

Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №1

Вариант: Метод Гаусса-Зейделя

Выполнил:

Совенко Е.В.

P32212

Преподаватель:

Перл Ольга

Вячеславовна

Санкт-Петербург, 2023 г.

Описание метода

Метод Зейделя представляет собой некоторую модификацию метода простой итерации. Основная его идея заключается в том, что при вычислении $(k+1)$ -го приближения неизвестной x_i учитываются уже вычисленные ранее $(k+1)$ -е приближения неизвестных x_1, x_2, \dots ,

$$\begin{cases} x_1^{(k+1)} = \alpha_{12}x_2^{(k)} + \alpha_{13}x_3^{(k)} + \dots + \alpha_{1n}x_n^{(k)} + \beta_1, \\ x_2^{(k+1)} = \alpha_{21}x_1^{(k+1)} + \alpha_{23}x_3^{(k)} + \dots + \alpha_{2n}x_n^{(k)} + \beta_2, \\ \dots \\ x_n^{(k+1)} = \alpha_{n1}x_1^{(k+1)} + \alpha_{n2}x_2^{(k+1)} + \dots + \alpha_{n,n-1}x_{n-1}^{(k+1)} + \beta_n. \end{cases}$$

Условием выхода из итерационного процесса является выполнение условия:

$$\left| x_i^{(k+1)} - x_i^{(k)} \right| \leq \varepsilon,$$

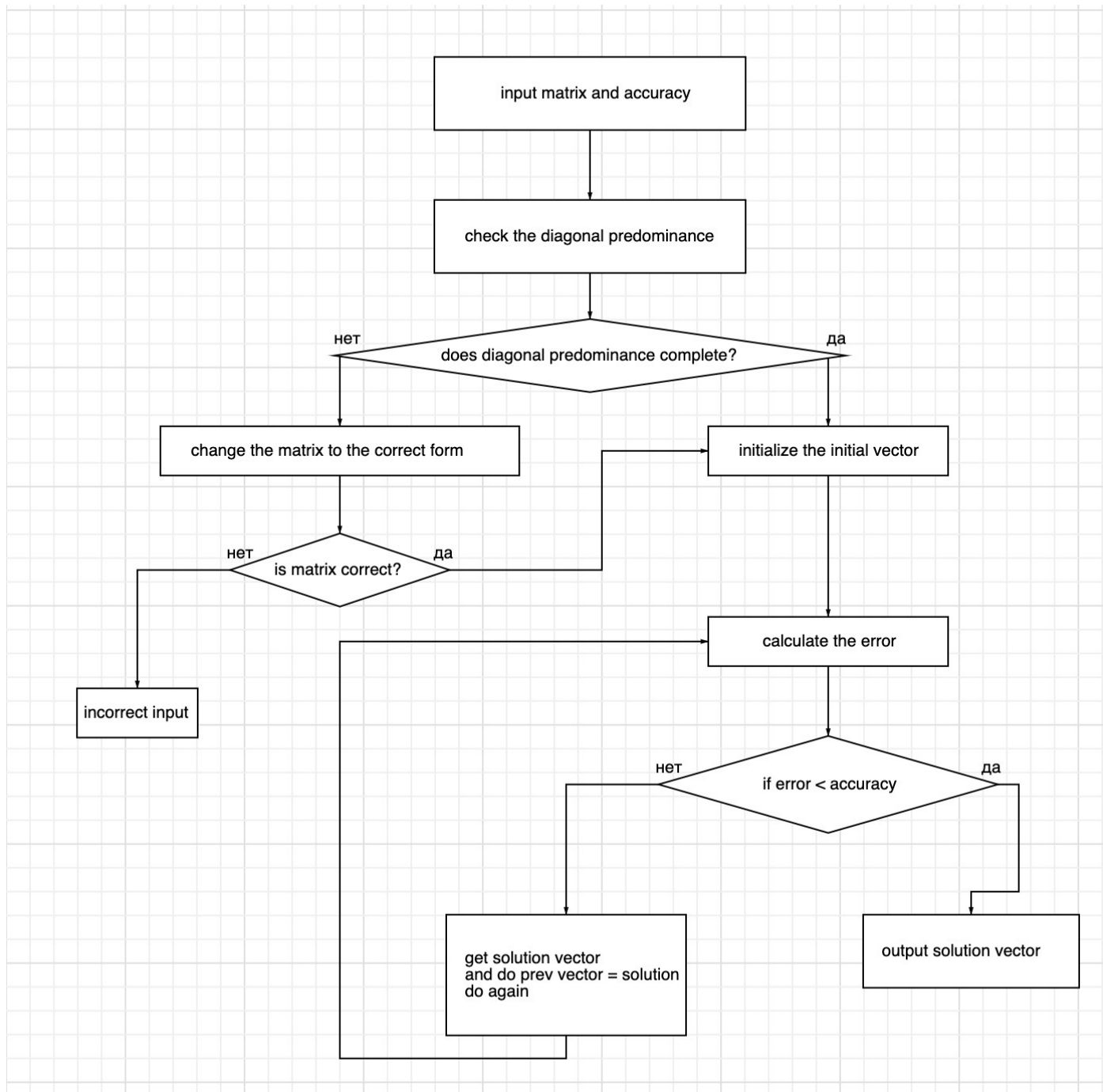
Причем, такая пара $(x(k+1), x(k))$, где их разность максимальна

Код программы

https://github.com/ooowler/Computational_Mathematics_2_course/tree/main/CM_lab

1

Блок схема



Функция, реализовывающая сам метод

Для квадратной матрицы ($N \times N$)

```
def __solve_square_matrix(self, matrix: Matrix, eps: float) -> Optional[list]:
    self.__check_square_matrix()
    self.eps = eps

    n = len(matrix.A)
    init_vector = [self.matrix.B[i] / self.matrix.A[i][i] for i in range(n)]
    prev_vector = init_vector[:]
    error = eps + 1
    iteration = 1
    solution_vector = []

    while error > eps:
        solution_vector = []
        for i in range(n):
            normalize = self.matrix.A[i][i]
            new_value = self.matrix.B[i] / normalize

            for k in range(i):
                new_value -= solution_vector[k] * (self.matrix.A[i][k] / normalize)

            for k in range(i + 1, n):
                new_value -= prev_vector[k] * (self.matrix.A[i][k] / normalize)

            solution_vector.append(new_value)

        new_error = max(abs(solution_vector[i] - prev_vector[i]) for i in range(n))
        if iteration > 5 and new_error > error:
            print(
                f'{"-" * 10}\nDoes not converge\nThe error increased by {round(100 * new_error / error, 1)}%\n{"-" * 10}')

            self.solution_vector = None
            return

        error = new_error

        prev_vector = solution_vector[:]
        iteration += 1

    print('SUCCESS!')
    print(f'Total count of iteration is: {iteration}')

    self.solution_vector = solution_vector[:]
    return solution_vector
```

Для не квадратной матрицы ($M \times N$), где $M \neq N$

```

def __solve_not_square_matrix(self, matrix: Matrix, eps: float) -> Optional[list]:
    rows = len(matrix.A)
    columns = len(matrix.A[0])
    if columns > rows:
        print('Probably inf solutions')
        return

    sub_A = self.matrix.A[:columns]
    self.matrix = Matrix(sub_A, self.matrix.B)
    self.__check_square_matrix()
    res = self.__solve_square_matrix(self.matrix, 10e-6)
    if not res:
        print('Can not solve matrix')
        return

    self.solution_vector = res[:]
    self.matrix = matrix

    if self.is_solution_correct():
        print(f'{"-" * 10}')
        print(f'Solution correct also for your {rows}x{columns} matrix!')
        print(f'{"-" * 10}')
        return self.solution_vector
    else:
        self.solution_vector = None
        print('Can not solve matrix')
        return

```

Проверка решения, путем подставления значений

```
def is_solution_correct(self) -> Optional[bool]:
    if not self.matrix.A:
        print('Matrix is None')
        return

    if not self.eps:
        print('please, solve the matrix firstly')
        return

    for i in range(len(self.matrix.A)):
        A_sum = 0
        sum_error = 0

        for j in range(len(self.matrix.A[i])):
            A_sum += self.matrix.A[i][j] * self.solution_vector[j]
            sum_error += abs(self.matrix.A[i][j]) * self.eps

        if not (self.matrix.B[i] - sum_error <= A_sum <= self.matrix.B[i] + sum_error):
            print('Solution is incorrect!')
            return False

    print('Solution is correct!')
    print(f'your accuracy is {self.eps}')
    return True
```


Пример работы программы

Входные данные:

Инициализация матрицы, удовлетворяющей условию
применения метода Гаусса-Зейделя

Результат работы:

```
1  from matrix import Matrix
2  from solve_matrix import Solution
3
4  matrix = Matrix([[10, 2, 3, 4], # sum 19
5                  [1, 15, 3, 5], # sum 24
6                  [-3, -2, -12, -2], # sum -19
7                  [-1, -2, -3, 8]], # sum 2
8
9                  [19, 24, -19, 2]) # solutions must be: [1, 1, 1, 1]
10
11  solution = Solution(matrix, 1e-7)
12
13  solution.solve_matrix()
14  print()
15  solution.print_solution_vector(accuracy=5)
16  print()
17  solution.is_solution_correct()
18
```

Run:  for_presentation x

```

/Users/evgeniy/Library/Caches/pypoetry/virtualenvs/cm-lab1-PeJ8onqB-py
SUCCESS!
Total count of iteration is: 11

x1: 1.0
x2: 1.0
x3: 1.0
x4: 1.0

Solution is correct!
your accuracy is 1e-07

Process finished with exit code 0
```

Вывод

Во время выполнения лабораторной работы я изучил работу метода Гаусса — Зейделя решения системы линейных уравнений. Обычно этот метод дает лучшую сходимость, чем метод простых итераций, потому что для своих вычислений учитывает уже вычисленные значения того же номера приближения. Но бывают случаи, что метод Гаусса — Зейделя расходится, когда сходится метод простых итераций, поэтому оба метода важны и нельзя выбрать однозначно, но благодаря более быстрому нахождению решения – метод Гаусса-Зейделя, все-таки, применяется чаще.

В качестве нулевых приближений можно брать любые значения, так как это не влияет на сходимость ряда, где сходимость ряда обеспечивается преобладанием диагональных элементов.

Итерационные методы отлично показывают себя при решении матриц больших размеров, где решения систем уравнения методом Гаусса занимало бы много места и времени. Мы можем решать систему уравнения с точностью, которую мы зададим, что делает метод Гаусса-Зейделя способным к получения точных решений.