Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Техники

**Лабораторная работа по дисциплине «Вычислительная математика» №2**

Вариант: 4

Преподаватель:   
Рыбаков Степан Дмитриевич

Выполнил: Васильченко Роман

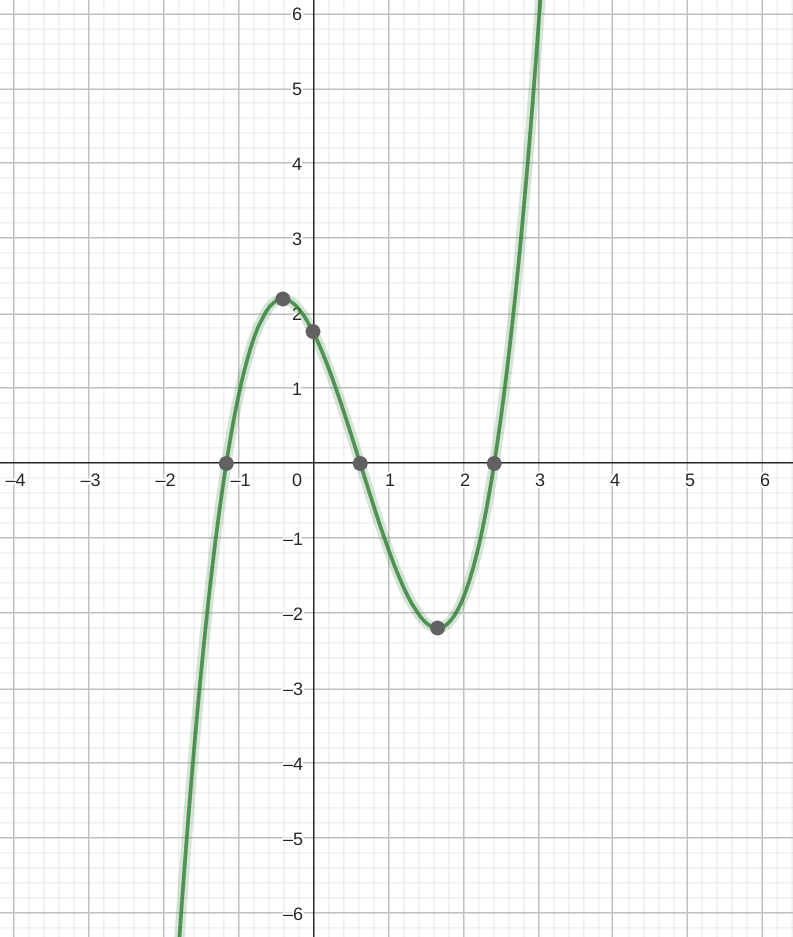
Группа: Р32081

Санкт-Петербург, 2023г

# Цель работы

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

Вычислительная часть:







Для определения интервалов изоляции корней данного уравнения, можно воспользоваться методом интервалов знакопеременности. Для этого нужно найти значения функции на различных интервалах и определить знак функции на каждом из них.  
  
Получим приближенные значения корней:  
x ≈ 0.6, x ≈ 1.2, x ≈ -0.3

Теперь нужно разбить ось x на четыре интервала: (-бесконечность, -0.3), (-0.3, 0.6), (0.6, 1.2) и (1.2, бесконечность). На каждом из этих интервалов нужно определить знак функции.

Для этого можем вычислить значения функции в произвольной точке каждого интервала. Например, для интервала (-бесконечность, -0.3) можно выбрать x = -1, для интервала (-0.3, 0.6) - x = 0, для интервала (0.6, 1.2) - x = 1, и для интервала (1.2, бесконечность) - x = 2.

Таким образом, получим следующие значения функции:

для x = -1: f(-1) = -2.55

для x = 0: f(0) = 1.76

для x = 1: f(1) = -1.13

для x = 2: f(2) = 2.76

Знаки функции на каждом интервале будут соответственно:

(-бесконечность, -0.3): -

(-0.3, 0.6): +

(0.6, 1.2): -

(1.2, бесконечность): +

Таким образом, мы получаем два интервала изоляции корней уравнения: (-бесконечность, -0.3) и (0.6, 1.2).

1. x1 ≈ -1,16  
   x2 ≈ 0,63  
   x3 ≈ 2,42
2. Метод простой итерации

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | x*k* | xk+1 | 𝝋(x*k+1*) | f(x*k+1*) | |x*k+1*-x*k*| |  | Корень |
| 0 | -1,0000 | -1,1820 | -1,1469 | -0,1680 | 0,1820 |  | **-1,1593** |
| 1 | -1,1820 | -1,1469 | -1,1593 | 0,0594 | 0,0352 |  |  |
| 2 | -1,1469 | -1,1593 | -1,1552 | -0,0195 | 0,0124 |  |  |
| 3 | -1,1593 | -1,1552 | -1,1566 | 0,0066 | 0,0041 |  |  |
| 4 | -1,1552 | -1,1566 | -1,1561 | -0,0022 | 0,0014 |  |  |
| 5 | -1,1566 | -1,1561 | -1,1563 | 0,0007 | 0,0005 |  |  |
| 6 | -1,1561 | -1,1563 | -1,1562 | -0,0002 | 0,0002 |  |  |
| Метод половинного деления |  |  |  |  |  |  |  |
|  | a | b |  | Точность |  | Корень |  |
| Интервал | 0 | 1 |  | 0,01 |  | **0,6328** |  |
|  |  |  |  |  |  |  |  |
| № | a | b | x | f(a) | f(b) | f(x) | |a-b| |
| 0 | 0,0000 | 1,0000 | 0,5000 | 1,7600 | -1,1300 | 0,4125 | 1,0000 |
| 1 | 0,5000 | 1,0000 | 0,7500 | 0,4125 | -1,1300 | -0,3813 | 0,5000 |
| 2 | 0,5000 | 0,7500 | 0,6250 | 0,4125 | -0,3813 | 0,0159 | 0,2500 |
| 3 | 0,6250 | 0,7500 | 0,6875 | 0,0159 | -0,3813 | -0,1834 | 0,1250 |
| 4 | 0,6250 | 0,6875 | 0,6563 | 0,0159 | -0,1834 | -0,0838 | 0,0625 |
| 5 | 0,6250 | 0,6563 | 0,6406 | 0,0159 | -0,0838 | -0,0340 | 0,0313 |
| 6 | 0,6250 | 0,6406 | 0,6328 | 0,0159 | -0,0340 | -0,0091 | 0,0156 |
| 7 | 0,6250 | 0,6328 | 0,6289 | 0,0159 | -0,0091 | 0,0034 | 0,0078 |
| Метод секущих |  |  |  |  |  |  |  |
| № | xk-1 | xk | xk+1 | f(xk-1) | f(xk) | f(xk+1) | |xk+1-xk| |
| 0 | 2,0000 | 3,0000 | 2,2384 | -1,8000 | 5,7500 | -0,9711 | 0,7616 |
| 1 | 3,0000 | 3,0000 | 2,2455 | 5,7500 | 5,7500 | -0,4556 | 0,4811 |
| 2 | 3,0000 | 3,0000 | 2,2467 | 5,7500 | 5,7500 | -0,0097 | 0,0078 |

Листинг программы  
import numpy as np

import matplotlib.pyplot as plt

from scipy.misc import derivative

# Определение функций

def f1(x):

return x \*\* 2 - 4

def f2(x):

return np.sin(x)

def f3(x):

return np.exp(x) - 2

# Здесь добавьте свои функции

functions = [f1, f2, f3]

# Реализация методов

def bisection\_method(func, a, b, tol):

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

raise ValueError("Функция должна иметь разные знаки на границах интервала")

iterations = 0

while (b - a) / 2 > tol:

iterations += 1

c = (a + b) / 2

if func(c) == 0:

break

elif func(c) \* func(a) < 0:

b = c

else:

a = c

return (a + b) / 2, iterations

def newton\_method(func, a, b, tol):

def find\_initial\_approximation():

if func(a) \* derivative(func, a) > 0:

return a

elif func(b) \* derivative(func, b) > 0:

return b

else:

raise ValueError("Невозможно выбрать начальное приближение")

x = find\_initial\_approximation()

iterations = 0

while True:

iterations += 1

x\_prev = x

x = x\_prev - func(x\_prev) / derivative(func, x\_prev)

if abs(x - x\_prev) < tol:

break

return x, iterations

def fixed\_point\_iteration(func, a, b, tol):

def g(x):

return x - func(x)

def g\_derivative(x):

return 1 - derivative(func, x)

if not (g\_derivative(a) < 1 and g\_derivative(b) < 1):

raise ValueError("Не выполняется достаточное условие сходимости")

x = (a + b) / 2

iterations = 0

while True:

iterations += 1

x\_prev = x

x = g(x\_prev)

if abs(x - x\_prev) < tol:

break

return x, iterations

# Ввод данных

def input\_data():

print("Выберите уравнение:")

for i, func in enumerate(functions, start=1):

print(f"{i}. {func.\_\_name\_\_}")

func\_idx = int(input("Введите номер уравнения: ")) - 1

selected\_func = functions[func\_idx]

a = float(input("Введите левую границу интервала a: "))

b = float(input("Введите правую границу интервала b: "))

tol = float(input("Введите погрешность: "))

if a >= b:

raise ValueError("Неверно задан интервал")

return selected\_func, a, b, tol

# Вывод результатов

def output\_results(func, results, to\_file=False):

root, iterations = results

if to\_file:

with open("output.txt", "w") as f:

f.write(f"Найденный корень уравнения: {root}\n")

f.write(f"Значение функции в корне: {func(root)}\n")

f.write(f"Число итераций: {iterations}\n")

else:

print(f"Найденный корень уравнения: {root}")

print(f"Значение функции в корне: {func(root)}")

print(f"Число итераций: {iterations}")

# Графики функций

def plot\_function(func, a, b):

x = np.linspace(a - 0.1 \* (b - a), b + 0.1 \* (b - a), 1000)

y = func(x)

plt.plot(x, y)

plt.axhline(0, color="black", lw=0.5)

plt.axvline(0, color="black", lw=0.5)

plt.xlabel("x")

plt.ylabel("y")

plt.title(f"График функции {func.\_\_name\_\_}")

plt.grid(True)

plt.show()

def main():

try:

selected\_func, a, b, tol = input\_data()

print("Выберите метод для нахождения корня:")

print("1. Метод половинного деления")

print("2. Метод Ньютона")

print("3. Метод простой итерации")

method\_idx = int(input("Введите номер метода: "))

if method\_idx == 1:

result = bisection\_method(selected\_func, a, b, tol)

elif method\_idx == 2:

result = newton\_method(selected\_func, a, b, tol)

elif method\_idx == 3:

result = fixed\_point\_iteration(selected\_func, a, b, tol)

else:

raise ValueError("Некорректный номер метода")

output\_choice = input("Вывести результаты на экран или в файл? (screen/file): ")

if output\_choice == "screen":

output\_results(selected\_func,result)

elif output\_choice == "file":

output\_results(selected\_func,result, to\_file=True)

else:

raise ValueError("Некорректный выбор вывода результатов")

plot\_function(selected\_func, a, b)

except ValueError as e:

print(f"Ошибка: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import fsolve

def system\_1(xy):

x, y = xy

return [x\*\*2 + y\*\*2 - 1, x\*\*2 - y - 0.5]

def system\_2(xy):

x, y = xy

return [x\*\*2 + y\*\*2 - 1, x - y\*\*2]

def plot\_system(system):

x = np.linspace(-2, 2, 400)

y = np.linspace(-2, 2, 400)

X, Y = np.meshgrid(x, y)

Z1 = np.array([system([x\_, y\_])[0] for x\_, y\_ in zip(np.ravel(X), np.ravel(Y))]).reshape(X.shape)

Z2 = np.array([system([x\_, y\_])[1] for x\_, y\_ in zip(np.ravel(X), np.ravel(Y))]).reshape(X.shape)

plt.contour(X, Y, Z1, levels=[0], colors='r')

plt.contour(X, Y, Z2, levels=[0], colors='b')

plt.xlabel('x')

plt.ylabel('y')

plt.show()

def simple\_iteration(system, x0, y0, epsilon=1e-5, max\_iter=1000):

def jacobian(xy):

x, y = xy

return np.array([[2\*x, 2\*y], [2\*x, -1]])

xy = np.array([x0, y0], dtype=float)

for i in range(max\_iter):

J\_inv = np.linalg.inv(jacobian(xy))

F = np.array(system(xy))

xy\_next = xy - np.dot(J\_inv, F)

error = np.linalg.norm(xy\_next - xy)

if error < epsilon:

return xy\_next, i + 1, error

xy = xy\_next

raise Exception("Solution not found in {} iterations".format(max\_iter))

if \_\_name\_\_ == "\_\_main\_\_":

systems = {'1': system\_1, '2': system\_2}

print("Выберите систему уравнений:")

for key, value in systems.items():

print(key, value.\_\_doc\_\_)

choice = input("Введите номер системы: ")

system = systems[choice]

x0, y0 = map(float, input("Введите начальные приближения x0, y0: ").split())

xy\_solution, iterations, error = simple\_iteration(system, x0, y0)

print(f"Вектор неизвестных: x1 = {xy\_solution[0]:.5f}, x2 = {xy\_solution[1]:.5f}")

print(f"Количество итераций: {iterations}")

print(f"Вектор погрешностей: {error:.5e}")

residuals = np.abs(np.array(system(xy\_solution)))

print("Проверка решения системы уравнений:")

print(f"Невязки: {residuals[0]:.5e}, {residuals[1]:.5e}")

plot\_system(system)

# Вывод:

В ходе выполнения лабораторной работы были изучены численные методы решения нелинейных уравнений и систем нелинейных уравнений с использованием Python. Была успешно реализована программа, предусматривающая выбор уравнений, методов решения, ввод исходных данных, проверку корректности данных и сходимости методов, а также вывод результатов на экран или в файл. В результате работы были найдены корни заданных уравнений и систем с использованием различных численных методов, а также были построены графики функций для полного представления исследуемых интервалов.