HAB 619 Introduction to Scientific Computing in Sports Science

#12

SERDAR ARITAN

serdar.aritan@hacettepe.edu.tr

Biyomekanik Araştırma Grubu
www.biomech.hacettepe.edu.tr
Spor Bilimleri Fakültesi
www.sbt.hacettepe.edu.tr
Hacettepe Universitesi, Ankara, Türkiye
www.hacettepe.edu.tr

De Motu Animalium G.Borelli (1680)

Linear Programming

Optimization - finding value of a parameter that maximizes or minimizes a function with that parameter

- Talking about mathematical optimization, not optimization of computer code!
- "function" is mathematical function, not python language `def`
- Can have multiple parameters
- Can have multiple functions
- Parameters can appear linearly or nonlinearly

Linear Programming

Linear programming

- Most often used kind of optimization

- Tremendous number of practical applications
- "Programming" means determining feasible programs (plans, schedules, allocations) that are optimal with respect to a certain criterion and that obey certain constraints
- A feasible program is a solution to a linear programming problem and that satisfies certain constraints
- In linear programming
- Constraints are linear inequalities
- Criterion is a linear expression
- Expression called the objective function
- In practice, objective function is often the cost of or profit from some activity

Linear Programming

Diet Problem

You are given a group of foods, their nutritional values and costs. You know how much nutrition a person needs. What combination of foods can you serve that meets the nutritional needs of a person but costs the least?

Linear Programming

Mathematical formulation

The variables $x_1, x_2, \ldots x_n$ satisfy the inequalities

$$\begin{gathered} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \le b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \le b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \le b_m \end{gathered}$$

and $x_1 \geq 0, x_2 \geq 0, \ldots x_n \geq 0$. Find the set of values of $x_1, x_2, \ldots x_n$ that minimizes (maximizes)

$$x_1 f_1 + x_2 f_2 + \cdots x_n f_n$$

Note that a_{pq} and f_i are known

Linear Programming

Mathematical matrix formulation

Find the value of *x* that minimizes (maximizes) $f^T x$ given that $x \geq 0$ and $Ax \leq b$, where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{21} & \cdots & a_{21} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \text{and } f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

Linear Programming

General procedure

- Restate problem in terms of equations and inequalities
- Rewrite in matrix and vector notation
- Call function `linprog` from `scipy.optimize` to solve

Linear Programming

NumPy

Example – diet problem

Nowadays kids diet comes from the four basic food groups - chocolate dessert, ice cream, soda, and cheesecake. They checks in a store and finds one of each kind of food, namely, a brownie, chocolate ice cream, Pepsi, and one slice of pineapple cheesecake. Each day they needs at least 500 calories, 6 gr of chocolate, 10 gr of sugar, and 8 gr of fat. Using the table on the next slide that gives the cost and nutrition of each item, figure out how much he should buy and eat of each of the four items he found in the store so that he gets enough nutrition but spends as little as possible.

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

What are unknowns?

x_1 = number of brownies to eat each day

x_2 = number of scoops of chocolate ice cream to eat each day

x_3 = number of bottles of Coke to drink each day

x_4 = number of pineapple cheesecake slices to eat each day

In linear programming "unknowns" are called decision variables

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

Objective is to minimize cost of food. Total daily cost is

Cost = (Cost of brownies) + (Cost of ice cream) + (Cost of Coke) + (Cost of cheesecake)

- Cost of brownies = (Cost/brownie) × (brownies/day) = 2.5 x_1
- Cost of ice cream = x_2
- Cost of Coke = $1.5x_3$
- Cost of cheesecake = $4x_4$

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

Therefore, need to minimize

$$\min(\; 2.5x_1 + x_2 + 1.5x_3 + 4x_4 \;)$$

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

Constraint 1 - calorie intake at least 500

- Calories from brownies = (calories/brownie)(brownies/day) = $400x_1$
- Calories from ice cream = $200x_2$
- Calories from Coke = $150x_3$
- Calories from cheesecake = $500x_4$

So constraint 1 is $400x_1 + 200x_2 + 150x_3 + 500x_4 \geq 500$

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

Constraint 2 - chocolate intake at least 6 gr

- Chocolate from brownies = (Chocolate/brownie)(brownies/day) = $3x_1$
- Chocolate from ice cream = $2x_2$
- Chocolate from Coke = $0x_3 = 0$
- Chocolate from cheesecake = $0x_4 = 0$

So constraint 2 is $3x_1 + 2x_2 \geq 6$

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

Constraint 3 - sugar intake at least 10 gr

- Sugar from brownies = (sugar/brownie)(brownies/day) = $2x_1$
- Sugar from ice cream = $2x_2$
- Sugar from Coke = $4x_3$
- Sugar from cheesecake = $4x_4$

So constraint 3 is $2x_1 + 2x_2 + 4x_3 + 4x_4 \geq 10$

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

Constraint 4 - fat intake at least 8 gr

- Fat from brownies = (fat/brownie)(brownies/day) = $2x_1$
- Fat from ice cream = $4x_2$
- Fat from Coke = $1x_3$
- Fat from cheesecake = $5x_4$

So constraint 4 is $2x_1 + 4x_2 + x_3 + 5x_4 \geq 8$

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

Finally, we assume that the amounts eaten are non-negative, i.e., we ignore throwing up. This means that we have

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ and\ x_4 \geq 0$$

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

Putting it all together, we have to minimize $2.5x_1 + x_2 + 1.5x_3 + 4x_4$

subject to the constraints

$$\begin{aligned} 400x_1 + 200x_2 + 150x_3 + 500x_4 &\geq 500 \\ 3x_1 + 2x_2 &\geq 6 \\ 2x_1 + 2x_2 + 4x_3 + 4x_4 &\geq 10 \\ 2x_1 + 4x_2 + x_3 + 5x_4 &\geq 8 \end{aligned}$$

And

$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$x_3 \geq 0$$
$$x_4 \geq 0$$

Linear Programming

NumPy

Example – diet problem

Food	Calories	Chocolate	Sugar	Fat	Cost (serving)
Brownie	400	3	2	2	$2.50 / brownie
Chocolate ice cream	200	2	2	4	$1.00 / scoop
Coke	150	0	4	1	$1.50 / bottle
Pineapple cheesecake	500	0	4	5	$4.00 / slice

In matrix notation, want to

$$\text{minimize } f^T x \text{ subject to } Ax \geq b \text{ and } x \geq 0$$

where

$$A = \begin{bmatrix} 400 & 200 & 150 & 500 \\ 3 & 2 & 0 & 0 \\ 2 & 2 & 4 & 4 \\ 2 & 4 & 1 & 5 \end{bmatrix}, \quad b = \begin{bmatrix} 500 \\ 6 \\ 10 \\ 8 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \quad \text{and } f = \begin{bmatrix} 2.5 \\ 1 \\ 1.5 \\ 4 \end{bmatrix}$$

Linear Programming

NumPy

`scipy.optimize.linprog` solves linear programming

$$\text{minimize } f^T x \text{ such that } \begin{cases} A \cdot x \le b \\ Aeq \cdot x = beq \\ lb \le x \le ub \end{cases}$$

where *x, b, beq, lb*, and *ub* are vectors and *A* and *Aeq* are matrices.

- Can use one or more of the constraints
- "*lb*" means "lower bound", "*ub*" means "upper bound"

  Often have lb = 0 and ub = ∞, i.e., no upper bound

Linear Programming

`scipy.optimize` programming solver is `linprog()`

```
scipy.optimize.linprog(f, A_ub=None, b_ub=None, A_eq=None, b_eq=None, bounds=None,
method='interior-point', callback=None, options=None, x0=None)
```

Linear programming: minimize a linear objective function subject to linear equality and inequality constraints.

f 1-D array

The coefficients of the linear objective function to be minimized.

A_ub 2-D array, optional

The inequality constraint matrix. Each row of A_ub specifies the coefficients of a linear inequality constraint on x.

b_ub 1-D array, optional

The inequality constraint vector. Each element represents an upper bound on the corresponding value of

Linear Programming

Us:

$$\text{minimize } f^T x \text{ subject to } Ax \geq b \text{ and } 0 \leq x$$

$$A = \begin{bmatrix} 400 & 200 & 150 & 500 \\ 3 & 2 & 0 & 0 \\ 2 & 2 & 4 & 4 \\ 2 & 4 & 1 & 5 \end{bmatrix}, \quad b = \begin{bmatrix} 500 \\ 6 \\ 10 \\ 8 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \quad \text{and } f = \begin{bmatrix} 2.5 \\ 1 \\ 1.5 \\ 4 \end{bmatrix}$$

NumPy:

$$\text{minimize } f^T x \text{ such that } \begin{cases} A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub \end{cases}$$

Note two differences:

Linear Programming

NumPy

linprog() solves linear programming problem

ISSUE 1 - We have $Ax \geq b$ but need $Ax \leq b$

One way to handle is to note that

if $Ax \geq b$ then $-Ax \leq -b$, so can have **`numpy.negative`** use constraint

$(-A)x \leq (-b)$

ISSUE 2 - We have $0 \leq x$ but linprog() wants $lb \leq x \leq ub$. Handle by omitting *ub* in call of **`linprog()`**. If omitted, linprog() assumes no upper bound.

Linear Programming

NumPy

```
import numpy as np
from scipy.optimize import linprog
# objective function
f = np.array([2.5, 1.0, 1.5, 4.0])
# constraint matrix
A = np.array([[400, 200, 150, 500],
              [3.0, 2.0, 0.0, 0.0],
              [2.0, 2.0, 4.0, 4.0],
              [2.0, 4.0, 1.0, 5.0]])
# inequality constraint vector
b = np.array([500.0, 6.0, 10.0, 8.0])

x1_bounds = (0, None)
x2_bounds = (0, None)
x3_bounds = (0, None)
x4_bounds = (0, None)

result = linprog(f, A_ub=np.negative(A), b_ub=np.negative(b), \
         bounds=[x1_bounds, x2_bounds, x3_bounds, x4_bounds])
```

Linear Programming

```
>>> print(result)
 con: array([], dtype=float64)
     fun: 4.5000000466017935
 message: 'Optimization terminated successfully.'
     nit: 7
   slack: array([2.50000005e+02, 4.03549727e-09, 4.82628302e-08,
4.99999998e+00])
  status: 0
 success: True
>>> x = np.array(result.x)
>>> print(x)
[1.42673815e-08 2.99999998e+00 1.00000001e+00 3.35245816e-09]
```

Linear Programming

NumPy

Optimal solution is `x = [ 0 3 1 0 ]` . In words, kids should eat 3 scoops of ice cream and drink 1 Coke each day.

A constraint is binding if both sides of the constraint inequality are equal when the optimal solution is substituted.

For `x = [ 0 3 1 0 ]` the set

$$400x_1 + 200x_2 + 150x_3 + 500x_4 \geq 500$$
$$3x_1 + 2x_2 \geq 6$$
$$2x_1 + 2x_2 + 4x_3 + 4x_4 \geq 10$$
$$2x_1 + 4x_2 + x_3 + 5x_4 \geq 8$$

Becomes

$$750 \geq 500$$
$$6 \geq 6$$
$$10 \geq 10$$
$$13 \geq 8$$

,

so the chocolate and sugar constraints are binding. The other two are nonbinding

Linear Programming

How many calories, and how much chocolate, sugar and fat will he get each day?

```
>> print(np.negative(A) @ x)
[-750.0000052    -6.           -10.00000005  -12.99999998]
```

750.0 calories
6.0 chocolate
10.0 sugar
13.0 fat

How much money will this cost?

```
>>> print(f @ x)
4.500000466017935 # dollars
```

Linear Programming

NumPy

Special Kinds of Solutions

Usually a linear programming problem has a unique (single) optimal solution. However, there can also be:

- No feasible solutions
- An unbounded solution. There are solutions that make the objective function arbitrarily large (max problem) or arbitrarily small (min problem)
- An infinite number of optimal solutions. The technique of goal programming is often used to choose among alternative optimal solutions.

Linear Programming

A scipy.optimize.OptimizeResult consisting of the fields:

x 1-D array
 The values of the decision variables that minimizes the objective function while satisfying the constraints.
fun float
 The optimal value of the objective function c @ x.
slack 1-D array
 The (nominally positive) values of the slack variables, b_ub - A_ub @ x.
con 1-D array
 The (nominally zero) residuals of the equality constraints, b_eq - A_eq @ x.
successbool
 True when the algorithm succeeds in finding an optimal solution.
statusint
 An integer representing the exit status of the algorithm.
 0 : Optimization terminated successfully.
 1 : Iteration limit reached.
 2 : Problem appears to be infeasible.
 3 : Problem appears to be unbounded.
 4 : Numerical difficulties encountered.
nitint
 The total number of iterations performed in all phases.
messagestr
 A string descriptor of the exit status of the algorithm.