Reduced Row Echelon Form explained

- Description of the `scale` variable and its role in the computation of the determinant
- Implementation of the *elementary row operations* on matrices
  - row_spap
  - scalar_multiply
  - row_combine
- A working example of transformation of a matrix in *rref* with a sequence of elementary row operations

Use of the `scale` variable

Let's consider a *non-singular squared* matrix `m`. Let's call:

- $row\_swap(m, i, r)$ the result of row swapping of m, for any pair of rows i, r
- $scalar\_multiply(m, r, alpha)$ the result of scalar multiplication for any row r and real not null value $alpha$
- $rref(m)$ the transformation of m in reduced row echelon form

The following equalities hold:

- $det(m) = -det(row\_swap(m, i, r))$
- $det(m) = det(scalar\_multiply(m, r, alpha)/alpha$
- $det(rref(m)) = 1$

Use of the `scale` variable (continued)

The implementations shown hereafter

- change the sign of `scale` for every use of `row_swap`
- divide the `scale` by `alpha` for every use of `scalar_multiply`
- let's call the final value $scale(rref(m))$,
- remember that the $rref$ of a squared matrix is the *identity* therefore:

$$det(m) = det(rref(m)) * scale(rref(m)) = 1 * scale(rref(m))$$

Implementation of the elementary row operations on matrices

Row swap

return matrix `m` with rows `i` and `j` swapped

In [1]:
```
row_swap <- function(m, i, j){
  if (i > nrow(m) | j > nrow(m)){
    return(NULL)
  }
  temp <- m[i,]
  m[i,] <- m[j,]
  m[j,] <- temp
  scale <<- -scale # update the global variable using <<-
  return(m)
} # row_swap: end
```

Implementation of the elementary row operations on matrices

Scalar multiplication

return `m` with row `i` multiplied by `alpha`

In [2]:
```
scalar_multiply <- function(m, i, alpha){
  if (i > nrow(m)){
    return(NULL)
  }
  m[i,] <- m[i,] * alpha
  scale <<- scale / alpha # update the global variable using <<-
  return(m)
} # scalar_multiply: end
```

Implementation of the elementary row operations on matrices

Row combination

return `m` with row `j` multiplied by `alpha` added to row `i`

```
In [3]:
row_combine <- function(m, i, j, alpha){
  if (i > nrow(m) | j > nrow(m)){
    return(NULL)
  }
  m[i,] <- m[i,] + m[j,] * alpha
  return(m)
} # row_combination: end
```

Service procedure

Find next pivot

find next pivot in the portion of `m` starting from `start_row` , start_col Use:

- `r` : row index
- `pivot` : column index Algorithm
- **if** `start_row` or `start_col` are out of matrix **return** null
- **repeat** varying `pivot` for all columns starting from `start_col`
  - **repeat** varying `r` for all rows starting from `start_row`
    - **if** element `r` , `pivot` of `m` is non zero the `pivot` is found, **return** its indexes `r` and `pivot`
- if we arrive here it means that the portion of matrix is all zero and there is no pivot, then **return** null

```
In [4]:
find_next_pivot <- function(m, start_row, start_col){
  norow <- nrow(m)
  nocol <- ncol(m)
  if (start_row > norow | start_col > nocol) {
    return(NULL)
  }
  for (pivot in start_col:nocol){
    for (r in start_row:norow){
      if (m[r,pivot] != 0){
        return(list(row=r, pivot=pivot))
      } # if (m[i,pivot] != 0)
    } #for (pivot in start_col:nocol)
  } # for (i in start_row:norow)
```

```
    return(NULL)
} # find_next_pivot: end
```

Working example

- The variable `scale` is manipulated inside the functions `row_swap` and `scalar_multiply`, but it must *survive* to the different calls of the functions therefore:
  - it must be initialized at the *top level* (i.e. out of the functions), we say that it is used as a *global variable*, it will be initialized to 1, since it will be updated by multiplications
  - inside the functions it will be updated with the `<<-` operator, meaning that the update will operate on the global variable, and not on a new variable local to the function

Initialization

- Initialize to 1 the variable `scale`
- Prepare the matrix `m`

In [5]:
```
scale <- 1
m0 <- matrix(data =             # save the original matrix
              c(0,20,0,40,20,0,60,0,0,30,0,0,60,-60,60,0)
             , nrow = 4, byrow = TRUE
            )
m <- m0
print(m)
```

```
     [,1] [,2] [,3] [,4]
[1,]    0   20    0   40
[2,]   20    0   60    0
[3,]    0   30    0    0
[4,]   60  -60   60    0
```

a) Find pivot, starting from row 1 and col 1

In [6]:
```
r <- 1                  # current row
pivot <- 1              # look for next pivot starting from the first column
pivot_rc <- find_next_pivot(m, r, pivot) # returns row and column of next pivot
str(pivot_rc)
```

```
List of 2
 $ row  : int 2
 $ pivot: int 1
```

b) Row swap

Pivot row (2) and current row (1) are swapped, to put the row with pivot in the current row

In [7]:
```
i <- pivot_rc$row
pivot <- pivot_rc$pivot  # update pivot column
if (i != r){ # row swap to put pivot in row r
    m <- row_swap(m, i, r)
}
print(m)
print(scale)
```

```
     [,1] [,2] [,3] [,4]
[1,]   20    0   60    0
[2,]    0   20    0   40
[3,]    0   30    0    0
[4,]   60  -60   60    0
[1] -1
```

c) Scalar multiplication

current row is multiplied by the inverse of the pivot of the current row

In [8]:
```
m <- scalar_multiply(m, r, 1/m[r, pivot])
print(m)
print(scale)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    3    0
[2,]    0   20    0   40
[3,]    0   30    0    0
[4,]   60  -60   60    0
[1] -20
```

d) Linear combination

The row(s) (other than the current one) with a non-zero in the pivot column (target) are linearly combined with the current row in order to have zero in the pivot column.

In this case, rows 1 and 4 are linearly combined, the combination factor is the opposite of the element of the target row in the pivot column

In [9]:
```
i <- 4 # besides the pivot, only row 4 is non-null in pivot column
m <- row_combine(m, i, r, -m[i, pivot])
print(m)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    3    0
[2,]    0   20    0   40
[3,]    0   30    0    0
[4,]    0  -60 -120    0
```

Second execution of main loop: repeat step a)

- increment current row and tentative pivot column
- find next pivot

In [10]:
```
r <- r + 1
pivot <- pivot + 1
pivot_rc <- find_next_pivot(m, r, pivot) # returns row and column of next pivot
str(pivot_rc)
```

```
List of 2
 $ row  : int 2
 $ pivot: int 2
```

b) Row swap

In this case no swap will be executed, because i is equal to r

c) Scalar multiplication

In [11]:
```
m <- scalar_multiply(m, r, 1/m[r, pivot])
print(m)
print(scale)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    3    0
[2,]    0    1    0    2
[3,]    0   30    0    0
[4,]    0  -60 -120    0
[1] -400
```

d) Linear combination

- Rows 3 and 4 are linearly combined with row 2 (the current)
- Row 1 has zero in the current pivot column, and doesn't need linear combination

In [12]:
```
i <- 3
m <- row_combine(m, i, r, -m[i, pivot])
i<- 4
m <- row_combine(m, i, r, -m[i, pivot])
print(m)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    3    0
[2,]    0    1    0    2
[3,]    0    0    0  -60
[4,]    0    0 -120  120
```

Third execution of main loop: repeat step a)

- increment current row and tentative pivot column
- find next pivot

In [13]:
```
r <- r + 1
pivot <- pivot + 1
pivot_rc <- find_next_pivot(m, r, pivot) # returns row and column of next pivot
str(pivot_rc)
```

```
List of 2
 $ row  : int 4
 $ pivot: int 3
```

b) Row swap

Rows 3 and 4 are swapped

In [14]:
```
i <- pivot_rc$row
pivot <- pivot_rc$pivot  # update pivot column
if (i != r){ # row swap to put pivot in row r
    m <- row_swap(m, i, r)
}
```

```
print(m)
print(scale)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    3    0
[2,]    0    1    0    2
[3,]    0    0 -120  120
[4,]    0    0    0  -60
[1] 400
```

c) Scalar multiplication

In [15]:
```
m <- scalar_multiply(m, r, 1/m[r, pivot])
print(m)
print(scale)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    3    0
[2,]    0    1    0    2
[3,]    0    0    1   -1
[4,]    0    0    0  -60
[1] -48000
```

d) Linear combination

- Row 1 is linearly combined with row 3 (the current)
- Rows 2 and 4 have zero in the current pivot column, and do not need linear combination

In [16]:
```
i <- 1
m <- row_combine(m, i, r, -m[i, pivot])
print(m)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    3
[2,]    0    1    0    2
[3,]    0    0    1   -1
[4,]    0    0    0  -60
```

Fourth execution of main loop: repeat step a)

- increment current row and tentative pivot column
- find next pivot

In [17]:
```
r <- r + 1
pivot <- pivot + 1
pivot_rc <- find_next_pivot(m, r, pivot) # returns row and column of next pivot
str(pivot_rc)
```

```
List of 2
 $ row  : int 4
 $ pivot: int 4
```

b) Row swap

In this case no swap will be executed, because i is equal to r

c) Scalar multiplication

In [18]:
```
m <- scalar_multiply(m, r, 1/m[r, pivot])
print(m)
print(scale)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    3
[2,]    0    1    0    2
[3,]    0    0    1   -1
[4,]    0    0    0    1
[1] 2880000
```

d) Linear combination

- Rows 1, 2 and 3 are linearly combined with row 4 (the current)

In [19]:
```
i <- 1
m <- row_combine(m, i, r, -m[i, pivot])
i <- 2
m <- row_combine(m, i, r, -m[i, pivot])
i <- 3
m <- row_combine(m, i, r, -m[i, pivot])
print(m)
```

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

End

Matrix `m` is now in reduced row echelon form

In [20]:
```
print(m)
```

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

In [21]:
```
print(scale)
```

```
[1] 2880000
```

In [22]:
```
print(det(m0)) # det() is the standard R function for the determinant
```

```
[1] 2880000
```