Федеральное государственное автономное образовательное учреждение высшего образования

***Университет ИТМО***

Факультет программной инженерии и компьютерной техники

Дисциплина: **Вычислительная математика**

Лабораторная работа №2

*“* **Численное решение нелинейных уравнений и систем***”*

# Вариант: 9

Выполнил: Кузнецов Максим Александрович

Группа: Р3211

Преподаватель: Малышева Татьяна Алексеевна

Санкт-Петербург 2022 г

**Цель работы:**

Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения, выполнить программную реализацию методов.

**Условия и задание:**

№ варианта определяется как номер в списке группы согласно ИСУ.

*Вычислительная реализация задачи:*

1. Отделить корни заданного нелинейного уравнения графически.
2. Определить интервалы изоляции корней.
3. Вычислительная реализация задачи (в отчет):
4. Уточнить корни нелинейного уравнения с точностью ε=10^-2.

Вычисления оформить в виде таблиц, удержать 3 знака после запятой.

1. Представить в отчете заполненные таблицы.

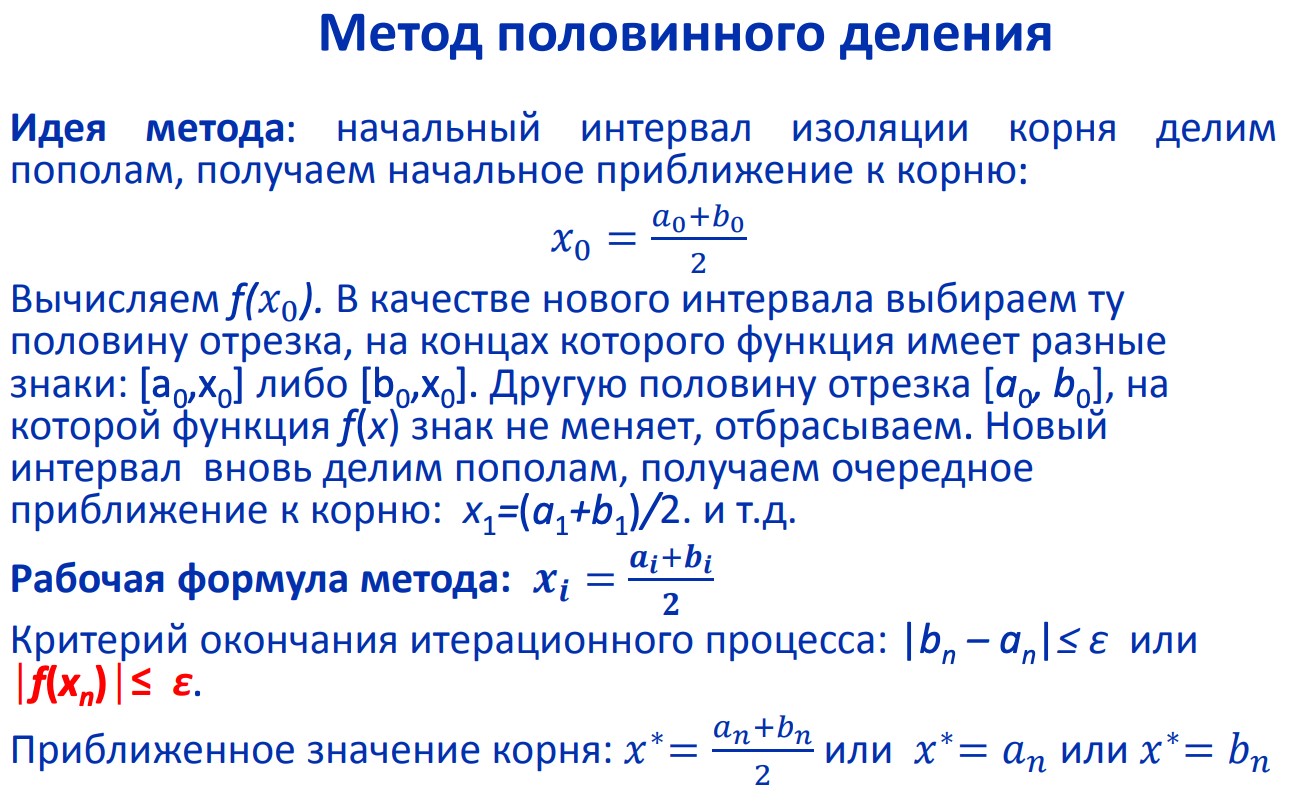
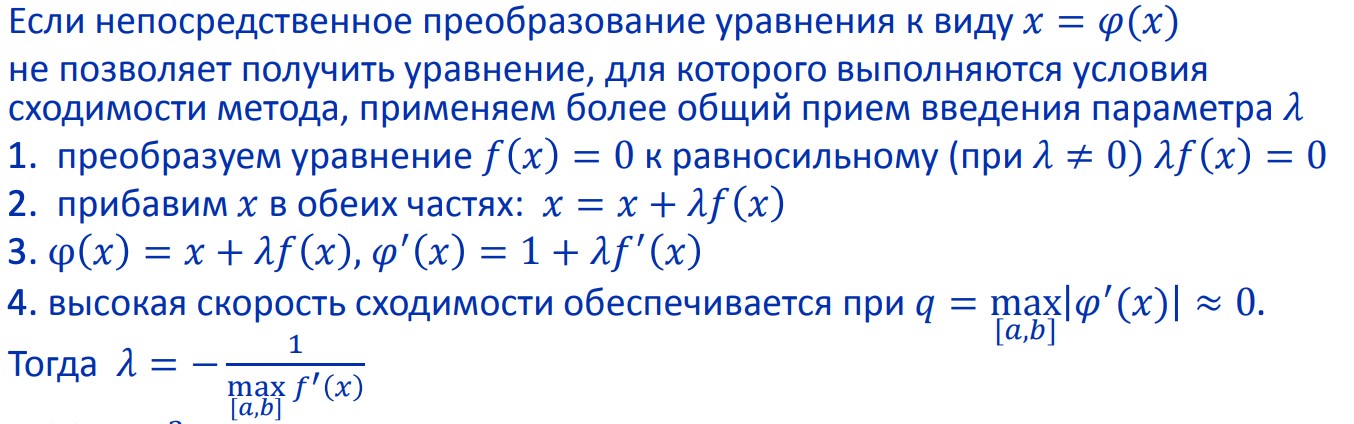
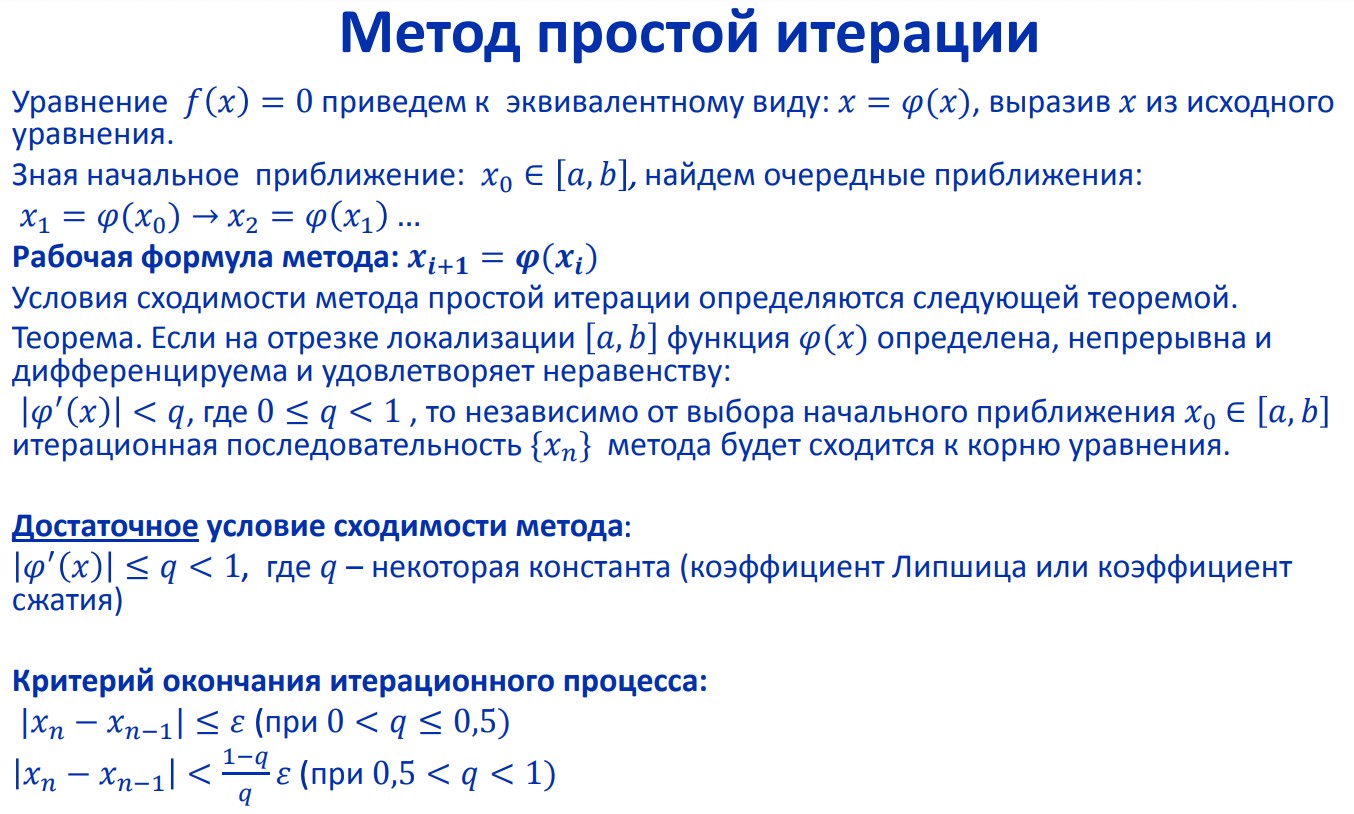
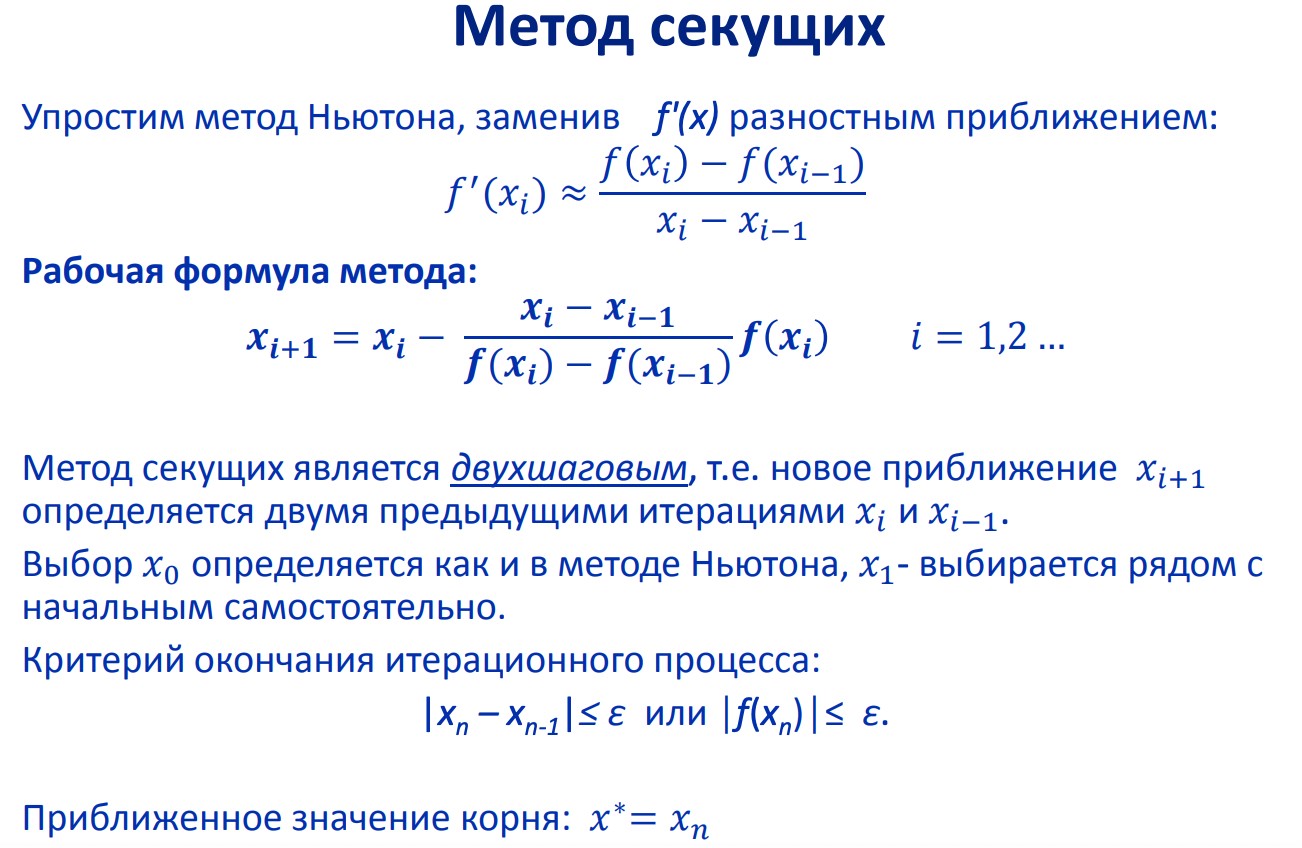
*Программная реализация задачи:*

1. Все численные методы должны быть реализованы в виде отдельных подпрограмм или классов.
2. Пользователь выбирает уравнение, корень/корни которого требуется вычислить (3–5 функций, в том числе и трансцендентные), из тех, которые предлагает программа.
3. Предусмотреть ввод исходных данных (границы интервала/начальное приближение к корню и погрешность вычисления) из файла или с клавиатуры по выбору конечного пользователя.
4. Выполнить верификацию исходных данных. Для метода половинного деления (метода хорд) анализировать наличие корня на введенном интервале. Для метода Ньютона (метода секущих) – выбор начального приближения (а или b). Для метода простой итерации – достаточное условие сходимости метода. Программа должна реагировать на некорректные введенные данные.
5. Предусмотреть вывод результатов (найденный корень уравнения, значение функции в корне, число итераций) в файл или на экран по выбору конечного пользователя.
6. Организовать вывод графика функции, график должен полностью отображать весь исследуемый интервал (с запасом).
7. Для систем нелинейных уравнений:
   1. Рассмотреть систему двух уравнений.
   2. Организовать вывод графика функций.
   3. Для метода простой итерации проверить условие сходимости.
   4. Вывод вектора неизвестных:

7.5Вывод количества итераций, за которое было найдено решение.

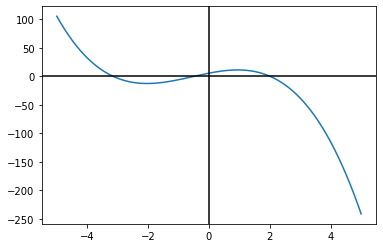
7.6 Вывод вектора погрешностей

**Описание «ручных» методов по варианту:**



Уравнение по варианту:

|  |  |
| --- | --- |
| 9 | −1,8𝑥! − 2,94𝑥" + 10,37𝑥 + 5,38 |



Таблицы:

Таблица 1

Уточнение корня уравнения методом половинного деления (левый)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № шага | a | b | x | f(a) | f(b) | f(x) | |a-b| |
| 0 | -4.000 | -3.000 | -3.500 | 32.06 | -3.59 | 10.245 | 1 |
| 1 | -3.500 | -3.000 | -3.250 | 10.245 | -3.59 | 2.414 | 0.5 |
| 2 | -3.250 | -3.000 | -3.125 | 2.414 | -3.59 | -0.806 | 0.25 |
| 3 | -3.250 | -3.125 | -3.188 | 2.414 | -0.806 | 0.749 | 0.125 |
| 4 | -3.188 | -3.125 | -3.156 | 0.749 | -0.806 | -0.042 | 0.0625 |
| 5 | -3.188 | -3.156 | -3.171 | 0.749 | -0.042 | 0.350 | 0.03125 |
| 6 | -3.171 | -3.156 | -3.164 | 0.350 | -0.042 | 0.151 | 0.015625 |
| 7 | -3.164 | -3.156 | -3.160 | 0.151 | -0.042 | 0.051 | 0.0078125 |

Таблица 3 Уточнение корня уравнения методом секущих (правый)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| №  итерации | *xk-1* | *f*(*xk-1* ) | *xk* | *f*(*xk*) | *xk*+1 | *f*(*xk+1*) | │*xk* − *xk*+1│ |
| 0 | 2.200 | -5.202 | 1.800 | 4.023 | 1.974 | 0.548 | 0.174 |
| 1 | 1.800 | 4.023 | 1.974 | 0.548 | 2.001 | -0.063 | 0.027 |
| 2 | 1.974 | 0.548 | 2.001 | -0.063 | 1.998 | 0.006 | 0.003 |

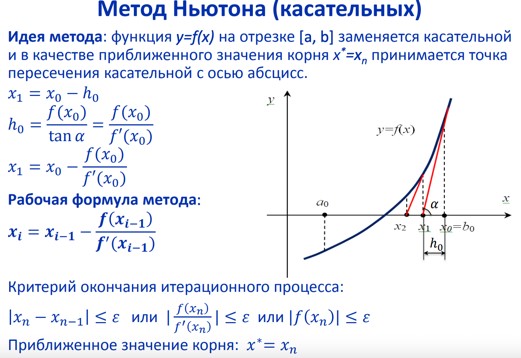
Таблица 4

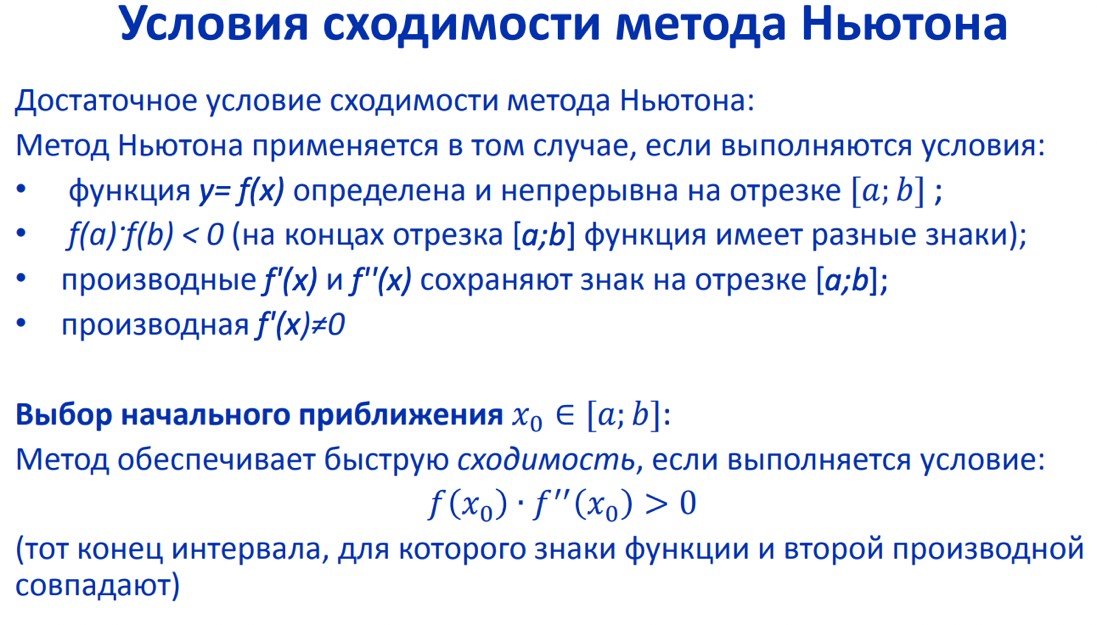
Уточнение корня уравнения методом простой итерации (центральный)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации | *xk* | *f* (*xk+1*) | *xk*+1 | 𝜑(𝑥#$%) | │*xk* − *xk*+1│ |
| 0 | -1.000 | 0.461 | -0.435 | -0.479 | 0.565 |
| 1 | -0.435 | 0.046 | -0.477 | -0.473 | 0.042 |
| 2 | -0.477 | 0.005 | -0.473 | -0.473 | 0.004 |

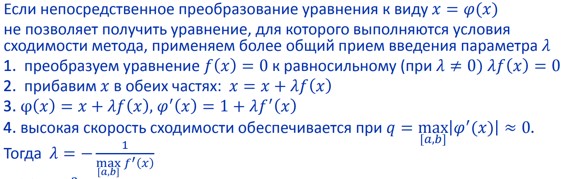
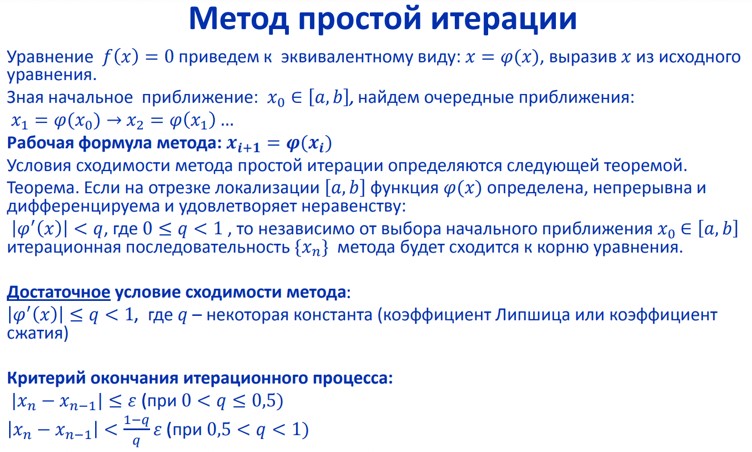
**Описание программных методов по варианту:**

* + 1. Метод Ньютона – для нелинейного уравнения



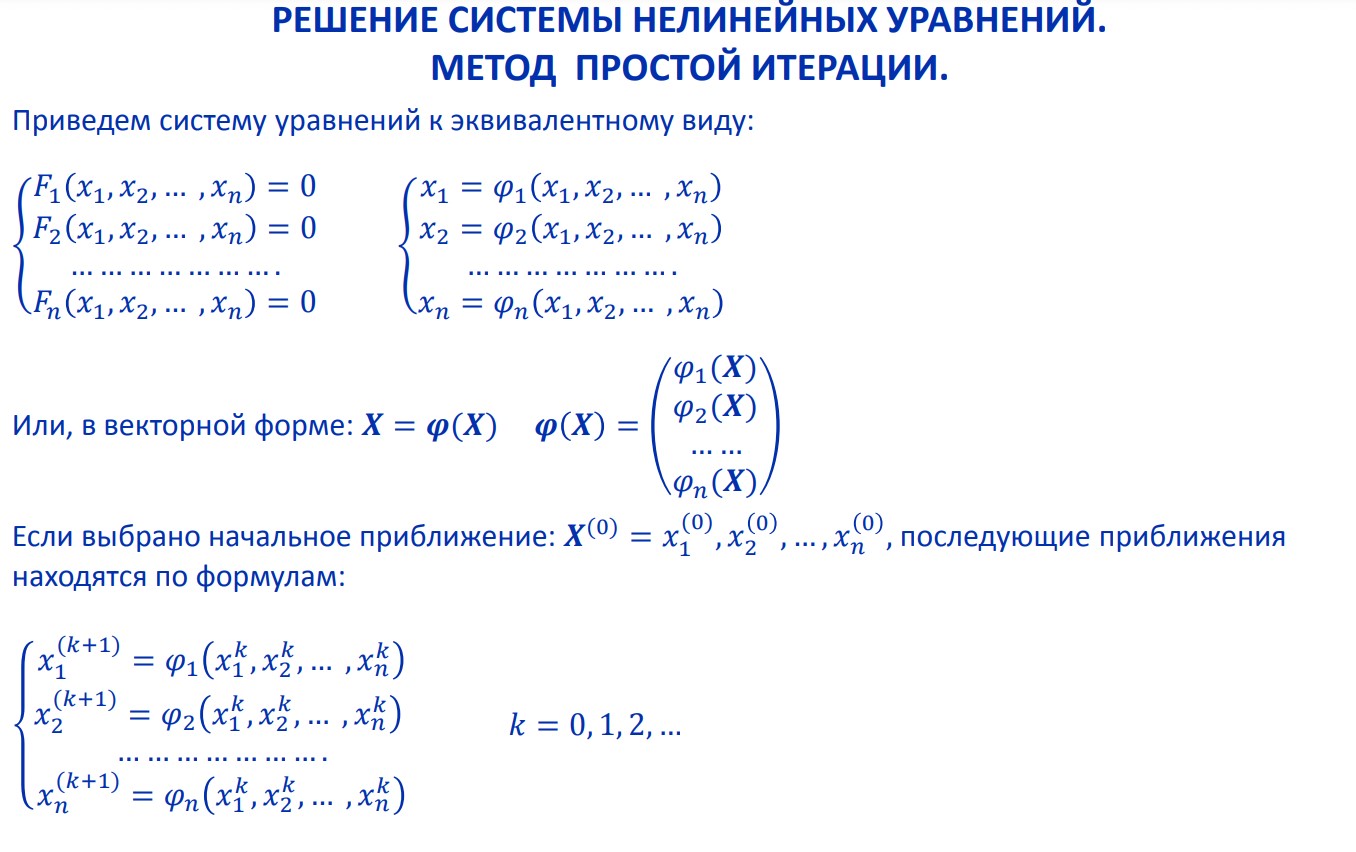


* + 1. Метод простой итерации – для нелинейного уравнения



В программе мной и используется указанный прием.

* + 1. Метод простой итерации – для системы нелинейных уравнений



**Листинг программы:**

Импорт необходимых библиотек.

|  |
| --- |
| import math import numpy as np import matplotlib  import matplotlib.pyplot as plt import matplotlib.lines as mlines from numpy import genfromtxt import sys |

|  |  |
| --- | --- |
| Раздел с предложенными функциями в программе, которые выбирает | |
| пользователь для решения. |  |
| def function1(x):  return -1.8 \* x \*\* 3 - 2.94 \* x \*\* 2 + 10.37 \* x + 5.38  def function1\_first\_derivative(x):  return -5.4 \* x \*\* 2 - 5.88 \* x + 10.37  def function1\_second\_derivative(x):  return -10.8 \* x - 5.88  def function2(x):  return math.sin(x) + 0.1  def function2\_first\_derivative(x):  return math.cos(x) | | | |
| def function2\_second\_derivative(x):  return -math.sin(x)  def function3(x):  return x \*\* 3 - x + 4  def function3\_first\_derivative(x):  return 3 \* x \*\* 2  def function3\_second\_derivative(x):  return 6 \* x  def f1(x1, x2):  return (0.3 - 0.1 \* x1 \*\* 2 - 0.2 \* x2 \*\* 2 - x1)  def f1\_derivative\_a(x1, x2):  return (-0.2 \* x1)  def f1\_derivative\_b(x1, x2):  return (-0.4 \* x2)  def f2(x1, x2):  return (0.7 - 0.2 \* x1 \*\* 2 - 0.1 \* x1 \* x2 - x2)  def f2\_derivative\_a(x1, x2):  return (-0.4 \* x1 + 0.1 \* x2)  def f2\_derivative\_b(x1, x2):  return (0.1 \* x1)  def system(x1, x2):  return (0.3 - 0.1 \* x1 \*\* 2 - 0.2 \* x2 \*\* 2), (0.7 - 0.2 \* x1 \*\* 2 - 0.1 \* x1 \* x2) | | | |

Методы для отрисовки графика.

|  |
| --- |
| def newline(p1, p2, color = 'black'):  ax = plt.gca()  xmin, xmax = ax.get\_xbound() if(p2[0] == p1[0]):  xmin = xmax = p1[0] ymin, ymax = ax.get\_ybound() else:  ymax = p1[1]+(p2[1]-p1[1])/(p2[0]-p1[0])\*(xmax-p1[0]) ymin = p1[1]+(p2[1]-p1[1])/(p2[0]-p1[0])\*(xmin-p1[0])    l = mlines.Line2D([xmin,xmax], [ymin,ymax], color = color) ax.add\_line(l) return l |
| def draw(a, b, Xs = [], func=function1):    x = [a + i\*(b-a)/10000 for i in range(10000)] y = [] for xx in x:  y.append(func(xx)) plt.plot(x, y) for xx in Xs:  plt.scatter(xx, 0, color = 'red') p1 = [min(x), 0] p2 = [max(x), 0] p3 = [0, min(y)] p4 = [0, max(y)]  newline(p1,p2, color = 'black') newline(p3,p4, color = 'black')    plt.show()  def draw\_system(a, b, Xs = [], Ys = [], fun1=f1, fun2=f2):     1. = [a + i\*(b-a)/10000 for i in range(10000)] y = [] for xx in x:   y.append(fun1(0, xx))  plt.plot(y, x)     1. = [a + i\*(b-a)/10000 for i in range(10000)] x = [] for yy in y:   x.append(fun2(yy, 0))  plt.plot(y, x)  for i in range(0, len(Xs)): plt.scatter(Xs[i], Ys[i], color = 'red')    # draw axes p1 = [min(x), 0] p2 = [max(x), 0] p3 = [0, min(y)] p4 = [0, max(y)]  newline(p1,p2, color = 'black') newline(p3,p4, color = 'black')    plt.show() |

|  |  |
| --- | --- |
| Функция для проверки смены знака на заданном участке, принимает границы | |
| интервала и названия метода. |  |

def check\_sign(a, b, method = '', func=function1):

if func(a) \* func(b) > 0:

raise ValueError(f'Ошибка вычисления методом {method}:\nВ интервале должен располагаться 1 корень') start = np.sign(func(a)) changing = 0 for i in range(1, 1001): x = a + i\*(b-a)/1000 f = func(x) if np.sign(f) != start: start = np.sign(f) changing +=1 if changing != 1:

raise ValueError(f'Ошибка вычисления методом {method}:\nВ

введенном интервале смена знака происходит не 1 раз')

Реализация метода Ньютона.

|  |
| --- |
| def newtone(a, b, eps, f=function1, |
| f\_first\_derivative=function1\_first\_derivative, |
| f\_second\_derivative=function1\_second\_derivative):  if b < a: a, b = b, a    check\_sign(a, b, 'Ньютона', f) x = 0 if f(a) \* f\_second\_derivative(a) > 0:  x = a elif f(b) \* f\_second\_derivative(b) > 0:  x = b else: raise ValueError('Ошибка вычисления Ньютона:\nНе сходится.')  n = 0  while n<1 or (abs(f(x)/f\_first\_derivative(x)) > eps or abs(f(x)) > eps): n += 1  x -= f(x)/f\_first\_derivative(x) return x, n |

Реализация метода простых итераций.

|  |
| --- |
| def iterations(a, b, eps, f=function1, |
| f\_first\_derivative=function1\_first\_derivative):  try: if b < a: |
| a, b = b, a  check\_sign(a, b, 'простых итераций', f) x = 0 for i in range(1, 1001): x = a + i/1000 \* (b-a)    lambda\_value = -1/max(f\_first\_derivative(a), f\_first\_derivative(b)) n = 0  x = a if f\_first\_derivative(a) > f\_first\_derivative(b) else b while abs(f(x)) > eps:  n += 1 x\_prev = x  x += lambda\_value \* f(x) return x, n except OverflowError:  raise ValueError('Ошибка методом вычисления простой итерации:\nНе cходится, или необходимо слишком много итераций.') |

Функция реализации методы простых итераций для системы:

|  |
| --- |
| def iteration\_for\_system(eps): a = f1\_derivative\_a(1, 1) b = f1\_derivative\_b(1, 1) c = f2\_derivative\_a(1, 1) d = f2\_derivative\_b(1, 1)  if(abs(a)+abs(b)<1 and abs(c)+abs(d)<1): print("Сходимость есть, решаем...") else: raise ValueError('Ошибка методом вычисления простой итерации:\nНе cходится.') iterations = 1  equation\_1\_previous, equation\_2\_previous = system(-3, -3) equation\_1\_current, equation\_2\_current = system(equation\_1\_previous, equation\_2\_previous)  while (abs(abs(equation\_1\_current) - abs(equation\_1\_previous))>eps or abs(abs(equation\_2\_current) - abs(equation\_2\_previous))>eps):  iterations += 1  equation\_1\_previous = equation\_1\_current equation\_2\_previous = equation\_2\_current equation\_1\_current, equation\_2\_current = system(equation\_1\_previous, equation\_2\_previous)  residuals\_1 = abs(equation\_1\_current - equation\_1\_previous) residuals\_2 = abs(equation\_2\_current - equation\_2\_previous) return equation\_1\_current, equation\_2\_current, iterations, residuals\_1, residuals\_2 |

Функция для считывания данных и преобразования во float.

|  |
| --- |
| def in\_float(s = ''):  should\_continue = True while should\_continue:  should\_continue = False try:  val = float(input('Введите значение ' + s +': ')) except ValueError:  should\_continue = True print('Попробуйте снова!\n') return val |

Функция "parse" отвечает за парсинг данных из файла.

|  |
| --- |
| def parse():  should\_continue = True while should\_continue:  path = input('Введите путь:\n') try:  a, b, eps = genfromtxt(path, delimiter=',') a = float(a) b = float(b) eps = float(eps) return a, b, eps except ValueError: print('В файле должно быть 3 числа, все через запятую.\n') except OSError: print('Такого файла нет.\n') print('Попробуйте снова!\n') |

Функция для подсчета выбранного уравнения:

|  |
| --- |
| def calculate\_nonlinear\_equation():  again = True f = function1  f\_first\_derivative = function1\_first\_derivative f\_second\_derivative = function1\_second\_derivative while again:  again = False  in\_type = input('Выберите уравнение:\n\t 1 - -1.8\*x^32.94\*x^2+10.37\*x+5.38\n\t 2 - sin(x)+0.1\n\t 3 - x^3-x+4\n\t') if in\_type.strip() == '1': f = function1  f\_first\_derivative = function1\_first\_derivative f\_second\_derivative = function1\_second\_derivative elif in\_type.strip() == '2':  f = function2  f\_first\_derivative = function2\_first\_derivative f\_second\_derivative = function2\_second\_derivative |
| elif in\_type.strip() == '3':  f = function3  f\_first\_derivative = function3\_first\_derivative f\_second\_derivative = function3\_second\_derivative else:  print('Введено неверно, попробуйте снова.') again = True again = True draw(-5, 5, [], f) while again:  again = False  in\_type = input('Введите:\n\t k - вводить с клавиатуры\n\t f - вводить из файла\n') if in\_type.strip() == 'k':  a = in\_float('a') b = in\_float('b') eps = in\_float('eps') elif in\_type.strip() == 'f':  a, b, eps = parse() else:  print('Введено неверно, попробуйте снова.') again = True again = True while again:  again = False  in\_type = input('Выберите способ:\n\t n - метод Ньютона\n\t i - метод итераций\n\t q - выход\n') try: if in\_type.strip() == 'n':  x, n = newtone(a, b, eps, f, f\_first\_derivative, f\_second\_derivative) elif in\_type.strip() == 'i':  x, n = iterations(a, b, eps, f, f\_first\_derivative) if in\_type.strip() == 'n' or in\_type.strip() == 'i': print(f'Решено за {n} итераций\nX = {x}\nf(x) =  {f(x)}\n График:\n')  draw(-5, 5, [x], f) elif in\_type.strip() == 'q':  break else:  raise ValueError('Введено неверно, попробуй снова.') except ValueError as err:  print(err) again = True |

Функция для взаимодействия с системой

def calculate\_nonlinear\_system(): again = True

|  |
| --- |
| draw\_system(-3, 3, [], [], f1, f2) while again:  again = False  in\_type = input('Введите:\n\t k - вводить с клавиатуры\n\t f - вводить из файла\n') if in\_type.strip() == 'k': eps = in\_float('eps') elif in\_type.strip() == 'f':  eps = parse() else:  print('Введено неверно, попробуйте снова.') again = True  x1, x2, n, res1, res2 = iteration\_for\_system(eps) print(f'Решено за {n} итераций\nX1 = {x1}\nX2 = {x2}\n Погрешности:  {res1}, {res2} \n График:\n')  draw\_system(-3, 3, [x1], [x2], f1, f2) |

Метод, запускающий программу.

|  |
| --- |
| def run(): again = True while again:  again = False  in\_type = input('Что считаем:\n\t 1 - уравнение\n\t 2 - систему\n') if in\_type.strip() == '1':  calculate\_nonlinear\_equation() elif in\_type.strip() == '2':  calculate\_nonlinear\_system() else:  print('Введено неверно, попробуйте снова.') again = True again = True |

run()

**Пример работы:**

Что считаем:

1. - уравнение
2. - систему

1

Выберите уравнение:

1. - -1.8\*x^3-2.94\*x^2+10.37\*x+5.38
2. - sin(x)+0.1
3. - x^3-x+4

1

Введите

:

k

-

вводить

с

клавиатуры

f

-

вводить

из

файла

k

Введите

значение

a: 2.2

Введите

значение

b: 1.8

Введите

значение

eps: 0.01

Выберите

способ

:

n

-

метод

Ньютона

i

-

метод

итераций

q

-

выход

i

Решено

за

5

итераций

X = 1.9984905630829255

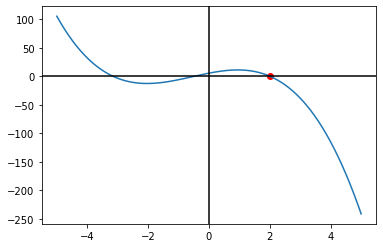
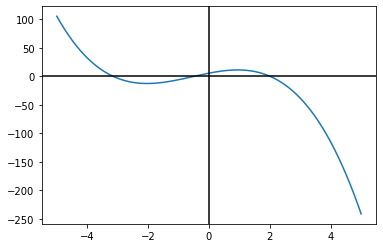
f(x) =

-

0.005329344299421557

График

:



Ч

то

считаем

:

1

-

уравнение

2

-

систему

2

Введите

:

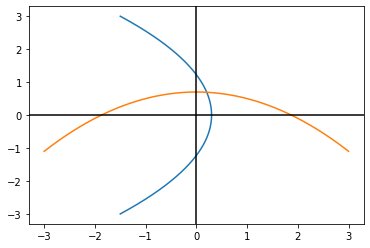
k

-

вводить

с

клавиатуры



f

-

вводить

из

файла

k

Введите

значение

eps: 0.01

Сходимость

есть

,

решаем

...

Решено

за

6

итераций

X1 = 0.20336259305918464

X2 = 0.6772908348386247

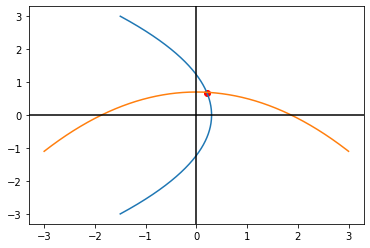
Погрешности

:

0.004132174510114273, 0.0021648262752982372

График

:



**Вывод:**

В результате выполнения лабораторной работы я:

* Познакомился с различными способами решения нелинейных уравнений, системы нелинейных уравнений
* Попрактиковался как в решении задачи на бумаге, так и написании программной реализации
* Поработал с Python, в особенности с библиотекой отрисовки графиков matplotlib