MATH 303
REPORT 1

11/11/2024

Revised Simplex Method

PREPARED BY :

Mohamed Ehab Yousri 202201236
Hamza Abdelmoreed 202201508
Amin Gamal 202202219

The Revised Simplex Method

1. Motivation:

The Revised Simplex Method is an efficient way used to improve Linear Programming Problems (LPPs), comparing by traditional simplex method this method works on recalculating an each table in the iterations. Moreover, it is more useful for large dataset problems by reducing unnecessary information and focusing only on the key parts solutions and indicating the basic matrix that is reduced from workload.

The process just depends on updating what's necessary at each step, the method avoids repetition, reworking and unessential operation.
This method is useful for more fields such as economics and engineering. In these fields the Revised Simplex methods make them take faster decisions and be more smarter and work with large datasets

2. Theory

By focusing on just the essential elements at each iteration, the Revised Simplex Method improves computing efficiency for solving Linear Programming Problems (LPPs). It preserves and updates only the basis and its related components, enabling faster convergence and lower memory consumption than the classical Simplex Method, which recalculates the complete tableau.

2.1 Non-Basis Representation and Basis Matrix:

The variables in an LPP are separated into basic and non-basic variables using the Revised Simplex Method. A basis matrix B, which represents the basic variables that meet the linear constraints, is used to represent the solution.
Non-basic variables stay at zero, and only the basis matrix B and its inverse B^−1 are used at each iteration. By separating the elements required to update the answer, this basis matrix optimizes processes.

2.2 reduced Cost Calculation and Optimality Condition:

The reduced costs for non-basic variables determines the direction of improvement in the objective function . The reduced cost vector c′ is calculated as follows: cN=cN−cB*B^−1N, where N is the matrix of columns that correspond to non-basic variables and cNc and cB are the cost coefficients for non-basic and basic variables, respectively.
The current solution is considered optimal if all entries in the reduced cost vector cN⩾0 (for a maximization problem). Otherwise, since it indicates the path with the highest potential for improving the objective function, the variable with the largest negative reduced cost is selected as the entering variable. If the entering variable goes up, this vector shows how the values of the basic variables will change.

2.3 Direction Vector and Minimum Ratio Test:

Then, to ensure viability, the minimum ratio test is used. Without going against any constraints, it calculates the maximum increment that can be made to the entering variable:
$\theta = \min(XB/D)$, where xB represents the basic variables' current values. In order to preserve feasibility, this phase determines the leaving variable.

2.4 Pivoting and Basis Update:

After selecting the entering and leaving variables, the basis matrix B is updated to reflect the new basis.
This updating process allows the method to enhance efficiently, reducing computational and memory requirements compared to the full Simplex Method.

3. Algorithm

The Recap on the algorithm

Step 1: Formalize the problem in standard form – I.
Step 2: In the revised simplex form, build the starting table.
Step 3: For $a1(1)$ and $a2(1)$, Compute Reduced Costs (Δj).
Step 4: Conduct an optimality test.
Step 5: Determine the direction vector XK column vector.
Step 6: Ratio test to Update the basic & non basic variables (Entering & Leaving).
Step 7: Recompute B-inverse for the new updates.

Revised Simplex Algorithm

1.Initiate with a Basic feasible solution

- Formalize the problem in standard form
- Inequalities are converted to equation with non-negative slack variables
- Define The basic matrix B for this solution and design CB as the objective coefficient vector corresponding to B.

2. **Compute the inverse of the Basic Matrix**

- Calculate the inverse of B of the basis matrix B

3. **Compute Reduced Costs for Non-Basic Vectors**

- For each non-Basic vector Pj
- Calculate the reduced cost Zj - Cj = CB . B^-1 . Pj - Cj.

4. **Optimality check :**

- If Zj - Cj >= 0 for maximization problems (or <= 0 for minimization problems) for all non-basic vectors.
- Stop if the current solution is optimal.
- The optimal solution values are XB = B^-1 * b and z = CB XB .
- If not optimal, determine the entering vector Pj

5.Calculate the Direction of Movement:

- Determine the Xk column vector
- Compute B^-1 . Pj

6.**Update the Basis**

- Form the next basis by replacing the leaving vector Pj with the entering vector Pj
- build New B inverse

Implementation Of The Revised Method

```python
def Revised_semplex(c, A, RHS):

  #C: coefficiant matrix
  #A : constraint matrix >> shape : m,n
  #b : np.array : Right-hand side vector
  m, n = A.shape
  basic_var = np.arange(m)  # initialize basis
  non_basic = np.arange(m, n)  # non-basic var

  # feasible solution
  B_inverse = np.linalg.inv(A[:, basic_var])
  basic_sol = B_inverse @ RHS
  solution = np.zeros(n)
  solution[basic_var] = basic_sol  # solution >> basic variables

  iteration = 0
  while True:
      # Reduced Costs for nonbasic Vectors (Δj)
      basic_cost = c[basic_var]
      non_basic_cost = c[non_basic]
      Updated_rcost = non_basic_cost - (basic_cost.T @ B_inverse @ A[:, non_basic]) # reduced cost

      # optimality check
      if all(Updated_rcost <= 0):  # Stop if all reduced costs
          z_optimal = c @ solution
          #print(f"Optimal solution found in {iteration} iterations.")
          return solution, z_optimal, "Optimal_value"

      # find entering variable
      enter_index = np.argmax(Updated_rcost)
      enter_var = non_basic[enter_index]

      # Compute direction vector (Xk column vector)
      dir_vector = B_inverse @ A[:, enter_var]

      if all(dir_vector <= 0):
          print("The solution is unbounded.")
          return solution, None, "Unbounded"

      # using ratio test
      ratios = np.where(dir_vector > 0, basic_sol / dir_vector, np.inf)
      leav_index = np.argmin(ratios)
      leav_var = basic_var[leav_index]

      # Update the basis
      non_basic[enter_index] = leav_var
      basic_var[leav_index] = enter_var

      # Recompute B_inverse
      B_inverse = np.linalg.inv(A[:, basic_var])
      basic_sol = B_inverse @ RHS
      solution[basic_var] = basic_sol  # Update solution > new basic var
      iteration +=1
```

Regular Simplex

Regular

```
[3] cof = np.array([5, 4, 0, 0, 0, 0 ])

    A = np.array([
        [6, 4, 1, 0, 0, 0],
        [1, 2, 0, 1, 0, 0],
        [-1, 1, 0, 0, 1, 0],
        [0, 1, 0, 0, 0, 1]

    ])
    RHS = np.array([24, 6, 1, 2])

    solution, optimal_value, status = Revised_semplex(cof, A, RHS)

    print("Solution:", solution)
    print("Optimal Value:", optimal_value)
    print("Status:", status)
```

```
Solution: [3.  1.5 4.  1.  2.5 0.5]
Optimal Value: 21.0
Status: Optimal_value
```

Built-In function - Regular

```
from scipy.optimize import linprog
cof = [-5,-4]
A = [[6, 4],[1, 2],[-1, 1],[0, 1]]

RHS = [24, 6, 1, 2]
x0_bounds = (0, None)
x1_bounds = (0, None)
result = linprog(cof, A_ub=A, b_ub=RHS, bounds=(x0_bounds,x1_bounds),method ='revised simplex
print(result)
```

```
 message: Optimization terminated successfully.
 success: True
  status: 0
     fun: -21.0
       x: [ 3.000e+00  1.500e+00]
     nit: 2
<ipython-input-4-ee5777cb7edf>:8: DeprecationWarning: `method='revised simplex'` is deprecated and will be
  result = linprog(cof, A_ub=A, b_ub=RHS, bounds=(x0_bounds,x1_bounds),method ='revised simplex' )
```

2-Phase-Simplex

```
[5] cof = np.array([2, 4, 0, 0, 0, 0])
    A = np.array([
        [1, 1, 1, 0, 0, 0],
        [6, 4, 0, 1, 0, -1],
        [1, 4, 0, 0, 1, 0]
    ])
    RHS = np.array([8, 12, 20])

    solution, z_optimal, status = Revised_semplex(cof, A, RHS)

    print("Optimal Solution:", solution)
    print("Optimal Value:", z_optimal)
    print("Status:", status)
```

```
Optimal Solution: [ 4.   4.   4.2  0.   0.  28. ]
Optimal Value: 24.0
Status: Optimal_value
```

2-phase-simplex - Built-In

```
[6] from scipy.optimize import linprog
    cof = [-2,-4]
    A = [[1, 1],[-6, -4],[1, 4]]

    RHS = [8, 12, 20]
    x0_bounds = (0, None)
    x1_bounds = (0, None)
    result = linprog(cof, A_ub=A, b_ub=RHS, bounds=[x0_bounds, x1_bounds],method='revised simplex
    print(result)
```

```
 message: Optimization terminated successfully.
 success: True
  status: 0
     fun: -24.0
       x: [ 4.000e+00  4.000e+00]
     nit: 2
<ipython-input-6-745505ba058f>:8: DeprecationWarning: `method='revised simplex'` is deprecated and will be
  result = linprog(cof, A_ub=A, b_ub=RHS, bounds=[x0_bounds, x1_bounds],method='revised simplex')
```

Degeneracy

Degeneracy

```
cof = np.array([3, 9, 0, 0])
A = np.array([
    [1, 4, 1, 0],
    [1, 2, 0, 1]])
RHS = np.array([8, 4])


solution, optimal_value, status = Revised_semplex(cof, A, RHS)

# Print the results
print("Solution", solution)
print("Optimal Value", optimal_value)
print("Status", status)
```

```
Solution [0. 2. 0. 0.]
Optimal Value 18.0
Status Optimal value
```

Built-In function - Degeneracy

```
[8] from scipy.optimize import linprog
    cof = [-3, -9]
    A = [[1, 4],[1, 2]]

    RHS = [8, 4]
    x0_bounds = (0, None)
    x1_bounds = (0, None)
    result = linprog(cof, A_ub=A, b_ub=RHS, bounds=(x0_bounds,x1_bounds),method = 'revised simplex
    print(result)
```

```
 message: Optimization terminated successfully.
 success: True
  status: 0
     fun: -18.0
       x: [ 0.000e+00  2.000e+00]
     nit: 2
<ipython-input-8-5bd4dc0d446c>:8: DeprecationWarning: `method='revised simplex'` is deprecated and will be
  result = linprog(cof, A ub=A, b ub=RHS, bounds=(x0 bounds,x1 bounds),method = 'revised simplex' )
```

Alternative-Optima

```
cof = np.array([2, 4, 0, 0])
A = np.array([[1, 2, 1, 0],
              [1, 1, 0, 1]])
RHS = np.array([5, 4])


solution, optimal_value, status = Revised_semplex2(cof, A, RHS)

# Print the results
print("Solution", solution)
print("Optimal Value", optimal_value)
print("Status", status)
```

```
Alternative optimal solutions exist.
Solution [3. 1. 0. 0.]
Optimal Value 10.0
Status Optimal_value
```

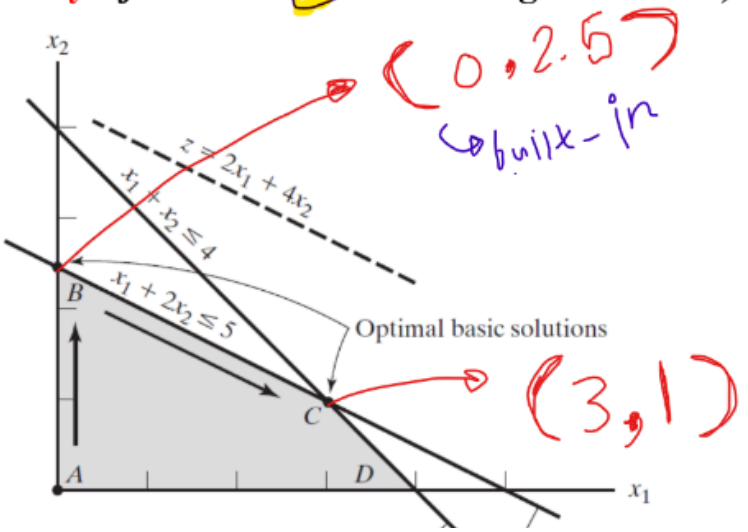
[10]

```
from scipy.optimize import linprog
cof = [-2, -4]
A = [[1, 2],[1, 1]]
RHS = [5, 4]
x0_bounds = (0, None)
x1_bounds = (0, None)
result = linprog(c = cof, A_ub=A, b_ub=RHS, bounds=(x0_bounds,x1_bounds),method = 'revised simplex' )
print(result)
```

```
 message: Optimization terminated successfully.
 success: True
  status: 0
     fun: -10.0
       x: [ 0.000e+00  2.500e+00]
     nit: 1
<ipython-input-11-63802a8aba78>:7: DeprecationWarning: `method='revised simplex'` is deprecated and will be removed in SciPy 1.11.0. Please use one of the HiGHS solv
  result = linprog(c = cof, A_ub=A, b_ub=RHS, bounds=(x0_bounds,x1_bounds),method = 'revised simplex' )
```

Simplex Method-Alternative Optima

Graphically: (*f*-contours // the binding constraint)



(0, 2.5)
built-in

(3, 1)

Unbounded solution

```
obj = np.array([2, 1, 0, 0])

lhs = np.array([
    [1, -1, 1, 0],
    [2, 0, 0, 1]
])
rhs = np.array([10, 40])

solution, optimal_value, status = Revised_semplex3(obj, lhs, rhs)

# Print the results
print("Solution:", solution)
print("Optimal Value:", optimal_value)
print("Status:", status)
```

```
The solution is unbounded.
Solution: [20. 10.  0.  0.]
Optimal Value: 50.0
Status: Unbounded
```

Unbounded - Built-in

```
obj = np.array([-2, -1])
lhs_ineq = np.array([
    [1, -1],
    [2, 0]
])
rhs_ineq = np.array([10, 40])

x0_bounds = (0, None)
x1_bounds = (0, None)
result = linprog(obj, A_ub=lhs_ineq, b_ub=rhs_ineq, bounds=(x0_bounds,x1_bounds),method = 'revised simplex' )
print(result)
```

```
 message: The problem is unbounded, as the simplex algorithm found a basic feasible solution from which there is a direction with negative reduced cost in which all decision variabl
 success: False
  status: 3
     fun: -50.0
       x: [ 2.000e+01  1.000e+01]
     nit: 2
C:\Users\HP\AppData\Local\Temp\ipykernel_18724\1599059805.py:10: DeprecationWarning: `method='revised simplex'` is deprecated and will be removed in SciPy 1.11.0. Please use one of
  result = linprog(obj, A_ub=lhs_ineq, b_ub=rhs_ineq, bounds=(x0_bounds,x1_bounds),method = 'revised simplex' )
```

Infeasible solution

```
c = np.array([3, 2, 0, 0 ])

A = np.array([
    [2, 1, 1, 0],
    [3, 4, 0, 1]
])
b = np.array([2, 12])

solution, optimal_value, status = Revised_simplex4(c, A, b)

# Print the results
print("Solution:", solution)
print("Optimal Value:", optimal_value)
print("Status:", status)
```

```
The problem is infeasible.
Solution: None
Optimal Value: None
Status: Infeasible
```

Infeasible - Built-in

```
obj = np.array([-3, -2])
lhs_ineq = np.array([
    [2, 1],
    [-3, -4]
])
rhs_ineq = np.array([2, -12])
x0_bounds = (0, None)
x1_bounds = (0, None)
result = linprog(obj, A_ub=lhs_ineq, b_ub=rhs_ineq, bounds=(x0_bounds,x1_bounds),method = 'revised simplex' )
print(result)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_18724\2286135431.py:9: DeprecationWarning: `method='revised simplex'` is deprecated and will be removed in SciPy 1.11.0. Please use one
  result = linprog(obj, A_ub=lhs_ineq, b_ub=rhs_ineq, bounds=(x0_bounds,x1_bounds),method = 'revised simplex' )
 message: The problem appears infeasible, as the phase one auxiliary problem terminated successfully with a residual of 4.0e+00, greater than the tolerance 1e-12 required for th
 success: False
  status: 2
     fun: -4.0
       x: [ 0.000e+00  2.000e+00]
     nit: 1
```

References

1- Taha, H. A. (2017). Operations Research: An Introduction (10th ed.). Pearson Education Limited
https://drive.google.com/file/d/1X8gAHz0TK-ynBiIEY3Gx9cQjgXOrPBbX/view

2- Boyd, S., & Vandenberghe, L. (2004). Convex Optimization. Cambridge University .
https://www.cambridge.org/us/universitypress/subjects/statistics-probability/optimization-or-and-risk/convex-optimization?format=HB&isbn=9780521833783

3- Rao, S. S. (2020). Engineering Optimization: Theory and Practice (5th ed.). John Wiley & Sons.
https://drive.google.com/drive/home?dmr=1&ec=wgc-drive-hero-goto

4- https://youtu.be/SVgMPjAqiEw?feature=shared

Colab Notebook Link :

https://colab.research.google.com/drive/1dPpzYedPpsheNNqlDqCDqdcwZsequtX9?usp=sharing