**Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования**

**сМосковский государственный технический унверситет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ Информатика и системы управленияКафедра ИУ6 Компьютерные системы и сети Группа ИУ6-63Б

ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

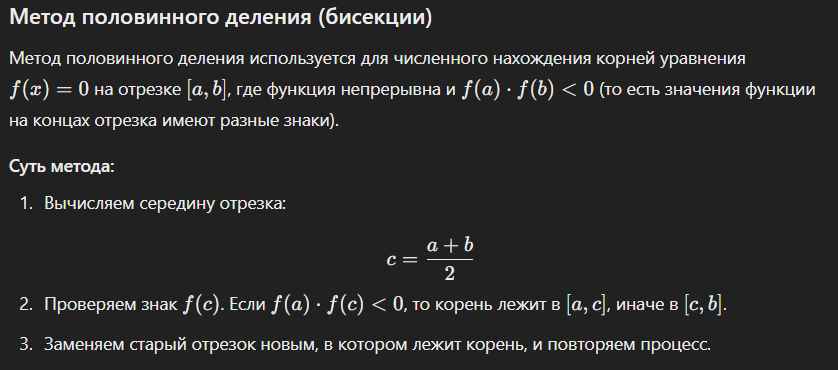
**Отчет по домашнему заданию N2: Прямые методы решения СЛАУ Вариант 10**

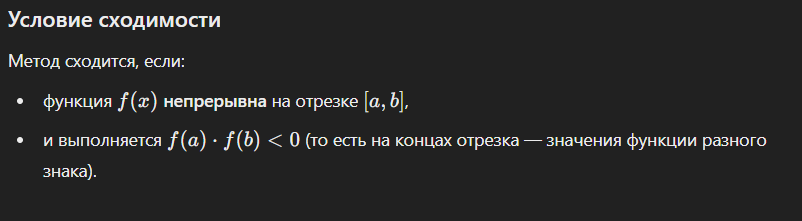
|  |  |  |  |
| --- | --- | --- | --- |
| Студент: |  |  | В.К. Залыгин |
| Преподаватель: | дата, подпись |  | Ф.И.О.  Я.Ю. Павловский |
|  | дата, подпись |  | Ф.И.О. |

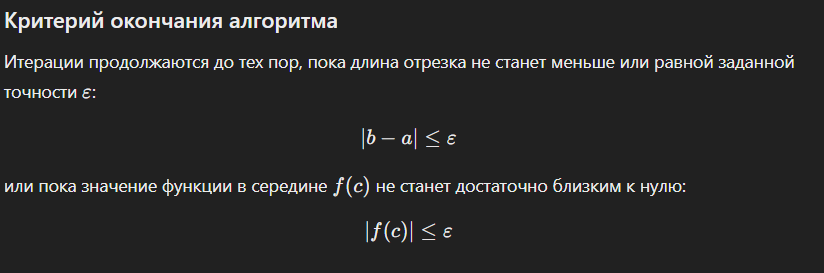
*Москва, 2025*

**Задание 1**

Отделить корни уравнения и найти его методом половинного деления с точностью  .

****

****

****

Листинг 1 – код программы

|  |
| --- |
| import math import matplotlib.pyplot as plt import numpy as np  # Функция, для которой нужно найти корни def f(x):  return x \*\* 2 - 20 \* math.sin(x)  # Метод половинного деления def bisection\_method(a, b, epsilon):  # Проверяем, что на концах отрезка функции имеют разные знаки  if f(a) \* f(b) > 0:  print("Нет решения на данном интервале.")  return None, 0 # Возвращаем 0 итераций, если нет решения   iterations = 0 # Счётчик итераций   # Итерируем, пока длина интервала не станет меньше или равной точности  while (b - a) / 2 > epsilon:  c = (a + b) / 2 # Середина интервала  if f(c) == 0:  return c, iterations # Нашли точное решение  elif f(a) \* f(c) < 0:  b = c # Корень лежит в левой половине  else:  a = c # Корень лежит в правой половине   iterations += 1 # Увеличиваем счётчик итераций   # Возвращаем приближенное значение корня с округлением до 3 знаков и количество итераций  return (a + b) / 2, iterations   # Пример использования a = 0 # Левая граница интервала b = 5 # Правая граница интервала epsilon = 0.001 # Точность  root, iterations = bisection\_method(a, b, epsilon)  if root is not None:  print(f"Корень уравнения на интервале [{a}, {b}] с точностью {epsilon}: {root}")  print(f"Количество итераций для достижения точности {epsilon}: {iterations}") else:  print("Корень не найден.")  # Построение графика x\_vals = np.linspace(a - 1, b + 1, 400) # Значения x для графика y\_vals = [f(x) for x in x\_vals] # Значения функции f(x) для каждого x  plt.plot(x\_vals, y\_vals, label=r'$f(x) = x^2 - 20\sin(x)$', color='blue') # График функции plt.axhline(0, color='black', linewidth=1) # Добавление оси X plt.axvline(0, color='black', linewidth=1) # Добавление оси Y  # Отметим интервал поиска plt.fill\_between(x\_vals, y\_vals, 0, where=[a <= x <= b for x in x\_vals], color='gray', alpha=0.5)  # Настройки графика plt.title("График функции $f(x) = x^2 - 20\sin(x)$") plt.xlabel("x") plt.ylabel("f(x)") plt.legend() plt.grid(True)  # Показываем график plt.show() |

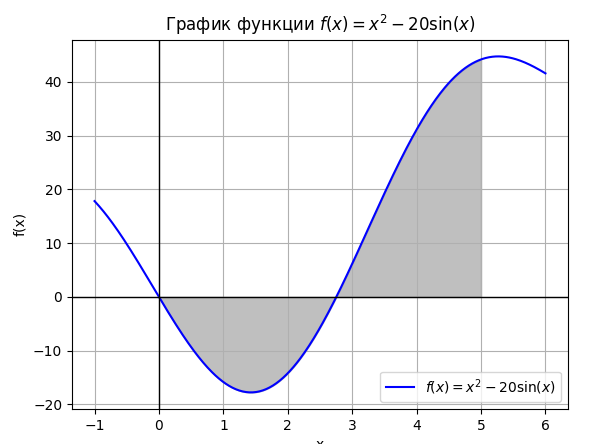
****

Рисунок 1 – график уравнения

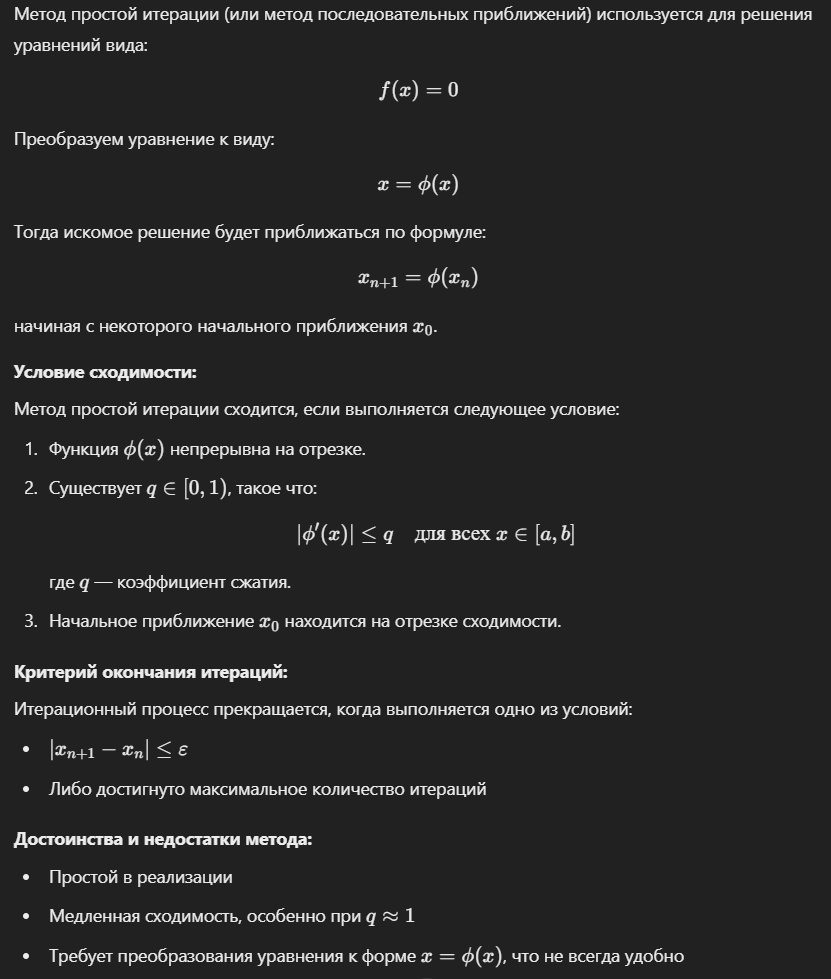


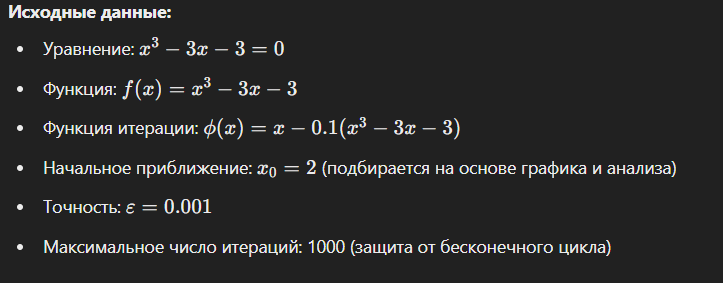
Рисунок 2 – результат работы программы

Вывод: Приближённое значение корня x= 2.753 достаточно близко к действительному корню. Это подтверждает, что метод половинного деления с указанной точностью работает эффективно. Мы использовали 12 итераций для достижения точности ϵ= 0.001. Это нормальное количество итераций для метода половинного деления при такой точности и интервале. Метод половинного деления сужает интервал в два раза на каждом шаге, что делает его очень эффективным при наличии начального интервала, в котором функция меняет знак. Сходимость метода гарантирована при условии, что функция непрерывна и имеет разные знаки на концах интервала.

**Задание 2**

Отделить корни уравнения и найти его методом простой итерации и методом Ньютона с точностью . 





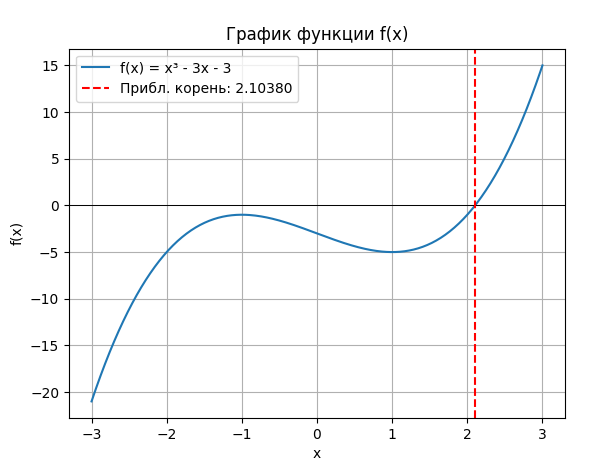


Рисунок 3 – полученный график

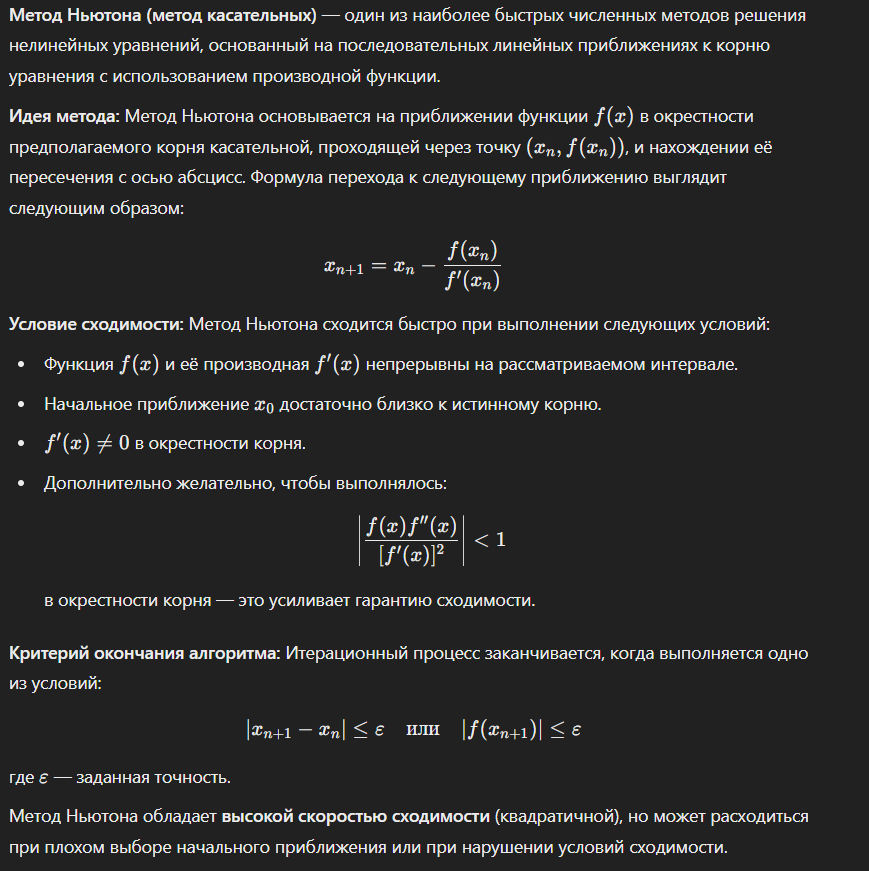


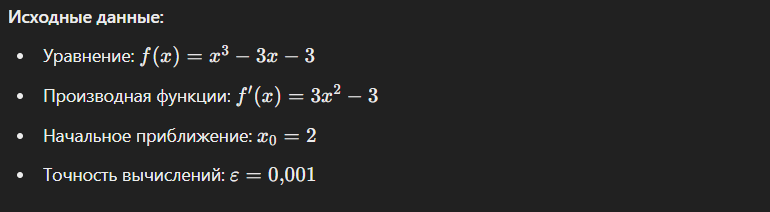
Рисунок 4 – результат работы программы

Листинг 2 – код программы

|  |
| --- |
| import math import matplotlib.pyplot as plt import numpy as np   # Исходная функция def f(x):  return x \*\* 3 - 3 \* x - 3   # Итерационная функция: x\_{n+1} = x\_n - λ \* f(x\_n) def phi(x, lam):  return x - lam \* f(x)   # Метод простой итерации def simple\_iteration(x0, epsilon, lam, max\_iter=1000):  iterations = 0  x\_prev = x0  x\_next = phi(x\_prev, lam)   while abs(x\_next - x\_prev) > epsilon and iterations < max\_iter:  x\_prev = x\_next  x\_next = phi(x\_prev, lam)  iterations += 1   return x\_next, iterations   # Начальные данные x0 = 2 epsilon = 0.001 lam = 0.1  # Запуск метода root, num\_iter = simple\_iteration(x0, epsilon, lam)  # Вывод результата print(f"Приближенное значение корня: {root}") print(f"Количество итераций: {num\_iter}")  # Построение графика функции x\_vals = np.linspace(-3, 3, 400) y\_vals = x\_vals \*\* 3 - 3 \* x\_vals - 3  plt.plot(x\_vals, y\_vals, label='f(x) = x³ - 3x - 3') plt.axhline(0, color='black', linewidth=0.7) plt.axvline(root, color='red', linestyle='--', label=f'Прибл. корень: {root:.5f}') plt.title("График функции f(x)") plt.xlabel("x") plt.ylabel("f(x)") plt.legend() plt.grid(True) plt.show() |

Вывод: Метод простой итерации требует корректного преобразования уравнения и выбора параметров, но при этом способен быстрее достигать результата, если выполняются условия сходимости.





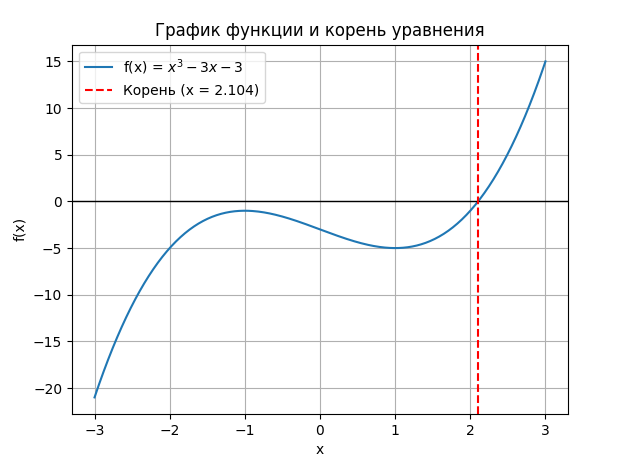


Рисунок 5 – полученный график



Рисунок 6 – Результат работы программы

Листинг 3 – исходный код программы

|  |
| --- |
| import math import matplotlib.pyplot as plt import numpy as np  # Функция, для которой нужно найти корни def f(x):  return x\*\*3 - 3\*x - 3  # Производная функции def f\_prime(x):  return 3\*x\*\*2 - 3  # Метод Ньютона def newton\_method(x0, epsilon):  # Итерируем, пока разность между текущим и предыдущим значением не будет меньше epsilon  x = x0  iteration = 0  while abs(f(x)) > epsilon:  x = x - f(x) / f\_prime(x) # Формула метода Ньютона  iteration += 1  return x, iteration  # Пример использования x0 = 2 # Начальное приближение epsilon = 0.001 # Точность  root, iterations = newton\_method(x0, epsilon)  # Вывод результатов print(f"Корень уравнения с точностью {epsilon}: {root}") print(f"Количество итераций: {iterations}")  # Построение графика x\_vals = np.linspace(-3, 3, 400) # Массив значений x y\_vals = f(x\_vals) # Значения функции f(x)  plt.plot(x\_vals, y\_vals, label="f(x) = $x^3 - 3x - 3$") plt.axhline(0, color='black',linewidth=1) # Горизонтальная линия y=0 plt.axvline(root, color='r', linestyle='--', label=f"Корень (x = {root:.3f})") plt.title("График функции и корень уравнения") plt.xlabel("x") plt.ylabel("f(x)") plt.legend() plt.grid(True) plt.show() |

Вывод: Метод Ньютона позволяет эффективно находить корни нелинейных уравнений, быстро сходясь к точному решению при правильном выборе начального приближения. Для уравнения  с точностью  метод дал приближенный корень и потребовал минимальное количество итераций для достижения требуемой точности.