HW1

The Algorithm and Implementation of Gauss-Jordan Elimination, Sweep Operator, and Power Method

陳凱騫

2024-11-10

Table of contents

1. Gauss-Jordan Elimination

Algorithm and Code

它包含了三個主要部分：行交換 (row_switch)，行處理（gjrp），以及高斯-喬登算法（Gauss_Jordan 和 Gauss_Jordan2）。我們可以將這段程式碼整理為一個清晰的算法流程，如下所示：

Gauss-Jordan Elimination Algorithm

Input:

A matrix (A) of size $m \times n$.

Output:

A matrix in **reduced row echelon form (RREF)**.

Step 1: Row Switching

Function: row_switch(matrix1, r)

1. Set row1 to be the r^{th} row of the matrix.
2. Starting from the r^{th} row, check each row below it:
   - If a non-zero element is found in the r^{th} column, switch the r^{th} row with that row.
   - If no such row is found, issue a warning that the matrix might be singular (no non-zero pivot in this column).
3. Return the updated matrix.

Step 2: Forward Elimination

Function: gjrp(matrix1, r)

1. If the value (assume a is the value) in r^{th} column and r^{th} row is not equal to zero then multiply $\frac{1}{a}$ to let it be 1.
2. Add/subtract multiples of the r^{th} row to the other rows(the row smaller than r) so that all other entries which below the r^{th} rows in the column are all zero.

Step 3: Full Gauss-Jordan Forward Elimination

Function: Gauss_Jordan(matrix1)

1. For each row (i) from 1 to (m):
   - If the diagonal element (matrix1[i, i]) is 0, call row_switch(matrix1, i) to swap rows and ensure a non-zero value occur.
   - Call gjrp(matrix1, i) to perform forward elimination on the matrix, processing the i^{th} row and eliminating entries below it.
2. Return the updated matrix.

Step 4: Backward Elimination (Refinement to RREF)

Function: Gauss_Jordan2(matrix1)

1. For each row (row1) from 2 to (m):
   - For each row (i) above (row1) (i.e., from ($row1$ -1) to 1):
   - Subtract a suitable multiple of the $row1^{th}$ row from the i^{th} row to eliminate all elements above the pivot.
2. Return the updated matrix.

Final Algorithm: Full Gauss-Jordan Elimination

1. Call Gauss_Jordan(matrix1) to perform the forward elimination, reducing the matrix to an upper triangular form (Gaussian elimination).
2. Call Gauss_Jordan2(matrix1) to perform the backward elimination, refining the matrix into **reduced row echelon form (RREF)**.

Code Implementation and example

3x3 matrix :

```
     [,1] [,2] [,3]
[1,]    4    1    2
[2,]    1    5    3
[3,]    2    3    6
```

3x3 matrix inverse:

```
     [,1]          [,2]       [,3]
[1,]  0.3  6.938894e-18 -0.1000000
[2,]  0.0  2.857143e-01 -0.1428571
[3,] -0.1 -1.428571e-01  0.2714286
```

4x4 matrix :

```
     [,1] [,2] [,3] [,4]
[1,]   10    2    3    4
[2,]    2    8    1    5
[3,]    3    1    9    6
[4,]    4    5    6   11
```

4x4 matrix inverse:

```
             [,1]         [,2]        [,3]        [,4]
[1,]  0.119142753 -0.009807483 -0.01997821 -0.02796949
[2,] -0.009807483  0.192880494  0.05957138 -0.11660007
[3,] -0.019978206  0.059571377  0.19542317 -0.12640756
[4,] -0.027969488 -0.116600073 -0.12640756  0.22302942
```

```r
FindInver(A3) %*% A3
```

```
              [,1]         [,2]          [,3]
[1,]  1.000000e+00 0.000000e+00 -1.110223e-16
[2,]  0.000000e+00 1.000000e+00  0.000000e+00
[3,] -1.110223e-16 1.110223e-16  1.000000e+00
```

```r
FindInver(A4) %*% A4
```

```
              [,1]          [,2]          [,3] [,4]
[1,]  1.000000e+00  0.000000e+00  0.000000e+00    0
[2,] -5.551115e-17  1.000000e+00 -1.110223e-16    0
[3,] -2.220446e-16 -1.110223e-16  1.000000e+00    0
[4,]  1.110223e-16  0.000000e+00  2.220446e-16    1
```

From the result above, we can really know the inverse matrix is what we want。

2. Sweep Operator

Algorithm and Code

此部分包含Sweep Operator的步驟說明和R程式碼。Sweep Operator可以有效計算對稱矩陣的逆矩陣。

Sweep Operator Algorithm

Suppose we have an $n \times n$ symmetric matrix A, and we want to compute its inverse A^{-1}. This algorithm uses the sweep operator to calculate $-A^{-1}$ and then takes the negative of the result to obtain

A^{-1}.

Variables

- **A**: An $n \times n$ symmetric matrix, assumed to be positive definite.
- **k**: The current index of the diagonal element to sweep.
- **inverse**: If TRUE, this performs a reverse sweep operation, restoring the matrix to its original state.

Algorithm Steps

1. **Input**: Define a symmetric $n \times n$ matrix A.
   - **Output**: A^{-1}
2. **Perform the sweep operation for each specified index k**:
   - Retrieve the diagonal element a_{kk}.
   - **Update the diagonal element**:
     $$a_{kk} = -\frac{1}{a_{kk}}$$
   - **Update other elements in the k-th row and k-th column** (excluding a_{kk}):
     - For each $i \neq k$, update the elements in the k-th row and k-th column:
       $$a_{ik} = \frac{a_{ik}}{a_{kk}}$$
       $$a_{ki} = a_{ik}$$
       If performing a reverse sweep (inverse = TRUE):
       $$a_{ik} = -\frac{a_{ik}}{a_{kk}}$$
       $$a_{ki} = a_{ik}$$
   - **Update all other elements not in the k-th row or k-th column**:
     - For each $i \neq k$ and $j \neq k$, update the element:
       $$a_{ij} = a_{ij} - \frac{a_{ik} \cdot a_{kj}}{a_{kk}}$$
3. **Take the negative of the matrix upon completion**:
   - After performing the sweep operation on each diagonal element, the final result is $-A^{-1}$.
   - To obtain A^{-1}, take the negative of the matrix, yielding A^{-1}.

Code Implementation

3x3 matrix :

```
     [,1] [,2] [,3]
[1,]    4    1    2
[2,]    1    5    3
[3,]    2    3    6
```

3x3 matrix inverse:

```
              [,1]          [,2]       [,3]
[1,]  3.000000e-01  6.938894e-18 -0.1000000
[2,]  6.938894e-18  2.857143e-01 -0.1428571
[3,] -1.000000e-01 -1.428571e-01  0.2714286
```

4x4 matrix :

```
     [,1] [,2] [,3] [,4]
[1,]   10    2    3    4
[2,]    2    8    1    5
[3,]    3    1    9    6
[4,]    4    5    6   11
```

4x4 matrix inverse:

```
             [,1]         [,2]        [,3]        [,4]
[1,]  0.119142753 -0.009807483 -0.01997821 -0.02796949
[2,] -0.009807483  0.192880494  0.05957138 -0.11660007
[3,] -0.019978206  0.059571377  0.19542317 -0.12640756
[4,] -0.027969488 -0.116600073 -0.12640756  0.22302942
```

```r
sweep_operator(A3, 1:3) %*% A3
```

```
              [,1]         [,2]          [,3]
[1,]  1.000000e+00 0.000000e+00 -1.110223e-16
[2,]  0.000000e+00 1.000000e+00  0.000000e+00
[3,] -1.110223e-16 1.110223e-16  1.000000e+00
```

```r
sweep_operator(A4, 1:4) %*% A4
```

```
              [,1]          [,2]          [,3]          [,4]
[1,]  1.000000e+00  2.775558e-17 -2.775558e-17  5.551115e-17
[2,]  0.000000e+00  1.000000e+00  0.000000e+00  2.220446e-16
[3,] -2.220446e-16 -2.220446e-16  1.000000e+00 -2.220446e-16
[4,]  1.110223e-16  0.000000e+00  0.000000e+00  1.000000e+00
```

From the result above, we can really know the inverse matrix is what we want。

3. Power Method

Algorithm and Code

此部分涵蓋 Power Method 的算法描述和 R 程式碼實現，用於計算方陣的最大和次大特徵值。

Power Method Algorithm

The Power Method to find the largest eigenvalue follows these steps: 1. **Input**: square matrix A - **Output**: λ

2. **Initialize** a random vector u_1 with the same dimension as the square matrix A.
3. **Normalize** the vector u_1 by dividing it by its norm to prevent numerical overflow or underflow.
4. **Iteratively apply** the matrix A to the vector u_1 :
   - $u_{k+1} = A \cdot u_k$
   - Normalize the new vector u_{k+1}.
5. **Estimate the eigenvalue** as the norm of the vector u_{k+1}:
   - $\lambda = \|A \cdot u_k\|$
6. **Repeat steps 3 and 4** until the difference between successive eigenvalue estimates is smaller than a given tolerance, or a maximum number of iterations is reached.
7. **Return** the dominant eigenvalue λ and its corresponding eigenvector u_k.

Code Implementation

舉例之4x4矩陣:

```
     [,1] [,2] [,3] [,4]
[1,]    4    1   -2    2
[2,]    1    3   -1    1
[3,]   -2   -1    3   -2
[4,]    2    1   -2    4
```

絕對值後最大特徵值： 8.268169

對應的特徵向量： 0.5736358 0.3116802 -0.4947114 0.5736358

迭代次數： 12

```
          [,1]
[1,]  4.742918
[2,]  2.577025
[3,] -4.090358
[4,]  4.742918
```

```
           [,1]       [,2]      [,3]       [,4]
[1,]  1.2792930 -0.4782745 0.3463754 -0.7207070
[2,] -0.4782738  2.1967917 0.2748838 -0.4782738
[3,]  0.3463753  0.2748844 0.9764534  0.3463753
[4,] -0.7207070 -0.4782745 0.3463754  1.2792930
```

Power Method之結果:

[1] 8.268169

[1] 2.446477

R function中的eigen()之結果:

[1] 8.268169 2.446477

qmd檔連結:

https://github.com/KaiChienChen/Simulation/blob/main/H24101222_HW1.qmd