Razvoj koda u programskom jeziku Python za izračun kvalitete plina uplinjenog na UPP terminalu Omišalj

Tisaj, Josipa

SVEUČILIŠTE U ZAGREBU

RUDARSKO-GEOLOŠKO-NAFTNI FAKULTET

Preddiplomski studij naftnog rudarstva

RAZVOJ KODA U PROGRAMSKOM JEZIKU PYTHON ZA IZRAČUN KVALITETE PLINA UPLINJENOG NA UPP TERMINALU OMIŠALJ

Završni rad

Josipa Tisaj

N4505

Zagreb, 2022.

Sveučilište u Zagrebu
Rudarsko-geološko-naftni fakultet

Završni rad

RAZVOJ KODA U PROGRAMSKOM JEZIKU PYTHON ZA IZRAČUN KVALITETE PLINA UPLINJENOG NA UPP TERMINALU OMIŠALJ

JOSIPA TISAJ

Završni rad izrađen: Sveučilište u Zagrebu
Rudarsko-geološko-naftni fakultet
Zavod za naftno-plinsko inženjerstvo i energetiku
Pierottijeva 6, 10000 Zagreb

Sažetak

U ovome radu opisan je postupak izračuna metanskoga broja u skladu s normom *HRN EN 16726 - Plinska infrastruktura - Kvaliteta plina - Grupa H.* Opisani je postupak zatim proveden u programskom jeziku Python s ciljem izračuna metanskoga broja za nekoliko različitih sastava UPP-a koji su uplinjeni na terminalu u općini Omišalj na otoku Krku. Rezultati dobiveni u Pythonu uspoređeni su izračunatim metanskim brojevima u Excelu.. U konačnici, donesen je zaključak o funkcionalnosti i efikasnosti ponuđenog koda za izračun metanskoga broja.

Ključne riječi: metanski broj, Python, sastav prirodnog plina, optimizacija, Excel

Završni rad sadrži: 25 stranica, 22 slika, 6 tablica i 10 referenci

Završni rad pohranjen: Knjižnica Rudarsko-geološko-naftnog fakulteta
Pierottijeva 6, 10000 Zagreb

Voditelji: Dr. sc. Daria Karasalihović Sedlar, redovita profesorica RGNF-a
Dr. sc. Domagoj Vulin, redoviti profesor RGNF-a

Pomoć pri izradi: Ivan Smajla, mag. ing. petrol.

Ocjenjivači: Dr. sc. Daria Karasalihović Sedlar, redovita profesorica RGNF-a
Dr. sc. Domagoj Vulin, redoviti profesor RGNF-a
Dr. sc. Luka Perković, izvanredni profesor RGNF-a
Dr. sc. Borivoje Pašić, izvanredni profesor RGNF-a
Dr. sc. Karolina Novak Mavar, docentica RGNF-a

Datum obrane: 9.9.2022., Rudarsko-geološko-naftni fakultet, Sveučilište u Zagrebu

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. UVOD

Na teritoriju Republike Hrvatske, u općini Omišalj na otoku Krku, smješten je terminal za ukapljeni prirodni plin koji je iznimno važan za osnaživanje europskog energetskog tržišta te doprinosi sigurnosti opskrbe prirodnim plinom zemalja Europske unije (EU). Terminalu su dodijeljena značajna bespovratna sredstva od 101,4 milijuna eura te spada u EU projekte od zajedničkog interesa.

Terminal se sastoji od kopnenog dijela i FSRU (engl. *Floating storage regasification unit*) broda koji je opremljen skladišnim spremnicima za UPP (ukapljeni prirodni plin) te jedinicama za uplinjavanje. Proces uplinjavanja sastoji se od izmjenjivanja topline morske vode i UPP-a preko glikola. Na taj način obrađeni prirodni plin šalje se u plinski transportni sustav Republike Hrvatske.

Hrvatska energetska regulatorna agencija (HERA) propisuje *Općim uvjetima opskrbe plinom (NN 100/2021)* da plin koji ulazi u transportni sustav mora imati metanski broj minimalno 75. Metanski broj izračunava se metodom u skladu sa standardom *HRN EN 16726 - Plinska infrastruktura - Kvaliteta plina - Grupa H.* Ukoliko plin ne zadovoljava prethodno spomenute kriterije, on nije podoban za ulazak u transportni sustav RH.

S obzirom na važnost metanskog broja kao parametra u procjenjivanju kvalitete plina, u ovome radu objašnjenja je procedura izračuna metanskog broja u skladu s prethodno spomenutim standardom u Pythonu. Za potrebe izračuna metanskog broja, upotrjebljeni su javno dostupni podaci o karakteristikama plinova objavljeni na internet stranicama od *Plinacro d.o.o.*

U radu se istražuje je li moguće postupak izračuna metanskog broja prevesti u Pythonu na način koji će omogućiti nastanak prihvatljivih rješenja, ali i eliminirati potrebu za korištenjem Excela. Naime, sam standard predlaže upotrebu Excela u obavljanju proračuna, no Excel je poprilično ograničen u smislu onoga što može samostalno obaviti te često zahtjeva pomoć od strane korisnika kako bi došao do rješenja.

2. METODOLOGIJA

2.1. Postupak izračuna metanskoga broja u skladu s normom *HRN EN 16726 – Plinska infrastruktura – Kvaliteta plina – Grupa H*

Metanski broj prirodnog plina moguće je izračunati korištenjem nekoliko različitih metoda. Većina članova EUROMOT-a (engl. *European Association of Internal Combustion Engine Manufacturers*) koristi *AVL* metodu iz 1970-ih godina koja se pokazala pouzdanom i dovoljno preciznom za kvalitetu plina današnjice, pod uvjetom da se vodik isključi iz sastava plina. Osim *AVL* metode, koristi se i *EON-GasCalculation* metoda. Oboje metode rezultiraju sličnim vrijednostima metanskoga broja, no *EON-GasCalculation* metoda pruža mogućnost jednostavnije korekcije za prisutnost vodika. Bitno je spomenuti kako, iako ove metode funkcioniraju za kvalitetu prirodnog plina koju imamo danas, u budućnosti će biti potrebno korigirati vrijednosti metanskih brojeva dobivene spomenutim metodama zbog pojave plinova koji sadrže ugljikovodike veće molekulske mase (u što spada i većina UPP-a trenutno na tržištu s manje od 95% metana).

Za potrebe ovoga rada, korištena je metoda opisana u H*RN EN 16726 - Plinska infrastruktura - Kvaliteta plina - Grupa H,* a se temelji na *AVL* metodi. Metoda zahtjeva ulazne parametre u obliku postotnih udjela pojedinih komponenata pri standardnim uvjetima od 101,325 kPa i 15 °C. Može se primijeniti na smjese plinova koje sadrže sljedeće komponente: ugljikov monoksid, butadien, butilen, etilen, propilen, vodikov sulfid, vodik, metan, etan, propan, butan, dušik i ugljikov dioksid. Metanski broj određene plinovite smjese računat je u šest koraka:

- Originalna plinovita smjesa pojednostavljena je na način da se iz smjese uklone sve negorive komponente poput ugljikovog dioksida i dušika, a sve ostale komponente molekulske mase veće od butana (uključujući i butan) svedu se na C4+ komponentu.
- Zatim se pristupa odabiru smjesa, tj. pojednostavljenih sustava koji sadrže određen postotak pojedinih, za taj tip smjese definiranih, komponenata. Postoji 18 tipova smjesa između kojih se biraju one koji sadrže tri komponente iz sastava, tj. odabiru se smjese koje najbolje predstavljaju sastav ukupne plinske smjese. Potrebno je još između tih

odabranih *pod-smjesa* izabrati samo one koje u konačnici dovode do zaključka da se između svih odabranih *pod-smjesa* svaka komponenta iz sastava pojavljuje barem dva puta. Naposljetku, pojednostavljeni sastav jednako se razdjeli u odabrane *pod-smjese* ovisno o tome sadrži li ili ne pojedina *pod-smjesa*, određenu komponentu iz sastava.

- Računa se metanski broj svake *pod-smjese*. U ovome koraku, metanski brojevi se razlikuju.
- Traži se razlika između maksimalne i minimalne vrijednosti od svih izračunatih metanskih brojeva te se nastoji, mijenjanjem udjela u *pod-smjesama*, razlika svesti na minimum.
- Određuje se metanski broj svake odabrane *pod-smjese* s promijenjenim udjelima pojedinih komponenata. Izračunava se srednja vrijednost od svih dobivenih metanskih brojeva.
- Naposljetku, izračunava se metanski broj originalne plinovite smjese na način da se vrijednost iz prethodnog koraka korigira za prisutnost inertnih plinova.

3. KOD U PROGRAMSKOM JEZIKU PYTHON

Postupak određivanja metanskog broja jasno je definiran ovisno o odabranoj metodi. Provedba takvog postupka traje dugo i ne preporuča se . Brzo i efikasno rješavanje matematičkih problema (poput odabira tipa pod-smjese, minimizacije razlike rezultata u pojedinim *pod-smjesama*) moguće je upotrebom komercijalnog softvera, a moguće je i korištenjem računalnog koda. U stotinjak linija koda, izrađen je Python računalni program koji određuje metanski broj pojedine plinovite smjese. Njegova struktura, ali i alati za razumijevanje samoga koda, dati su u ovome dijelu rada zajedno sa svim svojim ograničenjima.

3.1. Unos originalnih podataka i njihova obrada

3.1.1. Unos podataka o sastavu plina u programski kod

U funkciji *ucitaj*, prvo je stvorena lista *l* koja sadrži imena svih komponenti u sastavu plina. Pomoću *for* petlje, iterira se kroz listu *l*, i za svaki element u listi pozivamo *built-in* funkciju *input* koja zahtjeva od korisnika unos postotnog udjela pojedine komponente u sastavu plina. Na svaki uneseni brojčani podatak, poziva se metoda *float* čime se osigurao numerički decimalni unos varijable. Pri svakom koraku petlje, uneseni decimalni broj sprema se u rječnik *d* pod ključem koji predstavlja element iz liste *l*.

```
1   def ucitaj():
2       l = ["C1", "C2", "C3", "I-C4", "N-C4", "I-C5", "N-C5","C6+", "N2", "CO2", "H2"]
3       global d
4       d = dict()
5       for element in l:
6           a = float(input(element+": "))
7           d[element] = a
```

Slika 3-1. Kod za unos podataka o sastavu plina i spremanje u rječnik

Rječnik *d* označen je kao globalna varijabla pomoću ključne riječi *global* što omogućava korištenje rječnika izvan same funkcije. Naime, svim varijablama i strukturama za spremanje podataka poput lista i rječnika, koje su definirane unutar same funkcije, ne može se pristupiti izvan funkcije. Drugim riječima, ako bismo htjeli koristiti rječnik *d* s unesenim brojčanim podatcima negdje izvan funkcije *ucitaj*, program bi javljao grešku jer je rječnik definiran samo unutar funkcije. S obzirom da će biti koristan za neke druge izračune kasnije u programu, dodijeljen mu je status globalne varijable i kao takvom omogućen mu je pristup u bilo kojem dijelu programa.

3.1.2. Privremeno zanemarivanje negorivih komponenti u sastavu plina

Originalna plinovita smjesa pojednostavljuje se na način da se privremeno zanemari prisutnost negorivih komponenti (dušik, ugljikov dioksid). U kodu je to vidljivo stvaranjem liste *l2* u koju se stavljaju samo gorive komponente. Također, sve komponente veće molekulske mase od butana (uključujući i butan) svode se na C4+ komponentu. U tu svrhu, spomenute komponente potrebno je zamijeniti količinom butana koja je ekvivalentna količini svih komponenta veće molekulske mase, a koje su prisutne u sastavu plina. Dakle, postotni udio pentana množi se s 2,3, a postotni udjeli svih ugljikovodika većih od pentana množe se s 5,3.

```
9     d["C4+"] = d["I-C4"] + d["N-C4"] + 2.3 * (d["I-C5"] + d["N-C5"]) + d["C6+"] * 5.3
10    l2 = [d["C1"], d["C2"], d["C3"], d["C4+"]]
11    zbroj = sum(l2)
12    for i in range(len(l2)):
13        l2[i] = (l2[i] / zbroj) * 100
14    return l2
```

Slika 3-2. Pojednostavljenje originalne plinovite smjese i normalizacija

3.1.3. Normalizacija pojednostavljenog sastava prirodnog plina

Pojednostavljeni sastav plina sprema se u listu *l2*. S obzirom da zbroj elemenata iz liste nije 100, potrebno je pojednostavljeni sastav normalizirati. Normalizacija je u kodu provedena upotrebom *for* petlje koja svaki postotni udio iz *l2* dijeli sa sumom svih elemenata u *l2* i množi sa 100. Sada je zbroj u *l2* jednak 100 %. Funkcija vraća listu *l2* koja predstavlja temelj za sve daljnje proračune.

Slika 3-3. Lista *l2* s normaliziranim pojednostavljenim sastavom plina

```
def ucitaj():
    l = ["C1", "C2", "C3", "I-C4", "N-C4", "I-C5", "N-C5","C6+", "N2", "CO2", "H2"]
    global d
    d = dict()
    for element in l:
        a = float(input(element+": "))
        d[element] = a

    d["C4+"] = d["I-C4"] + d["N-C4"] + 2.3 * (d["I-C5"] + d["N-C5"]) + d["C6+"] * 5.3
    l2 = [d["C1"], d["C2"], d["C3"], d["C4+"]]
    zbroj = sum(l2)
    for i in range(len(l2)):
        l2[i] = (l2[i] / zbroj) * 100
    return l2
```

Slika 3-4. Konačni prikaz funkcije *ucitaj*

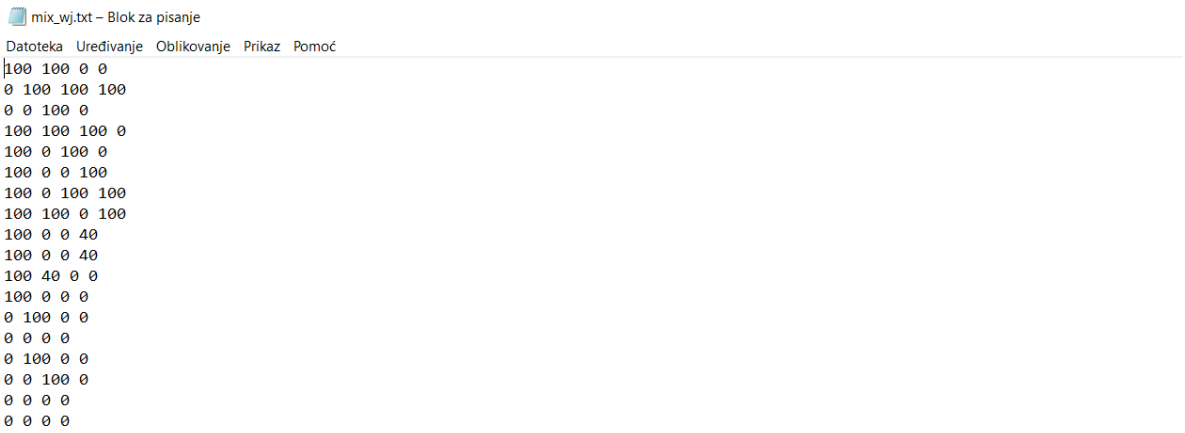
3.2. Odabir *pod-smjesa*

Sljedeći korak u izračunu metanskog broja odnosi se na odabir *pod-smjesa*. Na Tablici 3-1. prikazano je osamnaest *pod-smjesa* koje sadrže pojedine ugljikovodične i neugljikovodične komponente. Broj 0 znači da određena *pod-smjesa* ne sadrži pripadajuću komponentu, a broj 100 ili 40 znači kako određena *pod-smjesa* može sadržavati pripadajuću komponentu do maksimalno 100 ili 40 %. Ove vrijednosti, zajedno sa zbrojem stupaca pri dnu tablice, koriste se kako bi se odredilo poklapanje početne ugljikovodične smjese i pojedinih *pod-smjesa (fitnes).*

Tablica 3-1. Osamnaest sustava na raspolaganju za izračun metanskoga broja (EN ISO 16726, Gas infrastructure - Quality of gas - Group H (2015))

	komponente			
pod-smjese	metan	etan	propan	butan
1	100	100	0	0
2	0	100	100	100
3	0	0	100	0
4	100	100	100	0
5	100	0	100	0
6	100	0	0	100
7	100	0	100	100
8	100	100	0	100
9	100	0	0	40
10	100	0	0	40
11	100	40	0	0
12	100	0	0	0
13	0	100	0	0
14	0	0	0	0
15	0	100	0	0
16	0	0	100	0
17	0	0	0	0
18	0	0	0	0
suma	1000	640	600	480

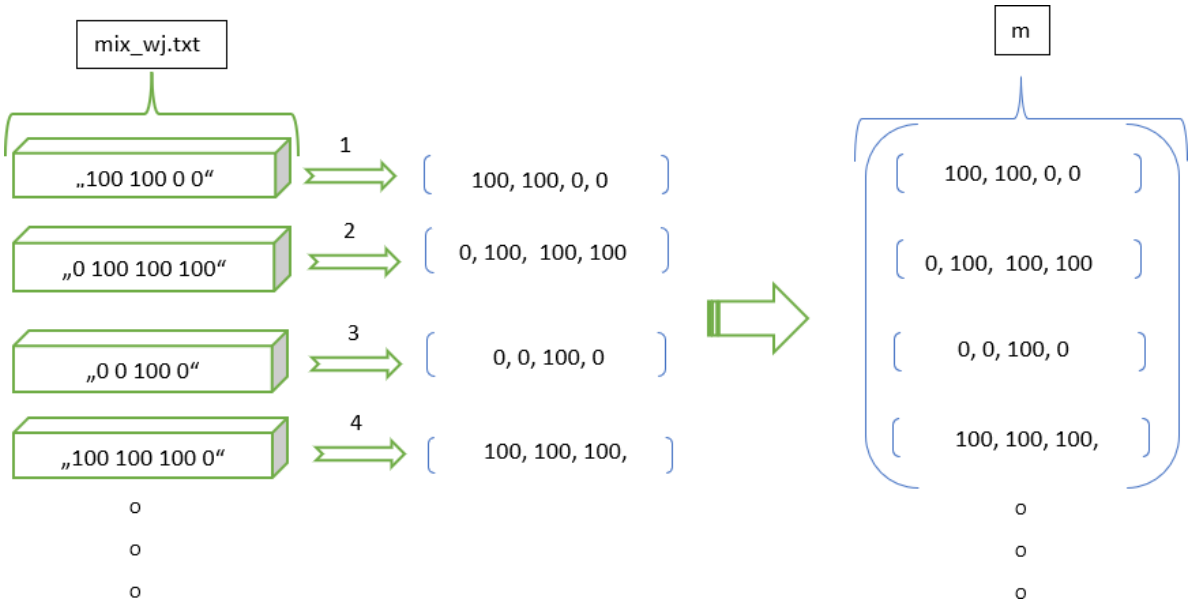
Kako bi se ovakav proračun mogao izvesti u Pythonu, potrebno je tablicu sa Tablice 3-1. prepisati i spremiti u datoteku. Iskorišteni su samo stupci koji se odnose na metan, etan, propan i butan.



Slika 3-5. Sadržaj datoteke mix_wj.txt potrebne za proračun *fitnesa*

3.2.1. Učitavanje podataka o pod-smjesama za četiri komponente iz sastava

U Pythonu, funkcija *racunaj_wj* uzima kao parametar normalizirani sastav plina dobiven pomoću funkcije *ucitaj*. Otvara datoteku mix_wj.txt, a zatim poziva metodu *readline()* koja čita prvi red u datoteci. Pomoću *while* petlje, prolazi kroz se svaki red u datoteci te se kreiraju manje liste koje se dodaju listi *m*. Po završetku *while* petlje, stvorena je lista *m* koja se sastoji od 18 manjih lista, a svaka manja lista predstavlja jedan red iz datoteke s 4 elementa.

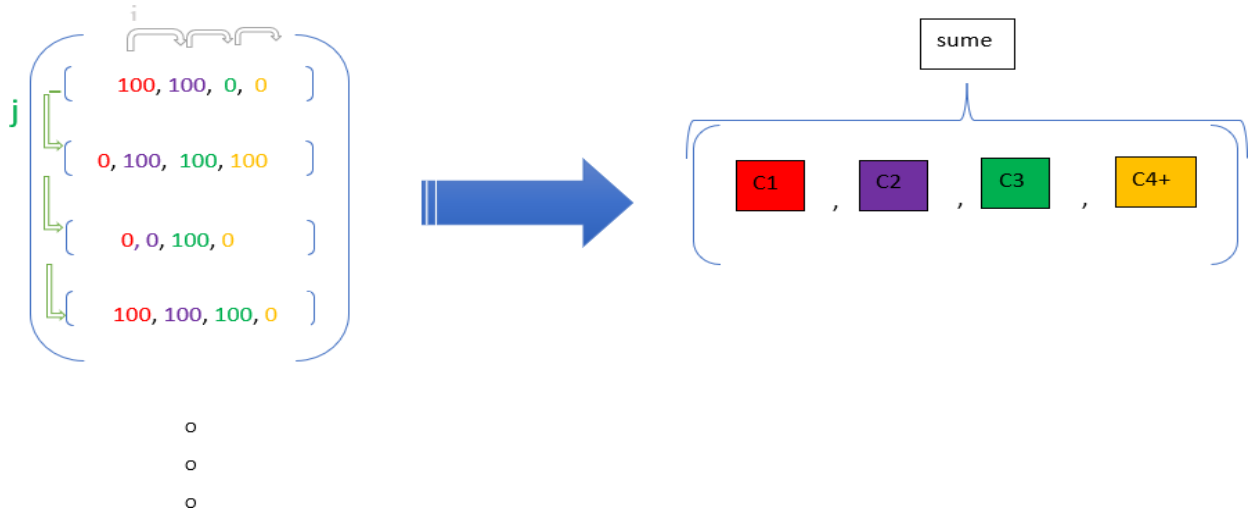


Slika 3-6. Vizualni prikaz nastanka liste *m*

```
32  def racunaj_wj(p):
33      dat = open("mix_wj.txt", "r")
34      m = []
35      red = dat.readline().strip()
36      while red != "":
37          m.append(list(map(float, red.split())))
38          red = dat.readline().strip()
39      dat.close()
```

Slika 3-7. Kod u Pythonu za kreiranje liste *m*

Nakon uspješnog kreiranja liste *m*, potrebno je zbrojiti stupce unutar iste liste. To se postiže dvjema *for* petljama od kojih je jedna smještena unutar druge. Vanjska petlja prolazi kroz elemente unutar manjih lista pomoću brojača *i*, dok se unutarnja kreće po elementima velike liste pomoću brojača *j*. Ovakav raspored kretanja unutarnje i vanjske petlje generira zbroj svakog od 4 stupca unutar velike liste. Zbrojevi se spremaju u listu *sume*.



Slika 3-8. Kreiranje liste *sume*

```
41      sume = []
42      for i in range(4):
43          z = 0
44          for j in range(len(m)):
45              z += m[j][i]
46          sume.append(z)
```

Slika 3-9. Kod u Pythonu za kreiranje liste *sume*

3.2.2. Izračun fitnesa i pod-smjese A2, A4, A7, A8

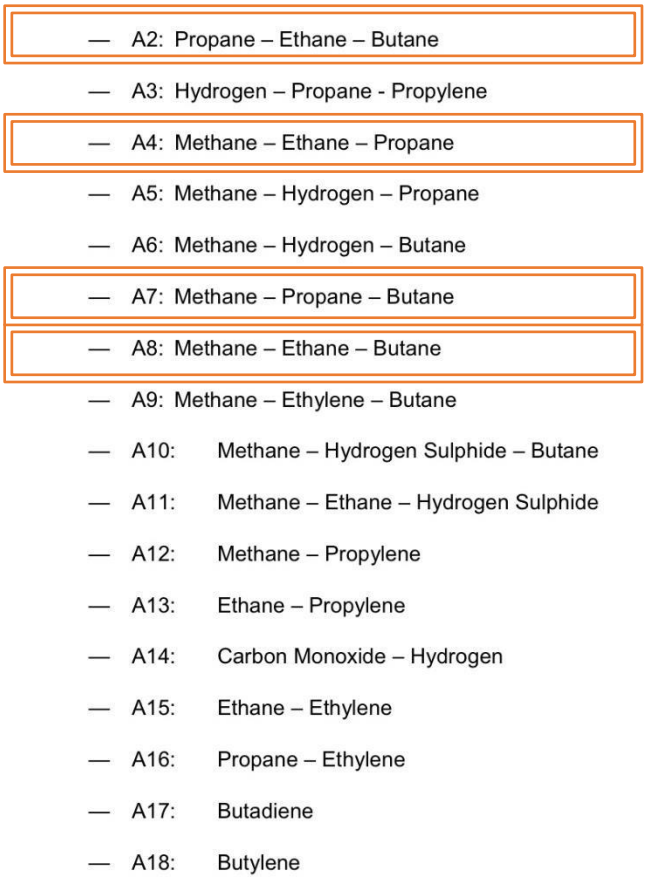
Konačno su stvoreni svi potrebni podatci za izračun *fitnesa*. Koristeći normalizirani sastav plina dobiven iz funkcije *ucitaj* te podatke iz liste *m* i liste *sume*, mijenjaju se podatci u listi *m* prema :

$$\frac{normalizirani\ sastav * m[i][j]}{sume[j]} \tag{3-1}$$

```
48      for i in range(len(m)):
49          for j in range(4):
50              m[i][j] = p[j] * m[i][j] / sume[j]
51      wj = []
52      for red in m:
53          wj.append(sum(red))
54
55      izdvoji = [m[1], m[3], m[6], m[7]]
56      izdvoji_wj = [wj[1], wj[3], wj[6], wj[7]]
```

Slika 3-10. Kod u Pythonu za izračun *fitnesa*

U funkciji *racunaj_wj*, lista s normaliziranim podatcima spremljena je u listu *p*. Pomoću unutarnje i vanjske *for* petlje mijenjamo svaki brojčani podatak na svakoj poziciji u listi *m* prema formuli (3-1). Zatim je kreirana nova lista *wj* u koju se sprema zbroj svakog elementa velike liste. U listu *izdvoji* dodani su podatci računati prema formuli (3-1) isključivo za *pod-smjese* A2, A4, A7 i A8. Ove četiri *pod-smjese* smatraju se kao najbolji kandidati za izračun metanskog broja jer u svim slučajevima najbolje predstavljaju sastav UPP-a, tj. svaki od njih sadrži tri komponente koje se može naći u svakom sastavu UPP-a. Također, u listu *izdvoji_wj* izdvojeni su podatci o *fitnesu* za četiri prethodno spomenute *pod-smjese* i proračun se nastavlja.



Slika 3-11. Odabrani sustavi koji sadrže tri komponente kao i u sastavu plina (EN ISO 16726, Gas infrastructure - Quality of gas - Group H (2015))

Nakon izračuna fitnesa, potrebno je između četiri odabrana sustava, izabrati one koji će osigurati da se svaka komponenta iz sastava plina pojavljuje barem dvaput. Prilikom odabira, ako postoji više sustava koji sadrže pojedinu komponentu, prednost ima onaj s najvećim *fitnesom*. U Pythonu, stvorena je dodatna funkcije *provjera* koja pamti koliko puta se pojedina komponenta pojavila u odabranim *pod-smjesama*. Funkcija vraća *True* ukoliko se svaka komponenta pojavljuje barem dva puta, a *False* ukoliko postoji jedna ili više komponenti koje se pojavljuju manje od dva puta.

```
16  def provjera(odabrani, izdvoji):
17      if len(izdvoji) == 0:
18          return False
19
20      brojac = [0, 0, 0, 0]
21      for x in odabrani:
22          for i in range(4):
23              if izdvoji[x][i] > 0:
24                  brojac[i] += 1
25      p = True
26      for x in brojac:
27          if x<2:
28              p = False
29      return p
```

Slika 3-12. Funkcija *provjera* u Pythonu

Funkcija *provjera* poziva se iz funkcije *racunaj_wj* kao uvjet u *while* petlji. Dakle, kada funkcija *provjera* zaključi kako se još uvijek sve komponente iz sastava ne pojavljuju dva puta, vraća na mjesto uvjeta u *while* petlji *False* što se zbog utjecaja ključne riječi *not* pretvara u *True*. Uslijed takvoga uvjeta, dolazi do definiranja varijable *koliko* koja pamti koliko *pod-smjesa* (od onih odabranih) sadrži tu komponentu koju trenutno gledamo, i pokretanja *for* petlje. Bitno je napomenuti kako je prije *while* petlje definirana varijabla *k* čija vrijednost može narasti do maksimalno broja 3, a odnosi se na metan (k=0), etan (k=1), propan (k=2) i butan (k=3). Također, definirana je i lista *odabrani* u koju će se spremati sve odabrane *pod-smjese* tijekom ovoga procesa.

Pri prvom koraku *while* petlje, lista *odabrani* je prazna pa se većina koda ni ne izvodi. No drugi po redu samostalni *if* uvjet ocjenjuje se kao *True* te dolazi do stvaranja lista *mogucnosti* i *mogucnosti_wj* u koje će se spremati indeksi i fitnesi svih *pod-smjesa* koje imaju priliku biti odabrane. Glavni zadatak *while* petlje u ovome dijelu koda sastoji se u traženju *pod-smjesa* koje sadrže pojedinu komponentu iz sustava, takve se prebacuju u listu *mogucnosti* te se od svih mogućih izabere samo ona s najvećim *fitnesom.* Zatim slijedi prijelaz na drugu komponentu po redu za koju se opet odrede *pod-smjese* koje ju sadrže, ali se uzme ona s najvećim fitnesom. *While* petlja se više ne izvodi kada funkcija *provjera* vrati vrijednost *True* što bi značilo da je nađen takav skup pod-smjesa koje omogućuju pojavljivanje svake komponente iz sastava plina barem dva puta.

```
k = 0
odabrani = []

while not provjera(odabrani, izdvoji):
    koliko = 0
    for x in odabrani:
        if izdvoji[x][k] > 0:
            koliko += 1
    if koliko > 0 and koliko == len(odabrani):
        k = (k + 1) % 4
        continue
    if koliko < 2:
        mogucnosti = []
        mogucnosti_wj = []
        for i in range(4):
            if i not in odabrani and izdvoji[i][k] > 0:
                mogucnosti.append(i)
                mogucnosti_wj.append(izdvoji_wj[i])
        odaberi = mogucnosti[mogucnosti_wj.index(max(mogucnosti_wj))]
        odabrani.append(odaberi)
    k = (k + 1) % 4
print(odabrani)
```

Slika 3-13. Kod u Pythonu za traženje najpogodnijih *pod-smjesa* kao dio funkcije *racunaj_wj*

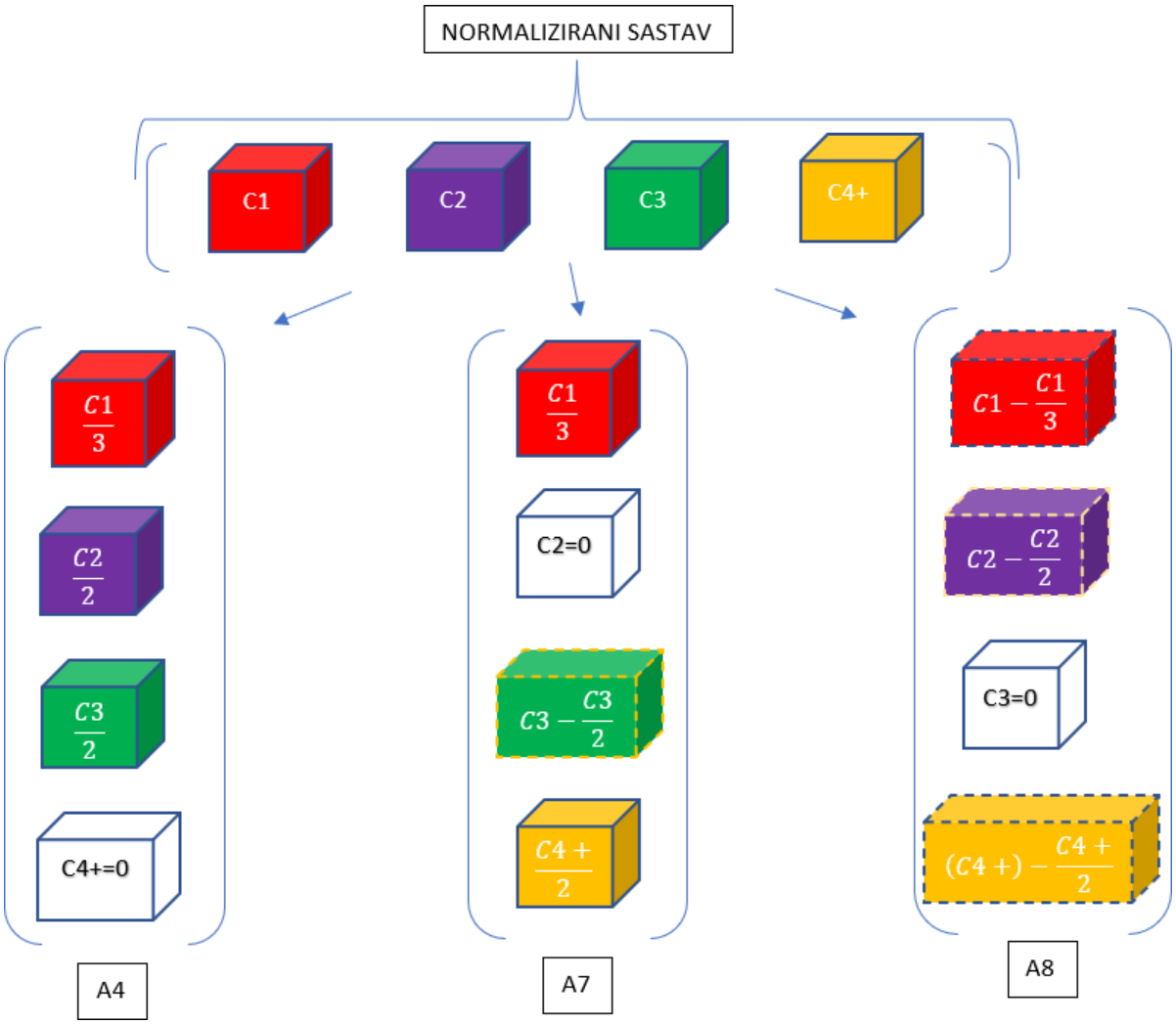
S obzirom na karakteristike sastava UPP-a, velika je vjerojatnost da će kod svakoga sastava odabrani sustavi biti A4, A7 i A8. Prema tome, u daljnjem računanju koriste se samo ova tri miksa. Bitno je napomenuti kako nakon provedbe procesa odabira najpogodnijih sustava, funkcija *racunaj_wj* prikazuje indekse *pod-smjesa* koje je odabrala. Na taj način možemo provjeriti jesu li odabrani sustavi A4, A7 i A8. Ukoliko nisu, potrebno je prilagoditi ostatak koda.

3.3. Raspodjela sastava u A4, A7, A8 i normalizacija

Prateći pretpostavku kako su odabrani sustavi zaista A4, A7 i A8, potrebno je normalizirani sastav dobiven funkcijom *ucitaj* jednako razdijeliti u sve tri *pod-smjese* ovisno o tome koju komponentu iz sastava sadrži pojedina *pod-smjesa*. S obzirom da se metan nalazi u sve tri *pod-smjese*, njegov normalizirani udio dijeli se s tri i raspodijeli na tri jednaka dijela. Etan je sadržan samo u A4 i A8 pa se njegov normalizirani udio dijeli s dva i raspodjeli u *pod-smjese* A4 i A8. Ista logika vrijedi i za butan i propan.

```
125  def main():
126      global normalizirani_podaci
127      normalizirani_podaci = ucitaj()
128      #racunaj_wj(normalizirani_podaci)
129      init = [normalizirani_podaci[0]/3, normalizirani_podaci[1]/2, normalizirani_podaci[2]/2, normalizirani_podaci[0]/3, normalizirani_podaci[3]/2]
130      print(normalizirani_podaci)
131      print(init)
```

Slika 3-14. Kod u Pythonu za raspodjelu normaliziranog sastava u *pod-smjese*



Slika 3-15. Princip raspodjele normaliziranog sastava u odgovarajuće sustave A4, A7 i A8

Slika 3-16. prikazuje raspodjelu normaliziranih udjela iz sastava plina u odabrane *pod-smjese*. No potrebno je spomenuti kako su vrijednosti koje se nalaze na mjestu geometrijskih oblika s isprekidanim rubovima dobivene oduzimanjem rezultata dijeljenja od ukupne količine kako bi se smanjila količina podataka s kojima program mora manevrirati. Udjeli pojedinih komponenti u *pod-smjesi* se najprije normaliziraju, a zatim koriste za izračune metanskih brojeva svake pojedine *pod-smjese*.

```
81  def norm(l):
82      s = sum(l)
83      for i in range(len(l)):
84          l[i] = l[i] / s * 100
85      return l
```

Slika 3-16. Funkcija *norm* za normalizaciju sastava u svakoj odabranoj *pod-smjesi*

3.4. Izračun metanskih brojeva odabranih *pod-smjesa* i funkcija *f*

Funkcija *racunaj* računa metanske brojeve svake pojedine *pod-smjese* pomoću formule:

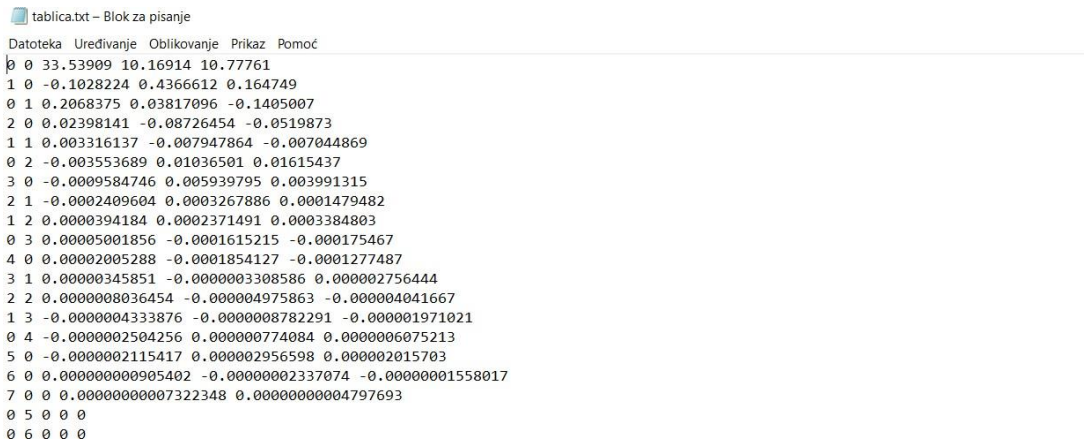
$$\sum_{i=0}^{i=7} \quad \sum_{j=0}^{j=6} a_{i,j} \cdot x^i \cdot y^j \tag{3-2}$$

gdje je $a_{i,j}$ odgovarajući multiplikator, a x i y predstavljaju normalizirane udjele komponenata iz svake odabrane *pod-smjese*.

Tablica 3-2. Multiplikatori za izračun metanskih brojeva sustava A4, A7, A8, A20

		A4	A7	A8	A20
i	j				
0	0	3,354E+01	1,017E+01	1,078E+01	2,992E+02
1	0	-1,028E-01	4,367E-01	1,647E-01	-1,512E+01
0	1	2,068E-01	3,817E-02	-1,405E-01	-3,116E-01
2	0	2,398E-02	-8,726E-02	-5,199E-02	7,636E-01
1	1	3,316E-03	-7,948E-03	-7,045E-03	4,548E-02
0	2	-3,554E-03	1,037E-02	1,615E-02	1,123E-02
3	0	-9,585E-04	5,940E-03	3,991E-03	-2,376E-02
2	1	-2,410E-04	3,268E-04	1,479E-04	-7,856E-04
1	2	3,942E-05	2,371E-04	3,385E-04	6,556E-04
0	3	5,002E-05	-1,615E-04	-1,755E-04	-2,147E-03
4	0	2,005E-05	-1,854E-04	-1,277E-04	4,355E-04
3	1	3,459E-06	-3,309E-07	2,756E-06	3,861E-06
2	2	8,036E-07	-4,976E-06	-4,042E-06	1,382E-06
1	3	-4,334E-07	-8,782E-07	-1,971E-06	-7,934E-06
0	4	-2,504E-07	7,741E-07	6,075E-07	6,699E-05
5	0	-2,115E-07	2,957E-06	2,016E-06	-4,608E-06
6	0	9,054E-10	-2,337E-08	-1,558E-08	2,611E-08
7	0	0	7,322E-11	4,798E-11	-6,144E-11
0	5	0	0	0	-8,369E-07
0	6	0	0	0	3,928E-09

Kao ulazni parametri, upotrijebljeni su multiplikatori za sustave A4, A7 i A8, zajedno s normaliziranim sastavom plina. Funkcija vraća tri različita metanska broja.



Slika 3-17. Datoteka za učitavanje podataka

```
def racunaj(par):
    a4 = norm([par[0], par[1], par[2], 0])
    a7 = norm([par[3], 0, normalizirani_podaci[2]-par[2], par[4]])
    a8 = norm([normalizirani_podaci[0]-par[0]-par[3], normalizirani_podaci[1]-par[1], 0, normalizirani_podaci[3]-par[4]])
    met_a4, met_a7, met_a8 = 0, 0, 0
    dat = open("tablica.txt", "r")
    red = dat.readline().strip().split()
    while red != []:
        i = int(red[0])
        j = int(red[1])
        met_a4 += float(red[2]) * (a4[0] ** i) * (a4[1] ** j)
        met_a7 += float(red[3]) * (a7[0] ** i) * (a7[2] ** j)
        met_a8 += float(red[4]) * (a8[0] ** i) * (a8[1] ** j)
        red = dat.readline().strip().split()
    dat.close()
    return met_a4, met_a7, met_a8
```

Slika 3-18. Funkcija racunaj vraća tri metanska broja za svaku odabranu *pod-smjesu*

Funkcija *f* pronalazi minimalnu i maksimalnu vrijednost od tri izračunata metanska broja, a zatim računa njihovu razliku.

```
def f(par):
    met_a4, met_a7, met_a8 = racunaj(par)
    return max(met_a4, met_a7, met_a8)-min(met_a4, met_a7, met_a8)
```

Slika 3-19. Funkcija *f* vraća razliku od najvećeg i najmanjeg metanskoga broja

3.5. Optimizacija u Pythonu

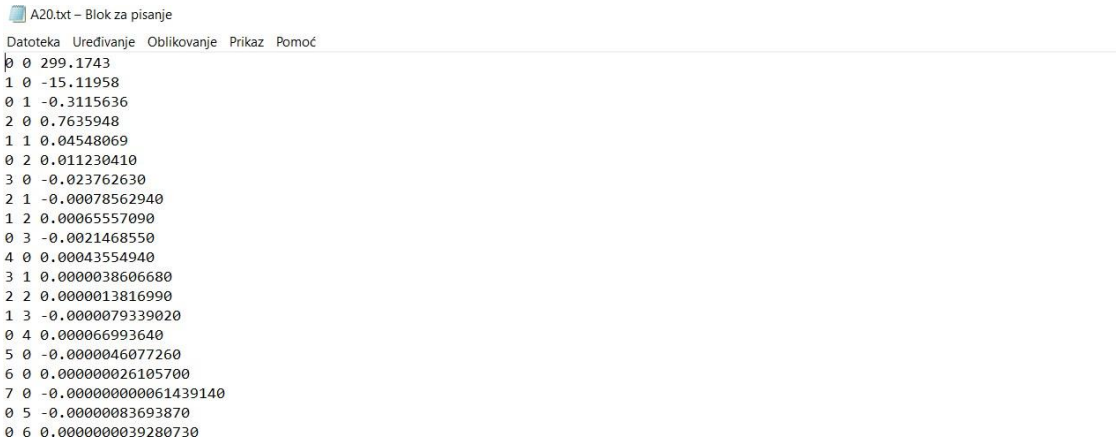
U ponuđenom kodu za izračun metanskog broja korištena je *SciPy library* koja nudi usluge rješavanja matematičkih, logičkih, znanstvenih i tehničkih problema. Naš problem sadržan je u samom sastavu prirodnog plina jer rezultira različitim vrijednostima metanskih brojeva. U nastojanju da izjednačimo njihove vrijednosti, posežemo za funkcijom *minimize* iz *SciPy library* koja generira odgovarajući sastav prirodnog plina. Potrebno je u samom kodu napomenuti kojom metodom će se funkcija poslužiti tijekom postupka računanja. Metode koje Python nudi za traženje minimuma neke funkcije, potkrijepljene su složenim matematičkim dokazima i objašnjenima te kao takve neće biti detaljno objašnjenje u ovome radu. No svakako je zanimljivo vidjeti vrijednosti metanskoga broja koje generiraju pojedine metode pri istom sastavu plina.

Upravo zbog činjenice da izračunati metanski brojevi sustava A4, A7 i A8 nisu jednaki, dobili smo razliku koja nije jednaka nuli. Takvo rješenje nije prihvatljivo pa je potrebno mijenjati normalizirane udjele komponenti u *pod-smjesama* kako bismo sveli razliku na minimum. U Pythonu je to izvedeno pomoću *minimize* funkcije. Za novi sastav dobiven u procesu optimizacije, ponovo se poziva funkcija *racunaj* koja generira nova tri metanska broja. Brojevi su sada gotovo jednaki, ali s obzirom da ne mogu biti 100 % isti, uzima se srednja vrijednost.

U završnom dijelu proračuna potrebno je uvesti *pod-smjesu* A20 koji se sastoji od metana, ugljikovog dioksida i dušika. S obzirom da smo na početku proračuna zanemarili negorive komponente, izračunata srednja vrijednost metanskog broja ne odgovara originalnoj plinovitoj smjesi. Kako bismo ispravili tu pogrešku, koristimo A20. Za *pod-smjesu* A20 računamo metanski broj po istom principu kao i za prethodno odabrane *pod-smjese*. Postupak se razlikuje u tome što ova *pod-smjesa* uzima kao ulazne parametre zbroj postotnih udjela prve četiri komponente originalne plinovite smjese, i postotni udio ugljikovog dioksida. Također, standard nalaže kako dušik treba izostaviti iz proračuna.

```python
def racunaj_a20():
    c1 = d["C1"] + d["C2"] + d["C3"] + d["C4+"]
    x = c1/(c1+d["CO2"]) * 100
    y = d["CO2"]/(c1+d["CO2"]) * 100
    a20 = 0
    dat = open("A20.txt", "r")
    red = dat.readline().strip().split()
    while red != []:
        i = int(red[0])
        j = int(red[1])
        a20 += float(red[2]) * (x ** i) * (y ** j)
        red = dat.readline().strip().split()
    return a20
```

Slika 3-20. Kod u Pythonu za izračun metanskog broja sustava A20



Slika 3-21. Multiplikatori za proračun metanskog broja sustava A20

Metanski broj originalne smjese plina dobiva se preko formule:

$$(metanski\ broj(A20) + prosječna\ vrijednost\ metanskih\ brojeva) - 100{,}0003 \quad (3\text{-}3)$$

gdje 100,0003 predstavlja metanski broj čistoga metana.

3.6. Sve proizlazi iz funkcije *main* – sažetak čitavoga proračuna

Funkcija *main* specifična je iz razloga što sadrži sve ostale funkcije tj. kompletan izračun metanskog broja polazi iz i završava s funkcijom *main*. Ona ponajprije poziva funkciju *učitaj* koja vraća listu s normaliziranim i pojednostavljenim sastavom originalne plinovite smjese. Zatim poziva funkciju *racunaj_wj* kojoj predaje normalizirani sastav plina te provodi postupak odabira odgovarajućih *pod-smjesa.* Proračun se nastavlja pod pretpostavkom da su odabrane *pod-smjese* A4, A7 i A8. Normalizirani sastav plina smješta u listu *init* za sve tri *pod-smjese.*

U sljedećem koraku slijedi optimizacija. Funkcija *f* prima podatke o udjelima komponenti u *pod-smjesama*. Poziva funkciju *racunaj* koja pak poziva funkciju *norm*. Funkcija *racunaj* vraća funkciji *f* podatke o tri metanska broja i zatim računa razliku između minimalnog i maksimalnog metanskog broja. Ukoliko razlika nije nula, mijenja sastav plina tako dugo dok ne nađe onaj za koji vrijedi da su sva tri metanska broja jednaka, tj. razlika između najvećeg i najmanjeg iznosi nula. Rezultati o novom optimiziranom sastavu spremaju se u listu *result*. Ponovno se poziva funkcija *racunaj*, ali ovoga puta joj se predaje optimizirani sastav plinovite smjese. Ona vraća tri metanska broja za promijenjene udjele komponenti u *pod-smjesama*. Kako bi dobiveni metanski broj korigirali za prisutnost negorivih komponenti, poziva se funkcija *racunaj_a20* koja vraća metanski broj *pod-smjese* A20. Konačna vrijednost metanskog broja dobivena je oduzimanjem metanskoga broja čistoga metana od zbroja metanskoga broja A20 i prosječne vrijednosti metanskoga broja.

```
125  def main():
126      global normalizirani_podaci
127      normalizirani_podaci = ucitaj()
128      #racunaj_wj(normalizirani_podaci)
129      init = [normalizirani_podaci[0]/3, normalizirani_podaci[1]/2, normalizirani_podaci[2]/2, normalizirani_podaci[0]/3, normalizirani_podaci[3]/2]
130      print(normalizirani_podaci)
131      print(init)
132      print(f(init))
133      result = optimize.minimize(f, init, method='SLSQP')
134      result = list(result['x'])
135      print(result)
136      met_a4, met_a7, met_a8 = racunaj(result)
137      print(met_a4, met_a7, met_a8)
138      prosjek = (met_a4 + met_a7 + met_a8) / 3
139      met_a20 = racunaj_a20()
140      met = (prosjek + met_a20) - 100.0003
141      print("a20: ", met_a20)
142      print("Konačno: ", met)
143
144
145
146  main()
```

Slika 3-22. Kod u Pythonu kao dio funkcije *main*

4. REZULTATI

Kroz predloženi programski kod provedeni su sastavi prirodnog plina za razdoblje od siječnja do srpnja 2022. godine. Upotrijebljeno je devet različitih metoda u procesu optimizacije: *Nelder-Mead, SLSQP, Powell, CG, BFGS, L-BFGS-B, TNC, COBYLA* i *trust-constr* u svrhu usporedbe rezultata. Također, navedeni su i metanski brojevi za pojedini sastav UPP-a dobiveni u Excelu.

Tablica 4-1. Sastavi za testiranje računalnog koda (1)

	sastav (y_i, mol %)					
komponenta	*Kinisis*	*Sean Spirit*	*Gaslog Gad.*	*QOGIR*	*Castillo de V.*	*Corcovado*
C1	97,189	97,173	95,341	97,352	97,394	97,307
C2	2,181	2,078	3,815	2,415	2,387	1,984
C3	0,08	0,079	0,498	0,108	0,103	0,083
i-C4	0,01	0,01	0,081	0,012	0,011	0,009
n-C4	0,119	0,126	0,085	0,009	0,008	0,118
i-C5	0,002	0,003	0,008	0,003	0,002	0,003
n-C5	0	0	0,003	0,001	0,001	0
neo-C5	0	0	0	0	0	0
C6+	0	0	0,002	0,001	0,001	0
N2	0,419	0,53	0,166	0,098	0,093	0,496
CO2	0	0	0	0	0	0
H2	0	0	0	0	0	0

Tablica 4-2. Sastavi za testiranje računalnog koda (2)

	sastav (y_i, mol %)						
komponenta	*Energy U.*	*BW P. Leeare*	*F. Aurora*	*Castillo de M.*	*SK Resolute*	*BW P. Aranda*	*LNG Endurance*
C1	93,671	97,063	94,943	96,238	99,311	95,613	95,457
C2	6,063	2,641	3,464	3,254	0,016	3,976	4,123
C3	0,189	0,109	1,046	0,281	0,004	0,28	0,293
i-C4	0,009	0,009	0,281	0,019	0,001	0,04	0,043
n-C4	0,016	0,007	0,225	0,03	0	0,039	0,041
i-C5	0,001	0,001	0,014	0,003	0,001	0,003	0,002
n-C5	0	0	0,002	0,002	0	0,001	0,001
neo-C5	0	0	0,002	0	0	0	0
C6+	0	0	0,001	0	0	0,001	0
N2	0,051	0,17	0,022	0,171	0,621	0,048	0,041
CO2	0	0	0	0,003	0,045	0	0
H2	0	0	0	0	0	0	0

Tablica 4-3. Metanski brojevi za razdoblje od siječnja do travnja 2022.

metode	*Kinisis*	*Sean Spirit*	*Gaslog Gad.*	*QOGIR*	*Castillo de V.*	*Corcovado*	*Energy U.*
Nelder-Mead	90,649	90,863	84,670	90,852	90,858	91,145	82,350
SLSQP	90,470	90,627	84,297	90,266	90,393	90,841	81,836
Powell	90,096	90,323	84,103	89,984	90,110	90,635	81,567
CG	90,311	903540	84,271	90,156	90,281	90,847	81,829
BFGS	90,226	90,442	84,270	90,160	90,250	90,878	81,805
L-BFGS-B	90,311	84,271	84,271	90,156	90,281	90,847	81,829
TNC	90,307	90,536	84,242	90,157	90,282	90,843	81,740
COBYLA	90,298	90,537	84,265	90,140	90,267	90,822	81,825
trust-constr	90,311	90,540	84,271	90,156	90,281	90,847	81,829
Excel	**90,617**	**90,794**	**84,447**	**90,653**	**90,784**	**91,104**	**82,326**

Tablica 4-4. Metanski brojevi za razdoblje od svibnja do srpnja 2022.

metode	*BW P. Leeare*	*F. Aurora*	*Castillo de M.*	*SK Resolute*	*BW P. Aranda*	*LNG Endurance*
Nelder-Mead	90,525	81,341	87,694	97,310	85,693	85,379
SLSQP	89,731	81,285	87,209	98,595	85,329	84,972
Powell	89,463	81,154	86,975	97,348	85,110	84,756
CG	89,637	81,294	87,137	98,586	85,305	84,951
BFGS	89,637	81,280	87,136	98,595	85,305	84,954
L-BFGS-S	89,637	81,295	87,136	98,586	85,305	84,950
TNC	90,307	81,298	87,108	98,598	85,257	84,957
COBYLA	89,600	81,279	87,127	98,648	85,363	84,974
trust-constr	89,636	81,295	87,136	98,586	85,305	84,950
Excel	**90,144**	**81,299**	**87,460**	**97,545**	**85,630**	**85,262**

5. ZAKLJUČAK

Cilj ovoga rada bio je samostalno kreirati program u Pythonu koji će generirati metanske brojeve za pojedine plinovite smjese. Glavni zaključci proizašli iz ovoga rada su sljedeći:

- Programski kod u Pythonu računa metanski broj prema postupku koji je definiran standardom *HRN EN 16726 - Plinska infrastruktura - Kvaliteta plina - Grupa H.* Vrijednosti metanskoga broja se ne poklapaju s vrijednostima koje su dobivene u Excelu.
- Vrijednost izračunatog metanskog broja u Pythonu ovisi o metodi koja je upotrijebljena u procesu optimizacije.
- Kako bi kod uspješno funkcionirao, potrebno je unositi one podatke o sastavu plina koji su definirani unutar samoga koda.
- Predloženi programski kod funkcionira samo ukoliko su odabrane *pod-smjese* A4, A7 i A8.
- Prilikom testiranja BW. P. Leeare sastava, razlika između najmanjeg i najvećeg metanskoga broja poprilično je velika što je onemogućilo *solveru* u Excelu da dovrši proračun. Ovakav problem se nije javljao u Pythonu.
- Uslijed izuzetno visokog udjela metana u SK Resolute sastavu, *solver* u Excelu nije bio u stanju ponuditi krajnje rješenje, dok Python s druge strane nije imao nikakvih problema s ovim sastavom.

6. POPIS LITERATURE

1. KOCHENDERFER, J. M., WHEELER, A. T., 2019. Algorithms for Optimization. MIT Press.
2. PALMER, G., 2017. Methane Number. *The Journal of Natural Gas Engineering*, vol. 2, no. 2, str. 134-142.
3. MELENSHEK, M., OLSEN, D.B., 2009. Methane number testing of alternative gaseous fuels. *Fuel*, vol. 88, str. 650-656.
4. RYAN, T.W. III, CALLAHAN, T.J., KING, S.R. 1993. Engine knock rating of natural gases—Methane number. *Journal of Engineering for Gas Turbines Power*, vol. 115, str. 769-776.
5. DE CARVALHO Jr, A.V. 1985. Natural gas and other alternative fuels for transportation purposes. *Energy*, vol. 10, no. 2, str. 187-215.
6. COMMITTEE FOR STANDARDIZATION, 2015. Gas infrastructure - Quality of gas - Group H. EN ISO 16726.
7. HRVATSKA ENERGETSKA REGULATORNA AGENCIJA, 2021. Opći uvjeti opskrbeplinom, NN 100/2021.

- Web izvori:

8. SCIPY.OPTIMIZE.MINIMIZE – SciPy v1.9.0 Manual.

   URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

9. LNG Hrvatska, 2022. URL: https://lng.hr/novosti/
10. PLINACRO, 2022. Kvaliteta prirodnog plina - objava podataka.
    URL: https://www.plinacro.hr/default.aspx?id=106

IZJAVA

Izjavljujem da sam ovaj rad izradila samostalno na temelju znanja stečenih na Rudarsko–geološko–naftnom fakultetu služeći se navedenom literaturom.

Josipa Tisaj

KLASA: 602-01/22-01/96
URBROJ: 251-70-12-22-2
U Zagrebu, 05.09.2022.

Josipa Tisaj, studentica

RJEŠENJE O ODOBRENJU TEME

Na temelju vašeg zahtjeva primljenog pod KLASOM 602-01/22-01/96, URBROJ: 251-70-12-22-1 od 30.04.2022. priopćujemo vam temu završnog rada koja glasi:

RAZVOJ KODA U PROGRAMSKOM JEZIKU PYTHON ZA IZRAČUN KVALITETE PLINA UPLINJENOG NA UPP TERMINALU OMIŠALJ

Za mentoricu ovog završnog rada imenuje se u smislu Pravilnika o izradi i ocjeni završnog rada Prof.dr.sc. Daria Karasalihović Sedlar nastavnik Rudarsko-geološko-naftnog-fakulteta Sveučilišta u Zagrebu i komentora Prof.dr.sc. Domagoj Vulin.

Mentorica:

(potpis)

Prof.dr.sc. Daria Karasalihović Sedlar

(titula, ime i prezime)

Predsjednik povjerenstva za završne i diplomske ispite:

(potpis)

Izv.prof.dr.sc. Luka Perković

(titula, ime i prezime)

Komentor:

(potpis)

Prof.dr.sc. Domagoj Vulin

(titula, ime i prezime)

Prodekan za nastavu i studente:

(potpis)

Izv.prof.dr.sc. Borivoje Pašić

(titula, ime i prezime)