Лабораторная работа No3 «Реализация метода Гаусса с перестановками»

Цель работы

Реализовать три варианта метода Гаусса с перестановками и научиться оценивать погрешности решения системы уравнения для матриц произвольной размерности.

Задание

1. Реализовать метод Гаусса с перестановками по столбцам, по строкам, по столбцам и строкам одновременно для действительных квадратных матриц произвольной размерности n.
2. Для проверки работоспособности алгоритмов необходимо использовать алгоритм тестирования задачи написанный в лабораторной работе No2 «Реализация метода Гаусса», который заключался в том, что мы заведомо определяем значения координат вектора x, данный вектор заведомо является решением уравнения $A \cdot x = b$, вычисляем b путем прямого перемножения матрицы A на вектор x и далее производим поиск решения уравнения $A \cdot x = b$ тем или иным методом Гаусса, получая xчисл, после чего производим сравнение полученного xчисл с заданным x, а также решением xбибл , полученным с использованием сторонней библиотеки выбранной студеном. При этом сравнение производится по евклидовой норме разности вектора (x- xчисл) и (x - xбибл).
3. Важно: на очной защите лабораторной работы студент должен показать умение оценивать погрешность вычислений в зависимости от диагонального преобладания матрицы. Сравнивать погрешности вычислений классическим методом Гаусса, методом с перестановкой строк, перестановкой столбцов и перестановкой строк и столбцов одновременно. Понимать связь теории с практикой. В случае, если студент не показывает такое умение или не сдает очно, то за лабораторную работу снимается один балл.

```
from copy import deepcopy

import numpy as np

from lab2 import prepare_to_gauss, gauss_method, calc_evklid_diff, increase_diag
from my_matrix import MyMatrix


def swap_rows(m, column):
    max_el = m[column][column]
```

```
        max_row = column
        for i in range(column + 1, len(m)):
            if abs(m[i][column]) > abs(max_el):
                max_el = m[i][column]
                max_row = i
        if max_row != column:
            m[column], m[max_row] = m[max_row], m[column]


    def gauss_with_column_permut(m):
        n = len(m)
        x = [0 for i in range(n)]

        for k in range(n - 1):
            swap_rows(m, k)
            for i in range(k + 1, n):
                div = m[i][k] / m[k][k]
                m[i][-1] -= div * m[k][-1]
                for j in range(k, n):
                    m[i][j] -= div * m[k][j]

        for k in range(n - 1, -1, -1):
            x[k] = (m[k][-1] - sum([m[k][j] * x[j] for j in range(k + 1, n)])) / m[k][k

        return x


    def swap_colums(m, row, swap_row):
        max_el = m[row][row]
        max_column = row
        for i in range(row + 1, len(m)):
            if abs(m[row][i]) > abs(max_el):
                max_el = m[row][i]
                max_column = i
        if max_column != row:
            for i in range(0, len(m)):
                m[i][max_column], m[i][row] = m[i][row], m[i][max_column]

        return max_column, row


    def get_key(mass, val):
        for i in range(0, len(mass)):
            if val == mass[i]:
                return i

        return "key doesn't exist"


    def make_swap(swapped_rows, vect):
        new_vect = []
        for i in range(0, len(swapped_rows)):
            ind = get_key(swapped_rows, i)
            new_vect.append(vect[ind])

        return new_vect
```

```
    return new_vect


def gauss_with_row_permut(m):
    swaped_rows = []
    for i in range(0, len(m)):
        swaped_rows.append(i)
    n = len(m)
    x = [0 for i in range(n)]

    for k in range(n - 1):
        max_column, row = swap_colums(m, k, swaped_rows)
        swaped_rows[max_column], swaped_rows[row] = swaped_rows[row], swaped_rows[m
        for i in range(k + 1, n):
            div = m[i][k] / m[k][k]
            m[i][-1] -= div * m[k][-1]
            for j in range(k, n):
                m[i][j] -= div * m[k][j]

    for k in range(n - 1, -1, -1):
        x[k] = (m[k][-1] - sum([m[k][j] * x[j] for j in range(k + 1, n)])) / m[k][k

    x = make_swap(swaped_rows, x)

    return x


def find_swap_on_matrix(m, column, row):
    max_el_row = m[column][column]
    max_row = column
    for i in range(column + 1, len(m)):
        if abs(m[i][column]) > abs(max_el_row):
            max_el_row = m[i][column]
            max_row = i

    max_el_column = m[row][row]
    max_column = row
    for i in range(row + 1, len(m)):
        if abs(m[row][i]) > abs(max_el_column):
            max_el_column = m[row][i]
            max_column = i

    if abs(max_el_row) >= abs(max_el_column) and max_row != column:
        m[column], m[max_row] = m[max_row], m[column]
        return column, max_row
    elif abs(max_el_column) > abs(max_el_row) and max_column != row:
        for i in range(0, len(m)):
            m[i][max_column], m[i][row] = m[i][row], m[i][max_column]
        return max_column, row
    return column, row


def gauss_with_all_permut(m):
    swaped_rows = []
    for i in range(0, len(m)):
        swaped_rows.append(i)
```

```
            swaped_rows.append(i)
        n = len(m)
        x = [0 for i in range(n)]

        for k in range(n - 1):
            col, row = find_swap_on_matrix(m, k, k)
            swaped_rows[col], swaped_rows[row] = swaped_rows[row], swaped_rows[col]
            for i in range(k + 1, n):
                div = m[i][k] / m[k][k]
                m[i][-1] -= div * m[k][-1]
                for j in range(k, n):
                    m[i][j] -= div * m[k][j]

        for k in range(n - 1, -1, -1):
            x[k] = (m[k][-1] - sum([m[k][j] * x[j] for j in range(k + 1, n)])) / m[k][k

        return make_swap(swaped_rows, x)


    def testing_algo(m, x,func):
        a = MyMatrix(
            deepcopy(m)
        )
        calc_b = a.mult_on_vector(x)

        a_prepared = prepare_to_gauss(deepcopy(a.matrix), calc_b)
        calc_x = func(a_prepared)

        a_numpy = np.array(deepcopy(m))
        x_numpy = np.array(x)
        b_numpy = np.matmul(a_numpy, x_numpy)
        x_numpy_calc = np.linalg.solve(a_numpy, b_numpy)

        evkl_diff = calc_evklid_diff(x, calc_x)
        print(func)
        print("Evklid difference between calculated x and given x: %.30f" % evkl_diff)
        print()
        evkl_diff_numpy = calc_evklid_diff(calc_x, x_numpy_calc)
        print("Evklid difference between calculated x and numpy x: %.30f" % evkl_diff_n
        print("-----")


    if __name__ == '__main__':
        # a = MyMatrix([
        #     [1, 1, 1, 4, 1],
        #     [2, -3, 4, 5, 5],
        #     [3, 4, 5, 10, 10],
        #     [3, 6, 7, 8, 8],
        #     [3, 3, 3, 3, 6]
        # ])
        m = [
            [1, 1, 1, 4, 1],
            [2, -3, 4, 5, 5],
            [3, 4, 5, 10, 10],
            [3, 6, 7, 8, 8],
            [3, 3, 3, 3, 6]
```

```
]

x = [1.0, 3.0, 5.0, 7.0, 9.0]
# calc_b = a.mult_on_vector(x)
#
# a_prepared = prepare_to_gauss(deepcopy(a.matrix), calc_b)
# calc_x = gauss_method(a_prepared)
testing_algo(m, x, gauss_method)
testing_algo(m, x, gauss_with_column_permut)
testing_algo(m, x, gauss_with_row_permut)
testing_algo(m, x, gauss_with_all_permut)
# print(calc_x)
# b_prepared = prepare_to_gauss(deepcopy(a.matrix), calc_b)
# calc_xx = gauss_with_column_permut(b_prepared)
# print(calc_xx)
# c_prepared = prepare_to_gauss(deepcopy(a.matrix), calc_b)
# calc_xxx = gauss_with_row_permut(c_prepared)
# print(calc_xxx)
# d_prepared = prepare_to_gauss(deepcopy(a.matrix), calc_b)
# calc_xxxx = gauss_with_all_permut(d_prepared)
# print(calc_xxxx)
```

```
<function gauss_method at 0x7fbaf88b5cb0>
Evklid difference between calculated x and given x: 0.00000000000031351565956

Evklid difference between calculated x and numpy x: 0.00000000000026472941518
-----
<function gauss_with_column_permut at 0x7fbaf893a320>
Evklid difference between calculated x and given x: 0.00000000000001538370149

Evklid difference between calculated x and numpy x: 0.00000000000004773959005
-----
<function gauss_with_row_permut at 0x7fbaf893a8c0>
Evklid difference between calculated x and given x: 0.00000000000003076740298

Evklid difference between calculated x and numpy x: 0.00000000000006959070628
-----
<function gauss_with_all_permut at 0x7fbaf893ac20>
Evklid difference between calculated x and given x: 0.00000000000003076740298

Evklid difference between calculated x and numpy x: 0.00000000000006959070628
-----
```

Платные продукты Colab - Отменить подписку