

Отчет для программы по численным методам – Худобин В. О.

Задача №3: (№12) МПИ Δ^2 -процесс Эйткена

Язык программирования: Python

Постановка задачи:

Рассматривается задача нахождения корня нелинейного уравнения $f(x) = 0$ с использованием метода простой итерации и процесса ускорения Эйткена для улучшения сходимости. Требуется реализовать алгоритм решения уравнения, который использует процесс Δ^2 -Эйткена для увеличения скорости сходимости.

Задача состоит из следующих этапов:

1. Построение итерационной схемы для нахождения приближенного значения корня.
2. Применение метода Эйткена для ускорения сходимости.
3. Оценка точности полученного решения на основе заданного критерия сходимости.

Метод решения:

Процесс Δ^2 -Эйткена используется для ускорения сходимости последовательностей. В данном случае метод применяется к последовательности, порождаемой методом простой итерации для уравнения $f(x) = 0$. Для этого уравнение преобразуется в форму $x = g(x)$, где $g(x) = x + f(x)$.

Δ^2 -алгоритм Эйткена

Шаг 0. Ввод x_0 (начального приближения), $\varphi(x)$ (исходной функции), q (оценки модуля производной), ε (допустимой абсолютной погрешности).

Шаг 1. Вычисление значений: $x_1 := \varphi(x_0)$, $x_2 := \varphi(x_1)$.

Шаг 2. Δ^2 -ускорение: $\tilde{x}_2 := \frac{x_0 x_2 - x_1^2}{x_2 - 2x_1 + x_0}$.

Шаг 3. Вычисление контрольного значения: $x_3 := \varphi(\tilde{x}_2)$.

Шаг 4. Проверка на точность: если $|x_3 - \tilde{x}_2| > \frac{1-q}{q} \varepsilon$, то положить

$x_0 := \tilde{x}_2$, $x_1 := x_3$, вычислить $x_2 := \varphi(x_1)$ и вернуться к шагу 2.

Шаг 5. Положить $\xi := x_3$ (с точностью ε).

Алгоритм:

1. Подготовка данных: Чтение уравнения, начального приближения и точности из файла `input.txt`.

2. Применение итерационного процесса:

Построение последовательности

- построение последовательности:

$$x_{n+1} = g(x_n)$$

- для ускорения сходимости применяется процесс Эйткена:

$$\chi = \frac{x_n * x_{n+2} - x_{n+1}^2}{x_{n+2} - 2x_{n+1} + x_n}$$

3. Проверка критерия остановки: Сходимость оценивается по условию

$$|x_{n+3} - \chi| < tol$$

4. Возвращение результата: Если решение найдено, оно сохраняется в `output.txt`.

Реализация программы

Код включает следующие функции:

- **parse_input(filename)**: Читает входные данные и извлекает уравнение, начальное приближение и точность.
- **aitken_delta_squared(f, x0, tol, q)**: Реализует процесс Эйткена для решения нелинейного уравнения.
- **main()**: Координирует процесс чтения данных, выполнения итераций, оценки точности и записи результатов.

Пример кода

```
import numpy as np
import sympy as sp

def parse_input(filename):
    """Парсинг данных из файла input.txt"""
    with open(filename, 'r') as file:
        lines = file.readlines()
```

```

    equation_line = next(line for line in lines if 'f(x) =' in
line)
    initial_guess_line = next(line for line in lines if 'x0 ='
in line)
    tolerance_line = next(line for line in lines if 'tol =' in
line)

```

```

    equation_str = equation_line.split('=')[1].strip()
    initial_guess =
float(initial_guess_line.split('=')[1].strip())
    tolerance = float(tolerance_line.split('=')[1].strip())

    return equation_str, initial_guess, tolerance

```

```

def aitken_delta_squared(f, x0, tol, q):
    """Решение нелинейного уравнения методом простой итерации
с процессом Эйткена"""

```

```

    x = sp.symbols('x')
    f_expr = sp.sympify(f)
    g_expr = x + f_expr
    g_func = sp.lambdify(x, g_expr, 'numpy')

```

```

    x1 = g_func(x0)
    x2 = g_func(x1)

```

```

    iteration = 0
    max_iter = 100

```

```

    while iteration < max_iter:
        try:
            x_tilde_2 = (x0 * x2 - x1**2) / (x2 - 2 * x1 + x0)
            x3 = g_func(x_tilde_2)

            if abs(x3 - x_tilde_2) < tol:
                return x_tilde_2

            x0, x1 = x_tilde_2, x3
            x2 = g_func(x1)

```

```

        except (ZeroDivisionError, RuntimeWarning):
            print("Ошибка: деление на ноль или числовая
нестабильность")
            return None

        iteration += 1

    print("Превышено максимальное количество итераций")
    return None

def main():
    equation_str, x0, tol = parse_input('input.txt')
    q = 0.5 # этот параметр теперь не используется
    root = aitken_delta_squared(equation_str, x0, tol, q)

    if root is not None:
        x = sp.symbols('x')
        f_expr = sp.sympify(equation_str)
        f_func = sp.lambdify(x, f_expr, 'numpy')
        residual = abs(f_func(root))

        result_str = f"Приближенное значение корня: {root}\n"
        result_str += f"Невязка |f(x)|: {residual}"

        print(result_str)

        with open('output.txt', 'w') as file:
            file.write(result_str)
    else:
        error_msg = "Не удалось найти корень. Попробуйте
другое начальное приближение."
        print(error_msg)
        with open('output.txt', 'w') as file:
            file.write(error_msg)

if __name__ == "__main__":
    main()

```

Тестовые данные

Для проверки работы программы используются следующие тестовые данные:

Функция: $f(x) = \sqrt{x} - \cos(0.378x)$

Начальное приближение: 0.5


Погрешность: 0.00001

Результат:

Программа успешно решает уравнение и выводит следующий результат:

Приближенное значение корня: 0.890830836894308

Невязка $|f(x)|$: 4.3259613013990617e-07



Построим график каждой части уравнения. Решение — абсцисса (координата x) точки пересечения.

$x \approx 0.89083149$

[Нажмите для просмотра решения...](#)

$\sqrt{x} - \cos(0.378x) = 0$

Заключение:

Метод Эйткена показал себя как эффективное средство для ускорения сходимости в решении нелинейных уравнений. Однако, к сожалению, данный метод не очень подходит для уравнений с несколькими корнями, так как его сходимость зависит от начального приближения. В таких случаях метод может сходиться к одному из корней, не гарантируя нахождения других, и выбор начального приближения становится критичным для успешного решения задачи.