Department of Mathematics and Computer Science
University of Southern Denmark, Odense

April 10, 2025
Marco Chiarandini

AI505 – Optimization

Sheet 06, Spring 2025

Solution:

Included.

Exercises with the symbol $^+$ are to be done at home before the class. Exercises with the symbol * will be tackled in class. The remaining exercises are left for self training after the exercise class. Some exercises are from the text book and the number is reported. They have the solution at the end of the book.

Exercise 1$^+$ (11.1)

Suppose you do not know any optimization algorithm for solving a linear program. You decide to evaluate all the vertices and determine, by inspection, which one minimizes the objective function. Give a loose upper bound on the number of possible minimizers you will examine. Furthermore, does this method properly handle all linear constrained optimization problems?

Solution:

We have chosen to minimize a linear program by evaluating every vertex in the convex polytope formed by the constraints. Every vertex is thus a potential minimizer. Vertices are defined by intersections of active constraints. As every inequality constraint can either be active or inactive, and assuming there are n inequality constraints, we do not need to examine more than 2^n combinations of constraints. This method does not correctly report unbounded linear constrained optimiza- tion problems as unbounded.

Exercise 2$^+$ (11.2)

If the program in example 11.1 is bounded below, argue that the simplex method must converge.

Solution:

The simplex method is guaranteed either to improve with respect to the objective function with each step or to preserve the current value of the objective function. Any linear program will have a finite number of vertices. So long as a heuristic, such as Bland's rule, is employed such that cycling does not occur, the simplex method must converge on a solution.

Exercise 3$^+$ (11.3)

Suppose we want to solve:

$$\begin{aligned} \text{minimize} \ & 6x_1 + 5x_2 \\ \text{subject to} \ & 3x_1 - 2x_2 \geq 5. \end{aligned}$$

How would we translate this problem into a linear program in equality form with the same minimizer?

Solution:

The problem is already in minimization form. Let's rewrite the inequality in less than equal form:

$$\begin{aligned} \text{minimize} \ & 6x_1 + 5x_2 \\ \text{subject to} \ & -3x_1 + 2x_2 \leq -5. \end{aligned}$$

Since both variables are free to put the problem in a form in which all variables are greater or equal to 0 we introduce two new variables for each original one:

$$x_1 = s_1 - s_2 \qquad x_2 = s_3 - s_4$$
$$s_1, s_2 \geq 0 \qquad s_3, s_4 \geq 0$$

which yields

$$\begin{aligned} \text{minimize} \;\; & 6s_1 - 6s_2 + 5s_3 - 5s_4 \\ \text{subject to} \;\; & -3s_1 + 3s_2 + 2s_3 - 2s_4 \leq -5 \\ & s_, s_2, s_3, s_4, s_5 \geq 0 \end{aligned}$$

Finally, adding a slack variable $s_5 \geq 0$:

$$\begin{aligned} \text{minimize} \;\; & 6s_1 - 6s_2 + 5s_3 - 5s_4 \\ \text{subject to} \;\; & -3s_1 + 3s_2 + 2s_3 - 2s_4 + s_5 = -5 \\ & s_, s_2, s_3, s_4, s_5 \geq 0 \end{aligned}$$

The simplex algorithm works by selecting a basis and then changing the basis. The initial basis has to be feasible. This implies that the martix A_B is invertible and that the solution $x = A^{-1}b \geq 0$. The basis in a problem of size 1×5 has size 1.
Let's select s_1 to be in basis.

```python
import numpy as np

c = np.array([6, -6,5,-5,0])
A = np.array([[-3,3,2,-2,1]])
b = np.array([-5])

# initial basis
B = np.array([0])
N = np.array([1,2,3,4])

#%%
print(c[B])
print(c[N])

#%%

def compute_solution(A,b,B):
    A_B_inv = np.linalg.inv(A[:,B])
    x_B = A_B_inv @ b
    return x_B, A_B_inv

def compute_reduced_costs(A, A_B_inv, b,c,B,N) -> np.array:
    muN = c[N]-np.transpose(A_B_inv @ A[:,N]) @ c[B]
    return muN

def choose_entering_var(muN, N):
    #q = np.argmin(muN)
    for i in range(N.shape[0]):
        if muN[i]<0:
            return N[i]

    raise SystemExit("Optimal")

def compute_leaving_var(A,A_B_inv, x_B,B,N,q):
    x_prime_q_s = np.zeros(A.shape[0])
    for i in range(B.shape[0]):
        # p=B[i]
        x_prime_q = x_B[i] / (A_B_inv @ A[:,q])[i]
        x_prime_q_s[i] = x_prime_q
    i = np.argmin(x_prime_q_s) # leaving var
    return (i, B[i], x_prime_q_s[i])

def run(A,b,c,B,N):
    niter = 1
    while True:
        print("Iteration",niter)
        x_B, A_B_inv = compute_solution(A,b,B)
        muN = compute_reduced_costs(A,A_B_inv,b,c,B,N)
        print("Reduced costs",muN)
        q = choose_entering_var(muN, N)
        print("Entering:",q)
        i, p, x_B_p = compute_leaving_var(A,A_B_inv,x_B,B,N,q)
        print("Leaving",p)
        x_B[i] = x_B_p
        B[i] = q
        N[N==q] = p
        print(B,N, x_B)
        niter+=1

run(A,b,c,B,N)
```

Exercise 4

Consider the following problem:

$$\begin{aligned}\text{minimize } & 5x_1 + 4x_2\\ \text{s.t. } & 2x_1 + 3x_2 \leq 5\\ & 4x_1 + x_2 \leq 11\end{aligned}$$

Solve the problem numerically implementing the simplex algorithm.

Solution:

$$\begin{aligned}\text{minimize } & 5x_1 + 4x_2\\ \text{s.t. } & 2x_1 + 3x_2 \leq 5\\ & 4x_1 + x_2 \leq 11\end{aligned}$$

Exercise 5$^+$ (11.4)

Suppose your optimization algorithm has found a search direction $\boldsymbol{d}$ and you want to conduct a line search. However, you know that there is a linear constraint $\boldsymbol{w}^T\boldsymbol{x} \geq \boldsymbol{0}$. How would you modify the line search to take this constraint into account? You can assume that your current design point is feasible.

Solution:
If the current iterate $\boldsymbol{x}$ is feasible, then $\boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{b} \geq \boldsymbol{0}$. We want the next point to maintain feasibility, and thus we require $\boldsymbol{w}^T(\boldsymbol{x} + \alpha\boldsymbol{d}) \geq \boldsymbol{0}$. If the obtained value for α is positive, that α is an upper bound on the step length. If the obtained value for α is negative, it can be ignored

Exercise 6$^+$ (11.5)

Reformulate the linear program

$$\begin{aligned}\text{minimize } & \boldsymbol{c}^T\boldsymbol{x}\\ \text{s.t. } & A\boldsymbol{x} \geq 0\end{aligned}$$

Drug	A	B	C		
raw material (Kg)	5	8	6		600
processing time (hours)	3	4	5		400
packaging units	2	3	1		200
profit	60	100	80	-5	

Table 1: Data from the pharmaceutical company

into an unconstrained optimization problem with a log barrier penalty.

Solution:

$$\text{minimize } \boldsymbol{c}^T \boldsymbol{x} - \mu \sum_i \ln(A^T i \boldsymbol{x})$$

Exercise 7*

A pharmaceutical company produces three types of drugs: A, B, and C. These drugs require raw materials, chemical processing time, and packaging units. The goal is to maximize profit, taking into account restrictions due to resource availability and production balance.
Drug A contributes 60 DKK per unit to the profit, drug B contributes 100 DKK per unit, drug C contributes 80 DKK per unit. The consumptions per units of drug of raw materials, chemical processing time, and packaging units are given in Table ?? together with the quantities available. While consumptions of raw material and processing time cannot exceed the amount available, excess of packaging units is allowed. Extra packaging units can be bought but each extra unit reduces profit at 5 DKK per unit while packaging units left can be reused in the next production period and hence contribute positively to the profit with the same amount of DKK per unit. Due to contractual obligations, Drug B must be produced in exactly twice the amount of Drug A.
Formulate the problem in linear programming terms. Write first the *instantiated* model and then the abstract, *general* model separating model from data.

Solution:
To model a problem in LP terms we need to define:

- the sets problem components and the known parameters
- the decision variables
- the objective function
- the constraints

Decision Variables

- x_1 Units of Drug A produced
- x_2 Units of Drug B produced
- x_3 Units of Drug C produced
- x_4 Additional packaging units (can be negative if surplus)

Objective Function (Maximization)

$$\text{maximize } Z = 60x_1 + 100x_2 + 80x_3 - 5x_4$$

Constraints
Raw Material Constraint (Inequality) The factory has at most 600 kg of raw material:

$$5x_1 + 8x_2 + 6x_3 \leq 600$$

Processing Time Constraint (Inequality) The available chemical processing time is 400 hours:

$$3x_1 + 4x_2 + 5x_3 \leq 400$$

Packaging Balance Constraint (Equality) The total packaging units used must match available packaging plus extra units:

$$2x_1 + 3x_2 + x_3 = 200 + x_4$$

If $x_4 > 0$, extra packaging was bought.
If $x_4 < 0$, excess packaging is left unused.
Drug B Production Requirement (Equality) Due to contractual obligations, Drug B must be produced in exactly twice the amount of Drug A:

$$x_2 = 2x_1$$

Non-negativity and Unbounded Variables
$x_1, x_2, x_3 \geq 0$ is unrestricted in sign
x_4 can be negative (if packaging is left over) or positive (if additional units are purchased).

Exercise 8*

Consider the following linear programming problem:

$$\begin{aligned}
\text{maximize } Z = 60x_1 + 100x_2 + 80x_3 - 5x_4 \\
\text{s.t. } 5x_1 + 8x_2 + 6x_3 \leq 600 \\
3x_1 + 4x_2 + 5x_3 \leq 400 \\
2x_1 + 3x_2 + x_3 = 200 + x_4 \\
x_2 = 2x_1 \\
x_1, x_2, x_3 \geq 0 \\
x_4 \in \mathbb{R}
\end{aligned}$$

Your tasks:

- Transform the problem in standard form
- Transform the problem in equality form
- Solve the problem numerically with `scipy.optimize.linprog`. Read this tutorial.

Solution:
The standard or the equality forms are already good for `scipy.optimize.linprog`. However, let's put the problem in yet another form that requires less manipulations from the initial given form.

$$\begin{aligned}
\text{minimize } -Z = -60x_1 - 100x_2 - 80x_3 + 5x_4 \\
\text{s.t. } 5x_1 + 8x_2 + 6x_3 + 0x_4 \leq 600 \\
3x_1 + 4x_2 + 5x_3 + 0x_4 \leq 400 \\
2x_1 + 3x_2 + x_3 - x_4 = 200 \\
2x_1 - x_2 + 0x_3 + 0x_4 = 0 \\
x_1, x_2, x_3 \geq 0 \\
x_4 \in \mathbb{R}
\end{aligned}$$

This corresponds to

$$c = \begin{bmatrix} -60 & -100 & -80 & 5 \end{bmatrix}^T$$

$$A_{ub} = \begin{bmatrix} 5 & 8 & 6 & 0 \\ 3 & 4 & 5 & 0 \end{bmatrix} \quad b_{ub} = \begin{bmatrix} 600 & 400 \end{bmatrix}^T$$

$$A_{eq} = \begin{bmatrix} 2 & 3 & 1 & -1 \\ 2 & -1 & 0 & 0 \end{bmatrix} \quad b_{eq} = \begin{bmatrix} 200 & 0 \end{bmatrix}^T$$

```
import numpy as np
from scipy.optimize import linprog
import array_to_latex as a2l

c = np.array([-60, -100, -80, 5])
A_ub = np.array([[5,8,6,0],
                 [3,4,5,0]])
b_ub = np.array([600,400])

A_eq = np.array([[2,3,1,-1],
                 [2,-1,0,0]])
b_eq = np.array([200, 0])

x1_bounds = (0, None)
x2_bounds = (0, None)
x3_bounds = (0, None) # +/- np.inf can be used instead of None
x4_bounds = (-np.inf, +np.inf)

bounds = [x1_bounds, x2_bounds, x3_bounds, x4_bounds]

result = linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bounds=bounds, options={"
    presolve": False})
print(result)

#%%
```

```
       message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
        success: True
         status: 0
            fun: -7846.153846153847
              x: [ 1.538e+01  3.077e+01  4.615e+01 -3.077e+01]
            nit: 4
          lower:  residual: [ 1.538e+01  3.077e+01  4.615e+01        inf]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00]
          upper:  residual: [       inf        inf        inf        inf]
                 marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00]
          eqlin:  residual: [ 0.000e+00  0.000e+00]
                 marginals: [-5.000e+00  2.436e+00]
        ineqlin:  residual: [ 0.000e+00  0.000e+00]
                 marginals: [-7.051e+00 -6.538e+00]
 mip_node_count: 0
 mip_dual_bound: 0.0
        mip_gap: 0.0
```

Elements can be accesssed individually. For example the solution can be accessed by:

```
print(result.x) # solution
print(result.fun) # objective function value
```

The residuals are the values of the slack variables and tell us whether the corresponding constraint is active or inactive in the optimal solution.
The marginals (or dual values, shadow prices, Lagrange multipliers) are the values of the dual variables that can be obtained as a by product from the simplex. They are called marginals because they tell us how much the objective function value would change if we increased by 1 the right hand side of the constraint. For example, the marginal associated with the second inequality constraint is -6.538, hence we expect the optimal value of the objective function to decrease by ϵ if we add a small amount ϵ to the right hand side of the second inequality constraint. Indeed:

```
print(result.x) # solution
print(result.fun) # objective function value
```

```
-7852.692307692308
```