Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Численные методы»

Студент:	П. А. Гамов
Преподаватель:	Д. Л. Ревизников
Группа:	М8О-407Б
Дата:	
Оценка:	
Подпись:	

Москва, 2021

Сингулярное разложение матриц

1 Введение

Сингулярное разложение — определённого типа разложение прямоугольной матрицы. Имеющее широкое применение, в силу своей наглядной геометрической интерпретации, при решении многих прикладных задач. Переформулировка сингулярного разложения, так называемое разложение Шмидта имеет приложения в квантовой теории информации, например в запутанности.

Сингулярное разложение матрицы M позволяет вычислять сингулярные числа данной матрицы, а также левые и правые сингулярные векторы матрицы M:

левые сингулярные векторы матрицы M — это собственные векторы матрицы M^*M.;

правые сингулярные векторы матрицы M — это собственные векторы матрицы M^*M.

Сингулярное разложение является удобным при вычислении ранга матрицы, ядра матрицы и псевдообратной матрицы.

Сингулярное разложение также используется для приближения матриц матрицами заданного ранга.

2 Определение

Сингулярным разложением матрицы M порядка $m \times n$ является разложение следующего вида

$$M = U\Sigma V^*$$

где Σ — матрица размера $m \times n$ с неотрицательными элементами, у которой элементы, лежащие на главной диагонали — это сингулярные числа (а все элементы, не лежащие на главной диагонали, являются нулевыми), а матрицы U (порядка m) и V (порядка n) — это две унитарные матрицы, состоящие из левых и правых сингулярных векторов соответственно (а V^* — это сопряжённо-транспонированная матрица к V).

Пусть дана матрица:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad V^* = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ -\sqrt{0.8} & 0 & 0 & 0 & \sqrt{0.2} \end{bmatrix},$$

3 Вращения Якоби

Для нахождения собственных чисел и сопряженных векторов используем алгоритм вращений Якоби.

```
def jacobi(A):
    err = 0.1
    U = None
    while True:
        i, j = find_max(A)
        P = math.pi / 4
        if A[i][i] - A[j][j] != 0:
            P = 2 * A[i][j] / (A[i][i] - A[j][j])
        c = math.cos(math.atan(P) / 2)
        s = math.sin(math.atan(P) / 2)
        rotate = rotate_matrix(len(A),s,c,i,j)
        if U == None:
            U = copy.deepcopy(rotate)
        else:
            U = prois(U, rotate)
        A = prois(prois(transpose(rotate),A),rotate)
        er = error(A)
        if er < err:
            break
    res = []
    for i in range(len(A)):
        vec = [U[j][i] for j in range(len(A))]
        vecs = sum([x**2 for x in vec])**0.5
        if vecs != 0:
            vec = [x / vecs for x in vec]
        res.append([A[i][i],vec])
    return res
```

Цепочка произведений матриц поворота дает нам матрицу состоящую из собственных векторов. Далее после нормировки они возвращаются из функции.

4 SVD

```
def SVD(A, cut=0):
    A1res = sorted(jacobi(prois(transpose(A),A)))[::-1]
    right = [[A1res[i][1][j] for j in range(len(A1res))] for i in range(len(A1res))]
    A2res = sorted(jacobi(prois(A,transpose(A))))[::-1]
    left = [[A2res[j][1][i] for j in range(len(A2res))] for i in range(len(A2res))]
    center = [[0 for j in range(len(A[i]))] for i in range(len(A))]
    for i in range(min(len(A),len(A[0]))):
        center[i][i] = A1res[i][0]**0.5
    if cut != 0 and cut < 100:
        f_vert = max(math.floor(len(center) * (100 - cut) / 100),1)
        f_hor = max(math.floor(len(center[0]) * (100 - cut) / 100),1)
        left = [[left[i][j] for j in range(f_vert)] for i in range(len(left))]
        right = [[right[i][j] for j in range(len(right[0]))] for i in range(f_hor)]
        center = [[center[i][j] for j in range(f_hor)] for i in range(f_vert)]
        return left, center, right
    else:
        return left, center, right
```

Если нет второго параметра, то алгоритм вернет 3 матрицы разложения, если есть параметр cut, то алгоритм обрежет разложение и вернет обрезаные компоненты матриц, которые содержат самые большие сингулярные числа.

```
A = [[1,0,0,0,2],
     [0,0,3,0,0],
     [0,0,0,0,0],
     [0,4,0,0,0]]
```

Рассмотрим на матрице из примера выше.

Левая компонента разложения.

```
array([[ 0., 0., 1., 0.],
       [ 0., 1., -0., 0.],
       [ 0., 0., 0., 1.],
       [ 1., 0., 0., 0.]])
```

Центральная компонента разложения. Содержит в себе синулярные числа в порядке убывания.

```
array([[4.  , 0.  , 0.  , 0.  , 0. ],
       [0.  , 3.  , 0.  , 0.  , 0. ],
       [0.  , 0.  , 2.23606798, 0.  , 0. ],
       [0.  , 0.  , 0.  , 0.  , 0. ]])
```

Правая компонента разложения.

```
array([[ 0.  , 1.  , 0.  , 0.  , 0. ],
       [ 0.  , 0.  , 1.  , 0.  , 0. ],
       [ 0.4472136 , 0.  , 0.  , 0.  , 0.89442719],
       [ 0.89442719, 0.  , 0.  , 0.  , -0.4472136 ],
       [ 0.  , 0.  , 0.  , 1.  , 0. ]])
```

Если умножить матрицы по порядку мы получим исходную матрицу.

```
[[1.0, 0.0, 0.0, 0.0, 2.0],
 [0.0, 0.0, 3.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 4.0, 0.0, 0.0, 0.0]]
```

Теперь попробуем обрезать одно сингулярное число.

Левая компонента разложения.

```
array([[0., 0.],
       [0., 1.],
       [0., 0.],
       [1., 0.]])
```

Центральная компонента разложения.

```
array([[4., 0.],
       [0., 3.]])
```

Правая компонента разложения.

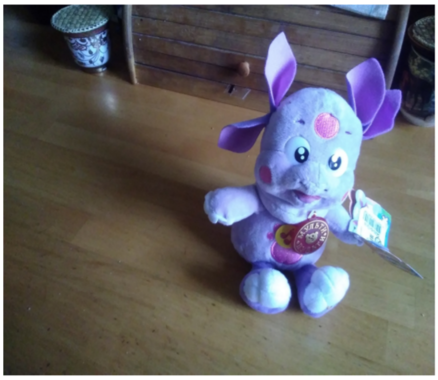
```
array([[0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.]])
```

Если умножить матрицы по порядку мы получим исходную матрицу.

```
[[0.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 3.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0, 0.0],
 [0.0, 4.0, 0.0, 0.0, 0.0]]
```

Для такой маленькой матрицы первая строчка обратилась в ноль. Данные потеряны. На большей матрице с большим рангом потеря окажется несущественной.

Такое отрезание менее значимых компонент составляет основу применений данного разложения: используя данный алгоритм, можно получить наилучную апроксимацию матрицы рангом не больше некого заданного числа. А значит данный алгоритм можно использовать например для сжатия картинок без существенной потери качества.



a) исходный массив

b) ранг k=60, q=10.8%

c) ранг k=30, q=5.4%

d) ранг k=15, q=2.7%

Материал взят из работы Карчевского М.М.

5 Выводы

Сингулярное разложение позволяет понизить ранг матрицы, что имеет свое применение в математике. Так же разложение имеет прикладное применение, такое как сжатие изображений, апроксимация графиков и данных.