Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
сМосковский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ	Фундаментальные Науки
Кафедра	ИУ-6 Математическое моделирование
Группа	ИУ6-63Б

ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

Отчет по домашнему заданию N2:

Прямые методы решения СЛАУ

Вариант 7

Студент:	05.04.2025	Д.Г. Донских
	дата, подпись	Ф.И.О.
Преподаватель:	05.04.2025	Я.Ю. Павловский
	дата, подпись	Ф.И.О.

Москва, 2025

Задание 1

Отделить корни уравнения и найти его методом половинного деления с точностью $\varepsilon = 0{,}001$ $x^2 - 20\sin(x) = 0$

Метод половинного деления (бисекции)

Метод половинного деления используется для численного нахождения корней уравнения $f(x) = 0$ на отрезке $[a, b]$, где функция непрерывна и $f(a) \cdot f(b) < 0$ (то есть значения функции на концах отрезка имеют разные знаки).

Суть метода:

1. Вычисляем середину отрезка:

$$c = \frac{a+b}{2}$$

2. Проверяем знак $f(c)$. Если $f(a) \cdot f(c) < 0$, то корень лежит в $[a, c]$, иначе в $[c, b]$.
3. Заменяем старый отрезок новым, в котором лежит корень, и повторяем процесс.

Условие сходимости

Метод сходится, если:

- функция $f(x)$ **непрерывна** на отрезке $[a, b]$,
- и выполняется $f(a) \cdot f(b) < 0$ (то есть на концах отрезка — значения функции разного знака).

Критерий окончания алгоритма

Итерации продолжаются до тех пор, пока длина отрезка не станет меньше или равной заданной точности ε:

$$|b - a| \leq \varepsilon$$

или пока значение функции в середине $f(c)$ не станет достаточно близким к нулю:

$$|f(c)| \leq \varepsilon$$

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()
```

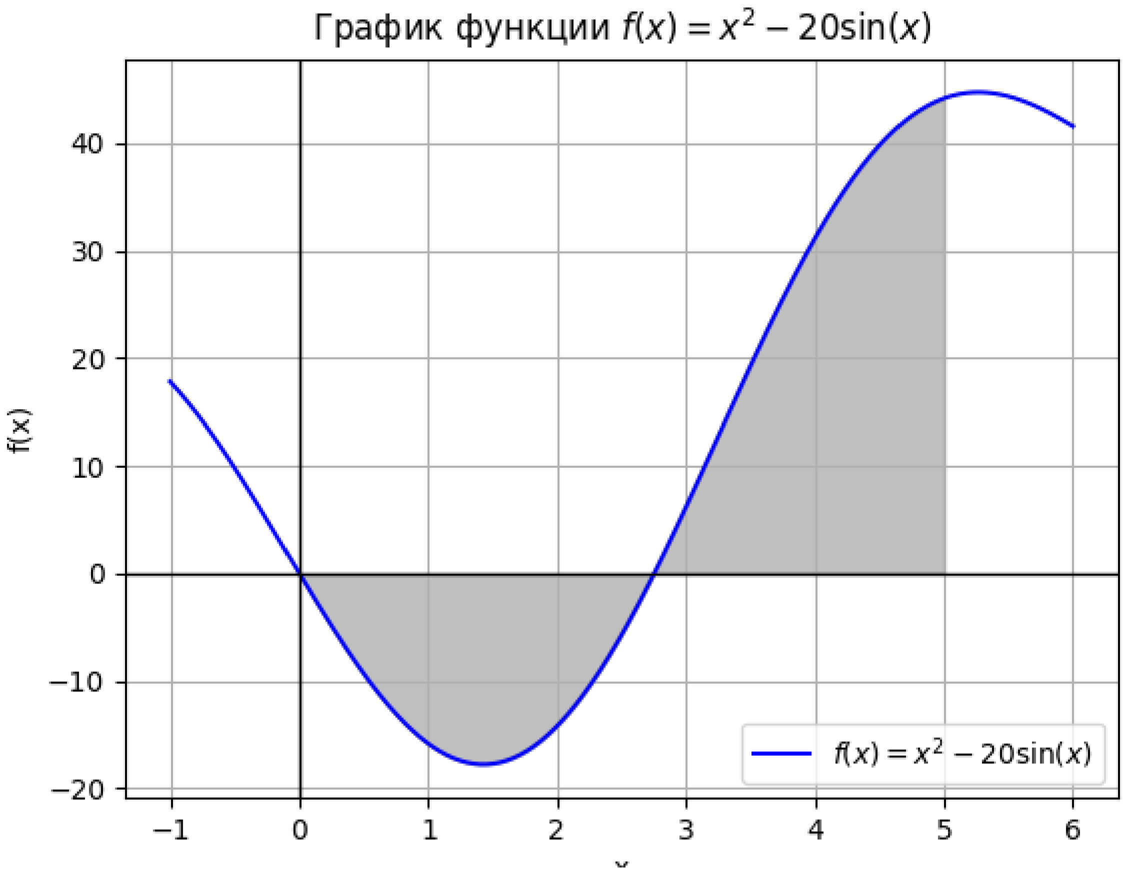

Рисунок 1 – график уравнения

```
Корень уравнения на интервале [0, 5] с точностью 0.001: 2.7532958984375
Количество итераций для достижения точности 0.001: 12
```

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 $\varepsilon = 0{,}001$. $x^3 - 3x - 3 = 0$

Метод простой итерации (или метод последовательных приближений) используется для решения уравнений вида:

$$f(x) = 0$$

Преобразуем уравнение к виду:

$$x = \phi(x)$$

Тогда искомое решение будет приближаться по формуле:

$$x_{n+1} = \phi(x_n)$$

начиная с некоторого начального приближения x_0.

Условие сходимости:

Метод простой итерации сходится, если выполняется следующее условие:

1. Функция $\phi(x)$ непрерывна на отрезке.
2. Существует $q \in [0, 1)$, такое что:

   $$|\phi'(x)| \leq q \quad \text{для всех } x \in [a, b]$$

   где q — коэффициент сжатия.
3. Начальное приближение x_0 находится на отрезке сходимости.

Критерий окончания итераций:

Итерационный процесс прекращается, когда выполняется одно из условий:

- $|x_{n+1} - x_n| \leq \varepsilon$
- Либо достигнуто максимальное количество итераций

Достоинства и недостатки метода:

- Простой в реализации
- Медленная сходимость, особенно при $q \approx 1$
- Требует преобразования уравнения к форме $x = \phi(x)$, что не всегда удобно

Исходные данные:

- Уравнение: $x^3 - 3x - 3 = 0$
- Функция: $f(x) = x^3 - 3x - 3$
- Функция итерации: $\phi(x) = x - 0.1(x^3 - 3x - 3)$
- Начальное приближение: $x_0 = 2$ (подбирается на основе графика и анализа)
- Точность: $\varepsilon = 0.001$
- Максимальное число итераций: 1000 (защита от бесконечного цикла)

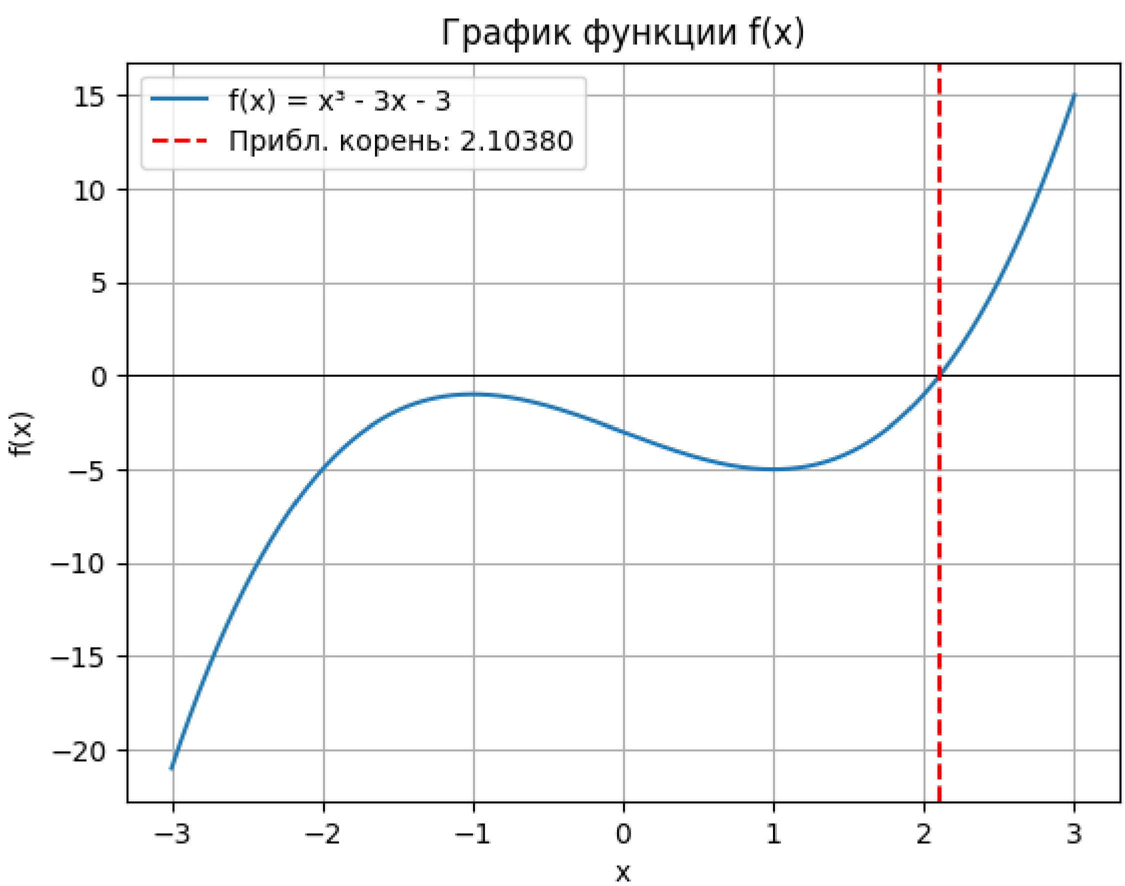

Рисунок 3 – полученный график

```
Приближенное значение корня: 2.1038007117681
Количество итераций: 2
```

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
```

```python
# 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.

Метод Ньютона (метод касательных) — один из наиболее быстрых численных методов решения нелинейных уравнений, основанный на последовательных линейных приближениях к корню уравнения с использованием производной функции.

Идея метода: Метод Ньютона основывается на приближении функции $f(x)$ в окрестности предполагаемого корня касательной, проходящей через точку $(x_n, f(x_n))$, и нахождении её пересечения с осью абсцисс. Формула перехода к следующему приближению выглядит следующим образом:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Условие сходимости: Метод Ньютона сходится быстро при выполнении следующих условий:

- Функция $f(x)$ и её производная $f'(x)$ непрерывны на рассматриваемом интервале.
- Начальное приближение x_0 достаточно близко к истинному корню.
- $f'(x) \neq 0$ в окрестности корня.
- Дополнительно желательно, чтобы выполнялось:

$$\left| \frac{f(x) f''(x)}{[f'(x)]^2} \right| < 1$$

  в окрестности корня — это усиливает гарантию сходимости.

Критерий окончания алгоритма: Итерационный процесс заканчивается, когда выполняется одно из условий:

$$|x_{n+1} - x_n| \leq \varepsilon \quad \text{или} \quad |f(x_{n+1})| \leq \varepsilon$$

где ε — заданная точность.

Метод Ньютона обладает **высокой скоростью сходимости** (квадратичной), но может расходиться при плохом выборе начального приближения или при нарушении условий сходимости.

Исходные данные:

- Уравнение: $f(x) = x^3 - 3x - 3$
- Производная функции: $f'(x) = 3x^2 - 3$
- Начальное приближение: $x_0 = 2$
- Точность вычислений: $\varepsilon = 0{,}001$

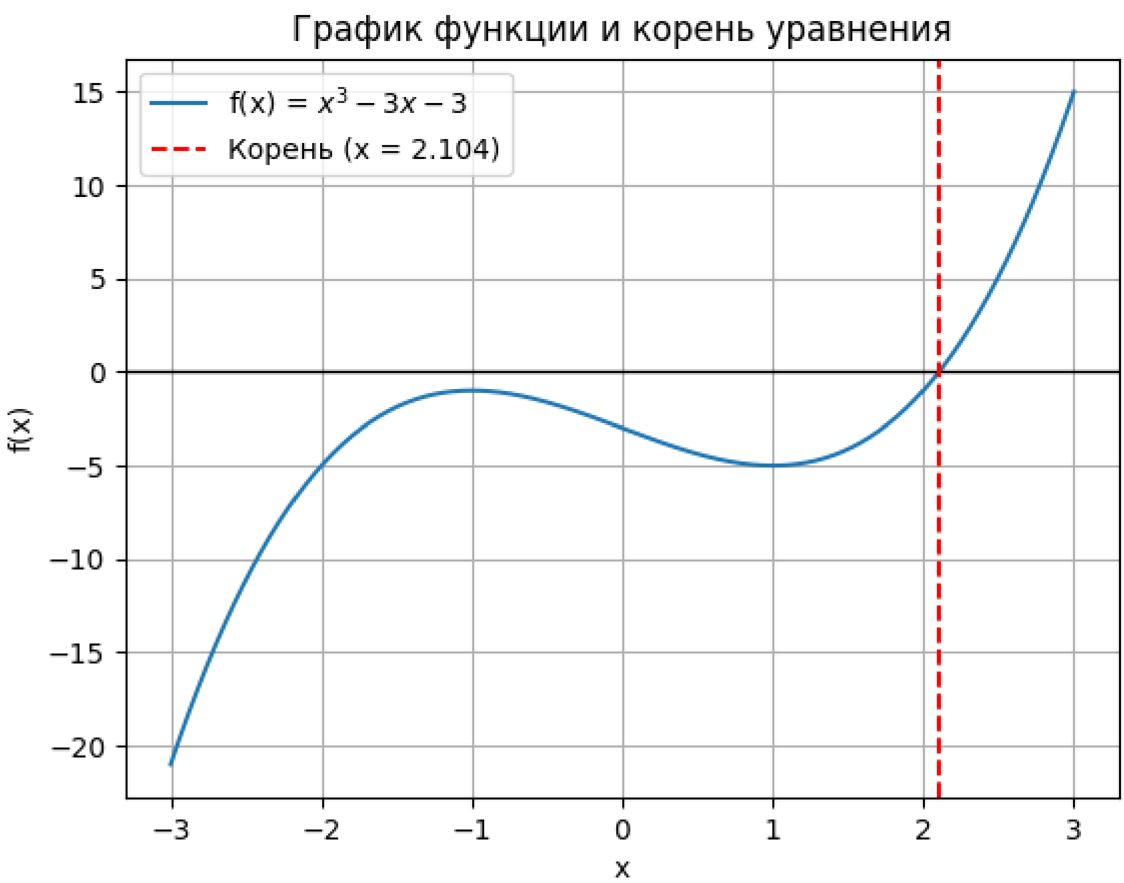

Рисунок 5 – полученный график

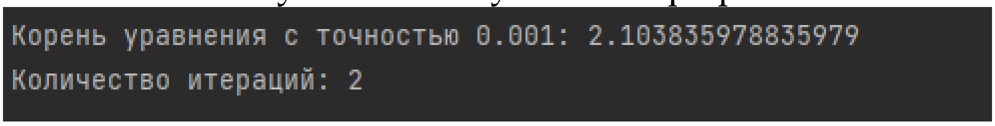

Рисунок 6 – Результат работы программы

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
```

```python
# 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 $x^3 - 3x - 3 = 0$ с точностью $\varepsilon = 0{,}001$ метод дал приближенный корень и потребовал минимальное количество итераций для достижения требуемой точности.