

Optimization Algorithms: Theory and Software Implementation

Prof. Thirumulanathan D

Department of Mathematics

Institute of IIT Kanpur

Lecture: 45

Hello everyone. This is lecture 5 of week 9. We have been learning about the simplex method, and in the last lecture, we presented the full algorithm.

You may have noticed that this algorithm is very different from the ones we have studied until now. All algorithms for unconstrained optimization followed a specific format: find a descent direction, determine a step size, and define a stopping criterion. Even for non-linear programming, methods like the quadratic penalty and augmented Lagrangian function had a similar structure, often involving inner and outer loops, but still based on searching for descent directions and step sizes.

The simplex algorithm is fundamentally different. It also has a stopping criterion, which is the condition that $c_N^T - c_B^T B^{-1}N \geq 0$ for all elements. However, the search is not for a descent direction and step size. Instead, the search is for the optimal vertex through the correct exchange of basic and non-basic variables.

We will now look at an example and complete the Python code for the simplex method.

Example

Consider the same example we have been working with:

Minimize: $-x_2$

Subject to:

$$-x_1 + x_2 \geq 0$$

$$x_1 + x_2 \leq 1$$

$$x_2 \geq 0$$

The first step is to convert this Linear Program (LP) into standard form. We introduce slack variables x_3 and x_4 and handle the free variable x_1 by splitting it into positive and negative parts ($x_1 = x_1^+ - x_1^-$, where $x_1^+ \geq 0, x_1^- \geq 0$).

The problem in standard form becomes:

Minimize: $-x_2$

Subject to:

$$-x_1^+ + x_1^- + x_2 - x_3 = 0$$

$$x_1^+ - x_1^- + x_2 + x_4 = 1$$

$$x_1^+ \geq 0, x_1^- \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0.$$

We can write this in matrix form:

$$\text{Minimize } c^T x$$

$$\text{Subject to } Ax = b, x \geq 0$$

Where:

$$* \quad c^T = [0, 0, -1, 0, 0]$$

$$* \quad A = [[-1, 1, 1, -1, 0], [1, -1, 1, 0, 1]]$$

$$* \quad b = [0, 1]$$

$$* \quad x = [x_1^+, x_1^-, x_2, x_3, x_4]^T$$

(Refer slide Time 3:49)

3. While $((c^T - c^T B^{-1} N)[i] < 0 \text{ for some } i)$:

- * $q = \text{argmin}_i ((c^T - c^T B^{-1} N)[i])$
- * $I = \{i: (B^{-1} N)[i, q] > 0\}$. If $I = \emptyset$, then print ("unbounded") and exit.
- * $j = \text{argmin}_{i \in I} \frac{(B^{-1} b)[i]}{(B^{-1} N)[i, q]}$
- * Make x_q basic, and x_j non-basic.

4. Solution: $(x_B^*, x_N^*) = (B^{-1} b, 0)$.

What happens if $\{i: (B^{-1} N)[i, q] > 0\}$ is empty?

The problem is UNBOUNDED.

$\max_x \quad x_1 + x_2 \quad \text{s.t.} \quad \{x_1 + x_2 \geq 1, x_1, x_2 \geq 0\} \rightarrow \text{UNBOUNDED.}$

$\min_x \quad (-x_2) \quad \text{s.t.} \quad \{-x_1 + x_2 \leq 0, x_1 + x_2 - 1 \leq 0, -x_2 \leq 0\}$

$\min \quad -x_2$

s.t. $\begin{bmatrix} -1 & 1 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1^+ \\ x_1^- \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, x_1^+, x_1^-, x_2, x_3, x_4 \geq 0.$

Simplex Iteration Walkthrough

Step 1: Choose an initial basic feasible solution.

We choose the columns for the slack variables x_3 and x_4 to form our initial basis matrix B , as they form an identity matrix.

Therefore, we define our sets:

$$* \quad \text{Basic variables } x_B : [x_3, x_4]$$

- * Non-basic variables $x_N : [x_1^+, x_1^-, x_2]$
- * Basis matrix $B : [[1, 0], [0, 1]]$
- * Non-basis matrix $N : [[-1, 1, 1], [1, -1, 1]]$
- * Cost vector for basic variables $c_B : [0, 0]$
- * Cost vector for non-basic variables $c_N : [0, 0, -1]$

The initial basic feasible solution is:

- * $x_B = B^{-1}b = I * [0, 1] = [0, 1]$
- * $x_N = [0, 0, 0]$

So, the full solution vector is $x = [0, 0, 0, 0, 1]$. This corresponds to the vertex $(x_1, x_2) = (0, 0)$.

Step 2: Check the Optimality Condition.

We compute the vector $c_N^T - c_B^T B^{-1}N$.

- * $c_N^T = [0, 0, -1]$
- * $c_B^T B^{-1}N = [0, 0] * I * N = [0, 0, 0]$
- * $c_N^T - c_B^T B^{-1}N = [0, 0, -1]$

This vector is not non-negative (the third element is -1). Therefore, the current solution is not optimal, and we proceed.

Step 3: Determine the Entering Variable (q).

The entering variable is the one with the most negative reduced cost. The argmin of $[0, 0, -1]$ is the third element. Since our $x_N = [x_1^+, x_1^-, x_2]$, the third non-basic variable, x_2 , enters the basis.

$q = \text{index of } x_2 \text{ in } N \text{ (which is 2)}$

Step 4: Determine the Leaving Variable (j).

We compute the direction of movement:

- * $B^{-1}N(:, q) = I * [1, 1] = [1, 1]$

Both elements are positive, so we compute the ratios to find the limiting constraint:

- * $B^{-1}b = [0, 1]$
- * Ratios: $0/1 = 0, 1/1 = 1$

The minimum ratio is 0, which corresponds to the first basic variable. Our $x_B = [x_3, x_4]$, so the first basic variable, x_3 , leaves the basis.

$j = \text{index of } x_3 \text{ in } B \text{ (which is 0)}$

Step 5: Pivot.

We update our sets by exchanging the entering and leaving variables:

- * New basic variables $x_B : [x_2, x_4]$
- * New non-basic variables $x_N : [x_1^+, x_1^-, x_3]$

(Refer Slide Time 13:14)

Handwritten mathematical derivation for the pivot step of the simplex algorithm:

$$A = \begin{bmatrix} -1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad N = \begin{bmatrix} -1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad x_B = \begin{bmatrix} x_2 \\ x_4 \end{bmatrix}, \quad x_N = \begin{bmatrix} x_1^+ \\ x_1^- \\ x_3 \end{bmatrix}$$

$$c^T x = 0x_1^+ + 0x_1^- + (-1)x_2 + 0x_3 + 0x_4, \quad c_B^T = [0, 0], \quad c_N^T = [0, 0, -1], \quad b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

$$(x_B, x_N) = (0, 1, 0, 0, 0) = (x_2, x_4, x_1^+, x_1^-, x_3)$$

$$c_N^T - c_B^T B^{-1} N = [0, 0, -1] - [0, 0] \begin{bmatrix} -1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = [0, 0, -1] - [0, 0, 0] = [0, 0, -1].$$

$$\therefore q = \arg \min_i (c_N^T - c_B^T B^{-1} N)[i] = 2 \Rightarrow x_2 \text{ becomes basic.}$$

$$I = \{i : (B^{-1} N)[i, q] > 0\}. \quad B^{-1} N = \begin{bmatrix} -1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad (B^{-1} N)[i, q] = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\therefore I = \{3, 4\}.$$

$$j = \arg \min_i \left(\frac{(B^{-1} b)[i]}{(B^{-1} N)[i, q]} \right) = \arg \min \left(\frac{0}{1}, \frac{1}{1} \right) = 3 \Rightarrow x_3 \text{ non-basic.}$$

Next iteration: Basic $\rightarrow x_2, x_4$, Non-basic $\rightarrow x_1^+, x_1^-, x_3$.

We will now perform this exchange computationally.

Python Implementation

We now code the simplex algorithm based on the steps above.

Python

```
import numpy as np

# - Problem Definition for the 5-Variable Example ---

# Variable indices:

# 0: x1_plus (x1^+),
# 1: x1_minus (x1^-),
# 2: x2,
# 3: x3 (slack),
# 4: x4 (slack)
```

```

# Initial basic set: [x3, x4] -> indices [3, 4]
# Initial non-basic set: [x1_plus, x1_minus, x2] -> indices [0, 1, 2]
# Initialize matrices and vectors

B = np.array([[1, 0], # Columns for x3, x4
              [0, 1]])

# Columns for x1_plus, x1_minus, x2 from matrix A
N = np.array([[ -1,  1,  1],
              [ 1, -1,  1]])

c_B = np.array([0, 0]) # Costs for basic variables [x3, x4]
c_N = np.array([0, 0, -1]) # Costs for non-basic variables [x1_plus, x1_minus, x2]
b = np.array([0, 1])

B_set = [3, 4] # Indices of basic variables
N_set = [0, 1, 2] # Indices of non-basic variables
# Initialize solution vector x (size = 5)

x = np.zeros(5)

# Set values for basic variables:  $x_B = B^{-1} * b$ 
x[B_set] = np.linalg.inv(B) @ b

iter = 0

print(f'Iteration {iter}: x = {x}')
print(f'B_set = {B_set}, N_set = {N_set}\n')

# --- Main Simplex Loop ---

# Continue while any reduced cost is negative
while np.any((c_N - c_B @ np.linalg.inv(B) @ N) < 0):

    # Step 1: Find entering variable (q)

    reduced_costs = c_N - c_B @ np.linalg.inv(B) @ N
    q = np.argmin(reduced_costs) # Index in N_set
    print(f'Most negative reduced cost: {reduced_costs[q]} for variable x{N_set[q]}')

    # Step 2: Find leaving variable (j)

    # Compute direction vector for the entering variable

```

```

d = np.linalg.inv(B) @ N[:, q]

# Compute ratios (B^{-1}b / d), avoid division by zero/negative
ratios = np.where(d > 0, (np.linalg.inv(B) @ b) / d, np.inf)

min_ratio = np.inf

j = -1

unbounded = True # Flag to check unboundedness

# Find the index j in the basic set with the minimum positive ratio
for i in range(len(B_set)):
    if d[i] > 0:
        unbounded = False

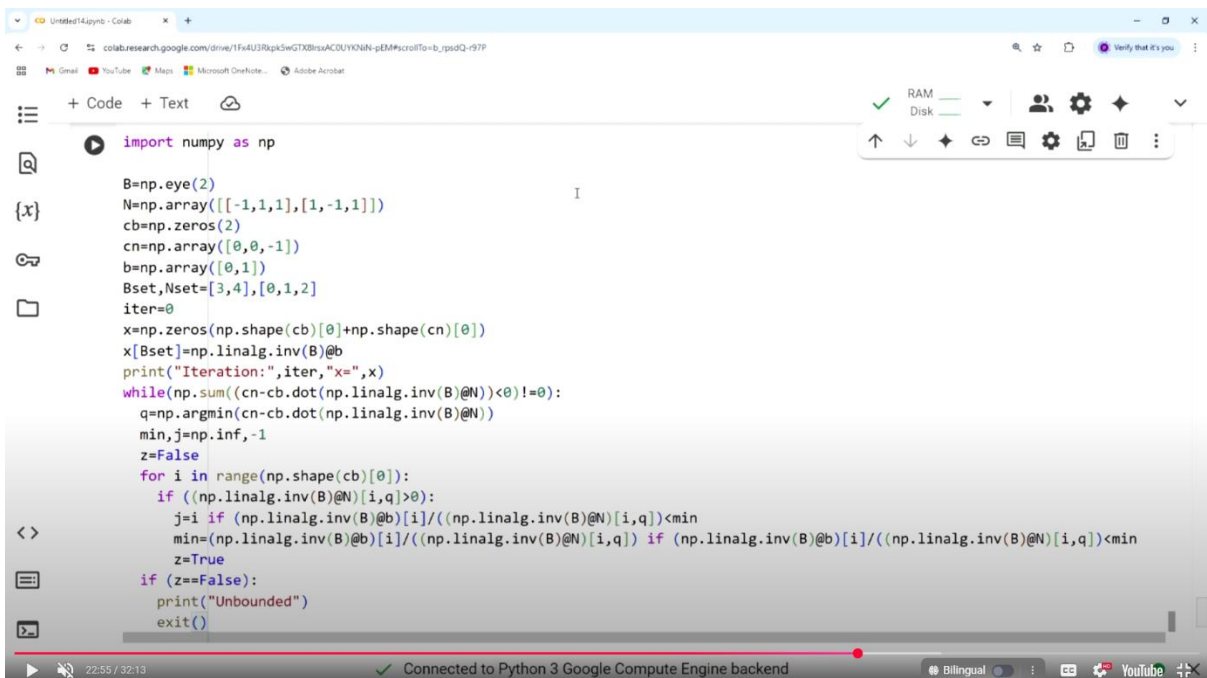
        if ratios[i] < min_ratio:
            min_ratio = ratios[i]
            j = i # Index in B_set

if unbounded:
    print("Problem is unbounded.")
    break

print(f'Minimum ratio: {min_ratio} for basic variable x_{B_set[j]}')

```

(Refer Slide Time 22:55)



```

import numpy as np

B=np.eye(2)
N=np.array([[ -1,1,1],[1,-1,1]])
cb=np.zeros(2)
cn=np.array([0,0,-1])
b=np.array([0,1])
Bset,Nset=[3,4],[0,1,2]
iter=0
x=np.zeros(np.shape(cb)[0]+np.shape(cn)[0])
x[Bset]=np.linalg.inv(B)@b
print("Iteration:", iter, "x=", x)
while(np.sum((cn-cb.dot(np.linalg.inv(B)@N))<0)!=0):
    q=np.argmax(cn-cb.dot(np.linalg.inv(B)@N))
    min,j=np.inf,-1
    z=False
    for i in range(np.shape(cb)[0]):
        if ((np.linalg.inv(B)@N)[i,q]>0):
            j=i if (np.linalg.inv(B)@b)[i]/((np.linalg.inv(B)@N)[i,q])<min
            min=(np.linalg.inv(B)@b)[i]/((np.linalg.inv(B)@N)[i,q]) if (np.linalg.inv(B)@b)[i]/((np.linalg.inv(B)@N)[i,q])<min
    z=True
    if (z==False):
        print("Unbounded")
        exit()

```

```

# Step 3: Pivot - Exchange variables between B_set and N_set

# Exchange the j-th basic variable with the q-th non-basic variable
print(f"Pivoting: x{B_set[j]} (leaves), x{N_set[q]} (enters)\n")

B_set[j], N_set[q] = N_set[q], B_set[j]

# Exchange the j-th column of B with the q-th column of N

B_col_j = B[:, j].copy()
N_col_q = N[:, q].copy()

B[:, j] = N_col_q
N[:, q] = B_col_j

# Exchange the corresponding cost coefficients

c_B_j = c_B[j]
c_N_q = c_N[q]
c_B[j] = c_N_q
c_N[q] = c_B_j

# Update the solution vector for the new basis

x = np.zeros(5)

x[B_set] = np.linalg.inv(B) @ b

iter += 1

print(f"Iteration {iter}: x = {x}")

print(f"B_set = {B_set}, N_set = {N_set}\n")

# Final output

print("--- Simplex Completed ---")

print("Optimal solution found:")

print(f"x1+ = {x[0]}")
print(f"x1- = {x[1]}")
print(f"x2 = {x[2]}")
print(f"x3 = {x[3]}")
print(f"x4 = {x[4]}")

print(f"\nTherefore, original x1 = x1+ - x1- = {x[0] - x[1]}")

```

```
print(f"Original  $x_2 = \{x[2]\}$ ")
```

```
print(f"Optimal objective value =  $\{-x[2]\}$ ")
```

Expected Output and Interpretation

The code will perform the simplex steps. The final optimal solution will be:

$$x_1^+ = 0.5$$

$$x_1^- = 0$$

$$x_2 = 0.5$$

$$x_3 = 0$$

$$x_4 = 0$$

Interpretation:

- * Original $x_1 = x_1^+ - x_1^- = 0.5 - 0 = 0.5$

- * Original $x_2 = 0.5$

- * The optimal point is $(x_1, x_2) = (0.5, 0.5)$, which matches our previous result. The slack variables are zero, meaning the constraints $-x_1 + x_2 = 0$ and $x_1 + x_2 = 1$ are active (binding) at the solution.

This code provides a foundational implementation of the simplex method. To solve other LPs, one must modify the initial definitions of `B`, `N`, `c_B`, `c_N`, `b`, `B_set`, and `N_set` accordingly.

(Refer Slide Time 30:46)

```
min=(np.linalg.inv(B)@b)[i]/((np.linalg.inv(B)@N)[i,q]) if (np.linalg.inv(B)@b)[i]
z=True
if (z==False):
    print("Unbounded")
    exit()
Bset[j],Nset[q]=Nset[q],Bset[j]
B[:,j],N[:,q]=N[:,q],B[:,j]
cb[j],cn[q]=cn[q],cb[j]
iter=iter+1
x=np.zeros(np.shape(cb)[0]+np.shape(cn)[0])
x[Bset]=np.linalg.inv(B)@b
print("Iteration:",iter,"x=",x,"Bset=",Bset,"Nset=",Nset)

Iteration: 0 x= [0. 0. 0. 0. 1.] Bset= [3, 4] Nset= [0, 1, 2]
Iteration: 1 x= [0. 0. 0. 0. 1.] Bset= [2, 4] Nset= [0, 1, 3]
Iteration: 2 x= [0.5 0. 0.5 0. 0.] Bset= [2, 0] Nset= [4, 1, 3]
```


We will continue with more examples in the next week before moving on to a new algorithm.
Thank you.