Algorithm solving system of linear equations

Gaussian Elimination: Algorithm

Since this is a book aimed at computer and data scientists, it is probably worth our time stating the algorithm of Gaussian Elimination as an algorithm.

Input: $\boldsymbol{A} = [a_{ij}] \in \mathbb{R}^{m \times n}$

Output: $\mathrm{RRF}(\boldsymbol{A}) \in \mathbb{R}^{m \times n}$

1: REF $\leftarrow \boldsymbol{A}$; nrows $\leftarrow m$
2: **while** $m > 0$ **and** $n > 0$ **do**
    If the first row does not start with a non-zero element,
    try to find a row that does:
3:     **if** $a_{11} = 0$ **then**
4:         **repeat**
5:             **if** $a_{11} = 0$ **then**
6:                 **for** $i = 1$ to m **do**
7:                     **if** $a_{i1} \neq 0$ **then**
                    Swap row 1 with row i:
8:                         $r_1 \leftrightarrow r_i$
9:                         Exit **repeat** loop
10:                     **end if**
11:                 **end for**
            Could not find a row with non-zero element in the first column
12:                 $\boldsymbol{A} \leftarrow \boldsymbol{A}\left[a_{22} : a_{mn}\right], m \leftarrow m - 1, n \leftarrow n - 1$
13:             **end if**
14:         **until** $a_{11} \neq 0$
15:     **else**
        Subtract scaled first row from other rows to get zero first column
16:         **for** $i = 2$ **to** m **do**
17:             $r_i \leftarrow r_i - \frac{a_{i1}}{a_{11}} r_1$
18:         **end for**
19:     **end if**
    Save the current row in the REF
20:     REF(nrows$-m$) $\leftarrow \boldsymbol{A}\left[a_{11} : a_{1n}\right]$
21:     $\boldsymbol{A} \leftarrow \boldsymbol{A}\left[a_{22} : a_{mn}\right], m \leftarrow m - 1, n \leftarrow n - 1$
22: **end while**
23: **return** REF

Note that instead of finding the first row with non-zero element (starting at line 5), it may be a better numerical strategy to unconditionally locate the row with the largest absolute value at the first element (starting at line 3), and swapping it with the first row. Most programs implement Gaussian Elimination that way.

Gauss-Jordan Elimination: Algorithm

We can state the Gauss-Jordan algorithm also using pseudo-code so that the steps may be clearer to the students of computer science.

Input: $\boldsymbol{A} = [a_{ij}] \in \mathbb{R}^{m \times n}$
Output: RRRF($\boldsymbol{A}$) $\in \mathbb{R}^{m \times n}$

Get the REF using Gaussian Elimination
$\boldsymbol{A} \leftarrow$ REF($\boldsymbol{A}$); pivots $\leftarrow$ pivots of $\boldsymbol{A}$
for $i = 1$ to m **do**
  if $\text{pivots}_i = 0$ **then**
    Exit **for** loop
  end if
  Scale the row to get unit pivot
  $r_i \leftarrow \frac{r_i}{pivots_i}$
  Subtract scaled pivot row from other rows to get zero pivot column
  for $j = 1$ to m **do**
    if $j \neq i$ **then**
      $k \leftarrow$ pivot column index
      $r_j \leftarrow r_j - a_{jk} r_i$
    end if
  end for
end for
return $\boldsymbol{A}$ as RREF

LU Factorization

......

......

65090500451 ภูมิไทย พรมโกฏิ

```
function solve_lu(A, b)
    n = size(A)
    L, U, p = lu(A) \* Compute the LU factorization
with partial pivoting*\
    y = zeros(n)
    x = zeros(n)
    # Forward substitution to solve Ly = b
    for i = 1 to n do
        y(i) = b(p(i))
        for j = 1 to i-1 do
            y(i) -= L(i, j) * y(j)
        end
    end
    # Backward substitution to solve Ux = y
    for i = n:-1:1 do
        x(i) = y(i)
        for j = i+1 to n do
            x(i) -= U(i, j) * x(j)
        end
        x(i) /= U(i, i)
    end
    return x
end
```

Code ภูมิไทย พรมโกฎิ 65090500451

1.CODE Gaussian_Elimination

```python
import numpy as np

def gaussian_elimination(A, b):
    n = len(A)
    Ab = np.hstack((A, b.reshape(-1, 1)))  # augment the matrix A with column vector b
    for i in range(n):
        # Find the row with the largest absolute value in the i-th column
        max_row = i
        for j in range(i+1, n):
            if abs(Ab[j,i]) > abs(Ab[max_row,i]):
                max_row = j
        # Swap the i-th and max_row-th rows
        Ab[[i,max_row],:] = Ab[[max_row,i],:]
        # Reduce the i-th column to zeros below the i-th row
        for j in range(i+1, n):
            c = Ab[j,i] / Ab[i,i]
            Ab[j,:] -= c * Ab[i,:]
    # Solve for the unknowns using back substitution
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = (Ab[i,-1] - np.dot(Ab[i,:-1], x)) / Ab[i,i]
    return x
# Example system of linear equations:
# 2x + y - z = 8
# -3x - y + 2z = -11
# -2x + y + 2z = -3
# x= 2, y= 3, z= -1
A = np.array([[2, 1, -1],
              [-3, -1, 2],
              [-2, 1, 2]])

b = np.array([8, -11, -3])
A = A.astype(np.float64)
b = b.astype(np.float64)
x = gaussian_elimination(A, b)
print(x)
```

1.1. OUTPUT Gaussian_Elimination

```
PS D:\python> & C:/Users/PC/AppData/Local/Microsoft/WindowsApps/python3.10.exe d:/python/Gaussian_Elimination.py
[ 2.  3. -1.]
```

2. CODE Gauss-Jondan

```
import numpy as np

def gauss_jordan(A, b):
    # augment the matrix A with column vector b
    Ab = np.hstack((A, b.reshape(-1, 1)))

    # apply Gauss-Jordan elimination
    n = Ab.shape[0]
    for i in range(n):
        # find the row with the largest pivot
        max_row = i
        for j in range(i+1, n):
            if abs(Ab[j,i]) > abs(Ab[max_row,i]):
                max_row = j

        # swap the rows to make the pivot the largest
        Ab[[i,max_row],:] = Ab[[max_row,i],:]

        # divide the pivot row by the pivot
        pivot = Ab[i,i]
        Ab[i,:] /= pivot

        # subtract multiples of the pivot row from the other rows
        for j in range(n):
            if j != i:
                c = Ab[j,i]
                Ab[j,:] -= c * Ab[i,:]

    # extract the solution vector from the augmented matrix
    x = Ab[:,n]

    return x
# Example system of linear equations:
# 2x + y - z = 8
# -3x - y + 2z = -11
# -2x + y + 2z = -3
# x= 2, y= 3, z= -1
A = np.array([[2, 1, -1],
              [-3, -1, 2],
              [-2, 1, 2]])

b = np.array([8, -11, -3])
A = A.astype(np.float64)
b = b.astype(np.float64)
x = gauss_jordan(A, b)
print(x)
```

2.1. OUTPUT Gauss-Jondan

```
PS D:\python> & C:/Users/PC/AppData/Local/Microsoft/WindowsApps/python3.10.exe d:/python/Gauss-Jondan.py
[ 2.  3. -1.]
```

3. CODE LU-Solve

```python
import numpy as np

def LU_decomposition(A):
    n = A.shape[0]
    L = np.identity(n)
    U = A.copy()

    for k in range(n-1):
        for i in range(k+1, n):
            L[i,k] = U[i,k]/U[k,k]
            for j in range(k+1, n):
                U[i,j] = U[i,j] - L[i,k]*U[k,j]
            U[i,k] = 0

    return L, U
def LU_solve(L, U, b):
    y = np.linalg.solve(L, b)
    x = np.linalg.solve(U, y)

    return x
A = np.array([[2, 1, -1], [-3, -1, 2], [-2, 1, 2]])
b = np.array([8, -11, -3])
A = A.astype(np.float64)
b = b.astype(np.float64)

# Perform LU decomposition
L, U = LU_decomposition(A)

# Solve the linear system Ax = b
x = LU_solve(L, U, b)

# Print the solution vector
print('The L is')
print(L)
print()
print('The U is')
print(U)
print()
print('The Solve is')
print(x)
```

3.1.OUTPUT LU-Solve

```
PS D:\python> & C:/Users/PC/AppData/Local/Microsoft/WindowsApps/python3.10.exe d:/python/LU-Solve.py
The L is
[[ 1.   0.   0. ]
 [-1.5  1.   0. ]
 [-1.   4.   1. ]]

The U is
[[ 2.   1.  -1. ]
 [ 0.   0.5  0.5]
 [ 0.   0.  -1. ]]

The Solve is
[ 2.  3. -1.]
```