Лабораторная работа №1
Алгоритм обращения матрицы

9 февраля 2021 г.

Выполнил: Совпель Дмитрий, гр. 853501

1 Реализация метода

Импортируем необходимые библиотеки.

[8]:
```
import warnings
import typing as tp

import numpy as np
import scipy.linalg as sla
```

Реализуем функции, необходимые для построения алгоритмы `compute_alpha` и `compute_inverse_step`. Затем реализуем сам алгоритм обращения матрицы `inverse`.

[72]:
```
def compute_alpha(inverse_matrix: np.array, column: np.array, column_no: int) ->␣
 ↪float:
    return inverse_matrix[column_no] @ column

def compute_inverse_step(previous_inverse_matrix: np.array, column: np.array,␣
 ↪column_no: int) -> np.array:
    d = previous_inverse_matrix @ column
    d_k = d[column_no]
    d[column_no] = -1
    d /= -d_k

    D = np.eye(previous_inverse_matrix.shape[0])
    D[:, column_no] = d

    return D @ previous_inverse_matrix

def inverse(matrix: np.array, verbose: bool = False) -> np.array:
    current_inverse_matrix = np.eye(matrix.shape[0])
    iteration = 0

    indexes_set = set(range(matrix.shape[0]))
```

```
    indexes_order = [0] * matrix.shape[0]
    if verbose:
        print(f'Initial unused indexes set {set([element + 1 for element in␣
↪indexes_set])}')
        print(f'Initial matrix \n {current_inverse_matrix}')

    while iteration < matrix.shape[0]:
        is_singular = True
        for current_column_no in indexes_set:
            current_column = matrix[: ,current_column_no]
            alpha = compute_alpha(current_inverse_matrix, current_column,␣
↪iteration)
            if not np.isclose(alpha, 0.0, rtol=1e-11):
                is_singular = False
                indexes_set.remove(current_column_no)
                indexes_order[current_column_no] = iteration
                break
        if is_singular:
            warnings.warn("Singular matrix", sla.LinAlgWarning)
            return None
        current_inverse_matrix = compute_inverse_step(current_inverse_matrix,␣
↪current_column, iteration)
        iteration += 1

        if verbose:
            print(f'Current iteration no. {iteration}')
            print(f'Current indexes set {set([element + 1 for element in␣
↪indexes_set]) if indexes_set else "{}"}')
            print(f'Current inverse matrix \n {current_inverse_matrix}')
    return current_inverse_matrix[np.array(indexes_order), :]
```

Для тестирования используем вспомогательную функцию `test`. Функция проверяет условие $|AA^{-1} - E|_F < \varepsilon$ (при этом по умолчанию $\varepsilon = 0.00001$), и если оно не соблюдается выдает ошибку.

[33]:
```
def test(matrix: np.array, inverse_matrix: np.array, eps = 1e-5) -> None:
    if inverse_matrix is not None:
        assert sla.norm(matrix @ inverse_matrix - np.eye(matrix.shape[0])) < eps
```

2 Тестирование алгоритма на матрице A

Матрица A имеет вид:

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Как видно, алгоритм корректно отрабатывает данный тестовый случай.

[11]:
```
matrix = np.array([[0, 0, 0, 1], [0, 0, 1, 0], [0, 1, 0, 0], [1, 0, 0, 0]])
inverse_matrix = inverse(matrix)
test(matrix, inverse_matrix)
```

3 Тестирование алгоритма на матрице B_ε для иследования точности метода

Рассмотрим матрицы вида B_{eps}

$$B_{eps} = \begin{pmatrix} 2 & 8 & -1 & 4 & 5 & 6 \\ 1 & -9 & 2 & -3 & 1 & -2 \\ 3 & -1 & 1 & 1+eps & 6 & 4 \\ 0 & 1 & 0 & 1 & 0 & 2 \\ 1 & 2 & -1 & 4 & 2 & 3 \\ -3 & 2 & 1 & 0 & 0 & 0 \end{pmatrix}$$

для различных $eps = \{1, 0.1, 0.0000001, 0.000000001, 0.000000000000001\}$.

[15]:
```
B = lambda eps: np.array([[2, 8, -1, 4, 5, 6],
                          [1, -9, 2, -3, 1, -2],
                          [3, -1, 1, 1 + eps, 6, 4],
                          [0, 1, 0, 1, 0, 2],
                          [1, 2, -1, 4, 2, 3],
                          [-3, 2, 1, 0, 0, 0]])
epses = [1, 0.1, 1e-6, 1e-9, 1e-15]
```

3.1 eps = 1

Рассмотрим B_{eps} при $eps = 1$. Как видно, данную ситуацию алгоритм отрабатывает корректно.

[36]:
```
matrix = B(epses[0])
inverse_matrix = inverse(matrix)
test(matrix, inverse_matrix)
```

3.2 eps = 0.1

Рассмотрим B_{eps} при $eps = 0.1$. Как видно, данную ситуацию алгоритм также отрабатывает корректно.

[22]:
```
matrix = B(epses[1])
inverse_matrix = inverse(matrix)
test(matrix, inverse_matrix)
```

3.3 eps = 0.000001

Рассмотрим B_{eps} при $eps = 0.000001$. В данном случае, условие $|BB^{-1} - E|_F < 0.00001$ не соблюдается. Попробуем условие $|BB^{-1} - E|_F < 0.1$.

[35]:
```
matrix = B(epses[2])
inverse_matrix = inverse(matrix)
test(matrix, inverse_matrix)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-35-af6f3b8d0591> in <module>
      1 matrix = B(epses[2])
      2 inverse_matrix = inverse(matrix)
----> 3 test(matrix, inverse_matrix)

<ipython-input-33-327f4077c160> in test(matrix, inverse_matrix, eps)
      1 def test(matrix: np.array, inverse_matrix: np.array, eps = 1e-5) -> None:
      2     if inverse_matrix is not None:
----> 3         assert sla.norm(matrix @ inverse_matrix - np.eye(matrix.shape[0]))␣
 ↪< eps

AssertionError:
```

Как видно, условие $|BB^{-1} - E|_F < 0.1$ соблюдается. Для сравнения посмотрим, как справляется с этим тестовым случаем встроенная функция библиотеки `scipy`.

[37]:
```
matrix = B(epses[2])
inverse_matrix = inverse(matrix)
test(matrix, inverse_matrix, 0.1)
```

Как видно, встроенная функция `inv` справляется с данным тестовым случаем (то есть точность метода, заложенного в функцию `inv` выше, чем метода, реализованного рассматриваемым алгоритмом).

[42]:
```
matrix = B(epses[2])
inverse_matrix = sla.inv(matrix)
test(matrix, inverse_matrix)
```

3.4 eps = 0.000000001

Рассмотрим B_{eps} при $eps = 0.000000001$. В данном случае, алгоритм выдает сообщение, о том что матрица является вырожденной.

[47]:
```
matrix = B(epses[3])
inverse_matrix = inverse(matrix)
test(matrix, inverse_matrix)
```

```
<ipython-input-9-089af5996201>:33: LinAlgWarning: Singular matrix
  warnings.warn("Singular matrix", sla.LinAlgWarning)
```

При этом, встроенный метод **inv** выдает корректный результат.

```
[51]: matrix = B(epses[3])
inverse_matrix = sla.inv(matrix)
test(matrix, inverse_matrix)
```

3.5 eps = 0.000000000000001

Рассмотрим B_{eps} при $eps = 0.000000000000001$. В данном случае, алгоритм выдает сообщение, о том что матрица является вырожденной.

```
[50]: matrix = B(epses[4])
inverse_matrix = inverse(matrix)
test(matrix, inverse_matrix)
```

```
<ipython-input-9-089af5996201>:33: LinAlgWarning: Singular matrix
  warnings.warn("Singular matrix", sla.LinAlgWarning)
```

При этом, точность встроенного метод **inv** падает, поэтому условие проверки в данном тестовом случае не соблюдено.

```
[55]: matrix = B(epses[4])
inverse_matrix = sla.inv(matrix)
test(matrix, inverse_matrix)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-55-63b1598fd2e0> in <module>
      1 matrix = B(epses[4])
      2 inverse_matrix = sla.inv(matrix)
----> 3 test(matrix, inverse_matrix)

<ipython-input-33-327f4077c160> in test(matrix, inverse_matrix, eps)
      1 def test(matrix: np.array, inverse_matrix: np.array, eps = 1e-5) -> None:
      2     if inverse_matrix is not None:
----> 3         assert sla.norm(matrix @ inverse_matrix - np.eye(matrix.shape[0]))␣
 ↪< eps

AssertionError:
```

4 Тестирование алгоритма на матрице C_{14} для определения соответствия реализованного алгоритма заложенному методу

Рассмотрим ход работы метода для матрицы вида:

$$C_{14} = \begin{pmatrix} -1 & -1 & -1 & -1 \\ 0 & 0 & 2 & 6 \\ 1 & 2 & 0 & 0 \\ 1 & -2 & 1 & 1 \end{pmatrix}$$

Итерация $i = 0$

Положим: - $C^0 = E$ - матрица, которая после замены столбцов на столбцы C_{14} (после всех итераций алгоритма) должна стать равной C_{14} - $B^0 = E$ - матрица, которая после всех итераций алгоритма должна стать равной C_{14}^{-1} - $J^0 = \{1, 2, 3, 4\}$ - множество номеров неиспользованных (невставленных) столбцев исходной матрицы C_{14}

При этом, столбы матрицы C_{14} обозначим как c_1, c_2, c_3 ,c_4

Шаг 1. Найдем ненулевое α_j, где j итерируется по номерам неиспользованных столбцев из множества J^0: - $\alpha_1 = e_1^T B^0 c_1 = (1, 0, 0, 0) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = 1$

Поскольку $\alpha_1 \neq 0$, то для вставки выберем 1 столбец.

Шаг 2. Обновим C^1, B^1, J^1: - $C^1 = (c_1, e_2, e_3, e_4)$ - $J^1 = J^0 \backslash \{1\} = \{2, 3, 4\}$ - $B^1 = D(1, B^0 c_1) B^0 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$

Так как номер итерации $i + 1 < 4$, перейдем на следующую итерацию $i = 1$.

Итерация $i = 1$

Шаг 1. Найдем ненулевое α_j, где j итерируется по номерам неиспользованных столбцев из множества J^1: - $\alpha_2 = e_2^T B^1 c_2 = (0, 1, 0, 0) \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ 2 \\ -2 \end{pmatrix} = 0$

Так как $\alpha_2 = 0$, то продолжим искать подходящий номер столбца

- $\alpha_3 = e_2^T B^1 c_3 = (0, 1, 0, 0) \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \\ 0 \\ 1 \end{pmatrix} = 2$

Поскольку $\alpha_3 \neq 0$, то для вставки выберем 3 столбец.

Шаг 2. Обновим C^2, B^2, J^2: - $C^2 = (c_1, c_3, e_3, e_4)$ - $J^2 = J^1 \backslash \{3\} = \{2, 4\}$ - $B^2 = D(2, B^1 c_3) B^1 = \begin{pmatrix} -1 & -0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 1 & 0.5 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$

Так как номер итерации $i + 1 < 4$, перейдем на следующую итерацию $i = 2$.

Итерация $i = 2$

Шаг 1. Найдем ненулевое α_j, где j итерируется по номерам неиспользованных столбцов из множества J^2: - $\alpha_2 = e_3^T B^2 c_2 = (0, 0, 1, 0) \begin{pmatrix} -1 & -0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 1 & 0.5 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ 2 \\ -2 \end{pmatrix} = 2$

Так как $\alpha_2 \neq 0$, то для вставки выберем 2 столбец.

Шаг 2. Обновим C^3, B^3, J^3: - $C^3 = (c_1, c_3, c_2, e_4)$ - $J^3 = J^2 \backslash \{2\} = \{4\}$ - $B^3 = D(3, B^2 c_2) B^2 = \begin{pmatrix} -2 & -1 & -1 & 0 \\ 0 & 0.5 & 0 & 0 \\ 1 & 0.5 & 1 & 0 \\ 4 & 1.5 & 3 & 1 \end{pmatrix}$

Так как номер итерации $i + 1 < 4$, перейдем на следующую итерацию $i = 3$.

Итерация $i = 3$

Шаг 1. Найдем ненулевое α_j, где j итерируется по номерам неиспользованных столбцов из множества J^3: - $\alpha_4 = e_4^T B^3 c_4 = (0, 0, 0, 1) \begin{pmatrix} -2 & -1 & -1 & 0 \\ 0 & 0.5 & 0 & 0 \\ 1 & 0.5 & 1 & 0 \\ 4 & 1.5 & 3 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 6 \\ 0 \\ 1 \end{pmatrix} = 1$

Так как $\alpha_4 \neq 0$, то для вставки выберем 4 столбец.

Шаг 2. Обновим C^4, B^4, J^4: - $C^4 = (c_1, c_3, c_2, c_4)$ - $J^4 = J^3 \backslash \{4\} = \{\}$ - $B^4 = D(4, B^3 c_4) B^3 = \begin{pmatrix} -6.66 & -5.55 & 1 & 6.67 \\ -2.00 & -2.50 & -1.5 & -5.00 \\ 3.33 & 2.78 & 5.55 & -3.33 \\ 6.66 & 2.50 & 5.00 & 1.67 \end{pmatrix}$

Так как номер итерации $i + 1 = 4$, работа метода окончена. Искомая C_{14}^{-1} матрица получается перестановкой строк матрицы B^4

Протестируем реализованный алгоритм на матрице C_{14}, при этом делая вывод текущей итерации i, множества J_i и матрицы B_i. Как видно, реализованный алгоритм показывает полное соответствие заложенному в него методу.

[73]:
```
matrix = np.array([[-1, -1, -1, -1], [0, 0, 2, 6], [1, 2, 0, 0], [1, -2, 1, 1]])
inverse_matrix = inverse(matrix, verbose=True)
test(matrix, inverse_matrix)
```

```
Initial unused indexes set {1, 2, 3, 4}
Initial matrix
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
Current iteration no. 1
Current indexes set {2, 3, 4}
Current inverse matrix
 [[-1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 1.  0.  1.  0.]
 [ 1.  0.  0.  1.]]
Current iteration no. 2
Current indexes set {2, 4}
Current inverse matrix
 [[-1.  -0.5  0.   0. ]
 [ 0.   0.5  0.   0. ]
 [ 1.   0.5  1.   0. ]
 [ 1.   0.   0.   1. ]]
Current iteration no. 3
Current indexes set {4}
Current inverse matrix
 [[-2.  -1.  -1.   0. ]
 [ 0.   0.5  0.   0. ]
 [ 1.   0.5  1.   0. ]
 [ 4.   1.5  3.   1. ]]
Current iteration no. 4
Current indexes set {}
Current inverse matrix
 [[ 6.66666667e-01 -5.55111512e-17  1.00000000e+00  6.66666667e-01]
 [-2.00000000e+00 -2.50000000e-01 -1.50000000e+00 -5.00000000e-01]
 [-3.33333333e-01  2.77555756e-17  5.55111512e-17 -3.33333333e-01]
 [ 6.66666667e-01  2.50000000e-01  5.00000000e-01  1.66666667e-01]]
```