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

ОТЧЁТ по лабораторной работе № 5

«Итерационные методы решения СЛАУ»

Вариант № 8

Выполнила: студент гр. Б9122-02.03.01сцт
Ф.И.О.
Ильяхова Алиса Алексеевна

Проверил: преподаватель
Ф.И.О.
Павленко Елизавета Робертовна

Владивосток

2024

Цели 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 ε = 10^{-4} и вывести полученное решение:
$$\begin{cases} 6.22x_1 + 1.42x_2 - 1.72x_3 + 1.91x_4 = 7.53 \\ 1.42x_1 + 5.33x_2 + 1.11x_3 - 1.82x_4 = 6.06 \\ -1.72x_1 + 1.11x_2 + 5.24x_3 + 1.42x_4 = 8.05 \\ 1.91x_1 - 1.82x_2 + 1.42x_3 + 6.55x_4 = 8.06 \end{cases}$$
3. Вывести число итераций, потребовавшихся для нахождения решений.
4. Сравнить скорость сходимости двух методов и сделать соответствующий вывод.

Основное:

1) Методы:

Метод Якоби:

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

где $\mathrm{x}^{(k+1)}$ — 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:

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

По итогу формула метода Якоби будет выглядеть следующим образом:

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

Метод последовательной верхней релаксации:

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

где $\mathrm{x}^{(k+1)}$ — 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:

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

2) Работа функций и основного кода:

Функция **jacobi**:

1. Определение размера: n = len(b)
   Получаем размерность вектора “b”, который соответствует количеству уравнений.
2. Инициализация начального приближения: if x0 is None: x0 = np.zeros(n) x = np.copy(x0)
   Если начальное приближение не задано, создается нулевой вектор размерности \(n\). Затем это значение копируется в переменную “x”.
3. Основной цикл итераций: for iteration in range(max_iterations)
   Начинается цикл, который будет выполняться до достижения максимального числа итераций.
4. Создание нового вектора: x_new = np.zeros(n)
   Создается новый вектор “x_new”, который будет содержать значения на текущей итерации.
5. Вычисление нового значения для каждого элемента: for i in range(n): sum_ = b[i] for j in range(n): if i != j: sum_ -= A[i][j] * x[j] x_new[i] = sum_ / A[i][i]
   Для каждого элемента “i” вектора “x_new” вычисляется новое значение. Сначала к “sum_” добавляется соответствующий элемент из вектора “b”. Затем из него вычитаются произведения элементов матрицы “A” и предыдущих значений вектора “x”, кроме самого элемента “i”. Наконец, результат делится на диагональный элемент матрицы “A”.
6. Вывод текущего значения вектора: print(f"Итерация {iteration + 1}: {x_new}")
7. Проверка на сходимость: if np.linalg.norm(x_new - x, ord=np.inf) < tol: return x_new
   Вычисляется норма разности между новым и старым вектором. Если она меньше заданной точности, метод завершает работу и возвращает результат.
8. Обновление значения вектора: x = x_new
9. Ошибка при превышении максимального числа итераций: raise ValueError("Метод не сошелся за заданное количество итераций.")

Функция **sor**:

1. Определение размера: n = len(b)
2. Инициализация начального приближения: if x0 is None: x0 = np.zeros(n) x = np.copy(x0)
3. Основной цикл итераций: for iteration in range(max_iterations):
4. Создание нового вектора: x_new = np.copy(x)
5. Вычисление нового значения для каждого элемента: for i in range(n): sum_ = b[i] for j in range(n): if j != i: sum_ -= A[i][j] * x_new[j] x_new[i] = (1 - omega) * x[i] + (omega / A[i][i]) * sum_
   Подобно методу Якоби, но здесь используется формула релаксации для обновления значений. Это позволяет учитывать как старое значение, так и новое значение для ускорения сходимости.
6. Вывод текущего значения вектора: print(f"Итерация {iteration + 1}: {x_new}")

7. Проверка на сходимость: if np.linalg.norm(x_new - x, ord=np.inf) < tol: return x_new
8. Обновление значения вектора: x = x_new
9. Ошибка при превышении максимального числа итераций: raise ValueError("Метод не сошелся за заданное количество итераций.")

Главная программа:

if __name__ == "__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]) omega = 1.25 print("Итерации метода Якоби:") j = jacobi(A, b) print("\nИтерации метода последовательной верхней релаксации:") s = sor(A, b, omega) print("\nМетод Якоби:", j) print("Метод последовательной верхней релаксации:", s)

Здесь задается система линейных уравнений через матрицу “A” и вектор “b”. Вызываются функции **jacobi** и **sor**, чтобы найти решение системы. Выводятся результаты и итерации для каждого метода. Таким образом, программа реализует два метода численного решения систем линейных уравнений и предоставляет информацию о процессе вычисления на каждой итерации.

3) Полученные результаты:

Итерации метода Якоби:

```
Итерации метода Якоби:
Итерация 1: [1.21061093 1.1369606  1.53625954 1.23053435]
Итерация 2: [0.99800028 0.91468235 1.35932553 0.86038364]
Итерация 3: [1.11348203 0.88177983 1.43693097 0.89897696]
Итерация 4: [1.13060256 0.84803006 1.47134851 0.83933536]
Итерация 5: [1.16613926 0.81593581 1.50027993 0.81750365]
Итерация 6: [1.18817052 0.7929884  1.52465946 0.7919511 ]
Итерация 7: [1.20799745 0.77331648 1.54367664 0.77386515]
Итерация 8: [1.22330096 0.75789815 1.55925302 0.75849466]
Итерация 9: [1.23584807 0.74532872 1.57170769 0.74637107]
Итерация 10: [1.24588452 0.73525245 1.58177422 0.73651962]
Итерация 11: [1.25399368 0.72711826 1.58987277 0.72861078]
Итерация 12: [1.26051875 0.7205707  1.59640087 0.72223023]
Итерация 13: [1.26577803 0.71529407 1.60165875 0.71709292]
Итерация 14: [1.27001415 0.71104373 1.60589501 0.71295324]
Итерация 15: [1.27342711 0.70761939 1.60930767 0.70961857]
Итерация 16: [1.27617656 0.70486075 1.61205701 0.706932  ]
```

```
Итерация 17: [1.27839159 0.70263832 1.61427191 0.70476769]
Итерация 18: [1.28017604 0.7008479  1.61605627 0.70302408]
Итерация 19: [1.28161363 0.69940551 1.61749378 0.7016194 ]
Итерация 20: [1.28277177 0.6982435  1.61865187 0.70048776]
Итерация 21: [1.28370479 0.69730736 1.61958484 0.6995761 ]
Итерация 22: [1.28445645 0.69655319 1.62033645 0.69884164]
Итерация 23: [1.285062   0.69594562 1.62094197 0.69824996]
Итерация 24: [1.28554984 0.69545615 1.62142978 0.69777329]
Итерация 25: [1.28594285 0.69506183 1.62182277 0.69738927]
Итерация 26: [1.28625946 0.69474416 1.62213937 0.6970799 ]
Итерация 27: [1.28651453 0.69448823 1.62239443 0.69683067]
Итерация 28: [1.28672002 0.69428206 1.62259991 0.69662988]
Итерация 29: [1.28688557 0.69411596 1.62276544 0.69646813]
Итерация 30: [1.28701894 0.69398215 1.6228988  0.69633781]
Итерация 31: [1.28712638 0.69387435 1.62300624 0.69623283]
Итерация 32: [1.28721293 0.6937875  1.62309279 0.69614825]
```

Итерации метода последовательной верхней релаксации:

```
Итерации метода последовательной верхней релаксации:
Итерация 1: [1.51326367 0.91725272 2.29834518 0.68233133]
Итерация 2: [1.40572789 0.41668735 1.58104789 0.5714673 ]
Итерация 3: [1.370072   0.6931089  1.71010286 0.67321611]
Итерация 4: [1.30565714 0.65528891 1.62695728 0.68065424]
Итерация 5: [1.30095844 0.69112786 1.63380638 0.69109923]
Итерация 6: [1.29026398 0.68840487 1.62488899 0.69385693]
Итерация 7: [1.28957376 0.69281391 1.62473352 0.69499261]
Итерация 8: [1.28799845 0.69276147 1.62375522 0.69552979]
Итерация 9: [1.28786288 0.69330368 1.62361863 0.69567025]
Итерация 10: [1.28764092 0.69333756 1.62350515 0.69575856]
Итерация 11: [1.28761362 0.69340541 1.62347444 0.69577832]
```

Результаты:

```
Метод Якоби: [1.28721293 0.6937875  1.62309279 0.69614825]
Метод последовательной верхней релаксации: [1.28761362 0.69340541 1.62347444 0.69577832]
```

Метод последовательной верхней релаксации показал более быструю сходимость, справился за меньшее количество итераций.

4) Вывод:

1. Метод последовательной верхней релаксации является более эффективным для задач, где важна скорость решения, благодаря использованию параметра релаксации, ускоряющего процесс сходимости (11 итераций с omega = 1.25).
2. Метод Якоби остается хорошим выбором для задач, где требуется простота реализации (32 итерации).
3. Сходимость обоих методов зависит от свойств матрицы A. Для метода Якоби важно наличие диагональной доминантности, а для метода последовательной верхней релаксации необходима оптимизация параметра omega.
   Оба метода предоставили решения с одинаковой точностью, удовлетворяющей заданному порогу $\varepsilon = 10^{-4}$. Однако результаты, полученные методами, были очень близки друг к другу.

5) Листинг программы:

```
import numpy as np

def jacobi(A, b, x0=None, tol=1e-4, max_iterations=100):
    n = len(b)
    if x0 is None:
        x0 = np.zeros(n)
    x = np.copy(x0)

    for iteration in range(max_iterations):
        x_new = np.zeros(n)
        for i in range(n):
            sum_ = b[i]
            for j in range(n):
                if i != j:
                    sum_ -= A[i][j] * x[j]
            x_new[i] = sum_ / A[i][i]

        #вывод текущего значения вектора
        print(f"Итерация {iteration + 1}: {x_new}")

        #проверка на сходимость
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            return x_new
        x = x_new

    raise ValueError("Метод не сошелся за заданное количество
итераций.")


def sor(A, b, omega=1.25, x0=None, tol=1e-4, max_iterations=100):
    n = len(b)
    if x0 is None:
        x0 = np.zeros(n)
    x = np.copy(x0)

    for iteration in range(max_iterations):
        x_new = np.copy(x)
        for i in range(n):
            sum_ = b[i]
            for j in range(n):
```

```
                if j != i:
                    sum_ -= A[i][j] * x_new[j]
            x_new[i] = (1 - omega) * x[i] + (omega / A[i][i]) *
sum_

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

        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            return x_new
        x = x_new

    raise ValueError("Метод не сошелся за заданное количество
итераций.")


if __name__ == "__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])
    omega = 1.25

    print("Итерации метода Якоби:")
    j = jacobi(A, b)

    print("\nИтерации метода последовательной верхней релаксации:")
    s = sor(A, b, omega)

    print("\nМетод Якоби:", j)
    print("Метод последовательной верхней релаксации:", s)
```