

# Введение

Алгоритм `svdlapack` представляет собой обёртку над подпрограммами библиотеки LAPACK для вычисления сингулярного разложения матрицы (SVD).

Библиотека LAPACK предоставляет высокоэффективные реализации алгоритмов линейной алгебры, оптимизированные для современных вычислительных архитектур. В проекте SLEPc (Scalable Library for Eigenvalue Problem Computations) реализован интерфейс к этим алгоритмам, позволяющий эффективно вычислять сингулярные разложения как плотных, так и разреженных матриц.

## Алгоритм вычисления SVD

Алгоритм `svdlapack` использует подпрограммы LAPACK для вычисления SVD плотных матриц. Основные этапы алгоритма следующие:

### 1. Приведение матрицы к двухдиагональной форме

Первым шагом является приведение исходной матрицы  $A$  к двухдиагональной форме  $B$  с помощью ортогональных преобразований (отражений Хаусхолдера). Это записывается как:

$$A = Q_L B Q_R^T,$$

где:

- $Q_L$  — ортогональная матрица размера  $m \times m$ .
- $Q_R$  — ортогональная матрица размера  $n \times n$ .
- $B$  — двухдиагональная матрица размера  $m \times n$ , которая имеет ненулевые элементы только на главной диагонали и первой поддиагонали (для  $m \geq n$ ) или наддиагонали (для  $m < n$ ):

$$B = \begin{pmatrix} d_1 & e_1 & & & \\ & d_2 & e_2 & & \\ & & \ddots & \ddots & \\ & & & d_{n-1} & e_{n-1} \\ & & & & d_n \end{pmatrix}.$$

Преобразование к двухдиагональной форме существенно сокращает вычислительную сложность последующих шагов, поскольку операции теперь выполняются над матрицей с простой структурой.

### 2. Вычисление SVD двухдиагональной матрицы

Для двухдиагональной матрицы  $B$  сингулярные значения и векторы вычисляются с использованием итеративных алгоритмов, специально адаптированных для такой структуры. Основные методы включают:

- **QR-алгоритм для двухдиагональных матриц:** Этот алгоритм применяет последовательность ортогональных преобразований для диагонализации двухдиагональной матрицы. Поскольку  $B$  имеет простую структуру, каждое итеративное преобразование может быть выполнено эффективно.

- **Алгоритм Кобылыша–Рейша (Divide and Conquer):** Этот метод разбивает матрицу на подматрицы меньшего размера, решает задачу для них и затем объединяет решения.
- **Метод обобщённых квадратов (Golub–Kahan):** Используется для вычисления сингулярных значений и векторов путём решения последовательности линейных систем.

В результате этого шага получаются диагональная матрица  $\Sigma$  с сингулярными значениями и ортогональные матрицы  $U_B$  и  $V_B$ , содержащие сингулярные векторы bidiagonal матрицы  $B$ :

$$B = U_B \Sigma V_B^\top.$$

**Нужно обратить внимание**, что код не содержит явной реализации конкретных алгоритмов, таких как QR-алгоритм для двухдиагональных матриц, Алгоритм Кобылыша–Рейша и метод Голуба–Кахана. Вместо этого, при вызове функции `DSSolve` через объект `DS`, LAPACK самостоятельно выбирает наиболее эффективный алгоритм для данной задачи.

### 3. Обратное преобразование сингулярных векторов

После вычисления SVD двухдиагональной матрицы  $B$  необходимо преобразовать полученные сингулярные векторы обратно к исходной матрице  $A$ :

$$\begin{aligned} U &= Q_L U_B, \\ V &= Q_R V_B. \end{aligned}$$

Это достигается умножением сингулярных векторов  $U_B$  и  $V_B$  на ортогональные матрицы  $Q_L$  и  $Q_R$ , которые были получены на первом шаге. Поскольку  $Q_L$  и  $Q_R$  являются ортогональными, это преобразование не влияет на ортогональность сингулярных векторов.

### Обобщённое сингулярное разложение (GSVD)

Обобщённое сингулярное разложение используется для пар матриц  $A$  и  $B$  размера  $m \times n$  и  $p \times n$  соответственно. Цель GSVD состоит в нахождении матриц  $U$ ,  $V$  и  $X$ , удовлетворяющих:

$$\begin{aligned} A &= U \Sigma_A X^\top, \\ B &= V \Sigma_B X^\top, \end{aligned}$$

где:

- $U$  — ортогональная матрица размера  $m \times m$ .
- $V$  — ортогональная матрица размера  $p \times p$ .
- $X$  — обратимая матрица размера  $n \times n$ .
- $\Sigma_A$  и  $\Sigma_B$  — диагональные матрицы размера  $m \times n$  и  $p \times n$  соответственно, удовлетворяющие условию:

$$\Sigma_A^\top \Sigma_A + \Sigma_B^\top \Sigma_B = I_n.$$

В GSVD сингулярные значения характеризуют относительные вклады матриц  $A$  и  $B$  в общее решение. Этот метод особенно полезен в задачах, где необходимо учитывать влияние двух различных систем или наборов данных.

## Гармоническое сингулярное разложение (HSVD)

Гармоническое сингулярное разложение применяется в задачах, где матрица  $A$  связана с метрикой, заданной положительно определённой матрицей  $\Omega$ . Цель HSVD — найти разложение:

$$A = U \Sigma V^T,$$

при условии, что:

- Векторы  $U$  ортонормированы относительно метрики  $\Omega$ :

$$U^T \Omega U = I_p,$$

где  $I_p$  — единичная матрица размера  $p \times p$ .

- Векторы  $V$  ортонормированы в стандартном смысле:

$$V^T V = I_p.$$

При этом сингулярные значения  $\sigma_i$  и векторы  $u_i$ ,  $v_i$  удовлетворяют следующим соотношениям:

$$\begin{aligned} A v_i &= \sigma_i \Omega u_i, \\ A^T \Omega u_i &= \sigma_i v_i. \end{aligned}$$

HSVD находит применение в задачах, где необходимо учитывать весовые коэффициенты или корреляции между элементами данных, например, во взвешенном методе наименьших квадратов или в обработке сигналов.

## 4. Имплементация

В этой секции рассмотрим, как алгоритм вычисления сингулярного разложения, описанный ранее, реализован в коде файла `svdlapack.c` из библиотеки SLEPc. Этот файл представляет собой обёртку над подпрограммами LAPACK для вычисления SVD, GSVD и HSVD.

Основные функции, реализующие алгоритм:

- `SVDSsetUp_LAPACK`
- `SVDSolve_LAPACK`
- `SVDSolve_LAPACK_GSVD`
- `SVDSolve_LAPACK_HSVD`
- `SVDSsetDSType_LAPACK`
- `SVDCreate_LAPACK`

## 4.1 Функция `SVDSetUp_LAPACK`

Эта функция отвечает за инициализацию и настройку объекта SVD для использования методов LAPACK. Основные действия функции:

### 1. Получение размеров матриц:

- Получение размеров исходной матрицы  $A$  с помощью `MatGetSize`.
- Если вычисляется обобщённое SVD (GSVD), то также получаются размеры матрицы  $B$ .

### 2. Установка параметров:

- Установка параметра `ncv`, который определяет количество сингулярных значений для вычисления.
- Для стандартного SVD `ncv` устанавливается равным  $N$  (числу столбцов матрицы  $A$ ).
- Для GSVD `ncv` устанавливается как минимальное из  $M$ ,  $N$  и  $P$ , где  $P$  — число строк матрицы  $B$ .

### 3. Выделение памяти:

- Выделение памяти для хранения сингулярных векторов и значений с помощью `SVDAAllocateSolution`.

### 4. Инициализация DS:

- Инициализация объекта DS (Direct Solver), который будет использоваться для решения задачи SVD с помощью LAPACK.

### 5. Проверка параметров:

- Проверка на неподдерживаемые функции и установка необходимых флагов.

## 4.2 Функция `SVDSolve_LAPACK`

Эта функция реализует основной алгоритм вычисления SVD для плотных матриц:

### 1. Сбор матрицы на одном процессе:

- Используется функция `MatCreateRedundantMatrix` для создания копии распределённой матрицы  $A$  на одном процессе.

### 2. Конвертация матрицы:

- Полученная матрица преобразуется в формат `MATSEQDENSE` с помощью `MatConvert`.

### 3. Настройка объекта DS:

- Матрица  $A$  копируется в объект `ds` с помощью `DSGetMat` и `MatCopy`.
- Устанавливается состояние объекта `ds` как `DS_STATE_RAW`.

### 4. Решение задачи SVD:

- Вызывается функция `DSSolve`, которая вычисляет сингулярные значения и векторы матрицы  $A$ .
5. **Сортировка результатов:**
    - Функция `DSSort` сортирует полученные сингулярные значения и векторы.
  6. **Копирование результатов:**
    - Сингулярные значения и векторы извлекаются из объекта `ds` и копируются в соответствующие структуры объекта `svd`.

### 4.3 Функция `SVDSolve_LAPACK_GSVD`

Эта функция отвечает за вычисление обобщённого сингулярного разложения (GSVD):

1. **Сбор матриц  $A$  и  $B$ :**
  - Матрицы  $A$  и  $B$  собираются на одном процессе и преобразуются в формат `MATSEQDENSE`.
2. **Настройка объекта `DS`:**
  - Матрицы  $A$  и  $B$  устанавливаются в объект `ds` с помощью `DSGetMat`.
  - Устанавливается состояние объекта `ds` как `DS_STATE_RAW`.
3. **Решение задачи GSVD:**
  - Вызывается функция `DSSolve`, которая вычисляет обобщённые сингулярные значения и векторы.
4. **Копирование результатов:**
  - Полученные сингулярные значения и векторы копируются в объект `svd`.

### 4.4 Функция `SVDSolve_LAPACK_HSVD`

Эта функция реализует гармоническое сингулярное разложение (HSVD):

1. **Сбор матрицы  $A$ :**
  - Матрица  $A$  собирается и конвертируется в последовательный формат.
2. **Установка весов  $\Omega$ :**
  - Вектор весов  $\omega$  устанавливается в объекте `ds` с помощью `DSGetMatAndColumn`.
3. **Решение задачи HSVD:**
  - Вызывается функция `DSSolve`, учитывающая метрику  $\Omega$ .
4. **Копирование результатов:**
  - Результаты копируются в объект `svd`, сохраняется информация о знаках сингулярных значений.

## 4.5 Функция `SVDSolve_LAPACK`

Эта функция устанавливает тип объекта `DS` в зависимости от решаемой задачи (SVD, GSVD, HSVD).

## 4.6 Функция `SVDCreate_LAPACK`

Эта функция создаёт контекст для использования метода LAPACK в SVD.

## 4.7 Приведение матрицы к двухдиагональной форме в коде

Важным шагом алгоритма вычисления SVD является приведение исходной матрицы  $A$  к двухдиагональной форме  $B$ . В коде файла `svdlapack.c` этот процесс реализован через вызовы подпрограмм LAPACK внутри объекта `DS` (Direct Solver).

В функции `SVDSolve_LAPACK` происходит следующее:

### 1. Подготовка матрицы:

- Матрица  $A$  собирается на одном процессе с помощью `MatCreateRedundantMatrix`.
- Конвертация матрицы в формат `MATSEQDENSE` с помощью `MatConvert`.
- Полученная матрица устанавливается в объекте `DS` с помощью `DSGetMat`.

### 2. Инициализация объекта `DS`:

- Тип объекта устанавливается как `DSSVD` через функцию `DSSetType(svd->ds, DSSVD)`.

### 3. Вызов функции `DSSolve`:

- Функция `DSSolve` вызывает соответствующие подпрограммы LAPACK для вычисления SVD.

### 4. Выбор алгоритма LAPACK:

- В зависимости от размеров матрицы и настроек, `DSSolve` может вызвать следующие подпрограммы:
  - `?GESVD` (например, `DGESVD`): Стандартная подпрограмма для вычисления SVD, использующая последовательную редукцию к двухдиагональной форме.
  - `?GESDD` (например, `DGESDD`): Подпрограмма, реализующая метод разделения и завоевания (Divide and Conquer).
- Здесь символ `?` заменяется на соответствующий префикс типа данных: `s`, `d`, `c` или `z`.

### 5. Редукция к двухдиагональной форме:

- Подпрограммы `?GESVD` и `?GESDD` внутри себя приводят матрицу  $A$  к двухдиагональной форме  $B$  с помощью ортогональных преобразований.
- Редукция выполняется с использованием подпрограммы `?GEBRD` (например, `DGEBRD`), которая применяет отражения Хаусхолдера.

- Результат редукции представляется в виде:

$$A = Q_L B Q_R^\top,$$

где  $Q_L$  и  $Q_R$  — ортогональные матрицы, а  $B$  — двухдиагональная матрица.

## 6. Вычисление SVD двухдиагональной матрицы:

- Для двухдиагональной матрицы  $B$  используются специализированные подпрограммы LAPACK, соответствующие трём основным методам:
  - **QR-алгоритм для двухдиагональных матриц:**
    - \* Реализован в подпрограмме ?BDSQR.
    - \* Использует последовательность вращений Гивенса для диагонализации  $B$ .
  - **Алгоритм разделения и завоевания (Divide and Conquer):**
    - \* Реализован в подпрограмме ?BDSDC.
    - \* Разбивает  $B$  на подматрицы меньшего размера, вычисляет SVD для них и объединяет результаты.
  - **Метод обобщённых квадратов (Golub–Kahan):**
    - \* Хотя LAPACK не предоставляет прямой реализации метода Голуба–Кахана для SVD двухдиагональной матрицы, аналогичные идеи используются в алгоритмах, оптимизированных для вычисления сингулярных значений и векторов.
- Выбор конкретной подпрограммы может зависеть от настроек и оптимизации LAPACK на данной системе.

## 7. Обратное преобразование сингулярных векторов:

- После получения сингулярных векторов  $U_B$  и  $V_B$  для матрицы  $B$ , необходимо преобразовать их к сингулярным векторам исходной матрицы  $A$ :

$$U = Q_L U_B, \quad V = Q_R V_B.$$

- Это преобразование выполняется внутри подпрограмм LAPACK, используя накопленные ортогональные преобразования.

## 8. Копирование результатов:

- После завершения работы DSSolve сингулярные значения  $\sigma_i$  и векторы  $U$ ,  $V$  извлекаются из объекта DS и копируются в соответствующие структуры объекта svd.

В коде функции SVDSolve\_LAPACK (листинг ??) можно увидеть вызов DSSolve, который внутри себя выполняет описанные шаги по редукции матрицы и вычислению SVD.

## 4.8 Структуры данных и их значение

- **SVD:** Главная структура, представляющая объект сингулярного разложения. Хранит информацию о матрицах  $A$ ,  $B$ , параметрах алгоритма и результатах вычислений.
- **DS:** Структура Direct Solver, используется для решения плотных матричных задач с помощью LAPACK.

- **Mat**: Структура для представления матриц в PETSc/SLEPc.
- **Vec**: Структура для представления векторов. Используется для хранения сингулярных векторов.
- **BV**: Блок векторов, представляет собой набор векторов для эффективных операций.

## 4.9 Referenses

- **MatCreateRedundantMatrix**: Создаёт копию распределённой матрицы на одном процессе. [Документация]
- **MatConvert**: Конвертирует матрицу в другой формат. [Документация]
- **DSSolve**: Решает задачу собственных значений или SVD. [Документация]
- **DSSort**: Сортирует вычисленные значения. [Документация]
- **DSSynchronize**: Синхронизирует данные между процессами. [Документация]
- **BVGetColumn** и **BVRestoreColumn**: Доступ к столбцу блок-вектора. [Документация]
- **VecGetArray** и **VecRestoreArray**: Доступ к внутреннему массиву вектора. [Документация]

## Реализация

```

1  /*
2      - - - - -
3      SLEPc - Scalable Library for Eigenvalue Problem Computations
4      Copyright (c) 2002-, Universitat Politecnica de Valencia, Spain
5
6      Этот файл является частью SLEPc.
7      SLEPc распространяется по лицензии BSD с 2 пунктами (см. LICENSE).
8      - - - - -
9  */
10
11 /*
12     Этот файл реализует обертку над подпрограммами LAPACK для вычисления SVD
13 */
14
15 #include <slepc/private/svdimpl.h>
16 #include <slepcblaslapack.h>
17
18 /* Функция для настройки SVD с использованием LAPACK */
19 static PetscErrorCode SVDSetup_LAPACK(SVD svd)
20 {
21     PetscInt      M,N,P=0;
22
23     PetscFunctionBegin;
24     PetscCall(MatGetSize(svd->A,&M,&N)); /* Получаем размеры матрицы A */
25
26     if (!svd->isgeneralized)              /* Если это не обобщенное SVD */
27         svd->ncv = N;                    /* Устанавливаем ncv равным числу ст
                                         олбцов N */

```



```

28 else {
29     PetscCall(MatGetSize(svd->OPb,&P,NULL)); /* Получаем число строк матриц
        ы B для GSVD */
30     svd->ncv = PetscMin(M,PetscMin(N,P)); /* ncv = min(M, N, P) */
31 }
32
33 if (svd->mpd!=PETSC_DETERMINE) /* Если параметр mpd задан */
34     PetscCall(PetscInfo(svd,"Warning: parameter mpd ignored\n")); /* Предуп
        реждаем, что mpd игнорируется */
35
36 SVDCheckUnsupported(svd,SVD_FEATURE_STOPPING); /* Проверяем на неподдержи
        ваемые функции */
37
38 if (svd->max_it==PETSC_DETERMINE) /* Если max_it не задан */
39     svd->max_it = 1; /* Устанавливаем max_it = 1 */
40
41 svd->leftbasis = PETSC_TRUE; /* Устанавливаем флаг использования
        левых сингулярных векторов */
42
43 PetscCall(SVDAllocateSolution(svd,0)); /* Выделяем память для решения */
44
45 PetscCall(DSAllocate(svd->ds,PetscMax(N,PetscMax(M,P)))); /* Выделяем пам
        ять для объекта DS */
46
47 PetscFunctionReturn(PETSC_SUCCESS); /* Завершаем функцию успешно */
48 }
49
50 /* Функция для решения SVD с использованием LAPACK */
51 static PetscErrorCode SVDSolve_LAPACK(SVD svd)
52 {
53     PetscInt M,N,n,i,j,k,ld,lowu,lowv,highu,highv;
54     Mat A,Ar,mat;
55     Vec u,v;
56     PetscScalar *pU,*pV,*pu,*pv,*w;
57
58     PetscFunctionBegin;
59     PetscCall(DSGetLeadingDimension(svd->ds,&ld)); /* Получаем ведущую размер
        ность для DS */
60
61     /* Создаем копию матрицы OP на одном процессе */
62     PetscCall(MatCreateRedundantMatrix(svd->OP,0,PETSC_COMM_SELF,
        MAT_INITIAL_MATRIX,&Ar));
63     PetscCall(MatConvert(Ar,MATSEQDENSE,MAT_INITIAL_MATRIX,&mat)); /* Конверт
        ируем в последовательный плотный формат */
64     PetscCall(MatDestroy(&Ar)); /* Уничтожаем промежуточную матрицу */
65
66     PetscCall(MatGetSize(mat,&M,&N)); /* Получаем размеры матрицы mat */
67
68     PetscCall(DSSetDimensions(svd->ds,M,0,0)); /* Устанавливаем размеры DS
        */
69     PetscCall(DSSVDSetDimensions(svd->ds,N)); /* Устанавливаем размерность
        для SVD в DS */
70
71     PetscCall(DSGetMat(svd->ds,DS_MAT_A,&A)); /* Получаем матрицу A из DS
        */
72     PetscCall(MatCopy(mat,A,SAME_NONZERO_PATTERN)); /* Копируем содержимое
        mat в A */
73     PetscCall(DSRestoreMat(svd->ds,DS_MAT_A,&A)); /* Возвращаем матрицу A в
        DS */
74

```

```

75 PetscCall(DSSetState(svd->ds,DS_STATE_RAW)); /* Устанавливаем состояни
   е DS как RAW */
76
77 n = PetscMin(M,N); /* n = min(M, N) */
78
79 PetscCall(PetscMalloc1(n,&w)); /* Выделяем память для мас
   сива сингулярных значений w */
80
81 PetscCall(DSSolve(svd->ds,w,NULL)); /* Вычисляем сингулярные з
   начения и векторы */
82 PetscCall(DSSort(svd->ds,w,NULL,NULL,NULL,NULL)); /* Сортируем сингулярны
   е значения */
83 PetscCall(DSSynchronize(svd->ds,w,NULL)); /* Синхронизируем данные м
   ежду процессами */
84
85 /* Копируем сингулярные векторы */
86 PetscCall(DSGetArray(svd->ds,DS_MAT_U,&pU)); /* Получаем массив с левым
   и сингулярными векторами */
87 PetscCall(DSGetArray(svd->ds,DS_MAT_V,&pV)); /* Получаем массив с правы
   ми сингулярными векторами */
88
89 for (i=0;i<n;i++) {
90     if (svd->which == SVD_SMALLEST) /* Определяем порядок сохр
   анения сингулярных значений */
91         k = n - i - 1;
92     else
93         k = i;
94
95     svd->sigma[k] = PetscRealPart(w[i]); /* Сохраняем сингулярное з
   начение */
96
97     PetscCall(BVGetColumn(svd->U,k,&u)); /* Получаем k-й столбец из
   блока векторов U */
98     PetscCall(BVGetColumn(svd->V,k,&v)); /* Получаем k-й столбец из
   блока векторов V */
99
100    PetscCall(VecGetOwnershipRange(u,&lowu,&highu)); /* Получаем диапазон и
   ндексов для вектора u */
101    PetscCall(VecGetOwnershipRange(v,&lowv,&highv)); /* Получаем диапазон и
   ндексов для вектора v */
102
103    PetscCall(VecGetArray(u,&pu)); /* Получаем указатель на д
   анные вектора u */
104    PetscCall(VecGetArray(v,&pv)); /* Получаем указатель на д
   анные вектора v */
105
106    if (M>=N) {
107        /* Если число строк больше или равно числу столбцов */
108        for (j=lowu;j<highu;j++) pu[j-lowu] = pU[i*ld+j]; /* Копируем данные
   из pU в pu */
109        for (j=lowv;j<highv;j++) pv[j-lowv] = pV[i*ld+j]; /* Копируем данные
   из pV в pv */
110    } else {
111        /* Если число столбцов больше числа строк */
112        for (j=lowu;j<highu;j++) pu[j-lowu] = pV[i*ld+j]; /* Копируем данные
   из pV в pu */
113        for (j=lowv;j<highv;j++) pv[j-lowv] = pU[i*ld+j]; /* Копируем данные
   из pU в pv */
114    }
115

```

```

116     PetscCall(VecRestoreArray(u,&pu));           /* Возвращаем доступ к дан
ным вектора u */
117     PetscCall(VecRestoreArray(v,&pv));           /* Возвращаем доступ к дан
ным вектора v */
118
119     PetscCall(BVRestoreColumn(svd->U,k,&u));     /* Возвращаем столбец u в
блок векторов U */
120     PetscCall(BVRestoreColumn(svd->V,k,&v));     /* Возвращаем столбец v в
блок векторов V */
121 }
122
123 PetscCall(DSRestoreArray(svd->ds,DS_MAT_U,&pU)); /* Возвращаем массив pU
в DS */
124 PetscCall(DSRestoreArray(svd->ds,DS_MAT_V,&pV)); /* Возвращаем массив pV
в DS */
125
126 svd->nconv = n;                                /* Устанавливаем число на
йденных сингулярных значений */
127 svd->its = 1;                                  /* Устанавливаем число ит
ераций (1, так как метод прямой) */
128 svd->reason = SVD_CONVERGED_TOL;               /* Устанавливаем причину
завершения */
129
130 PetscCall(MatDestroy(&mat));                   /* Уничтожаем матрицу mat
*/
131 PetscCall(PetscFree(w));                       /* Освобождаем память, вы
деленную под w */
132
133 PetscFunctionReturn(PETSC_SUCCESS);            /* Завершаем функцию успе
шно */
134 }
135
136 /* Функция для решения обобщенного SVD с использованием LAPACK */
137 static PetscErrorCode SVDSolve_LAPACK_GSVD(SVD svd)
138 {
139     PetscInt      nsv,m,n,p,i,j,mlocal,plocal,ld,lowx,lowu,lowv,highx;
140     Mat           Ar,A,Ads,Br,B,Bds;
141     Vec           uv,x;
142     PetscScalar   *U,*V,*X,*px,*puv,*w;
143
144     PetscFunctionBegin;
145     PetscCall(DSGetLeadingDimension(svd->ds,&ld)); /* Получаем ведущую ра
змерность DS */
146
147     /* Создаем копию матрицы OP на одном процессе */
148     PetscCall(MatCreateRedundantMatrix(svd->OP,0,PETSC_COMM_SELF,
MAT_INITIAL_MATRIX,&Ar));
149     PetscCall(MatConvert(Ar,MATSEQDENSE,MAT_INITIAL_MATRIX,&A)); /* Конверт
ируем в последовательный плотный формат */
150     PetscCall(MatDestroy(&Ar));                 /* Уничтож
аем промежуточную матрицу */
151
152     /* Создаем копию матрицы OPb на одном процессе */
153     PetscCall(MatCreateRedundantMatrix(svd->OPb,0,PETSC_COMM_SELF,
MAT_INITIAL_MATRIX,&Br));
154     PetscCall(MatConvert(Br,MATSEQDENSE,MAT_INITIAL_MATRIX,&B)); /* Конверт
ируем в последовательный плотный формат */
155     PetscCall(MatDestroy(&Br));                 /* Уничтож
аем промежуточную матрицу */
156

```

```

157 PetscCall(MatGetSize(A,&m,&n)); /* Получаем размеры матри
    цы A */
158 PetscCall(MatGetLocalSize(svd->OP,&mlocal,NULL)); /* Получаем локальный р
    азмёр матрицы OP */
159 PetscCall(MatGetLocalSize(svd->OPb,&plocal,NULL)); /* Получаем локальный р
    азмёр матрицы OPb */
160 PetscCall(MatGetSize(B,&p,NULL)); /* Получаем размер матриц
    ы B */
161
162 PetscCall(DSSetDimensions(svd->ds,m,0,0)); /* Устанавливаем размеры
    DS */
163 PetscCall(DSGSVDSsetDimensions(svd->ds,n,p)); /* Устанавливаем размеры
    для GSVD в DS */
164
165 PetscCall(DSGetMat(svd->ds,DS_MAT_A,&Ads)); /* Получаем матрицу A из
    DS */
166 PetscCall(MatCopy(A,Ads,SAME_NONZERO_PATTERN)); /* Копируем A в Ads */
167 PetscCall(DSRestoreMat(svd->ds,DS_MAT_A,&Ads)); /* Возвращаем Ads в DS */
168
169 PetscCall(DSGetMat(svd->ds,DS_MAT_B,&Bds)); /* Получаем матрицу B из
    DS */
170 PetscCall(MatCopy(B,Bds,SAME_NONZERO_PATTERN)); /* Копируем B в Bds */
171 PetscCall(DSRestoreMat(svd->ds,DS_MAT_B,&Bds)); /* Возвращаем Bds в DS */
172
173 PetscCall(DSSetState(svd->ds,DS_STATE_RAW)); /* Устанавливаем состояни
    е DS как RAW */
174
175 nsv = PetscMin(n,PetscMin(p,m)); /* nsv = min(n, p, m) */
176
177 PetscCall(PetscMalloc1(nsv,&w)); /* Выделяем память для ма
    ссива сингулярных значений w */
178
179 PetscCall(DSSolve(svd->ds,w,NULL)); /* Вычисляем сингулярные
    значения и векторы */
180 PetscCall(DSSort(svd->ds,w,NULL,NULL,NULL,NULL)); /* Сортируем сингулярные
    значения */
181 PetscCall(DSSynchronize(svd->ds,w,NULL)); /* Синхронизируем данные
    между процессами */
182 PetscCall(DSGetDimensions(svd->ds,NULL,NULL,NULL,&nsv)); /* Получаем обнов
    лённое значение nsv */
183
184 /* Копируем сингулярные векторы */
185 PetscCall(MatGetOwnershipRange(svd->OP,&lowu,NULL)); /* Получаем диапа
    зон индексов для матрицы OP */
186 PetscCall(MatGetOwnershipRange(svd->OPb,&lowv,NULL)); /* Получаем диапа
    зон индексов для матрицы OPb */
187
188 PetscCall(DSGetArray(svd->ds,DS_MAT_X,&X)); /* Получаем массив
    X из DS */
189 PetscCall(DSGetArray(svd->ds,DS_MAT_U,&U)); /* Получаем массив
    U из DS */
190 PetscCall(DSGetArray(svd->ds,DS_MAT_V,&V)); /* Получаем массив
    V из DS */
191
192 for (j=0;j<nsv;j++) {
193     svd->sigma[j] = PetscRealPart(w[j]); /* Сохраняем сингу
    лярное значение */
194
195     PetscCall(BVGetColumn(svd->V,j,&x)); /* Получаем j-й ст
    олбец из блока векторов V */

```

```

196     PetscCall(VecGetOwnershipRange(x,&lowx,&highx)); /* Получаем диапазон
он индексов для вектора x */
197     PetscCall(VecGetArrayWrite(x,&px)); /* Получаем указате
ель на данные вектора x */
198     for (i=lowx;i<highx;i++) px[i-lowx] = X[i+j*ld]; /* Копируем данные
из X в px */
199     PetscCall(VecRestoreArrayWrite(x,&px)); /* Возвращаем доступ
к данным вектора x */
200     PetscCall(BVRestoreColumn(svd->V,j,&x)); /* Возвращаем стол
бец x в блок векторов V */
201
202     PetscCall(BVGetColumn(svd->U,j,&uv)); /* Получаем j-й столбец
из блока векторов U */
203     PetscCall(VecGetArrayWrite(uv,&puv)); /* Получаем указате
ель на данные вектора uv */
204
205     for (i=0;i<mlocal;i++) puv[i] = U[i+lowu+j*ld]; /* Копируем данные
из U в puv для матрицы A */
206     for (i=0;i<plocal;i++) puv[i+mlocal] = V[i+lowv+j*ld]; /* Копируем данные
из V в puv для матрицы B */
207
208     PetscCall(VecRestoreArrayWrite(uv,&puv)); /* Возвращаем доступ
к данным вектора uv */
209     PetscCall(BVRestoreColumn(svd->U,j,&uv)); /* Возвращаем стол
бец uv в блок векторов U */
210 }
211
212 PetscCall(DSRestoreArray(svd->ds,DS_MAT_X,&X)); /* Возвращаем массив
ив X в DS */
213 PetscCall(DSRestoreArray(svd->ds,DS_MAT_U,&U)); /* Возвращаем массив
ив U в DS */
214 PetscCall(DSRestoreArray(svd->ds,DS_MAT_V,&V)); /* Возвращаем массив
ив V в DS */
215
216 svd->nconv = nsv; /* Устанавливаем число найденных сингулярных значений */
217 svd->its = 1; /* Устанавливаем число итераций (1, так как метод прямой) */
218 svd->reason = SVD_CONVERGED_TOL; /* Устанавливаем причину завершения */
219
220 PetscCall(MatDestroy(&A)); /* Уничтожаем матрицу A */
221 PetscCall(MatDestroy(&B)); /* Уничтожаем матрицу B */
222 PetscCall(PetscFree(w)); /* Освобождаем память, выделенную под w */
223
224 PetscFunctionReturn(PETSC_SUCCESS); /* Завершаем функцию успешно */
225 }
226
227 /* Функция для решения гармонического SVD с использованием LAPACK */
228 static PetscErrorCode SVDSolve_LAPACK_HSVD(SVD svd)
229 {
230     PetscInt M,N,n,i,j,k,ld,lowu,lowv,highu,highv;
231     Mat A,Ar,mat,D;
232     Vec u,v,vomega;
233     PetscScalar *pU,*pV,*pu,*pv,*w;
234     PetscReal *pD;

```

```

235
236 PetscFunctionBegin;
237 PetscCall(DSGetLeadingDimension(svd->ds,&ld)); /* Получаем ведущую
           ю размерность DS */
238
239 /* Создаем копию матрицы OP на одном процессе */
240 PetscCall(MatCreateRedundantMatrix(svd->OP,0,PETSC_COMM_SELF,
           MAT_INITIAL_MATRIX,&Ar));
241 PetscCall(MatConvert(Ar,MATSEQDENSE,MAT_INITIAL_MATRIX,&mat)); /* Конверт
           ируем в последовательный плотный формат */
242 PetscCall(MatDestroy(&Ar)); /* Уни
           чтожаем промежуточную матрицу */
243
244 PetscCall(MatGetSize(mat,&M,&N)); /* Получаем раз
           меры матрицы mat */
245
246 PetscCall(DSSetDimensions(svd->ds,M,0,0)); /* Устанавливаем
           размеры DS */
247 PetscCall(DSHSVDSVDSetDimensions(svd->ds,N)); /* Устанавливаем
           размеры для HSVD в DS */
248
249 PetscCall(DSGetMat(svd->ds,DS_MAT_A,&A)); /* Получаем мат
           рицу A из DS */
250 PetscCall(MatCopy(mat,A,SAME_NONZERO_PATTERN)); /* Копируем mat
           в A */
251 PetscCall(DSRestoreMat(svd->ds,DS_MAT_A,&A)); /* Возвращаем A
           в DS */
252
253 /* Устанавливаем вектор весов omega */
254 PetscCall(DSGetMatAndColumn(svd->ds,DS_MAT_D,0,&D,&vomega)); /* Получаем м
           атрицу D и столбец vomega из DS */
255 PetscCall(VecCopy(svd->omega,vomega)); /* Копируем в
           ектор весов omega в vomega */
256 PetscCall(DSRestoreMatAndColumn(svd->ds,DS_MAT_D,0,&D,&vomega)); /* Возвра
           щаем D и vomega в DS */
257
258 PetscCall(DSSetState(svd->ds,DS_STATE_RAW)); /* Устанавливаем
           состояние DS как RAW */
259
260 n = PetscMin(M,N); /* n = min(M,
           N) */
261
262 PetscCall(PetscMalloc1(n,&w)); /* Выделяем па
           мять для массива сингулярных значений w */
263
264 PetscCall(DSSolve(svd->ds,w,NULL)); /* Вычисляем с
           ингулярные значения и векторы */
265 PetscCall(DSSort(svd->ds,w,NULL,NULL,NULL)); /* Сортируем с
           ингулярные значения */
266 PetscCall(DSSynchronize(svd->ds,w,NULL)); /* Синхронизир
           уем данные между процессами */
267
268 /* Копируем сингулярные векторы и знаки */
269 PetscCall(DSGetArrayReal(svd->ds,DS_MAT_D,&pD)); /* Получаем ма
           ссив знаков из DS */
270 PetscCall(DSGetArray(svd->ds,DS_MAT_U,&pU)); /* Получаем ма
           ссив U из DS */
271 PetscCall(DSGetArray(svd->ds,DS_MAT_V,&pV)); /* Получаем ма
           ссив V из DS */
272

```

```

273 for (i=0;i<n;i++) {
274     if (svd->which == SVD_SMALLEST) /* Определяем
        порядок сохранения сингулярных значений */
275         k = n - i - 1;
276     else
277         k = i;
278
279     svd->sigma[k] = PetscRealPart(w[i]); /* Сохраняем с
        ингулярное значение */
280     svd->sign[k] = pD[i]; /* Сохраняем з
        нах сингулярного значения */
281
282     PetscCall(BVGetColumn(svd->U,k,&u)); /* Получаем k-
        й столбец из блока векторов U */
283     PetscCall(BVGetColumn(svd->V,k,&v)); /* Получаем k-
        й столбец из блока векторов V */
284
285     PetscCall(VecGetOwnershipRange(u,&lowu,&highu)); /* Получаем ди
        апазон индексов для вектора u */
286     PetscCall(VecGetOwnershipRange(v,&lowv,&highv)); /* Получаем ди
        апазон индексов для вектора v */
287
288     PetscCall(VecGetArray(u,&pu)); /* Получаем ук
        азатель на данные вектора u */
289     PetscCall(VecGetArray(v,&pv)); /* Получаем ук
        азатель на данные вектора v */
290
291     if (M>=N) {
292         /* Если число строк больше или равно числу столбцов */
293         for (j=lowu;j<highu;j++) pu[j-lowu] = pU[i*ld+j]; /* Копируем да
        нные из pU в pu */
294         for (j=lowv;j<highv;j++) pv[j-lowv] = pV[i*ld+j]; /* Копируем да
        нные из pV в pv */
295     } else {
296         /* Если число столбцов больше числа строк */
297         for (j=lowu;j<highu;j++) pu[j-lowu] = pV[i*ld+j]; /* Копируем да
        нные из pV в pu */
298         for (j=lowv;j<highv;j++) pv[j-lowv] = pU[i*ld+j]; /* Копируем да
        нные из pU в pv */
299     }
300
301     PetscCall(VecRestoreArray(u,&pu)); /* Возвращаем
        доступ к данным вектора u */
302     PetscCall(VecRestoreArray(v,&pv)); /* Возвращаем
        доступ к данным вектора v */
303
304     PetscCall(BVRestoreColumn(svd->U,k,&u)); /* Возвращаем
        столбец u в блок векторов U */
305     PetscCall(BVRestoreColumn(svd->V,k,&v)); /* Возвращаем
        столбец v в блок векторов V */
306 }
307
308 PetscCall(DSRestoreArrayReal(svd->ds,DS_MAT_D,&pD)); /* Возвращаем
        массив pD в DS */
309 PetscCall(DSRestoreArray(svd->ds,DS_MAT_U,&pU)); /* Возвращаем
        массив pU в DS */
310 PetscCall(DSRestoreArray(svd->ds,DS_MAT_V,&pV)); /* Возвращаем
        массив pV в DS */
311
312 svd->nconv = n; /* Устанавлива

```



```

313     ем число найденных сингулярных значений */
svd->its      = 1;                                     /* Устанавлива
314     ем число итераций (1, так как метод прямой) */
svd->reason = SVD_CONVERGED_TOL;                       /* Устанавлива
315     ем причину завершения */
316 PetscCall(MatDestroy(&mat));                         /* Уничтожаем
317     матрицу mat */
PetscCall(PetscFree(w));                             /* Освобождаем
318     память, выделенную под w */
319 PetscFunctionReturn(PETSC_SUCCESS);                 /* Завершаем ф
320     ункцию успешно */
321 }
322 /* Функция для установки типа объекта DS в зависимости от задачи */
323 static PetscErrorCode SVDSetsDType_LAPACK(SVD svd)
324 {
325     PetscFunctionBegin;
326     PetscCall(DSSetType(svd->ds, svd->OPb?DSGSVD:svd->omega?DSHSVD:DSSVD)); /*
327         Устанавливаем тип DS */
328     PetscFunctionReturn(PETSC_SUCCESS);              /*
329         Завершаем функцию успешно */
330 }
331 /* Функция для создания контекста SVD с использованием LAPACK */
332 SLEPC_EXTERN PetscErrorCode SVDCreate_LAPACK(SVD svd)
333 {
334     PetscFunctionBegin;
335     svd->ops->setup      = SVDSetsUp_LAPACK;          /* Устанавливаем функцию на
336         стройки */
337     svd->ops->solve       = SVDSolve_LAPACK;          /* Устанавливаем функцию ре
338         шения стандартного SVD */
339     svd->ops->solveg      = SVDSolve_LAPACK_GSVD;     /* Устанавливаем функцию ре
340         шения обобщенного SVD */
341     svd->ops->solveh      = SVDSolve_LAPACK_HSVD;     /* Устанавливаем функцию ре
342         шения гармонического SVD */
343     svd->ops->setdstype = SVDSetsDType_LAPACK;        /* Устанавливаем функцию ус
344         тановки типа DS */
345     PetscFunctionReturn(PETSC_SUCCESS);              /* Завершаем функцию успешн
346         о */
347 }

```

Листинг 1: Реализация с пояснениями построчно

## 5. References

1. **LAPACK Users' Guide**, Third Edition, E. Anderson et al., SIAM, Philadelphia, PA, 1999. [Разделы, касающиеся вычисления SVD]
2. **SLEPc Documentation**: Официальная документация библиотеки SLEPc. Доступна по адресу: <https://slepc.upv.es/documentation/>
  - Раздел DS Module: Подробное описание модуля Direct Solver.
  - Раздел BV Module: Документация по блок-векторам.
3. **PETSc Documentation**: Документация по библиотеке PETSc. Доступна по адресу: <https://petsc.org/release/docs/>



- Разделы по структурам `Mat` и `Vec`.
4. **Golub, G. H., Van Loan, C. F.**, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, 1996. [Глава 8: Singular Value Decomposition]
  5. **Исходный код файла `svdlapack.c`**: Доступен в репозитории SLEPc по адресу:  
<https://gitlab.com/slepc/slepc/-/blob/master/src/svd/impls/lapack/svdlapack.c>