



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования

**«Дальневосточный федеральный университет»
(ДВФУ)**

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ
ТЕХНОЛОГИЙ**

Департамент математического и компьютерного моделирования

ОТЧЕТ по лабораторной работе № 5

по дисциплине

«Вычислительная математика»

Направление подготовки
02.03.01 «Математика и компьютерные науки»

Вариант № 6

Выполнил(а): студент гр. Б9122-02.03.01сцт
Бекболот Отгонцэцэг
Проверил: преподаватель
_____ Ф.И.О.

**Владивосток
2024**

Цель работы :

Решить систему линейных алгебраических уравнений в виде $Ax = b$ двумя приближенными методами.

Ход работы:

1. Решить СЛАУ двумя итерационными методами: методом Якоби и методом верхней релаксации.
2. Решить систему с заданной точностью $\varepsilon = 10^{-4}$.

$$\begin{cases} 6.22x_1 + 1.42x_2 - 1.72x_3 + 1.91x_4 = 7.53 \\ 1.42x_1 + 5.33x_2 + 1.11x_3 - 1.82x_4 = 6.06 \\ -1.72x_1 + 1.11x_2 + 5.24x_3 + 1.42x_4 = 8.05 \\ 1.91x_1 - 1.82x_2 + 1.42x_3 + 6.55x_4 = 8.06 \end{cases}$$

3. Вывести полученные решения СЛАУ.
4. Вывести число итераций, потребовавшихся для нахождения решений.
5. Сравнить скорость сходимости двух методов и сделать соответствующий вывод.

Теоретические сведения:

Итерационные методы решения систем линейных алгебраических уравнений (СЛАУ) являются альтернативой прямым методам. Прямые методы обеспечивают точное решение, но их высокая вычислительная сложность и чувствительность к ошибкам делают их не столь эффективными для больших систем. Итерационные методы, в свою очередь, позволяют достигать приближённых решений с заданной точностью, что особенно удобно для разреженных матриц.

В этой работе анализируются два итерационных метода:

- **Метод Якоби** (простая итерация) - базовый подход, который хорошо работает с симметричными матрицами.
- **Метод верхней релаксации (SOR)** - расширяет метод Зейделя, вводя параметр ускорения ω .

Метод Якоби:

$$x^{(k+1)} = \tilde{A}x^{(k)} + \tilde{b},$$

где $x^{(k+1)}$ — это решение на новом итерационном шаге, а $x^{(k)}$ — это решение с предыдущего итерационного шага, \tilde{b} — вектор с компонентами

$$\tilde{b}_i = \frac{b_i}{a_{ii}},$$

\tilde{A} — матрица с компонентами:

$$\begin{cases} \tilde{a}_{ij} = -\frac{a_{ij}}{a_{ii}}, & i \neq j, \\ \tilde{a}_{ij} = 0, & i = j. \end{cases}$$

По итогу формула метода Якоби будет выглядеть следующим образом:

$$x_i^{(k+1)} = \sum_{j=1}^n \tilde{a}_{ij} x_j^{(k)} + \tilde{b}_i.$$

Метод верхней релаксации (SOR):

$$x_i^{(k+1)} = (1 - \omega)x_j^{(k)} + \omega \left(\sum_{j=i}^n \tilde{a}_{ij} x_j^{(k)} + \sum_{j=1}^i \tilde{a}_{ij} x_j^{(k+1)} + \tilde{b}_i \right),$$

где $x^{(k+1)}$ — это решение на новом итерационном шаге, а $x^{(k)}$ — это решение с предыдущего итерационного шага, ω — это параметр (обычно этот параметр находится в пределах $0 < \omega < 2$, но поскольку в данной лабораторной работе расписан процесс верхней релаксации, то выбираем $1 < \omega < 2$), \tilde{b} — вектор с компонентами

$$\tilde{b}_i = \frac{b_i}{a_{ii}},$$

\tilde{A} — матрица с компонентами:

$$\begin{cases} \tilde{a}_{ij} = -\frac{a_{ij}}{a_{ii}}, & i \neq j, \\ \tilde{a}_{ij} = 0, & i = j. \end{cases}$$

Код программы:

```
import numpy as np

def jacobi_method(A, b, tol=1e-4, max_iterations=1000):
    n = len(b)
    x = np.zeros(n)
    x_new = np.zeros(n)
    iterations = 0

    while True:
        for i in range(n):
            x_new[i] = (b[i] - sum(A[i, j] * x[j] for j in range(n) if j != i)) / A[i, i]

        iterations += 1
        print(f"Итерация {iterations}: {x_new}")

        if np.linalg.norm(x_new - x, ord=np.inf) < tol or iterations >= max_iterations:
            break
        x = x_new.copy()

    return x_new, iterations

def sor_method(A, b, omega, tol=1e-4, max_iterations=1000):
    n = len(b)
    x = np.zeros(n)
    iterations = 0

    while True:
        x_old = x.copy()
        for i in range(n):
            sigma = sum(A[i, j] * x[j] for j in range(i)) + sum(A[i, j] * x_old[j] for j in range(i + 1, n))
            x[i] = (1 - omega) * x_old[i] + (omega / A[i, i]) * (b[i] - sigma)

        iterations += 1
        print(f"Итерация {iterations}: {x}")

        if np.linalg.norm(x - x_old, ord=np.inf) < tol or iterations >= max_iterations:
            break

    return x, iterations

def main():
    A = np.array([[6.22, 1.42, -1.72, 1.91],
                  [1.42, 5.33, 1.11, -1.82],
                  [-1.72, 1.11, 5.24, 1.42],
                  [1.91, -1.82, 1.42, 6.55]])
    b = np.array([7.53, 6.06, 8.05, 8.06])
    tol = 1e-4

    print("Решение методом Якоби:")
    jacobi_solution, jacobi_iterations = jacobi_method(A, b, tol)
    print(f"Решение: {jacobi_solution}")
    print(f"Количество итераций: {jacobi_iterations}")

    omega = 1.1
    print("\nРешение методом верхней релаксации (SOR):")
    sor_solution, sor_iterations = sor_method(A, b, omega, tol)
    print(f"Решение: {sor_solution}")
    print(f"Количество итераций: {sor_iterations}")

    print("\nСравнение сходимости:")
    print(f"Метод Якоби: {jacobi_iterations} итераций")
    print(f"Метод верхней релаксации: {sor_iterations} итераций")

if __name__ == "__main__":
    main()
```

Описание:

Программа предназначена для численного решения систем линейных алгебраических уравнений (СЛАУ) в виде $Ax = b$ с использованием итерационных методов. Она осуществляет сравнение скорости сходимости и точности двух методов: метода Якоби и метода верхней релаксации (SOR). Оба метода предоставляют приближённое решение вектора x с заданной точностью $\epsilon = 10^{-4}$.

Основные этапы работы программы:

1. Функция `jacobi_method`:

```
def jacobi_method(A, b, tol=1e-4, max_iterations=1000)
```

Функция реализует метод Якоби для решения системы линейных алгебраических уравнений (СЛАУ) вида $Ax = b$.

1. Функция инициализирует вектор решений с нулевыми значениями и устанавливает счетчик итераций.
2. В каждом цикле происходит вычисление новых значений для вектора x_{new} по формуле Якоби, которая основана на предыдущих значениях вектора x .
3. Итерации продолжаются до тех пор, пока не будет достигнута заданная точность или предельное количество итераций.
4. `np.linalg.norm` используется для определения, насколько близки два последовательных вектора решений.

Параметры:

- A: Квадратная матрица коэффициентов (размерность $n \times n$).
- b: Вектор свободных членов (размерность n).
- tol: Допустимая точность решения (по умолчанию $\epsilon = 10^{-4}$).
- max_iterations: Максимальное количество итераций (по умолчанию 1000).

Возвращает:

- x_{new} : Вектор, представляющий приближенное решение системы $Ax = b$.
- iterations: Количество итераций, выполненных для достижения результата.

2. Функция sor_method:

```
def sor_method(A, b, omega, tol=1e-4, max_iterations=1000)
```

Функция реализует метод верхней релаксации (Successive Over-Relaxation, SOR) для решения системы линейных алгебраических уравнений (СЛАУ) вида $Ax = b$.

1. Функция инициализирует вектор решений с нулевыми значениями и устанавливает счетчик итераций.
2. В каждом цикле выполняется обновление значений для вектора x с использованием метода SOR. Этот метод сочетает в себе как текущие, так и предыдущие значения для расчета новых.
3. Как и в методе Якоби, итерации продолжаются до достижения заданной точности или предельного числа итераций.
4. np.linalg.norm используется для контроля сходимости между текущим и предыдущим значениями решения.

3. Основная функция main

```
def main():
```

Основная функция, которая служит для инициализации данных, вызова методов и вывода результата.

1. Здесь задаются матрица коэффициентов A и вектор свободных членов b .
2. Устанавливается допустимая точность.
3. Вызываются функции jacobi_method и sor_method, и выводятся результаты: найденные решения и количество итераций для каждого метода.
4. Также выводится сравнение количества итераций для обоих методов, что позволяет оценить их эффективность.

Полученные результаты:

Итерации метода Якоби:

```

Решение методом Якоби:
Итерация 1: [1.21061093 1.1369606 1.53625954 1.230
53435]
Итерация 2: [0.99800028 0.91468235 1.35932553 0.860
38364]
Итерация 3: [1.11348203 0.88177983 1.43693097 0.898
97696]
Итерация 4: [1.13060256 0.84803006 1.47134851 0.839
33536]
Итерация 5: [1.16613926 0.81593581 1.50027993 0.817
50365]
Итерация 6: [1.18817052 0.7929884 1.52465946 0.791
9511 ]
Итерация 7: [1.20799745 0.77331648 1.54367664 0.773
86515]
Итерация 8: [1.22330096 0.75789815 1.55925302 0.758
49466]
Итерация 9: [1.23584807 0.74532872 1.57170769 0.746
37107]
Итерация 10: [1.24588452 0.73525245 1.58177422 0.73
651962]
Итерация 11: [1.25399368 0.72711826 1.58987277 0.72
861078]
Итерация 12: [1.26051875 0.7205707 1.59640087 0.72
223023]
Итерация 13: [1.26577803 0.71529407 1.60165875 0.71
709929]
Итерация 14: [1.27001415 0.71104373 1.60589501 0.71
295324]
Итерация 15: [1.27342711 0.70761939 1.60930767 0.70
961857]
Итерация 16: [1.27617656 0.70486075 1.61205701 0.70
6932 ]
Итерация 17: [1.27839159 0.70263832 1.61427191 0.70
476769]
Итерация 18: [1.28017604 0.7008479 1.61605627 0.70
302408]
Итерация 19: [1.28161363 0.69940551 1.61749378 0.70
16194 ]
Итерация 20: [1.28277177 0.6982435 1.61865187 0.70
048776]
Итерация 21: [1.28370479 0.69730736 1.61958484 0.69
95761 ]
Итерация 22: [1.28445645 0.69655319 1.62033645 0.69
884164]
Итерация 23: [1.285062 0.69594562 1.62094197 0.69
824996]
Итерация 24: [1.28554984 0.69545615 1.62142978 0.69
777329]
Итерация 25: [1.28594285 0.69506183 1.62182277 0.69
738927]
Итерация 26: [1.28625946 0.69474416 1.62213937 0.69
70799 ]
Итерация 27: [1.28651453 0.69448823 1.62239443 0.69
683067]
Итерация 28: [1.28672002 0.69428206 1.62259991 0.69
662988]
Итерация 29: [1.28688557 0.69411596 1.62276544 0.69
646813]
Итерация 30: [1.28701894 0.69398215 1.6228988 0.69
633781]
Итерация 31: [1.28712638 0.69387435 1.62300624 0.69
623283]
Итерация 32: [1.28721293 0.6937875 1.62309279 0.69
614825]
Решение: [1.28721293 0.6937875 1.62309279 0.696148
25]
Количество итераций: 32

```

Итерации метода последовательной верхней релаксации:

```

Решение методом верхней релаксации (SOR):
Итерация 1: [1.33167203 0.86039931 1.97022441 0.719
57037]
Итерация 2: [1.33868224 0.59124229 1.62395295 0.645
67368]
Итерация 3: [1.32520543 0.67367526 1.65653388 0.674
81207]
Итерация 4: [1.30592016 0.67456467 1.63741933 0.682
9144 ]
Итерация 5: [1.29907425 0.68390407 1.63226748 0.688
38324]
Итерация 6: [1.29399913 0.68769177 1.62843739 0.691
53535]
Итерация 7: [1.29132569 0.69015784 1.62634085 0.693
3314 ]
Итерация 8: [1.28972934 0.69153394 1.62511808 0.694
37605]
Итерация 9: [1.28881859 0.69233573 1.62441328 0.694
97686]
Итерация 10: [1.28829099 0.69279729 1.62400661 0.69
532407]
Итерация 11: [1.28798686 0.69306384 1.62377186 0.69
552435]
Итерация 12: [1.28781128 0.69321765 1.62363639 0.69
563996]
Итерация 13: [1.28770995 0.69330642 1.62355821 0.69
570668]
Итерация 14: [1.28765148 0.69335765 1.62351309 0.69
574518]

```

Результаты:

Метода Якоби:

Решение: [1.28721293 0.6937875 1.62309279 0.696148
25]
Количество итераций: 32

Метода последовательной верхней релаксации:

Решение: [1.28765148 0.69335765 1.62351309 0.695745
18]
Количество итераций: 14

Сравнение сходимости:

Метод Якоби: 32 итераций

Метод верхней релаксации: 14 итераций

Выводы:

1. Эффективность методов: Оба метода (Якоби и SOR) показали способность находить приближённые решения системы линейных уравнений с заданной точностью. Однако эффективность этих методов может различаться в зависимости от конкретной системы.

2. Скорость сходимости: Метод верхней релаксации (SOR) продемонстрировал более высокую скорость сходимости по сравнению с методом Якоби. Это связано с тем, что метод SOR использует параметр релаксации, который помогает ускорить процесс приближения к точному решению. В некоторых случаях это может существенно уменьшить количество необходимых итераций.

3. Заключение: Применение итерационных методов, особенно метода SOR, является эффективным инструментом для численного решения СЛАУ. В зависимости от свойств конкретной задачи, выбор между методами Якоби и SOR может быть основан на требуемой скорости сходимости и необходимых ресурсах.