In [ ]:
```
import os

import numpy as np
```

Lab 2: Simplex Method

Bondar Petro (Variant 2)

Function: $F = 3x_1 - 2x_2 \to min$

Equasions:

$$\begin{aligned} 2x_1 \ &+ \ x_2 \ \leq 14 \\ -3x_1 \ &+ \ 2x_2 \ \leq 9 \\ 3x_1 \ &+ \ 4x_2 \ \leq 27 \\ x_{1,2} &\geq 0 \end{aligned}$$

Let's reduce inequalities to equalities with aux variables:

Equasions:

$$\begin{aligned} 2x_1 \ &+ \ x_2 \ + x_3 \ &&= 14 \\ -3x_1 \ &+ \ 2\,x_2 \ &+ \ x_4 \ &= 9 \\ 3x_1 \ &+ \ 4\,x_2 \ &+ \ x_5 \ &= 27 \\ x_i &\geq 0, \ \forall i = \overline{1,5} \end{aligned}$$

As result we get canonical form.

Create and print table

In [ ]:
```
# Method to create table for algorithm
def create_table(c, A, b, basis_idx):
    up_t = [x + [e] + [c[bas]] for x, e, bas in zip(A, b, basis_idx)]   # Upper rows are for eq coef and basis vector
    low_r = c + [0, 0]                                                  # Last row is for funct coef
    deltas = [0 for i in range(0, len(low_r))]
    return up_t + [low_r] + [deltas]

def print_t(t):
```

```
    for r in t:
        print(r)
    print()
```

Calculate deltas for columns

In [ ]:
```
# Calculate deltas
def calc_deltas(table):
    for j in range(0, len(table) - 2):
        table[-1][-2] += table[j][-2] * table[j][-1]

    # (bas, Ai) - Ci
    for i in range(0, len(table[-1]) - 2):
        for j in range(0, len(table) - 2):
            table[-1][i] += table[j][i] * table[j][-1]
        table[-1][i] -= table[-2][i]
```

Optimization process

In [ ]:
```
# Check delta row
def could_be_optimized_to_min(table):
    for d in table[-1]:
        if d > 0:
            return True

    return False
```

In [ ]:
```
# Find element to move to basis
def find_pivot_min(table):
    deltas = table[-1]
    d_plus_idx = 1000
    for i in range(0, len(deltas) - 2):
        if deltas[i] > 0:
            d_plus_idx = i
            break

    div = [table[i][-2] / table[i][d_plus_idx] for i in range(0, len(table) - 2)]
    for i in range(0, len(div)):
        if(div[i] <= 0):
```

```
            div[i] = np.inf

    row_idx = div.index(min(div))

    return row_idx, d_plus_idx
```

In [ ]:
```
# Recalculate table to change basis
def table_recalc(table, pivot):
    pivot_r, pivot_c = pivot
    new_table = [r.copy() for r in table]
    new_table[-1] = [0 for i in range(0, len(table[0]))]

    # Redefine basis
    new_table[pivot_r][-1] = table[-2][pivot_c]

    # X_new = X_old - (A*B)/V
    for r in range(0, len(table) - 2):
        for c in range(0, len(table[r]) - 1):
            new_table[r][c] = table[r][c] - (table[r][pivot_c] * table[pivot_r][c]) / table[pivot_r][pivot_c]

    # Redefine row for new basis
    for i in range(0, len(table[pivot_c]) - 1):
        new_table[pivot_r][i] = table[pivot_r][i] / table[pivot_r][pivot_c]

    return new_table
```

Receving solution after simplex method

In [ ]:
```
def is_basis(col):
    return (len([i for i in col[:-2] if i == 0]) == len(col) - 3) and (sum(col[:-2]) == 1)

# Recieve var values for optimized solution
def get_solution(table):
    cols = np.array(table).T
    solutions = []
    for c in cols[:-2]:
        if(is_basis(c)):
            one_index = c.tolist().index(1)
            solutions.append(cols[-2][one_index])
        else:
```

```
            solutions.append(0)

    return solutions
```

Main functions

In [ ]:
```
def simplex_calc_to_min(c, A, b, b_idx):
    table = create_table(c, A, b, b_idx)
    calc_deltas(table)
    print('Table created from input data: ')
    print_t(table)

    i = 0
    while(could_be_optimized_to_min(table)):
        # Receveing pivot element idxes and revalculate other values to change basis
        table = table_recalc(table, find_pivot_min(table))
        calc_deltas(table)
        i = i + 1
        print('Table after step', i)
        print_t(table)

    return get_solution(table)
```

In [ ]:
```
# To get min/max func value
def calc_function(coefs, var_values):
    return np.dot(coefs, var_values)
```

User input and data preparation

In [ ]:
```
# Setting up condition matrix and func vector
c = [3, -2, 0, 0, 0]    # F = 3x_1 - 2x_2
A = [
    [ 2, 1, 1, 0, 0],
    [-3, 2, 0, 1, 0],
    [ 3, 4, 0, 0, 1]
]                       # Condition matrix
b = [14, 9, 27]
basis_idx = [2, 3, 4]   # Select basis vector (x_3, x_4, x_5)
```

In [ ]:
```

```

In [ ]:
```
print('Function vector: c =', c)
print('Condition vector b =', b)
print('Condition matrix A:')
print_t(A)
print()

# Call simplex method with current input
print('Finding minimum:')
solution = simplex_calc_to_min(c, A, b, basis_idx)
print('X =', solution)
print('F(X) =', calc_function(c, solution))
```

```
Function vector: c = [3, -2, 0, 0, 0]
Condition vector b = [14, 9, 27]
Condition matrix A:
[2, 1, 1, 0, 0]
[-3, 2, 0, 1, 0]
[3, 4, 0, 0, 1]


Finding minimum:
Table created from input data:
[2, 1, 1, 0, 0, 14, 0]
[-3, 2, 0, 1, 0, 9, 0]
[3, 4, 0, 0, 1, 27, 0]
[3, -2, 0, 0, 0, 0, 0]
[-3, 2, 0, 0, 0, 0, 0]

Table after step 1
[3.5, 0.0, 1.0, -0.5, 0.0, 9.5, 0]
[-1.5, 1.0, 0.0, 0.5, 0.0, 4.5, -2]
[9.0, 0.0, 0.0, -2.0, 1.0, 9.0, 0]
[3, -2, 0, 0, 0, 0, 0]
[0.0, 0.0, 0.0, -1.0, 0.0, -9.0, 0]

X = [0, 4.5, 9.5, 0, 9.0]
F(X) = -9.0
```

Example for maximum (Variant 1)

In [ ]:
```
# Setting up condition matrix and func vector
```

```
c = [2, -1, 3, -2, 1]
A = [
    [-1, 1, 1, 0, 0],
    [ 1,-1, 0, 1, 0],
    [ 1, 1, 0, 0, 1]
]                       # Condition matrix
b = [1, 1, 2]
basis_idx = [2, 3, 4]   # Select basis vector (x_3, x_4, x_5)
```

In [ ]:

```
print('Function vector: c =', c)
print('Condition vector b =', b)
print('Condition matrix A:')
print_t(A)
print()

# Call simplex method with current input
print('Finding maximum:')
solution = simplex_calc_to_min([-x for x in c], A, b, basis_idx)
print('X =', solution)
print('F(X) =', calc_function(c, solution))
```

```
Function vector: c = [2, -1, 3, -2, 1]
Condition vector b = [1, 1, 2]
Condition matrix A:
[-1, 1, 1, 0, 0]
[1, -1, 0, 1, 0]
[1, 1, 0, 0, 1]


Finding maximum:
Table created from input data:
[-1, 1, 1, 0, 0, 1, -3]
[1, -1, 0, 1, 0, 1, 2]
[1, 1, 0, 0, 1, 2, -1]
[-2, 1, -3, 2, -1, 0, 0]
[6, -7, 0, 0, 0, -3, 0]

Table after step 1
[0.0, 0.0, 1.0, 1.0, 0.0, 2.0, -3]
[1.0, -1.0, 0.0, 1.0, 0.0, 1.0, -2]
[0.0, 2.0, 0.0, -1.0, 1.0, 1.0, -1]
[-2, 1, -3, 2, -1, 0, 0]
[0.0, -1.0, 0.0, -6.0, 0.0, -9.0, 0]

X = [1.0, 0, 2.0, 0, 1.0]
F(X) = 9.0
```