Алгоритм 4

Обозначения

В данной работе рассматривается и имплементируется предложенный Огита и Аишима итеративный метод уточнения сингулярных значений для полного сингулярного разложения [2].

Рассмотрим данный итеративный метод уточнения сингулярных значений для матрицы с действительными коэффициентами ,   
. В случае будет решаться задача для транспонированной матрицы , поскольку сингулярное разложение и совпадает [7].

Пусть – сингулярные числа. Полным сингулярным разложением матрицы будем считать такое разложение

матрицы и ортогональные, матрица диагональная и .

Далее будем считать, что , то есть все сингулярные значения различные. Также обозначает спектральную норму, т. е. . – единичная матрица. В тексте различаются приблизительные значения и посчитанные значения соответственно и .

Ускорение алгоритма

Известно, что вычисление сингулярного разложения преимущественно ограниченно матричным умножением, что и является основной вычислительной сложностью алгоритма. Есть несколько подходов к матричному умножению с высокой точностью: XBLAS [3], быстрые и точные вычисления скалярного произведения [4] и произведений матриц [5], основанные на error-free transformations.

Рассматриваемый алгоритм использует такие соотношения:

1. В силу ортогональности

1. В силу ортогональности :
2. В силу диагонализируемости A:

Пусть и приближенные значения матриц и . Корректирующими матрицами и будем называть такие матрицы, которые удовлетворяют равенствам и . Пусть определяется как

Полагаем, что мало и , тогда обе матрицы и невырожденные и можно разложить в ряд Тейлора обратные матрицы

Подставив в (1), получим

из последнего следует

Аналогичное выражение можно получить, подставив в (2)

Проделаем те же операции и подставим и в уравнение (3)

Проведем оценку остаточных членов:

Опустим слагаемые второго порядка у , , в (4), (5), (6) и получим систему матричных уравнений для и

Таким образом, остается решить систему уравнений и найти . Эффективнее всего это сделать, представив матрицы как блочные:

Тогда первое уравнение системы можно переписать:

третье уравнение системы перепишется так

Очевидно, что диагональные элементы матриц и достаточно просто выражаются

Третье уравнение системы для диагональных элементов выглядит так

**Замечание.** В общем случае возможно равенство , однако на самом деле и это невязки с точки зрения ортогональности *U* и *V*, поэтому зачастую на практике , и .

Остается найти недиагональные значения матриц и . Рассмотрим систему уравнений (7) и (8):

Умножим третье и четвертое уравнения системы на и соответственно,

Сложив полученные уравнения и подставив второе уравнение, получим

В найденное выражение подставим первое уравнение системы

Аналогичное выражение можно получить и для

Таким образом, получаем выражения для недиагональных элементов

Все эти рассуждения применимы для нахождения элементов остальных матриц . Например, совместив уравнения (7*e*) и (8), значения матрицы получатся

Из (7*b*) уравнения выражается :

И значения элементов соответственно

Благодаря условию (7*с*) определяется .

**Замечание.** В этом алгоритме мы полагаем, что для всех пар (*i*, *j*). Если же это условие не выполняется, то существует подход для решения этой проблемы, описанный в [6].

В имплементируемом алгоритме наиболее значащую роль играют быстрые и точные матричные умножения, поэтому часть вычислений переведена в матричный вариант. Заменим вычисление коэффициентов и в формуле (9) на аналогичное вычисление коэффициентов в матричном виде. Тогда получится

Введем матрицы *D* и *E*

Тогда значение недиагональных элементов матриц и примет вид

|  |
| --- |
| **Вход:**,  **Выход:**  ,    # счет приблизительных сингулярных значений  for to *n* do    sigma\_arr diag(sigma)  # счет диагональных элементов матриц и  for to *n* do  f  g g  # Выделение квадратных матриц n x n из T и R  T1 = T.topRows(n)  R11 = R.topLeftCorner(n, n)  # Вычисление матриц коэффициентов          for to *n* do  # Вычисление недиагональных элементов и  for j to *n* do  if i != j then  f  g  endif  # Вычисление  for to m do  f  for to m do  # Вычисление  for to n do  f  # Вычисление  for to m do  f  # Обновление значений , |

Листинг 1. Полный алгоритм итеративного уточнения сингулярных значений.

В представленном алгоритме есть функции, которые ранее не упоминались: *diag*(*sigma*) создает диагональную матрицу с переданными значениями *sigma*, *topRows*(*n*) возвращает первые *n* строк в матрице, *topLeftCorner*(*n*) возвращает верхнюю левую квадратную матрицу размера из исходной, *transpose*() транспонирует матрицу.

Данный алгоритм обладает квадратичной сходимостью, это следует из следующей теоремы.

**Теорема.** Пусть , при . , и , определяется как . Аналогично определим и , , где и это результат работы описанного алгоритма уточнения сингулярных значений. Если выполняется условие

тогда

Доказательство: [2].

Воспользуемся этой теоремой. Пусть , где . Тогда учитывая, что и , получаем

Применяя теорему, найдем, что

Таким, образом требуемая арифметическая точность алгоритма составляет около десятичных цифр. Хотя арифметическая точность в двух последних строках алгоритма составляет *d* десятичных цифр. Это происходит ввиду того, что только первые *d* десятичных цифр и точны, и только первые *d* десятичных цифр и могут повлиять на результат [1]. В итоге, вычислительная сложность алгоритма (количество требуемых операций) с точностью и *d* десятичных цифр соответственно  
 до и [1].

Тестирование алгоритма

Тестирование алгоритма проводилось на прямоугольных и квадратных матрицах разного размера. Сначала генерировались сингулярные значения в некотором интервале, по которым восстанавливались матрицы . После в матрицы добавлялся псевдослучайный шум, применялся алгоритм Огиты – Аишимы и оценивалась точность полученных ошибок для матриц . Полученные результаты представлены в файле *results.csv*.

Также было проведено следующее тестирование. Аналогично прошлому варианту генерировались сингулярные значения в некотором интервале, по которым восстанавливались матрицы . Далее применялся алгоритм сингулярного разложения *Jacobi* [8] с последующим уточнением результатов алгоритмом Огиты-Аишимы. Полученные результаты представлены в файле *results2.csv*.

При проведении тестов, представленных ниже, помимо уточнения сингулярных чисел, оценивалась невязка по фробениусовой норме исходной матрицы и восстановленной матрицы после 1-й итерации алгоритма.

Случай действительной матрицы

Приведем примеры работы алгоритма для 5 итераций на действительной матрице cо следующим тестированием: сгенерируем сингулярные значения на интервале и восстановим матрицы . В матрицы добавим псевдослучайный шум с уровнем

*Исходные матрицы.*

*A*:

*U*:

*S*:

*V*:

*Матрицы после применения алгоритма.*

:

:

:

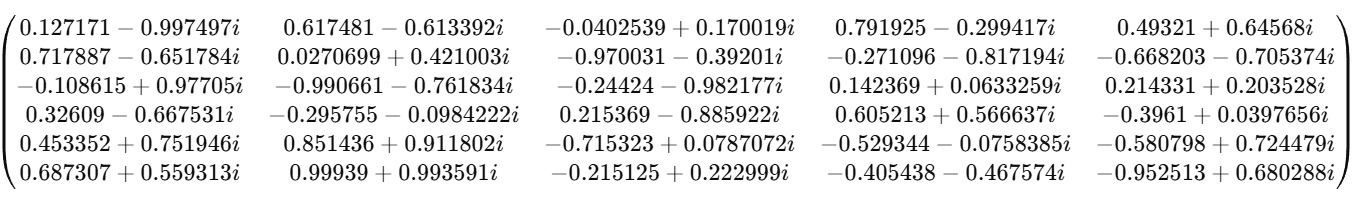
:

Посчитаем невязки матриц.

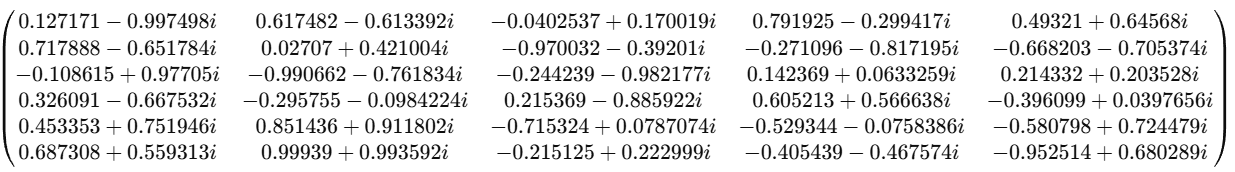
Случай комплексной матрицы.

Для вычисления сингулярного разложения комплексной матрицы 6×5 используется *Jacobi* [8] с последующим уточнением алгоритмом Огиты - Аишимы.

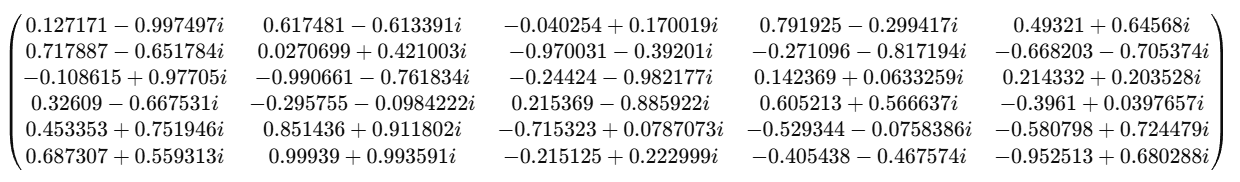
Оригинальная матрица:



Восстановленная матрица после *Jacobi*:



Восстановленная матрица после алгоритма Ogita-Aishima:



Таким образом, тест на комплексных матрицах показал, что имплементированный алгоритм работает не только на действительных матрицах, но и на комплексных матрицах. Однако это подлежит дальнейшему изучению.

Дополнительные тесты

1. Значение характеристик для псевдослучайных действительных матриц.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Matrix Size | Relative Error  (||A - U\*S\*Vᵀ|| / ||A||) | max(||F||, ||G||) | max(||R||, ||S||) | Elapsed Time (s) |
| 10×10 | 7.12345e-08 | 8.74816e-08 | 1.74963e-07 | 0.0032547 |
| 30×30 | 1.14187e-07 | 2.67087e-07 | 5.34173e-07 | 0.0243358 |
| 50×50 | 1.33093e-07 | 3.75116e-07 | 7.50232e-07 | 0.110236 |
| 60×45 | 6.47478e-07 | 3.02241e-06 | 6.04481e-06 | 0.182544 |

Применение безошибочного преобразования матричного умножения в алгоритме Огиты – Аишимы

Одним из существенных недостатков алгоритма (рисунок 1) является частое применение матричного умножения, что влияет на сложность алгоритма и его точность. Попробуем применить один из методов безошибочного матричного умножения [5], чтобы повысить точность алгоритма. В статье [5] применяются быстрые рутины в *BLAS*, в настоящей имплементации мы воспользуемся инструментами *eigen* [8].

При тестировании алгоритма для уточнения сингулярных значений с применением точного матричного умножения были получены те же результаты, что и при использовании встроенной функции *eigen* для умножения. Это может говорить о том, что *eigen* уже использует точный и быстрый алгоритм умножения матриц, и применение алгоритма [5] излишне.

1. Сравнение результатов работы алгоритма Огиты-Аишимы с применением встроенного матричного умножения и безошибочного матричного умножения [5] для действительных матриц.

|  |  |  |  |
| --- | --- | --- | --- |
| Размер матрицы |  | Время выполнения *eigen* (c) | Время выполнения алгоритм [5] (c) |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Список литературы

1. Uchino Y., Terao T., Ozaki K.: Acceleration of iterative refinement for singular value decomposition / Numerical Algorithms. — 2024. — № 95. — С. 979–1009.
2. Ogita, T., Aishima, K.: Iterative refinement for singular value decomposition based on matrix multiplication / J. Comput. Appl. —2020. — №369. – 112512.
3. Li X. S., Demmel J. W., Bailey D. H., Henry G., Hida Y., Iskandar J., Kahan W., Kang S. Y., Kapur A., Martin M. C., Thompson B. J., Tung T., Yoo D.: Design, implementation and testing of extended and mixed precision BLAS, ACM Trans. / Math. Software. —2002. — № 28. — С. 152–205.
4. Ogita T., Rump S. M., Oishi S.: Accurate sum and dot product / SIAM J. Sci. Comput. —2005. — № 26:6. — С. 1955–1988.
5. Ozaki K., Ogita T., Oishi S., Rump S. M.: Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications / Numerical Algorithms. —2012. — 59:1. — C. 95-118.
6. Ogita T., Aishima K.: Iterative refinement for symmetric eigenvalue decomposition / Japan J. Indust. Appl. Math. — 2018, № 35:3. — C. 1007–1035.
7. Golub G. Matrix Computations / Gene H. Golub, Charles F. Van Loan. — 4th ed. — Baltimore : Johns Hopkins University Press, 2013. — C. 756.
8. Eigen [Электронный ресурс] : библиотека для линейной алгебры на C++ / разработчики G. Guennebaud, B. Jacob и др. – URL: https://eigen.tuxfamily.org.