интегральное уравнение Вольтерра второго рода имеет единственное решение.*

ЗАДАЧА 3

Выяснить, при каких значениях параметра $\lambda \neq 0$ к интегральному уравнению Фредгольма второго рода

$$x(t) - \lambda \int_0^1 \frac{\sqrt{1+t}}{1+s}\, x(s)\, ds = 3.$$

применим принцип сжимающих отображений в пространстве $C[0,1]$ и в пространстве $L_2[0,1]$. При $\lambda = \lambda_0$ найти приближенное решение уравнения с точностью $\varepsilon = 10^{-3}$ и сравнить его с точным решением. Составить алгоритм и написать программный код, реализующий метод последовательных приближений, предусматривающий:

- приведение интегрального уравнения к специальному виду для применения метода последовательных приближений;
- вычисление коэффициента сжатия;
- вычисление априорной оценки количества итераций;
- выбор начального приближения;
- составление итерационного процесса в каждой фиксированной точке t_i, $i = 1, \ldots, n$ по правилу

  $$x_n(t_i) = \lambda \int_a^b \mathcal{K}(t_i, s)\, x_{n-1}(s)\, \mathrm{d}s + y(t_i)$$

  с приближенным вычислением интеграла по формуле Симсона с шагом 0.05;
- вывода на печать номера последней итерации, апостериорной погрешности, графика точного и приближенного решения.

Исходное уравнение представляет собой интегральное уравнение с вырожденным ядром, поэтому мы можем найти его точное решение. Перепишем уравнение в виде

$$x(t) = \lambda\sqrt{1+t} \int_0^1 \frac{1}{1+s}\, x(s)\, ds + 3.$$

Обозначим

$$c = \int_0^1 \frac{1}{1+s} x(s) \mathrm{d}s.$$

Тогда

$$x(t) = \lambda\sqrt{1+t}\, c + 3.$$

Подставим это выражение в предыдущее уравнение и получим

$$c = \int_0^1 \frac{1}{1+s}(\lambda\sqrt{1+s}\, c + 3)\mathrm{d}s = \lambda c\, 2\sqrt{s+1}\Big|_0^1 + 3\ln|s+1|\Big|_0^1 = \lambda c(2\sqrt{2}-2) + 3\ln 2.$$

Отсюда

$$c = \frac{3\ln 2}{1 - 2\lambda(\sqrt{2}-1)}.$$

Тогда точное решение исходного интегрального имеет вид

$$x(t) = \lambda\sqrt{1+t}\, \frac{3\ln 2}{1 - 2\lambda(\sqrt{2}-1)} + 3.$$

Теперь попробуем вычислить приближенное решение для данного интегрального уравнения. Приведем исходное интегральное уравнение к виду

$$x = F(x),$$

тогда мы сможем применить принцип сжимающих отображений, если в рассматриваемых банаховых пространствах отображение является сжимающим. Таким образом, исходное уравнение примет вид

$$x(t) = \underbrace{\lambda \int_0^1 \frac{\sqrt{1+t}}{1+s}\, x(s)\, ds + 3}_{F(x)}.$$

ПРОСТРАНСТВО $C[0,1]$

В пространстве $C[0,1]$ отображение $F : C[0,1] \to C[0,1]$, так как представляет собой сумму непрерывных функций. Покажем, что отображение F является сжимающим. Для этого покажем выполнение условия: $\exists \alpha \in \mathbb{R},\ 0 < \alpha < 1$:

$$\|F(x) - F(y)\|_{C[0,1]} \leqslant \alpha \|x - y\|_{C[0,1]}, \quad \forall x, y \in C[0,1].$$

Рассмотрим

$$\|F(x) - F(y)\|_{C[0,1]} = \max_{0 \leqslant t \leqslant 1} \left| \lambda \int_0^1 \frac{\sqrt{1+t}}{1+s} (x(s) - y(s)) \mathrm{d}s \right| \leqslant$$

$$\leqslant |\lambda| \sqrt{2} \cdot \max_{0 \leqslant t \leqslant 1} |x(s) - y(s)| \cdot \int_0^1 \frac{1}{1+s} ds = \underbrace{|\lambda| \sqrt{2} \ln 2}_{\alpha} \|x - y\|_{C[0,1]}.$$

Отсюда следует, что $\alpha = |\lambda| \sqrt{2} \ln 2 < 1$. Значит выбираем λ такое, что

$$|\lambda| < \frac{1}{\sqrt{2} \ln 2} \approx 1.02.$$

При таких λ можно применять принцип сжимающих отображений.

Вычислим априорную оценку количества итераций по формуле

$$n_{apr} = \left[\log_\alpha \frac{\varepsilon(1-\alpha)}{\|x_0 - x_1\|} \right] + 1.$$

Пусть $x_0 = 0$, $\lambda = \dfrac{1}{2\sqrt{2} \ln 2} \approx 0.51$. Тогда коэффициент сжатия равен $\alpha = 0.5$, а

$$x_1 = F(x_0) = 3.$$

Ниже приведен листинг программы, реализующей вычисление коэффициента сжатия с заданными параметрами:

```
def apriori(x_0, x_1, epsilon, alpha):
  return math.floor(math.log(epsilon * (1 - alpha) / (np.absolute(x_0
                                - x_1)), alpha)) + 1

lambda_ =  1 / (2 * np.sqrt(2) * np.log(2))
x_0 = 0
```

```
x_1 = 3
alpha = np.absolute(lambda_) * np.sqrt(2) * np.log(2)
epsilon = 1e-3

n_apr = apriori(x_0, x_1, epsilon, alpha)
print('apriori number of iterations: ' + str(n_apr))
```

Результат вывода

apriori number of iterations: 13

То есть, число $n_{apr} = 13$. Теперь нам необходимо составить итерационный процесс с приближенным вычислением интеграла по формуле Симсона с шагом

$$\frac{b-a}{m} = 0.05$$

Выясним, чему равно m в данной формуле:

```
a = 0
b = 1
step = 0.05
m = (b-a) / step
print('m=' + str(m))
```

Результат вывода: m=20.0

Теперь рассчитаем все t_i, $i = \overline{1,20}$ узлы сетки, расположенной на $[a,b]$. Выберем $t_0 = 0$ — левый край отрезка. Далее с шагом 0.05 будем двигаться, получая остальные t_i:

```
t_i = 0
grid = [t_i]
for i in range(int(m)):
  t_i = t_i + (b - a) / m
  grid.append(t_i)
```

В результате получили последовательность значений t_i для применения формулы Симсона. Далее реализуем функцию, которая рассчитывает значение интеграла по формуле Симсона

$$\int_0^1 f(t)\mathrm{d}t \approx \frac{b-a}{m}\left[f_0 + f_m + 2\left(f_2 + f_4 + \ldots + f_{m-2}\right) + 4\left(f_1 + f_3 + \ldots + f_{m-1}\right)\right],$$

где $f_m = f\left(t_m\right), t_m = t_0 + 0.05$. Причем в нашем случае

$$f(t) = \sqrt{1+t}x_{n-1}(t)$$

и $m = 20$.

Листинг программы:

```
def simson(a, b, x_n_1):
  step = 0.05
  f = np.sqrt(1 + a) * x_n_1[0];
  coord = a
  result = 0
  for i in range(1, m-1):
    if i % 2 == 1:
      result += 4 * np.sqrt(1 + coord) * x_n_1[i];
```

```
        else:
          result += 2 * np.sqrt(1 + coord) * x_n_1[i];
        coord += step;
      result += np.sqrt(1 + b) * x_n_1[m-1];
      result *= step;
      return result;
```

Остается реализовать непосредственно итерационный процесс

$$x_n(t_i) = \lambda\sqrt{1+t_i}\int_0^1 \frac{1}{1+s}\,x_{n-1}(s)\,ds + 3.$$

Процесс прекращается при выполнении неравенства

$$\frac{\alpha}{1-\alpha}\,\|x_{n-1} - x_n\| \leqslant \varepsilon.$$

Причем в нашем случае $x_n = (x_n(t_1), \ldots, x_n(t_m))^T$. Следовательно возьмем

$$\|x_n - x_{n+1}\| = \max_{t_i}|x_{n-1}(t) - x_n(t)|.$$

Листинг программы:

```
    def aposteriori(x_n, x_n1, alpha):
      return alpha / (1 - alpha) * np.linalg.norm(x_n - x_n1, 1)

    iterations = 0
    x_n = np.array([0. for _ in range(m + 1)])
    x_n1 = np.array([0. for _ in range(m + 1)])

    while True:
      iterations += 1
      for j in range(len(grid)):
        x_n[j] = lambda_ * np.sqrt(1 + grid[j]) * simson(a, b, x_n1) + 3
      if aposteriori(x_n1, x_n, alpha) <= epsilon:
        break
      x_n1 = np.copy(x_n)
```

Выведем на печать получившееся число итераций и соответственно шаг сетки, значение приближенного решения и значение точного решения.

```
def x_actual(t):
    return lambda_ * np.sqrt(1 + t) * (3 * np.log(2)) / (1 - 2 *
                                    lambda_ * (np.sqrt(2) - 1)) + 3
print('iterations=' + str(iterations))
for i in range(len(grid)):
  print('$t=' + str(round(grid[i], 2)) + '\t\t\quad x[approximate]='
                                    + str(x_n[i]) + '\t\t\quad x[actual]=
                                    ' + str(x_actual(grid[i])) + '$\\\\')
```

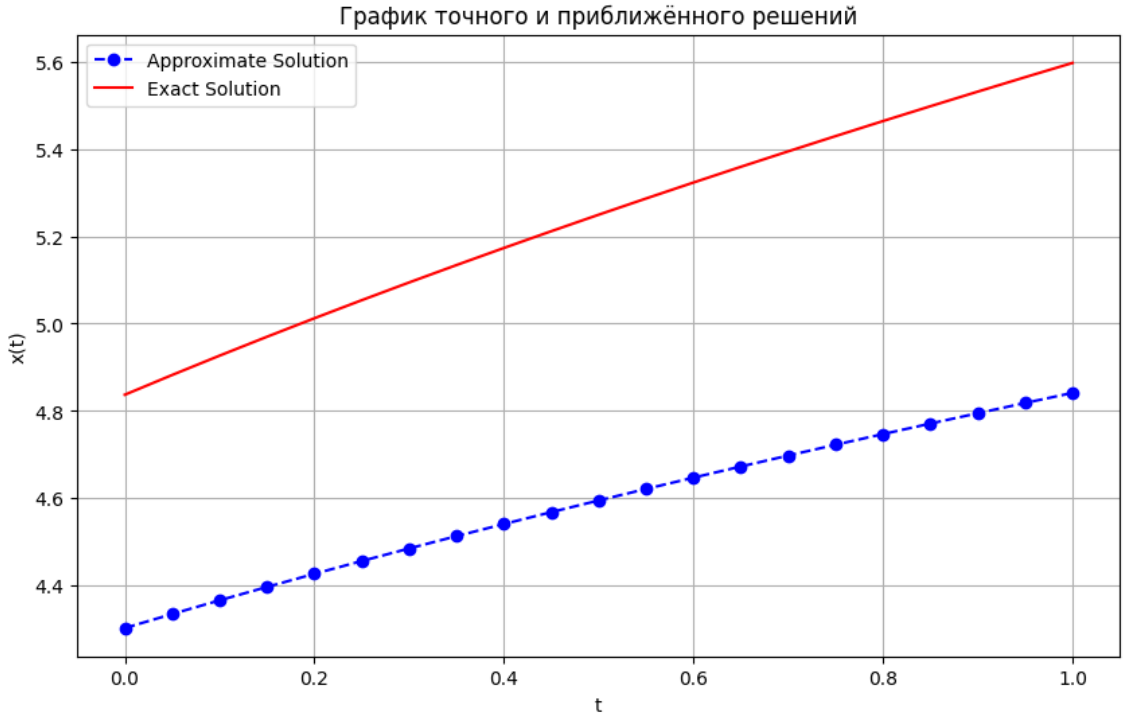
Результат вывода:

iterations=12

$$\begin{gathered}
t = 0 \quad x[approximate] = 4.301842 \quad x[actual] = 4.836818 \\
t = 0.05 \quad x[approximate] = 4.333991 \quad x[actual] = 4.882178 \\
t = 0.1 \quad x[approximate] = 4.365384 \quad x[actual] = 4.926471 \\
t = 0.15 \quad x[approximate] = 4.396070 \quad x[actual] = 4.969768 \\
t = 0.2 \quad x[approximate] = 4.426097 \quad x[actual] = 5.012133
\end{gathered}$$

$$
\begin{array}{lll}
t = 0.25 & x[approximate] = 4.455504 & x[actual] = 5.053625 \\
t = 0.3 & x[approximate] = 4.484329 & x[actual] = 5.094295 \\
t = 0.35 & x[approximate] = 4.512604 & x[actual] = 5.134189 \\
t = 0.4 & x[approximate] = 4.540361 & x[actual] = 5.173352 \\
t = 0.45 & x[approximate] = 4.567626 & x[actual] = 5.211822 \\
t = 0.5 & x[approximate] = 4.594425 & x[actual] = 5.249633 \\
t = 0.55 & x[approximate] = 4.620781 & x[actual] = 5.286820 \\
t = 0.6 & x[approximate] = 4.646715 & x[actual] = 5.323411 \\
t = 0.65 & x[approximate] = 4.672247 & x[actual] = 5.359435 \\
t = 0.7 & x[approximate] = 4.697395 & x[actual] = 5.394917 \\
t = 0.75 & x[approximate] = 4.722175 & x[actual] = 5.429882 \\
t = 0.8 & x[approximate] = 4.746605 & x[actual] = 5.464350 \\
t = 0.85 & x[approximate] = 4.770697 & x[actual] = 5.498342 \\
t = 0.9 & x[approximate] = 4.794466 & x[actual] = 5.531879 \\
t = 0.95 & x[approximate] = 4.817924 & x[actual] = 5.564976 \\
t = 1.0 & x[approximate] = 4.841083 & x[actual] = 5.597653
\end{array}
$$

Построим график приближенного и точного решений:



ПРОСТРАНСТВО $L_2[0,1]$

Рассмотрим пространство $L_2[0,1]$. Покажем, что отображение F является сжимающим. Для этого покажем выполнение условия: $\exists \alpha \in \mathbb{R},\ 0 < \alpha < 1$:

$$
\|F(x) - F(y)\|_{L_2[0,1]} \leqslant \alpha \|x - y\|_{L_2[0,1]}, \quad \forall x, y \in L_2[0,1].
$$

Сначала запишем в общем виде:

$$
\Big(\int_0^1 \Big| \int_0^1 K(t,s)\big(x(s) - y(s)\big)\mathrm{d}s\Big|^2 \mathrm{d}t\Big)^{\frac{1}{2}} \leqslant \Big(\int_0^1 \int_0^1 |K(t,s)|^2 \mathrm{d}s\mathrm{d}t\Big)^{\frac{1}{2}} \cdot \underbrace{\Big(\int_0^1 \int_0^1 |x(s) - y(s)|^2 \mathrm{d}s\mathrm{d}t\Big)^{\frac{1}{2}}}_{\|x-y\|}
$$

Теперь рассмотрим отдельно первый множитель:

$$\Big(\int_0^1\int_0^1 |K(t,s)|^2 \mathrm{d}s\mathrm{d}t\Big)^{\frac{1}{2}} = |\lambda|\cdot\Big(\int_0^1\int_0^1 \Big(\frac{\sqrt{1+t}}{1+s}\Big)^2 \mathrm{d}s\mathrm{d}t\Big)^{\frac{1}{2}} = |\lambda|\cdot\Big(\frac{3}{4}\Big)^{\frac{1}{2}} = |\lambda|\cdot\frac{\sqrt{3}}{2} = \alpha.$$

Следовательно, отображение F отображает пространство $L_2[0,1]$ на себя и является сжимающим, если

$$|\lambda| < \frac{2}{\sqrt{3}} \approx 1.1547$$

Таким образом, в данном пространстве у нас множество допустимых значений параметра λ шире. Соответственно мы можем выбрать

$$\lambda = \frac{1}{2\sqrt{2}\ln 2}$$

и так же применить принцип сжимающих отображений. Но теперь изменится априорное число итераций. Снова возьмем начальное приближение $x_0 = 0$. Тогда

$$x_1 = F(x_0) = 3.$$

Отсюда

$$\Big(\int_0^1 |x_1 - x_0|^2 ds\Big)^{\frac{1}{2}} = 3.$$

Тогда априорное число итераций вычисляем по формуле

$$n_{apr} = \Big[\log_\alpha \varepsilon(1-\alpha)\Big] + 1.$$

Листинг программы:

```
def apriori_L2(epsilon, alpha):
  return math.floor(math.log(epsilon * (1 - alpha), alpha)) + 1

lambda_ = 1 / (2 * np.sqrt(2) * np.log(2))
alpha = np.absolute(lambda_)*(np.sqrt(2) / 2
epsilon = 1e-3

n_apr_L2 = apriori_L2(epsilon, alpha)
print('apriori number of iterations: ' + str(n_apr_L2))
```

Результат вывода:

apriori number of iterations: 8

Таким образом, априорное число итераций при одинаковом выборе параметра λ в пространстве $L_2[0,1]$ меньше чем в пространстве $C[0,1]$. Однако реальное число итераций останется неизменным так же, как и результаты вычислений. Соответственно все полученные итерации и график мы можем перенести из пространства $C[0,1]$ в пространство $L_2[0,1]$.