МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Лабораторная работа 1
по дисциплине
«Линейная алгебра»

Выполнили:
Трифонов Василий Максимович
гр. J3111
ИСУ 467758,
Соловьев Матвей Михайлович
гр. J3111
ИСУ 467551,
Ежов Дмитрий Александрович
гр. J3111
ИСУ 471242,

Отчет сдан: 06.05.2025

Санкт-Петербург 2025

1. Математическое обоснование

1.1. Постановка задачи

Пусть даны центрированные данные $X = \{x_1, x_2, ..., x_n\} x_i \in \mathbb{R}^d$

Требуется найти ортонормированные векторы $w_1, w_2, ..., w_k$ максимизирующие дисперсию проекций данных на эти направления. Формально, для первого главного компонента задача формулируется как $w_1 = \arg\max_{\|w\|=1} Var(x_w) = \arg\max_{\|w\|=1} \left(w^T \frac{1}{n} X^T X \right)$

1.2. Формулировка через метод Лагранжа

для максимизации $w^T \frac{1}{n} X^T X$ при условии что $\|w\|^2 = 1$ вводим функцию Лагранжа:

$$L(w, \lambda) = w^T \frac{1}{n} X^T X w - \lambda(w^T w - 1)$$

Производная по w: $\frac{\delta L}{\delta w} = 2\frac{1}{n} X^T X w - 2\lambda w = 0 \Rightarrow \frac{1}{n} X^T X w = \lambda w$ Таким образом мы получаем что это уравнение совпадает с определением собсвтенных векторов и значений ковариационная матрицы $\Rightarrow w$ - собственное значение матицы ковариаций, а λ - соотвествующим собственным значением

1.3. Выбор максимального собственного значения

из условия выше следует $w^T \frac{1}{n} X^T X w = \lambda w^T = \lambda$ Чтобы максимизирвоать дисперсию, необходимо выбрать наибольшее собственное значение, а сооьвествующий собственный вектор будет первой главной компонентой

1.4. итеративное построение следующих компонент

Для нахождения k-го главного компонента ($k > 1$) задача модифицируется: требуется максимизировать дисперсию проекций данных на w_k, ортогональный предыдущим компонентам $\{w_1, ..., w_k - 1\}$ и тогда метод Лагранжа с условиями ортоганальностти будет выглядеть так:

$$L(w, \lambda, u_1, ..., u_k - 1) = w^T \frac{1}{n} X^T X w - \lambda(w^T w - 1) - \sum_{i=1}^{k-1} u_{i(w^T w_i)}$$

а производная по w

$$\frac{1}{n} X^T X w - \lambda w - \sum_{i=1}^{k-1} u_i w_i = 0$$

Что и требовалось доказать

Классы для работы с матрицами

`ABCMatrix` (Абстрактный базовый класс)

- **Описание:** Этот класс служит интерфейсом для всех матричных классов. Он определяет обязательные методы, которые должны быть реализованы в любом классе, представляющем матрицу. Это обеспечивает единообразие в работе с разными типами матриц.
- **Важные методы:**
  - `__init__` : Конструктор матрицы.
  - `size` : Возвращает размеры матрицы (число строк и столбцов).
  - `dtype` : Возвращает тип данных элементов матрицы.
  - `to_list` : Преобразует матрицу в стандартный список списков.
  - `__str__` : Возвращает строковое представление матрицы.
  - `__eq__` : Проверяет равенство двух матриц.
  - Арифметические операции: `__neg__` (отрицание), `__add__` (сложение), `__sub__` (вычитание), `__mul__` (умножение), `__rmul__` (умножение справа).
  - `__getitem__` , `__setitem__` : Получение и установка элементов матрицы по индексам.
  - `get_row` , `get_loc` , `set_row` , `set_loc` : Работа с строками и элементами по индексам.
  - `augment` : Расширение матрицы другой матрицей (добавление столбцов).
  - `transpose` : Транспонирование матрицы.
  - `determinant` : Вычисление определителя матрицы.
- **Значение:** `ABCMatrix` задает основу, которой должны следовать все классы матриц, что упрощает разработку и поддержку матричных операций.

`BaseMatrix` (Реализация плотной матрицы)

- **Описание:** Этот класс реализует матрицу, в которой все элементы хранятся в списке списков. Это стандартное представление, подходящее для матриц, где большинство элементов ненулевые.
- **Реализация методов:**
  - `__init__` : Принимает список списков и инициализирует матрицу. Преобразует элементы в нужный тип (`Decimal` , `complex` или `float`).
  - `size` , `dtype` , `to_list` , `__str__` , `__eq__` : Стандартные реализации, работающие со списком списков.

- Арифметические операции: Реализованы поэлементно или с использованием стандартных алгоритмов матричной арифметики.
- `__getitem__`, `__setitem__`, `get_row`, `get_loc`, `set_row`, `set_loc`: Осуществляют доступ к элементам списка списков.
- `augment`: Создает новую матрицу, объединяя столбцы.
- `transpose`: Создает новую матрицу с транспонированными элементами.
- `determinant`: Вычисляет определитель с использованием метода Гаусса.

- **Особенности:**
  - Простота реализации.
  - Неэффективность для больших разреженных матриц (тратится много памяти на хранение нулей).

`CSRMatrix` (Реализация разреженной матрицы)

- **Описание:** Этот класс реализует матрицу, используя формат Compressed Sparse Row (CSR). CSR - это эффективный способ хранения матриц, в которых большинство элементов равны нулю. Он хранит только ненулевые значения, их индексы столбцов и информацию о расположении в строках.
- **Реализация методов:**
  - `__init__`: Принимает либо список списков (и преобразует его в CSR), либо данные в формате CSR (значения, индексы столбцов, индексы начала строк).
  - `size`, `dtype`, `to_list`, `__str__`, `__eq__`: Реализованы с учетом структуры CSR. `to_list` преобразует CSR в плотное представление.
  - Арифметические операции: Часто преобразуют матрицы в плотное представление (`BaseMatrix`), выполняют операцию, а затем (иногда) преобразуют результат обратно в CSR. Это может быть неэффективно.
  - `__getitem__`, `__setitem__`, `get_row`, `get_loc`, `set_row`, `set_loc`: Реализованы для работы с представлением CSR, обеспечивая эффективный доступ к ненулевым элементам.
  - `augment`, `transpose`, `determinant`: Также часто используют преобразование в плотное представление для выполнения операций.
- **Особенности:**
  - Экономия памяти для разреженных матриц.
  - Более сложная реализация, чем `BaseMatrix`.
  - Операции могут быть неоптимизированы для разреженного формата.

Реализация лабораторной и PCA

Easy level

Метод Гаусса (`gauss_solver`)

```
def gauss_solver(A: 'Matrix', b: 'Matrix') -> List['Matrix']:
    # ... (код метода Гаусса)
    return basis
```

- **Описание:** Реализует метод Гаусса для решения систем линейных уравнений. Важен как вспомогательный алгоритм, который может пригодиться в более сложных реализациях PCA (например, для нахождения базиса подпространства).
- **Математическая корректность:**
  - Алгоритм приводит расширенную матрицу к ступенчатому виду.
  - Обрабатывает случаи несовместных и неопределенных систем.
  - Корректно находит базисные векторы для неопределенных систем.
- **Эффективность:**
  - Имеет сложность O(n^3) в общем случае.
  - Может быть оптимизирован для больших разреженных матриц, но текущая реализация этого не делает.

Центрирование данных (`center_data`)

```
def center_data(X: 'Matrix') -> 'Matrix':
    # ... (код центрирования)
    return X_centered
```

- **Описание:** Центрирует матрицу данных `X` путем вычитания среднего значения каждого столбца. Это критически важный шаг в PCA, поскольку он гарантирует, что главные компоненты отражают дисперсию данных относительно их среднего.
- **Математическая корректность:**
  - Среднее значение каждого столбца вычисляется правильно.
  - Вычитание среднего из каждого элемента столбца реализовано верно.
- **Эффективность:**
  - Имеет сложность O(n*m), где n - число строк, m - число столбцов.
  - В целом, достаточно эффективна, но можно оптимизировать доступ к элементам матрицы.
- **Ясность и стиль:**

Вычисление матрицы ковариаций (`covariance_matrix`)

```
def covariance_matrix(X_centered: 'Matrix') -> 'Matrix':
    # ... (код вычисления ковариационной матрицы)
    return C
```

- **Описание:** Вычисляет матрицу ковариаций для центрированных данных. Матрица ковариаций показывает, как различные признаки данных изменяются вместе. Это ключевой шаг в PCA, поскольку собственные векторы этой матрицы определяют главные компоненты.
- **Математическая корректность:**
  - Формула для вычисления ковариационной матрицы (X^T *X* 1/(n-1)) реализована верно.
  - Обработка случая с одной выборкой корректна (возвращает нулевую матрицу).
- **Эффективность:**
  - Умножение матриц имеет сложность, зависящую от реализации (в `matrix.py` это может быть O(n^3) для плотных матриц).
  - Для больших матриц это может стать узким местом.

Normal level

Методы нахождения собственных значений и векторов (`jacobi_eigen_decomposition`, `find_eigenvalues_and_vectors`, `find_eigenvalues`, `find_eigenvectors`)

```
def jacobi_eigen_decomposition(C: "Matrix", tol: float = 1e-7, max_iterations:
int = 100) -> tuple['Matrix', 'Matrix']:
    # ... (код метода Якоби)
    return A, V

def find_eigenvalues_and_vectors(C: "Matrix", tol: float = 1e-7,
max_iterations: int = 100) -> list[tuple[float, 'Matrix']]:
    # ...
    return eigen_pairs

def find_eigenvalues(C: "Matrix", tol: float = 1e-6, max_iterations: int =
100) -> list[float]:
    # ...

def find_eigenvectors(C: 'Matrix', tol: float = 1e-6, max_iterations: int =
```

```
100) -> List['Matrix']:
    # ...
```

- **Описание:** Реализуют метод Якоби для вычисления собственных значений и собственных векторов симметричной матрицы. Метод Якоби итеративно применяет вращения к матрице, чтобы привести её к диагональному виду, где диагональные элементы являются собственными значениями, а матрица вращений - матрицей собственных векторов.
- **Математическая корректность:**
  - Алгоритм Якоби корректно реализован.
  - Вычисление угла поворота (`angle`) выполнено правильно, с учетом особых случаев.
  - Функции `find_eigenvalues` и `find_eigenvectors` правильно извлекают собственные значения и векторы из результата `jacobi_eigen_decomposition` .
- **Эффективность:**
  - Метод Якоби хорошо подходит для симметричных матриц, но может быть медленнее, чем другие методы (например, метод степеней) для очень больших матриц.
  - Сходимость зависит от матрицы.

Вычисление доли объясненной дисперсии (`explained_variance_ratio`)

```
def explained_variance_ratio(eigenvalues: List[float], k: int) -> float:
    # ... (код вычисления доли объясненной дисперсии)
    return variance_k_components / total_variance
```

- **Описание:** Вычисляет, какая доля общей дисперсии данных объясняется первыми `k` главными компонентами. Это важная метрика для оценки того, сколько информации сохраняется при уменьшении размерности.
- **Математическая корректность:**
  - Формула для вычисления доли объясненной дисперсии реализована правильно.
  - Обработка краевых случаев (пустой список собственных значений, некорректное `k`) выполнена корректно.
- **Эффективность:**
  - Вычисление суммы - это операция O(n), где n - число собственных значений.
  - В целом, функция очень эффективна.

Hard level

Применение PCA к наборам данных (`apply_pca_to_dataset`)

```
def apply_pca_to_dataset(dataset_name: str, k: int) -> tuple[Matrix, float]:
    # ... (код применения PCA)
    return X_proj, explained_ratio
```

- **Описание:** Эта функция объединяет все предыдущие шаги для применения PCA к заданному набору данных. Она загружает данные, центрирует их, вычисляет матрицу ковариаций, находит собственные значения и векторы, выбирает главные компоненты и проецирует данные в новое пространство.
- **Математическая корректность:**
  - Все шаги PCA выполняются в правильном порядке.
  - Выбор главных компонент на основе собственных векторов корректен.
  - Проекция данных выполнена правильно (умножение центрированных данных на матрицу главных компонент).
- **Эффективность:**
  - Эффективность зависит от эффективности отдельных функций (особенно вычисления матрицы ковариаций и метода Якоби).
  - Загрузка данных также может влиять на производительность.

Визуализация результатов PCA

- В Notebook присутствуют примеры визуализации результатов PCA для наборов данных Iris и Wine.
- Визуализации строятся с использованием `matplotlib`.
- Графики позволяют увидеть проекции данных на главные компоненты, что помогает понять структуру данных и уменьшить их размерность.

Вычисление ошибки реконструкции (MSE) (`reconstruction_error`)

```
def reconstruction_error(X_orig: "Matrix", X_recon: "Matrix") -> float:
    # ... (код вычисления MSE)
    return mse
```

- **Описание:** Вычисляет среднеквадратическую ошибку (MSE) между исходной матрицей данных `X_orig` и её восстановленной версией `X_recon`. MSE является метрикой, которая показывает, насколько хорошо восстановленные данные соответствуют исходным. В контексте PCA, это позволяет оценить, сколько информации теряется при уменьшении размерности.
- **Математическая корректность:**

    - Формула MSE реализована верно: сумма квадратов разностей между соответствующими элементами матриц, делённая на общее число элементов.
    - Обработка случая несовпадающих размеров матриц (`ValueError`) корректна.
    - Обработка случая пустых матриц (возврат 0.0) логична.
- **Эффективность:**
    - Имеет сложность O(n*m), где n - число строк, m - число столбцов.
    - В целом, достаточно эффективна, но можно оптимизировать доступ к элементам матрицы.

Expert level

Автоматический выбор числа главных компонент (`auto_select_k`)

```
def auto_select_k(eigenvalues: List[float], threshold: float = 0.95) -> int:
    # ... (код автоматического выбора k)
    return k
```

- **Описание:** Определяет оптимальное число главных компонент `k` на основе заданного порога (`threshold`) объясненной дисперсии. Алгоритм выбирает минимальное `k` , при котором суммарная доля объясненной дисперсии превышает порог. Это полезно для автоматизации выбора `k` и избежания ручной настройки.
- **Математическая корректность:**
    - Алгоритм правильно сортирует собственные значения и вычисляет долю объясненной дисперсии на каждой итерации.
    - Обработка некорректных входных данных (`threshold` вне диапазона, отрицательные собственные значения) реализована с использованием `ValueError` .
    - Обработка случая нулевых собственных значений (возврат 0) логична.
- **Эффективность:**
    - Сортировка собственных значений - O(n*log(n)), где n - число собственных значений.
    - Вычисление суммы и итерация по собственным значениям - O(n).
    - В целом, функция достаточно эффективна.

Обработка пропущенных значений (`handle_missing_values`)

```
def handle_missing_values(X: list[list[float | None]]) -> list[list[float]]:
    # ... (код обработки пропущенных значений)
    return X_filled
```

- **Описание:** Заменяет пропущенные значения (`None`) в матрице данных средним значением по соответствующему столбцу. Это важный шаг предварительной обработки данных, поскольку PCA не работает с пропущенными значениями.
- **Математическая корректность:**
  - Среднее значение вычисляется правильно (сумма значений, делённая на их количество).
  - Обработка случая столбцов, содержащих только пропущенные значения (присваивание 0), корректна.
  - Функция корректно работает с матрицами, содержащими `float` и `None` .
- **Эффективность:**
  - Вычисление суммы и количества значений для каждого столбца - O(n*m), где n - число строк, m - число столбцов.
  - Заполнение пропущенных значений - O(n*m).
  - В целом, функция достаточно эффективна.

Обработка пропущенных значений (`handle_missing_values`)

```
def handle_missing_values(X: list[list[float | None]]) -> list[list[float]]:
    # ... (код обработки пропущенных значений)
    return X_filled
```

- **Описание:** Заменяет пропущенные значения (`None`) в матрице данных `X` средним значением по каждому столбцу. Представляет собой важный этап предварительной обработки данных, так как PCA (и многие другие алгоритмы) не могут обрабатывать пропущенные значения напрямую.
- **Математическая корректность:**
  - Среднее значение для каждого столбца вычисляется как сумма всех не- `None` значений в столбце, делённая на количество этих значений. Это стандартное и корректное определение среднего.
  - Если столбец содержит только пропущенные значения (`None`), функция присваивает 0 этому столбцу. Это разумный подход, чтобы избежать ошибок деления на ноль и обеспечить численную стабильность.
- **Эффективность:**
  - Вычисление суммы и количества не- `None` значений для каждого столбца требует прохода по всем элементам матрицы, что даёт временную сложность O(n*m), где n - число строк, а m - число столбцов.
  - Заполнение пропущенных значений также требует прохода по всем элементам матрицы в худшем случае, что также даёт временную сложность O(n*m).

- Следовательно, общая временная сложность функции составляет O(n*m). В целом, для обработки матриц это считается достаточно эффективным, особенно если учесть, что это операция предварительной обработки данных.

Добавление шума и сравнение результатов PCA (`add_noise_and_compare`)

```
def add_noise_and_compare(X: "Matrix", noise_level: float = 0.1):
    # ... (код добавления шума и сравнения PCA)
    # Выводит результаты сравнения
```

- **Описание:** Исследует влияние добавления случайного шума к данным на результаты PCA. Принимает матрицу данных X и уровень шума noise_level в качестве входных данных. Выполняет PCA на исходных данных и на данных с добавленным шумом, а затем сравнивает долю объясненной дисперсии и собственные значения.
- **Математическая корректность:**
  - PCA вычисляется как на исходных, так и на зашумленных данных, обеспечивая корректное сравнение.
  - Шум добавляется с масштабированием относительно стандартного отклонения признаков, что является статистически обоснованным подходом.
  - Функция корректно обрабатывает случай нулевого стандартного отклонения, предотвращая ошибки.
- **Эффективность:**
  - Вычисление PCA имеет сложность, зависящую от реализации функции `pca` (не предоставлена).
  - Вычисление стандартного отклонения - O(n*m).
  - Генерация и добавление шума - O(n*m).
  - В целом, сложность определяется операцией PCA, но остальные операции линейны по размеру данных.

Применение PCA к датасету (`apply_pca_to_dataset`)

```
def apply_pca_to_dataset(dataset_name: str, k: int):
    # ... (код применения PCA к датасету)
    return X_proj, explained_ratio
```

- **Описание:** Применяет PCA к указанному датасету из scikit-learn (`iris` или `wine`). Принимает название датасета (`dataset_name`) и количество главных компонент (`k`) в качестве входных данных. Загружает датасет, выполняет PCA и возвращает спроецированные данные и долю объясненной дисперсии.

- **Математическая корректность:**
  - Загрузка данных и применение PCA (через функцию `pca`) выполнены корректно.
  - Возвращаемая доля объясненной дисперсии является стандартной метрикой для оценки результатов PCA.
- **Эффективность:**
  - Сложность PCA определяется функцией `pca`.
  - Остальные операции (преобразование данных) линейны по размеру данных.