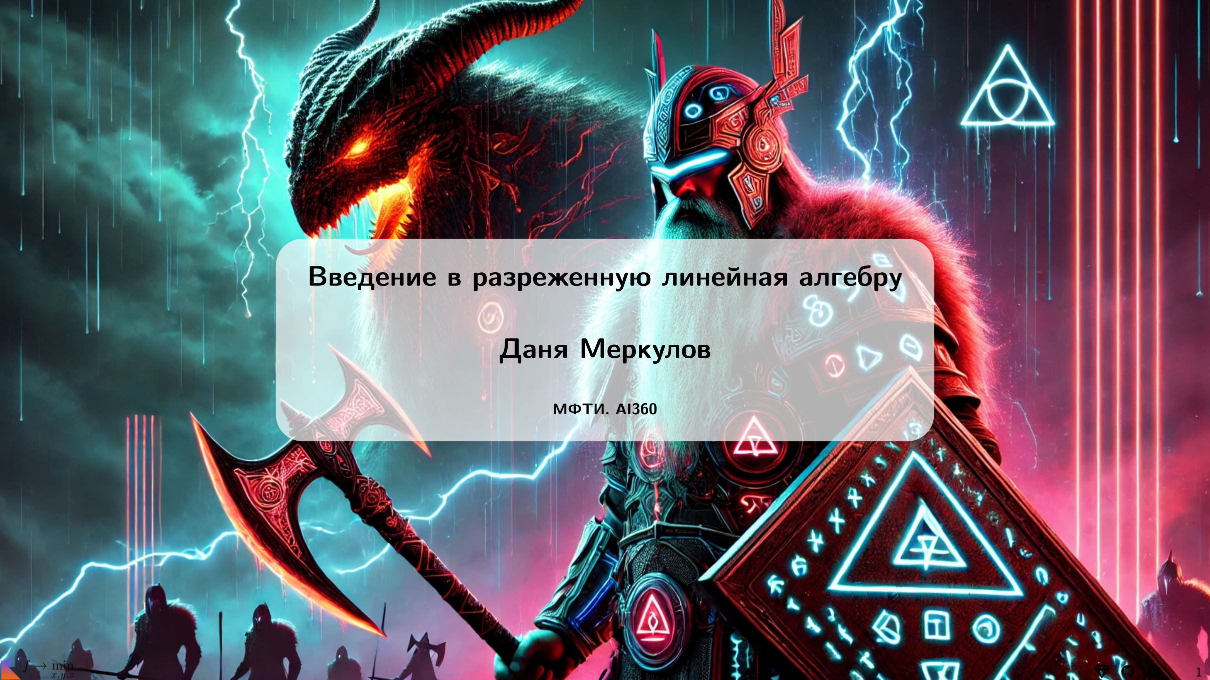


Введение в разреженную линейная алгебру

Даня Меркулов

МФТИ. AI360

Введение в разреженные матрицы

Что такое разреженная матрица?

Матрица называется **разреженной** (sparse), если большинство её элементов равны нулю. Точного порога нет, но обычно имеется в виду, что число ненулевых элементов (nnz) **значительно меньше** общего числа элементов $(m \times n)$.

$$\mathrm{nnz} \ll m \times n$$

Что такое разреженная матрица?

Матрица называется **разреженной** (sparse), если большинство её элементов равны нулю. Точного порога нет, но обычно имеется в виду, что число ненулевых элементов (nnz) **значительно меньше** общего числа элементов $(m \times n)$.

$$\mathrm{nnz} \ll m \times n$$

Зачем они нужны?

- **Экономия памяти:** Хранение только ненулевых элементов позволяет работать с матрицами огромных размеров, которые не поместились бы в память в плотном виде.

Что такое разреженная матрица?

Матрица называется **разреженной** (sparse), если большинство её элементов равны нулю. Точного порога нет, но обычно имеется в виду, что число ненулевых элементов (nnz) **значительно меньше** общего числа элементов $(m \times n)$.

$$\mathrm{nnz} \ll m \times n$$

Зачем они нужны?

- **Экономия памяти:** Хранение только ненулевых элементов позволяет работать с матрицами огромных размеров, которые не поместились бы в память в плотном виде.
- **Ускорение вычислений:** Операции (умножение на вектор, решение СЛАУ) можно выполнять быстрее, пропуская операции с нулевыми элементами.

Где встречаются разреженные матрицы?

Разреженные матрицы возникают естественным образом во многих областях:

- **Дискретизация дифференциальных уравнений:** Например, при решении уравнения Лапласа методом конечных разностей или конечных элементов матрица системы получается ленточной или блочно-ленточной.

Где встречаются разреженные матрицы?

Разреженные матрицы возникают естественным образом во многих областях:

- **Дискретизация дифференциальных уравнений:** Например, при решении уравнения Лапласа методом конечных разностей или конечных элементов матрица системы получается ленточной или блочно-ленточной.
- **Анализ графов:** Матрица смежности графа (например, социальной сети или веб-графа) обычно разрежена, так как каждый узел связан лишь с небольшим числом других узлов.

Где встречаются разреженные матрицы?

Разреженные матрицы возникают естественным образом во многих областях:

- **Дискретизация дифференциальных уравнений:** Например, при решении уравнения Лапласа методом конечных разностей или конечных элементов матрица системы получается ленточной или блочно-ленточной.
- **Анализ графов:** Матрица смежности графа (например, социальной сети или веб-графа) обычно разрежена, так как каждый узел связан лишь с небольшим числом других узлов.
- **Машинное обучение:** Например, в рекомендательных системах (матрица ``пользователь-товар'') или при прореживании (pruning) нейронных сетей.

Где встречаются разреженные матрицы?

Разреженные матрицы возникают естественным образом во многих областях:

- **Дискретизация дифференциальных уравнений:** Например, при решении уравнения Лапласа методом конечных разностей или конечных элементов матрица системы получается ленточной или блочно-ленточной.
- **Анализ графов:** Матрица смежности графа (например, социальной сети или веб-графа) обычно разрежена, так как каждый узел связан лишь с небольшим числом других узлов.
- **Машинное обучение:** Например, в рекомендательных системах (матрица ``пользователь-товар'') или при прореживании (pruning) нейронных сетей.
- **Научные вычисления:** Моделирование физических процессов, схемы электроники и т.д.

Форматы хранения разреженных матриц

Coordinate Format (COO)

Хранить все нули неэффективно. Существуют специальные форматы для хранения только ненулевых элементов и их позиций.

Самый простой формат: храним три массива одинаковой длины (nnz):

- `row`: массив номеров строк для каждого ненулевого элемента.

Coordinate Format (COO)

Хранить все нули неэффективно. Существуют специальные форматы для хранения только ненулевых элементов и их позиций.

Самый простой формат: храним три массива одинаковой длины (nnz):

- `row`: массив номеров строк для каждого ненулевого элемента.
- `col`: массив номеров столбцов.

Coordinate Format (COO)

Хранить все нули неэффективно. Существуют специальные форматы для хранения только ненулевых элементов и их позиций.

Самый простой формат: храним три массива одинаковой длины (nnz):

- `row`: массив номеров строк для каждого ненулевого элемента.
- `col`: массив номеров столбцов.
- `data`: массив значений ненулевых элементов.

Coordinate Format (COO)

Хранить все нули неэффективно. Существуют специальные форматы для хранения только ненулевых элементов и их позиций.

Самый простой формат: храним три массива одинаковой длины (nnz):

- `row`: массив номеров строк для каждого ненулевого элемента.
- `col`: массив номеров столбцов.
- `data`: массив значений ненулевых элементов.

Coordinate Format (COO)

Хранить все нули неэффективно. Существуют специальные форматы для хранения только ненулевых элементов и их позиций.

Самый простой формат: храним три массива одинаковой длины (nnz):

- `row`: массив номеров строк для каждого ненулевого элемента.
- `col`: массив номеров столбцов.
- `data`: массив значений ненулевых элементов.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

Coordinate Format (COO)

Хранить все нули неэффективно. Существуют специальные форматы для хранения только ненулевых элементов и их позиций.

Самый простой формат: храним три массива одинаковой длины (nnz):

- `row`: массив номеров строк для каждого ненулевого элемента.
- `col`: массив номеров столбцов.
- `data`: массив значений ненулевых элементов.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

В формате COO:

```
row  = [0, 0, 1, 2, 2]
col  = [0, 3, 1, 0, 2]
data = [1, 2, 3, 4, 5]
```

Coordinate Format (COO)

Хранить все нули неэффективно. Существуют специальные форматы для хранения только ненулевых элементов и их позиций.

Самый простой формат: храним три массива одинаковой длины (nnz):

- `row`: массив номеров строк для каждого ненулевого элемента.
- `col`: массив номеров столбцов.
- `data`: массив значений ненулевых элементов.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

В формате COO:

```
row  = [0, 0, 1, 2, 2]
col  = [0, 3, 1, 0, 2]
data = [1, 2, 3, 4, 5]
```

Плюсы: - Простота и легкость добавления новых элементов.

Минусы: - Неэффективен для арифметических операций (например, умножения на вектор). - Избыточное хранение индексов строк (могут повторяться).

List of Lists (LIL)

Представление в виде списка списков:

- Внешний список длины m (число строк).

List of Lists (LIL)

Представление в виде списка списков:

- Внешний список длины m (число строк).
- Каждый элемент `rows[i]` - это список пар `(индекс_столбца, значение)` для ненулевых элементов i-й строки.

List of Lists (LIL)

Представление в виде списка списков:

- Внешний список длины m (число строк).
- Каждый элемент `rows[i]` - это список пар `(индекс_столбца, значение)` для ненулевых элементов i-й строки.

List of Lists (LIL)

Представление в виде списка списков:

- Внешний список длины m (число строк).
- Каждый элемент `rows[i]` - это список пар `(индекс_столбца, значение)` для ненулевых элементов i-й строки.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

List of Lists (LIL)

Представление в виде списка списков:

- Внешний список длины m (число строк).
- Каждый элемент `rows[i]` - это список пар `(индекс_столбца, значение)` для ненулевых элементов i-й строки.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

```
rows = [
  [(0, 1), (3, 2)],  # Строка 0
  [(1, 3)],          # Строка 1
  [(0, 4), (2, 5)]   # Строка 2
]
```

List of Lists (LIL)

Представление в виде списка списков:

- Внешний список длины m (число строк).
- Каждый элемент `rows[i]` - это список пар `(индекс_столбца, значение)` для ненулевых элементов i-й строки.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

```
rows = [
  [(0, 1), (3, 2)],  # Строка 0
  [(1, 3)],          # Строка 1
  [(0, 4), (2, 5)]   # Строка 2
]
```

Плюсы:

- Удобно добавлять/удалять элементы и изменять структуру матрицы.

Минусы:

List of Lists (LIL)

Представление в виде списка списков:

- Внешний список длины m (число строк).
- Каждый элемент `rows[i]` - это список пар `(индекс_столбца, значение)` для ненулевых элементов i-й строки.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

```
rows = [
  [(0, 1), (3, 2)],  # Строка 0
  [(1, 3)],          # Строка 1
  [(0, 4), (2, 5)]   # Строка 2
]
```

Плюсы:

- Удобно добавлять/удалять элементы и изменять структуру матрицы.
- Эффективен для построения матрицы по элементам.

Минусы:

List of Lists (LIL)

Представление в виде списка списков:

- Внешний список длины m (число строк).
- Каждый элемент `rows[i]` - это список пар `(индекс_столбца, значение)` для ненулевых элементов i-й строки.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

```
rows = [
  [(0, 1), (3, 2)],  # Строка 0
  [(1, 3)],          # Строка 1
  [(0, 4), (2, 5)]   # Строка 2
]
```

Плюсы:

- Удобно добавлять/удалять элементы и изменять структуру матрицы.
- Эффективен для построения матрицы по элементам.

Минусы:

- Неэффективен для арифметических операций.

List of Lists (LIL)

Представление в виде списка списков:

- Внешний список длины m (число строк).
- Каждый элемент `rows[i]` - это список пар `(индекс_столбца, значение)` для ненулевых элементов i-й строки.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

```
rows = [
  [(0, 1), (3, 2)],  # Строка 0
  [(1, 3)],          # Строка 1
  [(0, 4), (2, 5)]   # Строка 2
]
```

Плюсы:

- Удобно добавлять/удалять элементы и изменять структуру матрицы.
- Эффективен для построения матрицы по элементам.

Минусы:

- Неэффективен для арифметических операций.
- Потребляет больше памяти, чем COO или CSR из-за списков Python.

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.
- `indices`: номера столбцов для каждого элемента в `data` (длина nnz).

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.
- `indices`: номера столбцов для каждого элемента в `data` (длина nnz).
- `indptr` (index pointer): массив длины $m + 1$. `indptr[i]` указывает на начало i-й строки в массивах `data` и `indices`. `indptr[i+1] - indptr[i]` - это количество ненулевых элементов в i-й строке. `indptr[m] = nnz`.

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.
- `indices`: номера столбцов для каждого элемента в `data` (длина nnz).
- `indptr` (index pointer): массив длины $m+1$. `indptr[i]` указывает на начало i-й строки в массивах `data` и `indices`. `indptr[i+1] - indptr[i]` - это количество ненулевых элементов в i-й строке. `indptr[m] = nnz`.

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.
- `indices`: номера столбцов для каждого элемента в `data` (длина nnz).
- `indptr` (index pointer): массив длины $m+1$. `indptr[i]` указывает на начало i-й строки в массивах `data` и `indices`. `indptr[i+1] - indptr[i]` - это количество ненулевых элементов в i-й строке. `indptr[m] = nnz`.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.
- `indices`: номера столбцов для каждого элемента в `data` (длина nnz).
- `indptr` (index pointer): массив длины $m + 1$. `indptr[i]` указывает на начало i-й строки в массивах `data` и `indices`. `indptr[i+1] - indptr[i]` - это количество ненулевых элементов в i-й строке. `indptr[m] = nnz`.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

В формате CSR:

```
data    = [1, 2, 3, 4, 5]   # Ненулевые элементы по строкам
indices = [0, 3, 1, 0, 2]   # Номера столбцов для них
indptr  = [0, 2, 3, 5]   # Указатели: строка 0 начинается
                        # с индекса 0, строка 1 с 2,
                        # строка 2 с 3, конец - индекс 5
```

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.
- `indices`: номера столбцов для каждого элемента в `data` (длина nnz).
- `indptr` (index pointer): массив длины $m + 1$. `indptr[i]` указывает на начало i-й строки в массивах `data` и `indices`. `indptr[i+1] - indptr[i]` - это количество ненулевых элементов в i-й строке. `indptr[m] = nnz`.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

В формате CSR:

```
data    = [1, 2, 3, 4, 5]   # Ненулевые элементы по строкам
indices = [0, 3, 1, 0, 2]   # Номера столбцов для них
indptr  = [0, 2, 3, 5]   # Указатели: строка 0 начинается
                        # с индекса 0, строка 1 с 2,
                        # строка 2 с 3, конец - индекс 5
```

Плюсы:

- Эффективное хранение ($2 \cdot \mathrm{nnz} + m + 1$ чисел).

Минусы:

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.
- `indices`: номера столбцов для каждого элемента в `data` (длина nnz).
- `indptr` (index pointer): массив длины $m + 1$. `indptr[i]` указывает на начало i-й строки в массивах `data` и `indices`. `indptr[i+1] - indptr[i]` - это количество ненулевых элементов в i-й строке. `indptr[m] = nnz`.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

В формате CSR:

```
data    = [1, 2, 3, 4, 5]   # Ненулевые элементы по строкам
indices = [0, 3, 1, 0, 2]   # Номера столбцов для них
indptr  = [0, 2, 3, 5]   # Указатели: строка 0 начинается
                        # с индекса 0, строка 1 с 2,
                        # строка 2 с 3, конец - индекс 5
```

Плюсы:

- Эффективное хранение ($2 \cdot \mathrm{nnz} + m + 1$ чисел).
- **Быстрое умножение матрицы на вектор (SpMV).**

Минусы:

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.
- `indices`: номера столбцов для каждого элемента в `data` (длина nnz).
- `indptr` (index pointer): массив длины $m+1$. `indptr[i]` указывает на начало i-й строки в массивах `data` и `indices`. `indptr[i+1] - indptr[i]` - это количество ненулевых элементов в i-й строке. `indptr[m] = nnz`.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

В формате CSR:

```
data    = [1, 2, 3, 4, 5]   # Ненулевые элементы по строкам
indices = [0, 3, 1, 0, 2]   # Номера столбцов для них
indptr  = [0, 2, 3, 5]   # Указатели: строка 0 начинается
                        # с индекса 0, строка 1 с 2,
                        # строка 2 с 3, конец - индекс 5
```

Плюсы:

- Эффективное хранение ($2 \cdot \text{nnz} + m + 1$ чисел).
- **Быстрое умножение матрицы на вектор (SpMV).**
- Быстрый доступ к строкам.

Минусы:

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.
- `indices`: номера столбцов для каждого элемента в `data` (длина nnz).
- `indptr` (index pointer): массив длины $m + 1$. `indptr[i]` указывает на начало i-й строки в массивах `data` и `indices`. `indptr[i+1] - indptr[i]` - это количество ненулевых элементов в i-й строке. `indptr[m] = nnz`.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

В формате CSR:

```
data    = [1, 2, 3, 4, 5]   # Ненулевые элементы по строкам
indices = [0, 3, 1, 0, 2]   # Номера столбцов для них
indptr  = [0, 2, 3, 5]   # Указатели: строка 0 начинается
                        # с индекса 0, строка 1 с 2,
                        # строка 2 с 3, конец - индекс 5
```

Плюсы:

- Эффективное хранение ($2 \cdot \mathrm{nnz} + m + 1$ чисел).
- **Быстрое умножение матрицы на вектор (SpMV).**
- Быстрый доступ к строкам.

Минусы:

- Медленное добавление/удаление элементов (требует сдвигов в `data` и `indices`).

Compressed Sparse Row (CSR)

Один из самых популярных и эффективных форматов для вычислений. Хранит 3 массива:

- `data`: значения ненулевых элементов (длина nnz), упорядоченные по строкам.
- `indices`: номера столбцов для каждого элемента в `data` (длина nnz).
- `indptr` (index pointer): массив длины $m+1$. `indptr[i]` указывает на начало i-й строки в массивах `data` и `indices`. `indptr[i+1] - indptr[i]` - это количество ненулевых элементов в i-й строке. `indptr[m] = nnz`.

Пример:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

В формате CSR:

```
data    = [1, 2, 3, 4, 5]   # Ненулевые элементы по строкам
indices = [0, 3, 1, 0, 2]   # Номера столбцов для них
indptr  = [0, 2, 3, 5]   # Указатели: строка 0 начинается
                        # с индекса 0, строка 1 с 2,
                        # строка 2 с 3, конец - индекс 5
```

Плюсы:

- Эффективное хранение ($2 \cdot \mathrm{nnz} + m + 1$ чисел).
- **Быстрое умножение матрицы на вектор (SpMV).**
- Быстрый доступ к строкам.

Минусы:

- Медленное добавление/удаление элементов (требует сдвигов в `data` и `indices`).
- Медленный доступ к столбцам.

Compressed Sparse Column (CSC)

Аналогичен CSR, но хранит матрицу по столбцам.

- `data`: значения ненулевых элементов, упорядоченные по столбцам.

Compressed Sparse Column (CSC)

Аналогичен CSR, но хранит матрицу по столбцам.

- `data`: значения ненулевых элементов, упорядоченные по столбцам.
- `indices`: номера строк для каждого элемента в `data`.

Compressed Sparse Column (CSC)

Аналогичен CSR, но хранит матрицу по столбцам.

- `data`: значения ненулевых элементов, упорядоченные по столбцам.
- `indices`: номера строк для каждого элемента в `data`.
- `indptr`: массив длины $n+1$. `indptr[j]` указывает на начало j-го столбца.

Compressed Sparse Column (CSC)

Аналогичен CSR, но хранит матрицу по столбцам.

- `data`: значения ненулевых элементов, упорядоченные по столбцам.
- `indices`: номера строк для каждого элемента в `data`.
- `indptr`: массив длины $n + 1$. `indptr[j]` указывает на начало j-го столбца.

Compressed Sparse Column (CSC)

Аналогичен CSR, но хранит матрицу по столбцам.

- `data`: значения ненулевых элементов, упорядоченные по столбцам.
- `indices`: номера строк для каждого элемента в `data`.
- `indptr`: массив длины $n+1$. `indptr[j]` указывает на начало j-го столбца.

Плюсы: - Эффективное хранение. - Быстрое умножение транспонированной матрицы на вектор $(A^T x)$. - Быстрый доступ к столбцам.

Минусы: - Медленное добавление/удаление элементов. - Медленный доступ к строкам.

Упражнение: Преобразование в CSR

Преобразуйте следующую матрицу в формат CSR вручную:

$$B = \begin{pmatrix} 0 & 7 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 \\ 8 & 0 & 2 & 0 \end{pmatrix}$$

Запишите массивы `data`, `indices` и `indptr`.

Упражнение: Преобразование в CSR

Преобразуйте следующую матрицу в формат CSR вручную:

$$B = \begin{pmatrix} 0 & 7 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 \\ 8 & 0 & 2 & 0 \end{pmatrix}$$

Запишите массивы `data`, `indices` и `indptr`.

Решение:

1. **data:** Записываем ненулевые элементы по строкам: `[7, 5, 1, 9, 8, 2]`

Упражнение: Преобразование в CSR

Преобразуйте следующую матрицу в формат CSR вручную:

$$B = \begin{pmatrix} 0 & 7 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 \\ 8 & 0 & 2 & 0 \end{pmatrix}$$

Запишите массивы `data`, `indices` и `indptr`.

Решение:

1. **data:** Записываем ненулевые элементы по строкам: `[7, 5, 1, 9, 8, 2]`
2. **indices:** Записываем столбцы для этих элементов: `[1, 0, 2, 3, 0, 2]`

Упражнение: Преобразование в CSR

Преобразуйте следующую матрицу в формат CSR вручную:

$$B = \begin{pmatrix} 0 & 7 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 \\ 8 & 0 & 2 & 0 \end{pmatrix}$$

Запишите массивы `data`, `indices` и `indptr`.

Решение:

1. **data:** Записываем ненулевые элементы по строкам: `[7, 5, 1, 9, 8, 2]`
2. **indices:** Записываем столбцы для этих элементов: `[1, 0, 2, 3, 0, 2]`
3. **indptr:** Указатели на начало строк (и конец последней):

Упражнение: Преобразование в CSR

Преобразуйте следующую матрицу в формат CSR вручную:

$$B = \begin{pmatrix} 0 & 7 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 \\ 8 & 0 & 2 & 0 \end{pmatrix}$$

Запишите массивы `data`, `indices` и `indptr`.

Решение:

1. **data:** Записываем ненулевые элементы по строкам: `[7, 5, 1, 9, 8, 2]`
2. **indices:** Записываем столбцы для этих элементов: `[1, 0, 2, 3, 0, 2]`
3. **indptr:** Указатели на начало строк (и конец последней):
    - Строка 0: начинается с индекса 0 (1 элемент: 7)

Упражнение: Преобразование в CSR

Преобразуйте следующую матрицу в формат CSR вручную:

$$B = \begin{pmatrix} 0 & 7 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 \\ 8 & 0 & 2 & 0 \end{pmatrix}$$

Запишите массивы `data`, `indices` и `indptr`.

Решение:

1. **data:** Записываем ненулевые элементы по строкам: `[7, 5, 1, 9, 8, 2]`
2. **indices:** Записываем столбцы для этих элементов: `[1, 0, 2, 3, 0, 2]`
3. **indptr:** Указатели на начало строк (и конец последней):
    - Строка 0: начинается с индекса 0 (1 элемент: 7)
    - Строка 1: начинается с индекса 1 (2 элемента: 5, 1)

Упражнение: Преобразование в CSR

Преобразуйте следующую матрицу в формат CSR вручную:

$$B = \begin{pmatrix} 0 & 7 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 \\ 8 & 0 & 2 & 0 \end{pmatrix}$$

Запишите массивы `data`, `indices` и `indptr`.

Решение:

1. **data:** Записываем ненулевые элементы по строкам: `[7, 5, 1, 9, 8, 2]`
2. **indices:** Записываем столбцы для этих элементов: `[1, 0, 2, 3, 0, 2]`
3. **indptr:** Указатели на начало строк (и конец последней):
    - Строка 0: начинается с индекса 0 (1 элемент: 7)
    - Строка 1: начинается с индекса 1 (2 элемента: 5, 1)
    - Строка 2: начинается с индекса 3 (1 элемент: 9)

Упражнение: Преобразование в CSR

Преобразуйте следующую матрицу в формат CSR вручную:

$$B = \begin{pmatrix} 0 & 7 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 \\ 8 & 0 & 2 & 0 \end{pmatrix}$$

Запишите массивы `data`, `indices` и `indptr`.

Решение:

1. **data:** Записываем ненулевые элементы по строкам: `[7, 5, 1, 9, 8, 2]`
2. **indices:** Записываем столбцы для этих элементов: `[1, 0, 2, 3, 0, 2]`
3. **indptr:** Указатели на начало строк (и конец последней):
    - Строка 0: начинается с индекса 0 (1 элемент: 7)
    - Строка 1: начинается с индекса 1 (2 элемента: 5, 1)
    - Строка 2: начинается с индекса 3 (1 элемент: 9)
    - Строка 3: начинается с индекса 4 (2 элемента: 8, 2)

Упражнение: Преобразование в CSR

Преобразуйте следующую матрицу в формат CSR вручную:

$$B = \begin{pmatrix} 0 & 7 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 \\ 8 & 0 & 2 & 0 \end{pmatrix}$$

Запишите массивы `data`, `indices` и `indptr`.

Решение:

1. **data:** Записываем ненулевые элементы по строкам: `[7, 5, 1, 9, 8, 2]`
2. **indices:** Записываем столбцы для этих элементов: `[1, 0, 2, 3, 0, 2]`
3. **indptr:** Указатели на начало строк (и конец последней):
    - Строка 0: начинается с индекса 0 (1 элемент: 7)
    - Строка 1: начинается с индекса 1 (2 элемента: 5, 1)
    - Строка 2: начинается с индекса 3 (1 элемент: 9)
    - Строка 3: начинается с индекса 4 (2 элемента: 8, 2)
    - Конец: индекс 6 (всего 6 ненулевых элементов) Итого: `[0, 1, 3, 4, 6]`

Пример:

Базовые операции с разреженными матрицами

Умножение матрицы на вектор (SpMV)

Это ключевая операция для многих алгоритмов (например, итерационных методов решения СЛАУ). В формате CSR она выполняется эффективно.

```
# A - матрица в CSR (массивы ia, ja, sa)
# x - вектор для умножения
# y - результирующий вектор (инициализирован нулями)
n = len(ia) - 1
for i in range(n):
    # y[i] = 0 # Если не инициализирован нулями
    for k in range(ia[i], ia[i+1]):
        # Доступ к элементам i-й строки
        # sa[k] - значение элемента
        # ja[k] - номер столбца
        y[i] += sa[k] * x[ja[k]]
```

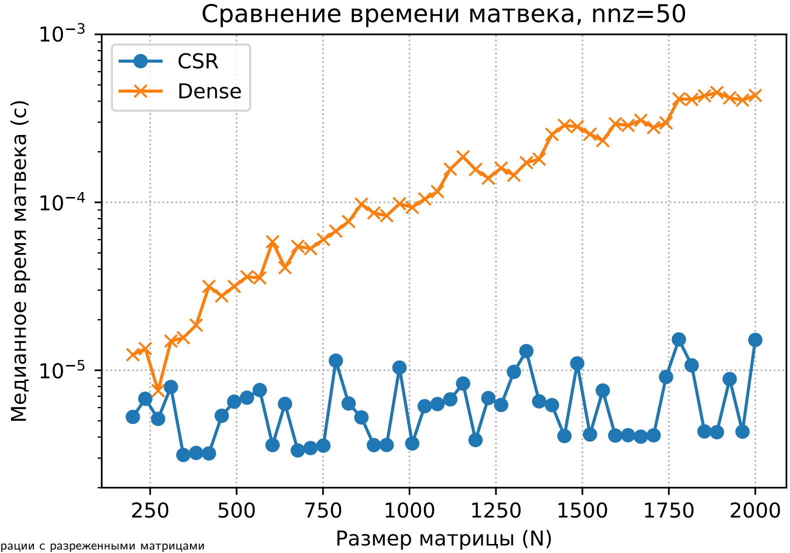
Умножение матрицы на вектор (SpMV)

Это ключевая операция для многих алгоритмов (например, итерационных методов решения СЛАУ). В формате CSR она выполняется эффективно.

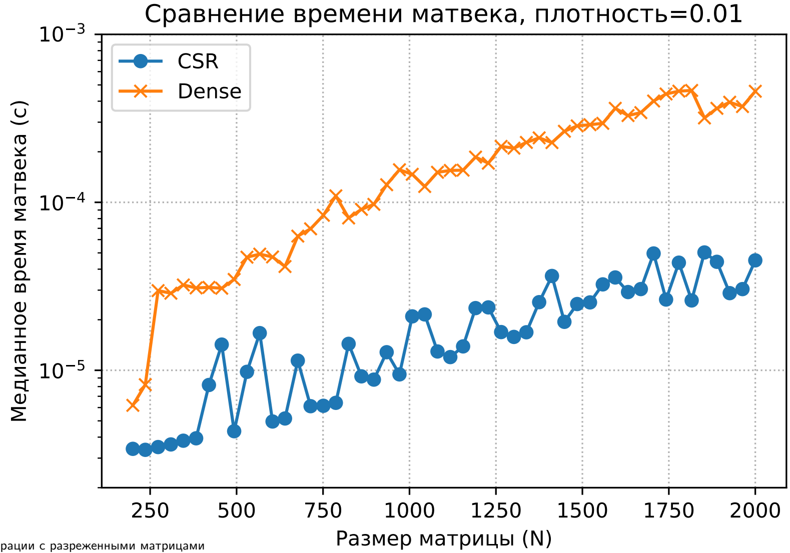
```
# A - матрица в CSR (массивы ia, ja, sa)
# x - вектор для умножения
# y - результирующий вектор (инициализирован нулями)
n = len(ia) - 1
for i in range(n):
    # y[i] = 0 # Если не инициализирован нулями
    for k in range(ia[i], ia[i+1]):
        # Доступ к элементам i-й строки
        # sa[k] - значение элемента
        # ja[k] - номер столбца
        y[i] += sa[k] * x[ja[k]]
```

Сложность: $\mathcal{O}(\mathrm{nnz})$ операций, что гораздо быстрее $\mathcal{O}(N^2)$ для плотного формата, если $\mathrm{nnz} \ll N^2$.

Пример: Сравнение скорости SpMV (Плотный vs CSR) Фиксированное количество ненулевых элементов



Пример: Сравнение скорости SpMV (Плотный vs CSR) Фиксированная плотность



Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов $(PAP^T = LU)$ для минимизации заполнения.

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение ($A = LU$) приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов ($PAP^T = LU$) для минимизации заполнения.
- Алгоритмы: Nested Dissection, Minimum Degree.

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов $(PAP^T = LU)$ для минимизации заполнения.
- Алгоритмы: Nested Dissection, Minimum Degree.
- Библиотеки: `scipy.sparse.linalg.spsolve` (использует UMFPACK или SuperLU), PARDISO, MUMPS.

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов $(PAP^T = LU)$ для минимизации заполнения.
- Алгоритмы: Nested Dissection, Minimum Degree.
- Библиотеки: `scipy.sparse.linalg.spsolve` (использует UMFPACK или SuperLU), PARDISO, MUMPS.
- Эффективны для *умеренно* больших задач.

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов $(PAP^T = LU)$ для минимизации заполнения.
- Алгоритмы: Nested Dissection, Minimum Degree.
- Библиотеки: `scipy.sparse.linalg.spsolve` (использует UMFPACK или SuperLU), PARDISO, MUMPS.
- Эффективны для *умеренно* больших задач.

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов $(PAP^T = LU)$ для минимизации заполнения.
- Алгоритмы: Nested Dissection, Minimum Degree.
- Библиотеки: `scipy.sparse.linalg.spsolve` (использует UMFPACK или SuperLU), PARDISO, MUMPS.
- Эффективны для *умеренно* больших задач.

Итерационные методы:

- Строят последовательность приближений $x_0, x_1, \ldots$, сходящуюся к решению.

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов $(PAP^T = LU)$ для минимизации заполнения.
- Алгоритмы: Nested Dissection, Minimum Degree.
- Библиотеки: `scipy.sparse.linalg.spsolve` (использует UMFPACK или SuperLU), PARDISO, MUMPS.
- Эффективны для *умеренно* больших задач.

Итерационные методы:

- Строят последовательность приближений $x_0, x_1, \dots$, сходящуюся к решению.
- Основная операция - **SpMV** $(Ax_k$ или $A^T x_k)$.

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов $(PAP^T = LU)$ для минимизации заполнения.
- Алгоритмы: Nested Dissection, Minimum Degree.
- Библиотеки: `scipy.sparse.linalg.spsolve` (использует UMFPACK или SuperLU), PARDISO, MUMPS.
- Эффективны для *умеренно* больших задач.

Итерационные методы:

- Строят последовательность приближений $x_0, x_1, \ldots$, сходящуюся к решению.
- Основная операция - **SpMV** $(Ax_k$ или $A^T x_k)$.
- Не требуют явного хранения факторов L, U.

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов $(PAP^T = LU)$ для минимизации заполнения.
- Алгоритмы: Nested Dissection, Minimum Degree.
- Библиотеки: `scipy.sparse.linalg.spsolve` (использует UMFPACK или SuperLU), PARDISO, MUMPS.
- Эффективны для *умеренно* больших задач.

Итерационные методы:

- Строят последовательность приближений $x_0, x_1, \dots$, сходящуюся к решению.
- Основная операция - **SpMV** $(Ax_k$ или $A^T x_k)$.
- Не требуют явного хранения факторов L, U.
- Примеры: Метод сопряженных градиентов (CG), GMRES, BiCGStab.

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов $(PAP^T = LU)$ для минимизации заполнения.
- Алгоритмы: Nested Dissection, Minimum Degree.
- Библиотеки: `scipy.sparse.linalg.spsolve` (использует UMFPACK или SuperLU), PARDISO, MUMPS.
- Эффективны для *умеренно* больших задач.

Итерационные методы:

- Строят последовательность приближений $x_0, x_1, ...$, сходящуюся к решению.
- Основная операция - **SpMV** $(Ax_k$ или $A^T x_k)$.
- Не требуют явного хранения факторов L, U.
- Примеры: Метод сопряженных градиентов (CG), GMRES, BiCGStab.
- Часто требуют **предобуславливания** для ускорения сходимости.

Решение линейных систем $Ax = b$

Если A разреженная, стандартное LU-разложение $(A = LU)$ приведет к **заполнению** (fill-in) - факторы L и U могут стать плотными или гораздо менее разреженными, чем A.

Прямые методы:

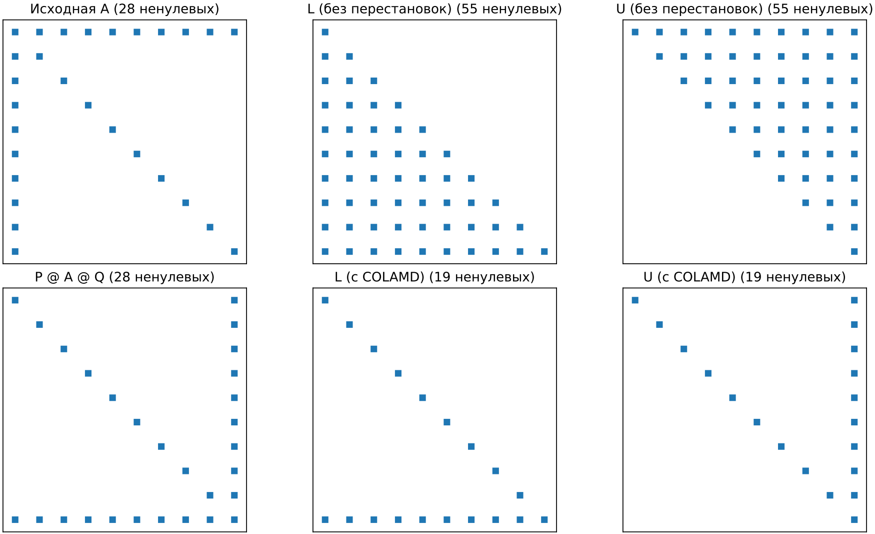
- Используют модификации LU-разложения (или Холецкого для SPD матриц).
- **Ключевая идея: Перестановки** строк и столбцов $(PAP^T = LU)$ для минимизации заполнения.
- Алгоритмы: Nested Dissection, Minimum Degree.
- Библиотеки: `scipy.sparse.linalg.spsolve` (использует UMFPACK или SuperLU), PARDISO, MUMPS.
- Эффективны для *умеренно* больших задач.

Итерационные методы:

- Строят последовательность приближений $x_0, x_1, \ldots$, сходящуюся к решению.
- Основная операция - **SpMV** $(Ax_k$ или $A^T x_k)$.
- Не требуют явного хранения факторов L, U.
- Примеры: Метод сопряженных градиентов (CG), GMRES, BiCGStab.
- Часто требуют **предобуславливания** для ускорения сходимости.
- Предпочтительны для очень больших задач.

Пример: потеря разреженности

Влияние перестановок на заполненность при LU-разложении



Заключение

- Разреженные матрицы экономят память и ускоряют вычисления для многих задач.

Заключение

- Разреженные матрицы экономят память и ускоряют вычисления для многих задач.
- Формат CSR является стандартом де-факто для эффективного SpMV.

Заключение

- Разреженные матрицы экономят память и ускоряют вычисления для многих задач.
- Формат CSR является стандартом де-факто для эффективного SpMV.
- Решение СЛАУ с разреженными матрицами требует специальных прямых или итерационных методов.

Заключение

- Разреженные матрицы экономят память и ускоряют вычисления для многих задач.
- Формат CSR является стандартом де-факто для эффективного SpMV.
- Решение СЛАУ с разреженными матрицами требует специальных прямых или итерационных методов.
- `scipy.sparse` предоставляет базовые инструменты для работы с разреженными матрицами в Python.

Упражнения для самостоятельной работы

Упражнение 1: Форматы COO и CSC

Преобразуйте матрицу B из предыдущего упражнения в форматы COO и CSC вручную:

$$B = \begin{pmatrix} 0 & 7 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 \\ 8 & 0 & 2 & 0 \end{pmatrix}$$

Запишите соответствующие массивы для каждого формата.

Упражнение 2: SpMV в Python

Используя `scipy.sparse`, создайте матрицу A из примера в формате CSR:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 5 & 0 \end{pmatrix}$$

Создайте случайный вектор x подходящего размера. Вычислите произведение $y = Ax$ с помощью функции `A.dot(x)` или оператора `A @ x`. Проверьте результат вручную для небольшого примера.

Упражнение: разреженность факторов

Проверьте численно сохраняется ли разреженность факторов у следующих матричных разложений:

- LU-разложение ✖

Упражнение: разреженность факторов

Проверьте численно сохраняется ли разреженность факторов у следующих матричных разложений:

- LU-разложение ✗
- QR-разложение

Упражнение: разреженность факторов

Проверьте численно сохраняется ли разреженность факторов у следующих матричных разложений:

- LU-разложение ✖
- QR-разложение
- SVD

Упражнение: разреженность факторов

Проверьте численно сохраняется ли разреженность факторов у следующих матричных разложений:

- LU-разложение ✖
- QR-разложение
- SVD
- Разложение Шура

Упражнение 4: Заполнение (Fill-in)

Рассмотрим матрицу

$$C = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 1 & 0 & 1 & 4 \end{pmatrix}$$

Найдите её LU-разложение (например, с помощью `scipy.linalg.lu`). Сравните количество ненулевых элементов в исходной матрице C и в факторах L и U. Обсудите наблюдаемое явление заполнения. Как перестановка строк/столбцов могла бы повлиять на заполнение (теоретически)?