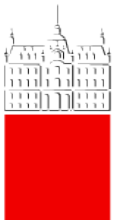
Univerza *v Ljubljani*
Fakulteta za *matematiko in fiziko*



Linearno programiranje

2. naloga pri Modelski analizi 1

Avtor: Marko Urbanč (28232019)
Predavatelj: prof. dr. Simon Širca
Asistent: doc. dr. Miha Mihovilovič

17.10.2023

Kazalo

1 Uvod

Linearna optimizacija je zadnje čase zelo vroča tema, tako da me veseli, da sem se je lahko lotil tudi sam. Linearno programiranje oz. linearna optimizacija je metoda za iskanje maksimuma ali minimuma linearnega izraza, ki je podvržen linearnim omejitvam. Omenjen linearni izraz je funkcija več spremenljivk, ki najpogosteje ovrednoti oz. oceni neko količino, ki jo želimo optimizirati. To je lahko npr. dobiček, strošek, količina proizvodnje, itd. zato to funkcijo imenujemo `cost function` oz. `objective function`. Linearno programiranje je torej metoda, ki nam omogoča, da z linearno funkcijo in linearnimi omejitvami poiščemo optimalno rešitev. Uporablja v različnih panogah, kot so ekonomija, logistika, telekomunikacije, transport, kot pa tudi v znanosti, kot je npr. fizika.

Torej če si to pogledamo v matematični notaciji imamo našo cost funkcijo $f(x_1, x_2, \ldots, x_n)$, definirano kot neko linearno kombinacijo spremenljivk:

$$f(x_1, x_2, \ldots, x_n) = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \, . \tag{1}$$

in set vezi, ki so pogosto izražene kot neenačbe:

$$\begin{aligned} a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n &\leq b_1 \, , \\ a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n &\leq b_2 \, , \\ &\vdots \\ a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n &\leq b_m \, . \end{aligned} \tag{2}$$

To je pravzaprav to kar se tiče matematičnega opisa osnovnega problema. Seveda so potem izvedenke postopkov oz. algoritmov malo bolj zapletene, ampak to žal ni namen te naloge.

Dotično pri tej nalogi si bomo pogledali problem optimizacije diete. Posebna omemba ni potrebna, kako pomembno je to, da se prehranjujemo zdravo in uravnoteženo. Vendar pa je to v današnjem času vse težje, saj je na voljo ogromno različnih živil, ki so vse prej kot zdrava. Zato je toliko bolj pomembno, da se zavedamo, kaj jemo in da se prehranjujemo zdravo. Vendar pa je to včasih težko, saj je veliko ljudi prezaposlenih in nimajo časa, da bi se ukvarjali s tem. Skratka ideja je taka, da bomo za dane želje kar se tiče hranilnih snovi, poskusili najti optimalno dieto. Optimalno pa je tu mišljeno v tem smislu, kot ga uporabnik določi. Zagotovo bo razlika med tem, ali se minimizira količina hrane, ki jo je potrebno pojesti, cena hrane, količina določene hranilne snovi ali pa kalorična vrednost.

2 Naloga

Naloga torej zahteva, da za dano tabelo živil in njihovih hranilnih vrednosti, najdemo optimalno dieto, kjer minimiziramo kalorije, če zahtevamo sledeče:

1.
   - Vsaj 70 g maščob
   - Vsaj 310 g ogljikovih hidratov

- Vsaj 50 g proteinov
- Vsaj 1000 mg kalcija
- Vsaj 18 mg železa
- Dnevni obroki naj ne presežejo 2000 g

Ta osnovni model lahko potem nadgradimo s tem da dodamo še:

- Vsaj 60 mg Vitamina C
- Vsaj 3500 mg Kalija
- Med 500 mg in 2400 mg Natrija

2. Potem pa poglejmo kako se rezultat razlikuje še zahtevamo vsaj 2000 kcal in minimiziramo vnos maščob.
3. Minimizirajmo zdaj ceno diete
4. Dodatne izvedbe omejitev za izboljšave uravnoteženosti diete

3 Opis reševanja

Reševanja sem se lotil v klasično v Pythonu. Pri tej nalogi je močno prišel v upoštev pythonian princip, da za vse že obstaja neka knjižnica. In res je tako, za reševanje linearne optimizacije obstaja knjižnica `PuLP`, ki je zelo enostavna za uporabo in ima še kar solidno dokumentacijo. Pravzaprav je njihov prvi demonstracijski zgled zelo podoben našemu, z razliko da oni mešajo meso v mačji hrani. Poleg tega pa seveda pridejo zraven še standardni `numpy`, `pandas` in tokrat zaradi malo alterative izbire vizualizacije še `holoviews`.

3.1 `PuLP`: basic cookbook

Kot sem že omenil, je knjižnica `PuLP` zelo enostavna za uporabo. Lahko na hitro povzamem postopek ker tako ali tako trenutno čakam, da se mi predprocesirajo podatki za (hopefully uspe) zadnji del naloge. Torej začnemo s tem da uvozimo knjižnico in definiramo problem.

```
from pulp import *
import pandas as pd

# Create the "prob"  variable to contain the problem data
prob = LpProblem("Diet Problem", LpMinimize)
```

Sledi nalaganje podatkov iz tabele živil in ustvarjanje spremenljivk. Spremenljivke so v tem primeru količina živila, ki ga bomo pojedli.

```
df = pd.read_table('Data/table.dat', sep=',', skiprows=2, index_col=0)

# Create a list of the food items
food_items = list(df.index)

# Create variables. These variables are the amounts of each food item to
↪  buy
food_vars = LpVariable.dicts("Food", food_items, lowBound=0,
↪  cat='Continuous')
```

Okay sedaj pa definiramo še našo cost funkcijo. V tem osnovnem primeru bo to količina kalorij, ki jih bomo zaužili.

```
# Define objective function and add it to the problem
prob += lpSum([df.loc[i, 'Energija[kcal]'] * food_vars[i] for i in
↪  food_items]), "Total energy intake per person"
```

Zelo praktično je, da v bistvu kar prištevamo izraze k našemu problemu. Tako lahko zelo enostavno dodajamo vezi. Storimo to.

```
# And now we can add the constraints
prob += lpSum([df.loc[i, 'Mascobe[g]'] * food_vars[i] for i in
↪  food_items]) >= 70, "FatRequirement"
prob += lpSum([df.loc[i, 'Ogljikovi_Hidrati[g]'] * food_vars[i] for i in
↪  food_items]) >= 310, "CarbohydrateRequirement"
prob += lpSum([df.loc[i, 'Proteini[g]'] * food_vars[i] for i in
↪  food_items]) >= 50, "ProteinRequirement"
prob += lpSum([df.loc[i, 'Ca[mg]'] * food_vars[i] for i in food_items])
↪  >= 1000, "CalciumRequirement"
prob += lpSum([df.loc[i, 'Fe[mg]'] * food_vars[i] for i in food_items])
↪  >= 18, "IronRequirement"
prob += lpSum([100 * food_vars[i] for i in food_items]) <= 2000,
↪  "MassLimit"
```

Zdaj ko smo model tako lepo definirali je smiselno, da ga tudi spravimo v datoteko. Nato pa kličemo reševanje. `PuLP` bo sam poskrbel za izbiro algoritma, ki bo najbolj primeren za naš problem.

```
model_name = "diet-model_no-weight-con.lp"
prob.writeLP(f"Models/{model_name}")

# Slove the problem
prob.solve()
```

In na koncu še spravimo rešitev v neko obliko, ki jo lahko potem uporabimo za vizualizacijo.

```
# Save the solution to a file with numpy
var_names = np.array([v.name for v in prob.variables()])
var_values = np.array([v.varValue for v in prob.variables()])
solution = np.column_stack((var_names, var_values))

np.savetxt(f"Solutions/{model_name}-sol.dat", solution, delimiter=",",
↪   fmt="\%s", header="Item,Value")
```

Če bi nas zanimalo še takojšnje ovrednotenje lahko z spodnjim blokom kode izpišemo nekaj uporabnih informacij.

```
# Print the status of the solution
print("Status:", LpStatus[prob.status])

# Print the optimal solution
for v in prob.variables():
    print(v.name, "=", v.varValue)
print("Total energy intake per person = ", value(prob.objective))
```

Od tod naprej pa lahko poljubno zapletemo vezi (dokler ostanejo linearne seveda) ali pa spremenimo vhodne podatke.

4 Rezultati

5 Komentarji in izboljšave

Literatura