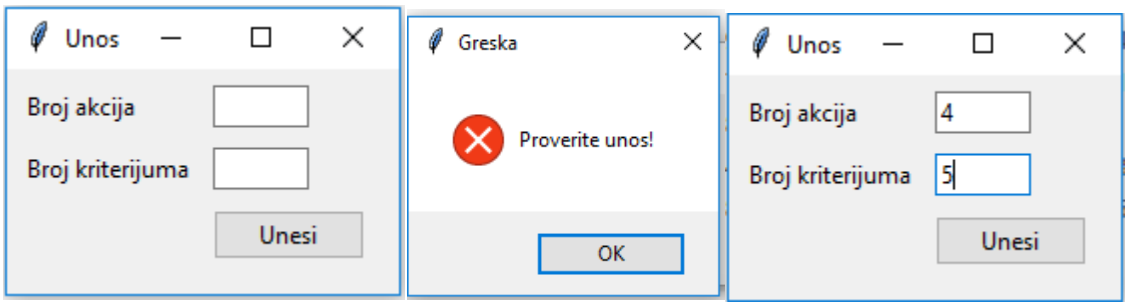
AHPElectreI softver

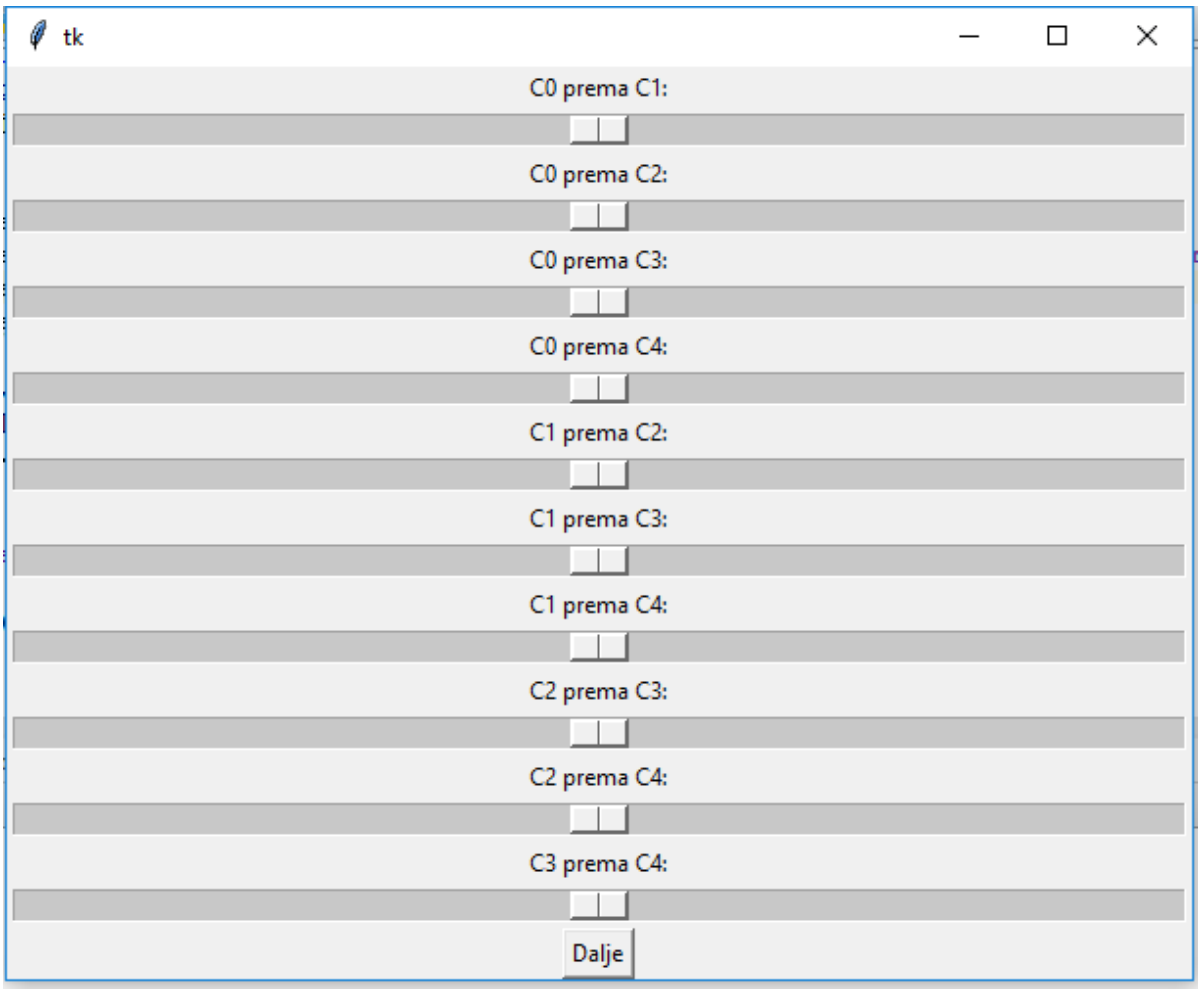
Slobodan Nedeljković
Mihailo Jovanović

UI modul

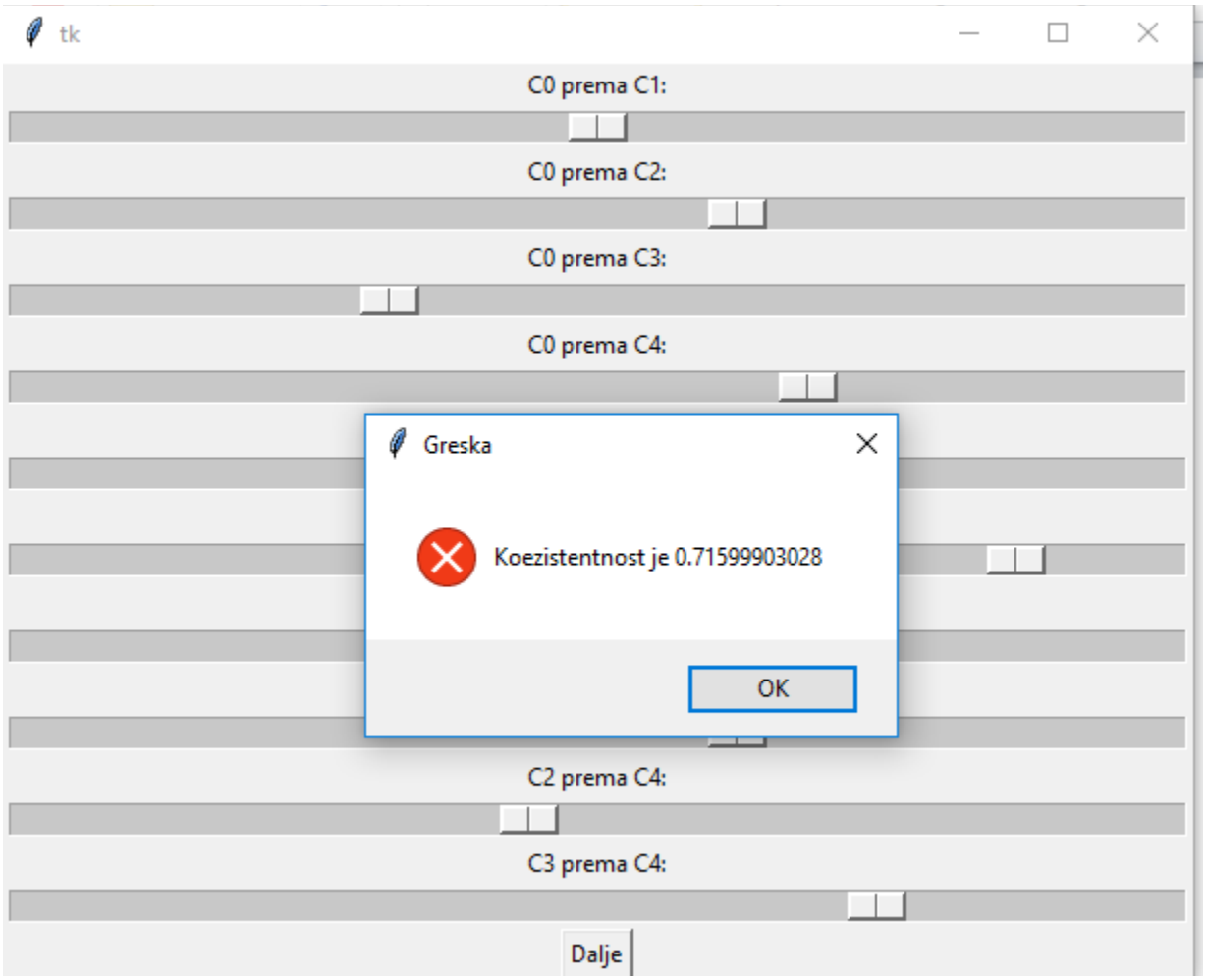
Korisnik kad pokrene program od njega se traži da kaže koliko akcija i kriterijuma ima odluka koju želi da donese. Ukoliko ne unese a korisnik hoće da nastavi ispisati grešku i čekati ponovo unos. Kada unese preći na sledeći ekran.



Sledeći korak u algoritmu je određivanje težina za svaki od zadatih kriterijuma. Težine se određuju pomoću AHP metode, odnosno, određuju se na način gde se svaki kriterijum poredi sa svakim na način da poređenja budu koezistentna.



Da bi unost težina bio valjan potrebno je računati i koezistentnost težina koje se unose i za slučaj da je koezistentnost veća od 0.1 treba obavestiti korisnika i tražiti od njega da ponovo unese odnose težina.



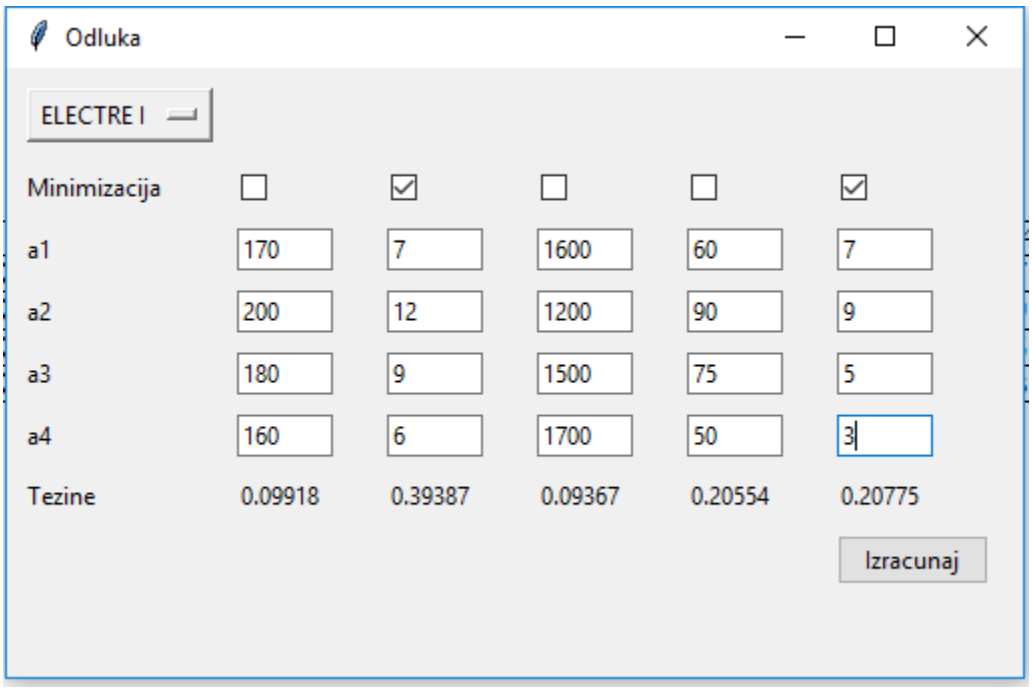
Da bi podaci bili koezistentni, odnosno da imaju logički sled, da ako je C0 bitnije od C2, a manje bitnije od C4 da C2 ne bude bitnije od C4.

Tj ako je C4 > C0 > C2 onda C4 < C2 nije moguće odnosno nije koezistentno. Bilo bi dobro implementirati i način da se grafički ne dopusti korisniku da unese ovakve vrednosti.

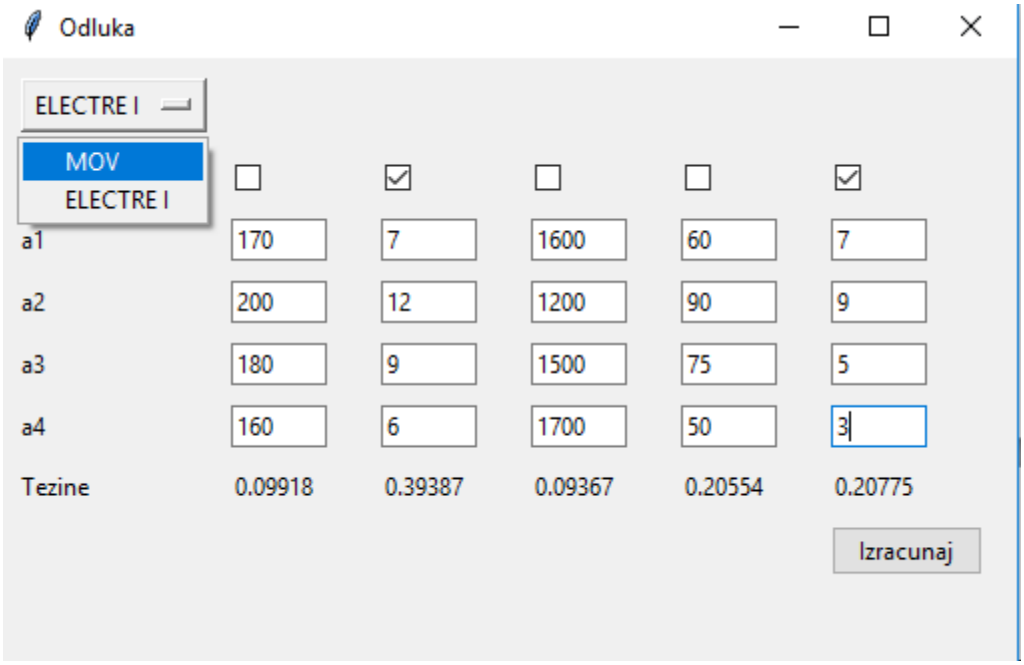
Kada korisnik unese koezistentne vrednosti za poređenje težina treba mu dati mogućnost da unese za svaki od mogućih akcija.

Ispod svake od kolona treba ispisati težine za dat kriterijum, odnosno kolonu.

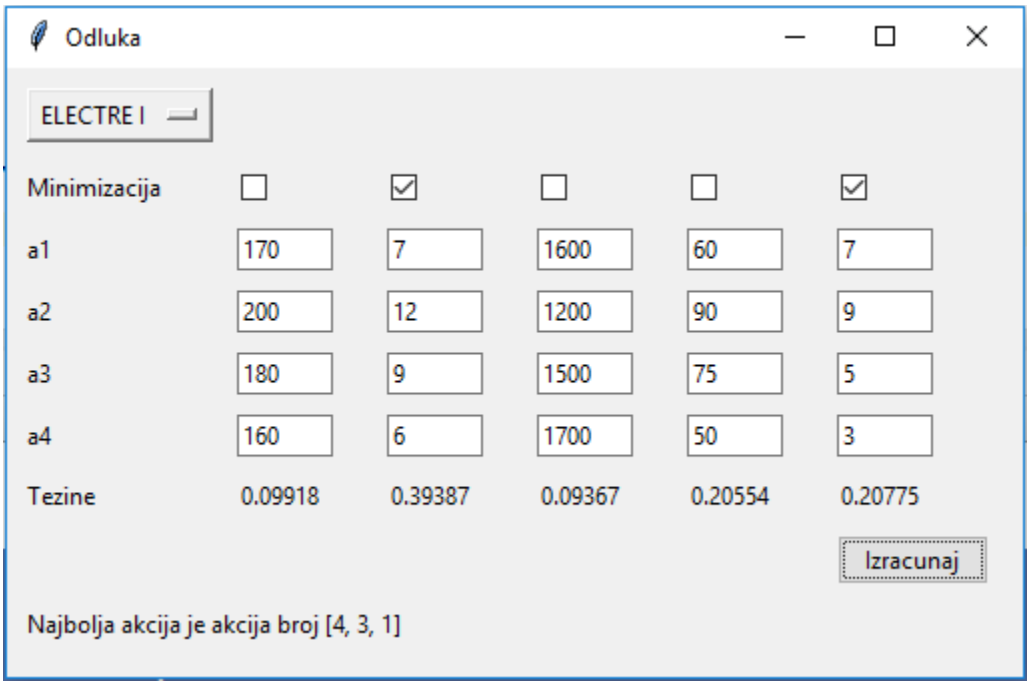
Korisnik pored vrednosti za svaki od kriterijuma unosi i to da li određeni kriterijum treba minimizovati. Odnosno ako se radi o minimizaciji tog kriterijuma treba čekirati polje iznad kolone, odnosno kriterijuma.



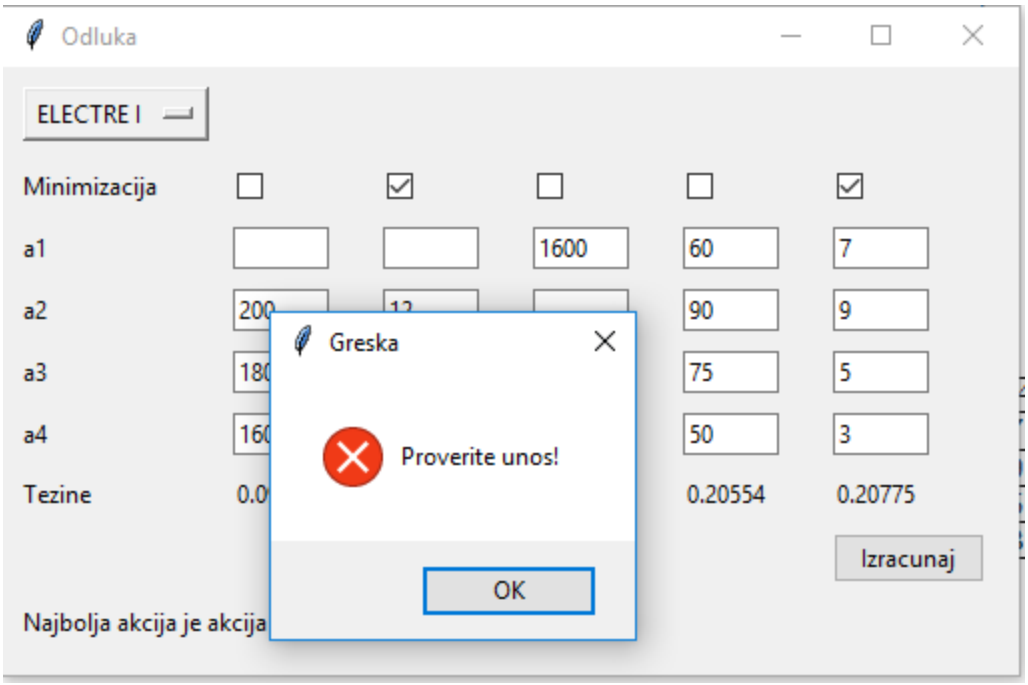
Pored ELECTREI metode postoji mogućnost korišćenja drugih metoda, ovde je dat kao primer MOV metoda je za sad implementirana u softver.



Kada unese vrednosti za sve kriterijume, odabere koji kriterijumi se minimizuju i koju metodu koristi korisnik klikom na dugme izračunaj dobija predlog najboljeg rešenja, odnosno najbolja tri rešenja.



Ukoliko korisnik ne unese sve vrednosti za kriterijume treba ga obavestiti o tome da ne bi došlo do greške priliko rada programa.



Grafika u aplikaciji se radi pomoću biblioteke grafičke biblioteke **tkinter.**Da bi uvezli ovu biblioteku u kod koristimo sledeće linije koda:

```
#Graficke biblioteke
from tkinter import * #importuje sve podrazumevane module iz ove biblioteke
from tkinter import ttk #importuje ttk modul iz ove biblioteke
from tkinter import messagebox #importuje modul za ispis poruka kao što su greške
```

Ova biblioteka za grafiku je izabrana zato što predstavlja standard za GUI programiranje u python, postoji najviše dokumentacije za nju i zato što dolazi uz osnovni paket prilikom instalacije pythona.Importovanje ttk i messagebox modula je potrebno zato što oni ne spadaju u podrazumevane module koji se importuju prilikom importovanja putem *.

Opis biblioteke, preuzet sa vikipedije:

Tkinter je Python biblioteka za Tk GUI toolkit. To je standardni Python interfejs za Tk GUI toolkit i de facto standard za GUI. Tkinter je uključen u standardni Microsoft Windows i Mac OS instalaciju Pyithon-a. Ime Tkinter dolazi iz Tk interfejsa. Tkinter je napisao Fredrik Lundh. Kao i kod većine savremenih Tk vezivanja, Tkinter se implementira kao Pithon omotač oko kompletnog Tcl tumača ugrađenog u Python interpreter. Pozivi Tkinter-a prevedeni su u Tcl komande koji se prenose ovom ugrađenom tumaču, čime se omogućava mešanje Pythona i Tcl-a u jednu aplikaciju. Python 2.7 i Python 3.1 sadrže funkcionalnost "themed Tk" ("ttk") Tk 8.5. Ovo omogućava da se Tk vidgeti lako tematizuju kako bi izgledali kao izvorno okruženje u kojem aplikacija radi, čime se bavi dugoročnom kritikom Tk-a (a time i Tkinter-a). Postoji nekoliko popularnih alternativnih GUI biblioteka dostupnih, kao što je vkPython , PiKt (PiSide), Pigame, Piglet i PiGTK. Tkinter je besplatni softver objavljen pod Python licencom.

Modul ELECTREI

NumPi je biblioteka za Python programski jezik, dodajući podršku za velike, višedimenzionalne matrice i matrice sa velikom zbirkom matematičkih funkcija na visokom nivou radi na ovim nizovima. Preci NumPi, Numeric, originalno je kreirao Jim Hugunin sa doprinosima nekoliko drugih programera. Travis Oliphant je 2005. godine stvorio NumPi tako što je uključio karakteristike konkurentskog Numarraia u Numeric, sa obimnim izmenama. NumPi je softver otvorenog koda i ima mnogo doprinosa.

```
#biblioteka za rad sa matricama u pythonu
#najvise koriscena zbog inicijalizacije samih matrica
import numpy as np
#matematicka biblioteka, koristi se za koriscenje nekih kompleksnijih matematickih
operacija
importmath as mt
#za određivanje minimalnog mogućeg broja i maksimalnog
importsys
```

Prvi korak u electrei metodi je normalizacija matrice.Na osnovu sledeće formule treba normalizovati matricu.

$$\mathbf{n_{ij}} = \frac{\mathbf{x_{ij}}}{\sqrt{\sum_{j=1}^{m} \mathbf{x_{ij}^2}}}$$

Gde je n_{ij} element normalizovane matrice koji se nalazi u i-toj koloni i j-tom redu.
X_{ij} element ulazne matrice matrice koji se nalazi u i-toj koloni i j-tom redu.

Na primer za datu matricu

$$O := \begin{pmatrix} 170 & 7 & 1600 & 60 & 7 \\ 200 & 12 & 1200 & 90 & 9 \\ 180 & 9 & 1500 & 75 & 5 \\ 160 & 6 & 1700 & 50 & 3 \end{pmatrix}$$

Izračunavanje n_{11} izgledati ovako

$$n_{11} = \frac{x_{11}}{\sqrt{\sum_{j=1}^{m} x_{ij}^2}} = \frac{170}{\sqrt{170^2 + 200^2 + 180^2 + 160^2}} = \frac{170}{356.23} = 0.477$$

Ovo je najbolje odraditi kroz jednu metodu koja će za ulaz da prihvati jednu matricu n*m. Ovu metodu ćemo nazvati normalizovanjeMatrice.

```
defnormalizovanjeMatrice(matrica):
```

Da bi mogli da kao rezultat vratimo novu normalizovanu matriuc potrebno je definisati novu matricu iste visine i širine kao i početnu matricu

```
novaMatrica = np.zeros((len(matrica), len(matrica[0])))
```

Možemo da uočimo iz formule da će deljenik uvek biti isti broj za isti kriterijum, odnosno biće koren sume kvadrata j-te kolone elemenata ulazne matrice.Koren računamo pomoću spoljne matematičkog modula.

Za $n_{1,1}$ $n_{2,1}$ $n_{3,1}$ $n_{4,1}$ će uvek biti $\sqrt{(170^2+200^2+180^2+160^2)}$

$$O := \begin{pmatrix} 170 & 7 & 1600 & 60 & 7 \\ 200 & 12 & 1200 & 90 & 9 \\ 180 & 9 & 1500 & 75 & 5 \\ 160 & 6 & 1700 & 50 & 3 \end{pmatrix}$$

Pa tako za svaku kolonu prvo to računamo

```
fori in range(0, len(matrica[0])):
        suma = 0
for j in range(0, len(matrica)):
            suma = suma + matrica[j][i] * matrica[j][i]
        suma = mt.sqrt(suma)
```

To predstavlja najteži deo metode, sledeći korak je da za svaki element u koloni podelimo sa korenom sumom.

```
fork in range(0, len(matrica)):
            novaMatrica[k][i] = matrica[k][i] / suma
```

Nova matrica se ovako za svaki element podesi.Na kraju se samo ova nova matrica vrati kao rezultat funkcije

```
returnnovaMatrica
```

$$N := \begin{pmatrix} 0{,}477 & 0{,}397 & 0{,}529 & 0{,}426 & 0{,}546 \\ 0{,}561 & 0{,}681 & 0{,}396 & 0{,}639 & 0{,}702 \\ 0{,}505 & 0{,}511 & 0{,}496 & 0{,}532 & 0{,}390 \\ 0{,}449 & 0{,}340 & 0{,}562 & 0{,}355 & 0{,}234 \end{pmatrix}$$

Za gore prikazanu matricu ovo je normalizovana matrica.

Sledeči korak je težinska normalizacija matrice. Ovo se radi tako što se svaki element matrice pomnoži sa težinom za taj kriterijum. Ovako izgledaju težinski koeficijenti.Ima ih isto koliko i kriterijuma jer je svaki težinski koeficijent zadužen za određeni kriterijum.

$$T := (0{,}1 \quad 0{,}2 \quad 0{,}1 \quad 0{,}3 \quad 0{,}3)$$

Ovo se odradi tako što se funkciji proslede matrica i niz tezinskih koeficijenta.Pa se inicijalizuje nova matrica iste velicine kao i prosledjena matrica. I onda se pomoću dve for petlje kreće kroz matricu i svaki element se množi sa tezinskim komponentom koja je na istoj sirinskoj lokaciji kao i sirinska kordinata za tu matricu.

```
# tezinska normalizacija
# normalizovana matrica svaki od svojih elemenata mnozi sa tezinskim koeficijentom za
dat kriterijum
# funkcija prihvata matricu i tezinske koeficijente
def tezinskoNormalizovanje(matrica, t):
    novaMatrica = np.zeros((len(matrica), len(matrica[0])))
for i in range(0, len(matrica[0])):
for k in range(0, len(matrica)):
            novaMatrica[k][i] = t[i] * matrica[k][i]

return novaMatrica
```

$$TN := \begin{pmatrix} 0{,}0477 & 0{,}0794 & 0{,}0529 & 0{,}1278 & 0{,}1639 \\ 0{,}0561 & 0{,}1362 & 0{,}0396 & 0{,}1917 & 0{,}2108 \\ 0{,}0505 & 0{,}1022 & 0{,}0496 & 0{,}1596 & 0{,}1171 \\ 0{,}0449 & 0{,}0680 & 0{,}0562 & 0{,}1065 & 0{,}0702 \end{pmatrix}$$

Ovako izgleda težinski normalizovana matrica.

Određivanje skupa saglasnosti i nesaglasnost. Skupovi saglasnosti, odnosno nesaglasnosti su dvodimenzione liste.U ovom koraku upoređuju se parovi akcija p i r (p,r = 1,2,...,m i p≠ r). Najpre se formira skup saglasnosti S_{pr} za akcije a_p i a_r, koji se sastoji od svih kriterijuma (J = {j|j=1,...,n}), za koje jeakcija ap poželjnija od akcije ar, odnosno:

S_{pr} = {j | x_{pj} [3] >= x_{rj}}

Ukoliko se radi o kriterijumu tipa minimizacije, znak jednakosti je suprotan.Zatim se formira komplementaran skup nesaglasnosti:

NS_{pr} = J - S_{pr} = {j | x_{pi}< x_{ri}}

```
# odredjivanje skupova saglasnosti S i nesaglasnosti NS
# funkciji se prosledjuje matrica na osnovu koje se ordedjuju skupovi saglasnosti i
nesaglasnosti
#
def skupoviSaglasnosti(matrica):
    listaSaglasnosti = []
    listaNesaglasnosti = []

for p in range(0, len(matrica)):
for r in range(0, len(matrica)):
if p != r:
                redSaglasnih = []
                redNesaglasnih = []
```

```
for j in range(0, len(matrica[0])):
if matrica[p][j] >= matrica[r][j]:
                        redSaglasnih.append(j)
else:
                        redNesaglasnih.append(j)

                listaSaglasnosti.append(redSaglasnih)
                listaNesaglasnosti.append(redNesaglasnih)
return listaSaglasnosti,listaNesaglasnosti
```

	Skup saglasnosti	Skup nesaglasnosti
p = 1, r = 2	2, 3, 4	1, 5
p = 1, r = 3	2, 3, 4, 5	1
p = 1, r = 4	1, 5	2, 3, 4
p = 2, r = 1	1, 5	2, 3, 4
p = 2, r = 3	1, 5	2, 3, 4
p = 2, r = 4	1, 5	2, 3, 4
p = 3, r = 1	1	2, 3, 4, 5

	Skup saglasnosti	Skup nesaglasnosti
p = 3, r = 2	2, 3, 4	1, 5
p = 3, r = 4	1, 5	2, 3, 4
p = 4, r = 1	2, 3, 4	1, 5
p = 4, r = 2	2, 3, 4	1, 5
p = 4, r = 3	2, 3, 4	1, 5

Ovako izgledaju skpovi saglasnosti odnosno nesaglasnosti za našu matricu ukoliko se minimizacija vrši za drugi i cetvrti kriterijum.

Matrica saglasnosti odreduje se na osnovu skupa saglasnosti. Elemente matrice cine indeksisaglasnosti, cija je vrednost jednaka sumi težinskih koeficijenata koji odgovaraju pripadajucim elementima skupova saglasnosti.Za dati primer indeksi saglasnosti jednaki su:

$$s_{pr} = \sum_{j \in S_{pr}} t_j$$

```
#Pravljenje matrice saglasnosti
# na osnovu tezinem matrice i liste Saglasnosti pravi se matrica saglasnosti
def matricaSaglasnostiFunkcija(listaSaglasnosti, tezine, matrica):
    matricaSaglasnosti = np.zeros((len(matrica), len(matrica)))
# brojac se definiše da bi moglo kroz listu saglasnosti da se kreće
    brojac = 0
for i in range(0,len(matrica)):
for j in range(0,len(matrica)):
            suma = 0
if i != j:
for s in range(0, len(listaSaglasnosti[brojac])):
                    suma= suma + tezine[listaSaglasnosti[brojac][s]]
                brojac=brojac+1
matricaSaglasnosti[i][j]=suma
return matricaSaglasnosti
```

$$MS := \begin{pmatrix} 0 & 0{,}6 & 0{,}9 & 0{,}4 \\ 0{,}4 & 0 & 0{,}4 & 0{,}4 \\ 0{,}1 & 0{,}6 & 0 & 0{,}4 \\ 0{,}6 & 0{,}6 & 0{,}6 & 0 \end{pmatrix}$$

Matrica saglasnosti za naš primer.

Matrica nesaglasnosti odreduje se na osnovu skupa nesaglasnosti. Elemente matrice cineindeksi nesaglasnosti, koji se odreduju na osnovu formule:

$$\mathbf{ns_{pr}} = \frac{\max\limits_{\mathbf{j \in NS_{pr}}} \left| \mathbf{tn_{pj} - tn_{rj}} \right|}{\max\limits_{\mathbf{j \in J}} \left| \mathbf{tn_{pj} - tn_{rj}} \right|}$$

gde je tn – element težinske normalizovane matrice odlucivanja.

$$ns_{12} = \frac{\max\limits_{j \in NS_{12}} \left(\left| tn_{11} - tn_{21} \right|, \left| tn_{15} - tn_{25} \right| \right)}{\max\limits_{j \in J} \left(\left| tn_{11} - tn_{21} \right|, \left| tn_{12} - tn_{22} \right|, \left| tn_{13} - tn_{23} \right|, \left| tn_{14} - tn_{24} \right|, \left| tn_{15} - tn_{25} \right| \right)}$$

Primer za našu matricu

```
#Pravljenje matrice nesaglasnosti
# na onsonuv liste nesaglasnosti, tezine i matrice prave se matrica nesaglasnosti
def matricaNesaglasnostiFunkcija(listaNesaglasnosti,tezine,matrica):
    matricaNesaglasnosti = np.zeros((len(matrica), len(matrica)))

    brojac = 0
for p in range(0, len(matrica)):
for r in range(0, len(matrica)):
            rez = 0

if p != r:
                maksimalnog = -sys.maxsize
for s in range(0, len(listaNesaglasnosti[brojac])):
                    trenutni =abs(matrica[p][listaNesaglasnosti[brojac][s]]-
matrica[r][listaNesaglasnosti[brojac][s]])
if trenutni > maksimalnog:
                        maksimalnog = trenutni

                maks = -sys.maxsize
for k in range(0, len(matrica[0])):
if maks <abs(matrica[p][k] - matrica[r][k]):
                        maks = abs(matrica[p][k] - matrica[r][k])
```

```
                brojac = brojac + 1
if maks != 0:
                    rez = maximalnog/maks
            matricaNesaglasnosti[p][r]=rez
return matricaNesaglasnosti
```

$$MNS := \begin{Bmatrix} 0 & 0{,}7329 & 0{,}0599 & 0{,}2274 \\ 1 & 0 & 0{,}3637 & 0{,}6064 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0{,}8794 & 0 \end{Bmatrix}$$

Za naš primer

Matrica saglasne dominacije odreduje se na osnovu vrednosti praga indeksa saglasnosti, koji se može definisati kao prosecni indeks saglasnosti:

$$\mathbf{PIS} = \frac{\sum_{p=1}^{m}\sum_{r=1}^{m} s_{pr}}{m(m-1)}$$

pri cemu je p ¹ r.

Potom se formira matrica saglasne dominacije na osnovu sledeceg kriterijuma:

$$\mathbf{msd_{pr} = 1 \text{ za } s_{pr} \geq PIS}$$

$$\mathbf{msd_{pr} = 0 \text{ za } s_{pr} < PIS}$$

U ovom slucaju,

$$PIS = \frac{0+0.6+0.9+0.4+0.4+0+0.4+0.4+0.1+0.6+0+0.4+0.6+0.6+0.6+0}{4(4-1)} = \frac{6}{12} = 0{,}50$$

$$MSD := \begin{Bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{Bmatrix}$$

```
#Pravljenje matrice saglasne dominacije
# kada se odredi matrica saglasnosti treba se na osnovu nje izracunati matrica
saglasne dominacije
# saglasna dominacija se racuna na osnovu matrice saglasnosti
def matricaSaglasneDominacijeFunkcija(matricaSaglasnosti):
    novaMatrica = np.zeros((len(matricaSaglasnosti), len(matricaSaglasnosti)))
    suma = 0
for i in range(0,len(matricaSaglasnosti)):
for j in range(0, len(matricaSaglasnosti)):
            suma = suma + matricaSaglasnosti[i][j]
#Ovde je izračunata PIS
    pis = suma/((len(matricaSaglasnosti)-1)*len(matricaSaglasnosti))

for i in range(0, len(matricaSaglasnosti)):
for j in range(0, len(matricaSaglasnosti)):
if i != j:
if matricaSaglasnosti[i][j] >= pis:
                    novaMatrica[i][j]=1
return novaMatrica
```

Matrica nesaglasne dominacije izracunava se analogno MSD; najpre se racuna prosecanindeks nesaglasnosti:

$$\mathbf{PINS} = \frac{\sum_{p=1}^{m}\sum_{r=1}^{m} \mathbf{ns_{pr}}}{\mathbf{m(m-1)}}$$

pri cemu je p ¹ r.
Potom se formira matrica nesaglasne dominacije na osnovu sledeceg kriterijuma:

$$\mathbf{mnsd_{pr} = 1 \text{ za } ns_{pr} \leq PINS}$$

$$\mathbf{mnsd_{pr} = 0 \text{ za } ns_{pr} > PINS}$$

U ovom slucaju je:

$$PINS = \frac{0+0.7329+0.0599+0.2274+1+0+0.3637+0.6064+1+1+0+1+1+1+0.8794+0}{4(4-1)} =$$

$$= \frac{8{,}9599}{12} = 0.7391$$

$$MNSD := \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

```
#Pravljenje matrice nesaglasne dominacije
# Matrica nesaglasne dominacije se racuna uz pomoc matrice nesaglasnosti
def matricaNesaglasneDominacijeFunkcija(matricaNesaglasnosti):
    novaMatrica = np.zeros((len(matricaNesaglasnosti), len(matricaNesaglasnosti)))
    suma = 0
for i in range(0,len(matricaNesaglasnosti)):
for j in range(0, len(matricaNesaglasnosti)):
            suma = suma + matricaNesaglasnosti[i][j]
    pins = suma/((len(matricaNesaglasnosti)-1)*len(matricaNesaglasnosti))

for i in range(0, len(matricaNesaglasnosti)):
for j in range(0, len(matricaNesaglasnosti)):
if i != j:
if matricaNesaglasnosti[i][j] <= pins:
                    novaMatrica[i][j]=1
return novaMatrica
```

Elementi matrice agregatne dominacije jednaki su proizvodu elemenata na odgovarajucoj poziciji u matricama saglasne i nesaglasne dominacije:

$$\mathbf{ad}_{\mathbf{pr}} = \mathbf{sd}_{\mathbf{pr}} \cdot \mathbf{nsd}_{\mathbf{pr}}$$

```
#Pravljenje matrice agregatne dominacije
# matrica agregatne dominacije je u sustini logicko sabiranje dveju matrica
def agregatnaDominacija(c, d):
    res = np.zeros((len(c), len(c)))
for i in range(0, len(c)):
for j in range(0, len(c)):
if c[i][j] == d[i][j] == 1:
                res[i][j] = 1
return res
```

tako da, u ovom slucaju, matrica ima sledece vrednosti:

$$MAD := \begin{Bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{Bmatrix}$$

	A_1	A_2	A_3	A_4	
A_1	0	1	1	0	$A_1 \rightarrow A_2, A_3$
A_2	0	0	0	0	A_2 ne dominira
A_3	0	0	0	0	A_3 ne dominira
A_4	0	0	0	0	A_4 ne dominira

Iz ovoga vidimo da akcija A1 dominira u odnosu na akcije A2 i A3. To vidimo zato što u redu za akciju A1 imamo najviše jedinica.Pa tako i računamo u svakom redu koliko ima jedinica i vraćamo one tri koje su najbolje.

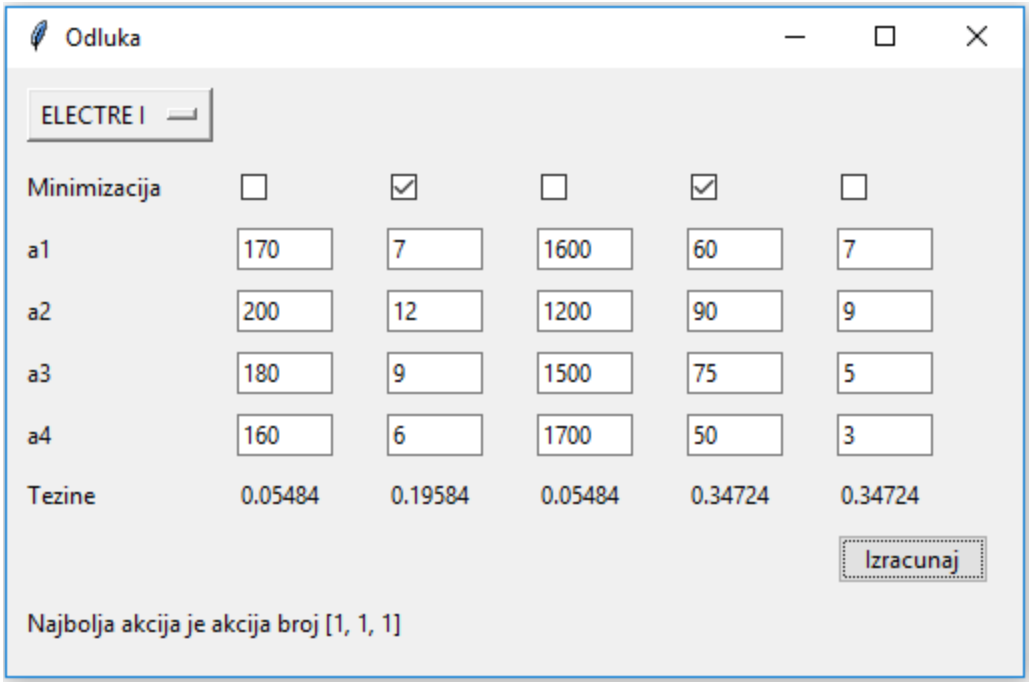
```
#metoda odluka racuna tri najoptimalnije odluke na osnovu matrice agregatne dominacije
def odluka(matrica):
    max = 0
```

```
max1 = 0
max2 = 0
red = 0
red1 = 0
red2 = 0
for i in range(len(matrica)):
        trenutan = 0
for j in range(len(matrica[0])):
if(matrica[i][j] != 0):
                trenutan = trenutan + 1

if trenutan > max:
            max = trenutan
            red = i
elif trenutan > max1:
            max1 = trenutan
            red1 = i
elif trenutan > max2:
            max2 = trenutan
            red2 = i
#ispisujemo +1 zato što smo svuda u kodu radili od 0 a za korisnika je intuativnije da
#kreće od 1 pa tako zbog prikaza se dodaje ta jedinica
return [red+1, red1+1, red2+1]
```

Ovim zaključujemo sve korake u ELECTREI metodi. Možemo da vidimo da je svaki korak uradjen kao jedna funkcija odnosno metoda. Sad sve što je potrebno uraditi je pozvati svaku od medota u pravom redosledu i proslediti joj prave parameter.To radimo kroz jednu metodu main koja objedinjuje to sve.

```
def main(x, tezine):
    x = normalizovanjeMatrice(x)
# pozivanje funkcije za tezinsku normalizaciju matrice
x = tezinskoNormalizovanje(x, tezine)
# pozivanje funkcije za generisanje skupova saglasnosti
    # skupovi su skladisteni u liste jer su promenjivih duzina
listaSaglasnosti, listaNesaglasnosti = skupoviSaglasnosti(x)
# pozivanje funkcija za generisanje matrice saglasnosti
matricaSaglasnosti = matricaSaglasnostiFunkcija(listaSaglasnosti, tezine, x)
# pozivanje funkcije za generisanje matrice nesaglasnosti
matricaNesaglasnosti = matricaNesaglasnostiFunkcija(listaNesaglasnosti, tezine, x)
# pozivanje funkcije za konstruisanje matrice saglasne dominacije
matricaSaglasneDominacije = matricaSaglasneDominacijeFunkcija(matricaSaglasnosti)
# pozivanje funkcije za konstruisanje matrice neseglasne dominacije
matricaNesaglasneDominacije =
matricaNesaglasneDominacijeFunkcija(matricaNesaglasnosti)
#donosnje odluke na osnovu matrice koje se dobija kao prozivod funkcije agregatne
dominacije
return odluka(agregatnaDominacija(matricaSaglasneDominacije,
matricaNesaglasneDominacije))
```

Sve se ovo odigra kad se pritisne dugme izračunaj.



Konačan kod izgleda ovako

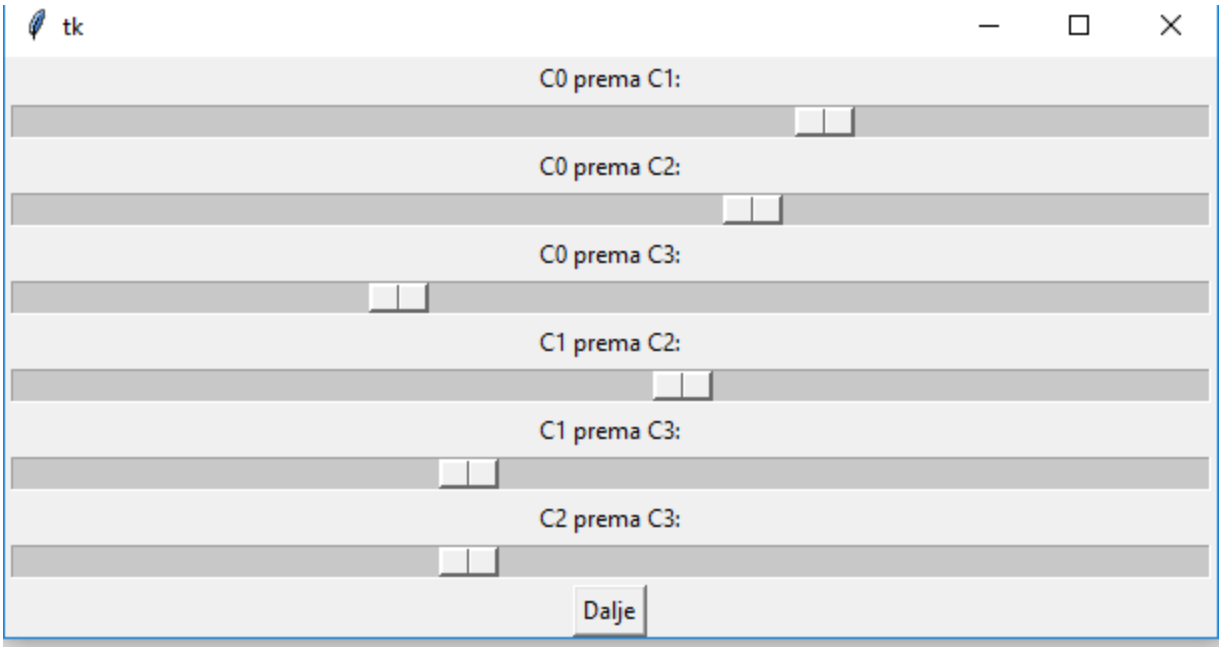
```
#biblioteka za rad sa matricama u pythonu
#najvise koriscena zbog inicijalizacije samih matrica
import numpy as np
#matematicka biblioteka, koristi se za koriscenje nekih kompleksnijih matematickih
operacija
import math as mt

# normalizacija matrice
# postupak normalizacije matrice je prvi postupak u electre metodi
# funkcija za normalizaciju matrice prihvata matricu koju treba da normalizuje
# a kao rezultat vraca normalizovanu matricu
def normalizovanjeMatrice(matrica):

#inicijalizacija nove matrice koja treba da bude iste visine i sirine kao prosledjena
matrice
novaMatrica = np.zeros((len(matrica), len(matrica[0])))

#normalizovanje matrice po formuli n(i,j) = x(i,j)/sqrt(suma(x(i,j)*x(i,j)))
    #gde je n(i,j) element normalizovane matrice
    # x(i,j) element prosledjene matrice
for i in range(0, len(matrica[0])):
        suma = 0
for j in range(0, len(matrica)):
            suma = suma + matrica[j][i] * matrica[j][i]

        suma = mt.sqrt(suma)
for k in range(0, len(matrica)):
            novaMatrica[k][i] = matrica[k][i] / suma

#kao rezultat se vraca nova matrica
return novaMatrica

# tezinska normalizacija
# normalizovana matrica svaki od svojih elemenata mnozi sa tezinskim koeficijentom za
```

```
dat kriterijum
# funkcija prihvata matricu i tezinske koeficijente
def tezinskoNormalizovanje(matrica, t):
    novaMatrica = np.zeros((len(matrica), len(matrica[0])))
for i in range(0, len(matrica[0])):
for k in range(0, len(matrica)):
            novaMatrica[k][i] = t[i] * matrica[k][i]

return novaMatrica


# odredjivanje skupova saglasnosti S i nesaglasnosti NS
# funkciji se prosledjuje matrica na osnovu koje se ordedjuju skupovi saglasnosti i
nesaglasnosti
#
def skupoviSaglasnosti(matrica):
    listaSaglasnosti = []
    listaNesaglasnosti = []

for p in range(0, len(matrica)):
for r in range(0, len(matrica)):
if p != r:
                redSaglasnih = []
                redNesaglasnih = []
for j in range(0, len(matrica[0])):
if matrica[p][j] >= matrica[r][j]:
                        redSaglasnih.append(j)
else:
                        redNesaglasnih.append(j)

                listaSaglasnosti.append(redSaglasnih)
                listaNesaglasnosti.append(redNesaglasnih)
return listaSaglasnosti,listaNesaglasnosti

#Pravljenje matrice saglasnosti
# na osnovu tezinem matrice i liste Saglasnosti pravi se matrica saglasnosti
def matricaSaglasnostiFunkcija(listaSaglasnosti, tezine, matrica):
    matricaSaglasnosti = np.zeros((len(matrica), len(matrica)))
    brojac = 0
for i in range(0,len(matrica)):
for j in range(0,len(matrica)):
            suma = 0
if i != j:
for s in range(0, len(listaSaglasnosti[brojac])):
                    suma= suma + tezine[listaSaglasnosti[brojac][s]]
                brojac=brojac+1
matricaSaglasnosti[i][j]=suma
return matricaSaglasnosti

#Pravljenje matrice nesaglasnosti
# na onsonuv liste nesaglasnosti, tezine i matrice prave se matrica nesaglasnosti
def matricaNesaglasnostiFunkcija(listaNesaglasnosti,tezine,matrica):
    matricaNesaglasnosti = np.zeros((len(matrica), len(matrica)))

    brojac = 0
for p in range(0, len(matrica)):
for r in range(0, len(matrica)):
            rez = 0

if p != r:
                maximalnog = -999
for s in range(0, len(listaNesaglasnosti[brojac])):
                    trenutni =abs(matrica[p][listaNesaglasnosti[brojac][s]]-
```

```
matrica[r][listaNesaglasnosti[brojac][s]])
if trenutni > maximalnog:
                        maximalnog = trenutni

                maks = -999
for k in range(0, len(matrica[0])):
if maks <abs(matrica[p][k] - matrica[r][k]):
                        maks = abs(matrica[p][k] - matrica[r][k])

                brojac = brojac + 1

if maks != 0:
                    rez = maximalnog/maks

            matricaNesaglasnosti[p][r]=rez

return matricaNesaglasnosti

#Pravljenje matrice saglasne dominacije
# kada se odredi matrica saglasnosti treba se na osnovu nje izracunati matrica
saglasne dominacije
# saglasna dominacija se racuna na osnovu matrice saglasnosti
def matricaSaglasneDominacijeFunkcija(matricaSaglasnosti):
    novaMatrica = np.zeros((len(matricaSaglasnosti), len(matricaSaglasnosti)))
    suma = 0
for i in range(0,len(matricaSaglasnosti)):
for j in range(0, len(matricaSaglasnosti)):
            suma = suma + matricaSaglasnosti[i][j]
    pis = suma/((len(matricaSaglasnosti)-1)*len(matricaSaglasnosti))

for i in range(0, len(matricaSaglasnosti)):
for j in range(0, len(matricaSaglasnosti)):
if i != j:
if matricaSaglasnosti[i][j] >= pis:
                    novaMatrica[i][j]=1
return novaMatrica

#Pravljenje matrice nesaglasne dominacije
# Matrica nesaglasne dominacije se racuna uz pomoc matrice nesaglasnosti
def matricaNesaglasneDominacijeFunkcija(matricaNesaglasnosti):
    novaMatrica = np.zeros((len(matricaNesaglasnosti), len(matricaNesaglasnosti)))
    suma = 0
for i in range(0,len(matricaNesaglasnosti)):
for j in range(0, len(matricaNesaglasnosti)):
            suma = suma + matricaNesaglasnosti[i][j]
    pins = suma/((len(matricaNesaglasnosti)-1)*len(matricaNesaglasnosti))

for i in range(0, len(matricaNesaglasnosti)):
for j in range(0, len(matricaNesaglasnosti)):
if i != j:
if matricaNesaglasnosti[i][j] <= pins:
                    novaMatrica[i][j]=1
return novaMatrica

#Pravljenje matrice agregatne dominacije
# matrica agregatne dominacije je u sustini logicko sabiranje dveju matrica
def agregatnaDominacija(c, d):
    res = np.zeros((len(c), len(c)))
for i in range(0, len(c)):
for j in range(0, len(c)):
if c[i][j] == d[i][j] == 1:
                res[i][j] = 1
return res
```

```
#metoda odluka racuna tri najoptimalnije odluke na osnovu matrice agregatne dominacije
def odluka(matrica):
    max = 0
max1 = 0
max2 = 0
red = 0
red1 = 0
red2 = 0
for i in range(len(matrica)):
        trenutan = 0
for j in range(len(matrica[0])):
if(matrica[i][j] != 0):
                trenutan = trenutan + 1

if trenutan > max:
            max = trenutan
            red = i
elif trenutan > max1:
            max1 = trenutan
            red1 = i
elif trenutan > max2:
            max2 = trenutan
            red2 = i

return [red+1, red1+1, red2+1]

def main(x, tezine):
    x = normalizovanjeMatrice(x)
# pozivanje funkcije za tezinsku normalizaciju matrice
x = tezinskoNormalizovanje(x, tezine)

# pozivanje funkcije za generisanje skupova saglasnosti
    # skupovi su skladisteni u liste jer su promenjivih duzina
listaSaglasnosti, listaNesaglasnosti = skupoviSaglasnosti(x)

# pozivanje funkcija za generisanje matrice saglasnosti
matricaSaglasnosti = matricaSaglasnostiFunkcija(listaSaglasnosti, tezine, x)
# pozivanje funkcije za generisanje matrice nesaglasnosti
matricaNesaglasnosti = matricaNesaglasnostiFunkcija(listaNesaglasnosti, tezine, x)

# pozivanje funkcije za konstruisanje matrice saglasne dominacije
matricaSaglasneDominacije = matricaSaglasneDominacijeFunkcija(matricaSaglasnosti)
# pozivanje funkcije za konstruisanje matrice neseglasne dominacije
matricaNesaglasneDominacije =
matricaNesaglasneDominacijeFunkcija(matricaNesaglasnosti)

#donosnje odluke na osnovu matrice koje se dobija kao prozivod funkcije agregatne
dominacije
return odluka(agregatnaDominacija(matricaSaglasneDominacije,
matricaNesaglasneDominacije))
```

AHP modul



Korisnik pmoću slajdera unosi odnose kriterijuma. Vrednosti za odnose se određuju na osnovu skale koja izgleda

Skala	Objašnjenje rangiranja
9	Apsolutno najznačajnije/najpoželjnije
8	Veoma snažno ka apsolutno najznačajnijem
7	Veoma snažno ka veoma značajnom/poželjnom
6	Snažno ka veoma snažnom
5	Snažnije više značajno/poželjno
4	Slabije ka više snažnijem
3	Slabije više značajno/poželjno
2	Podjednako ka slabije višem
1	Podjednako značajno/poželjno
0.50	Podjednako ka slabije manjem
0.33	Slabije manje značajno/poželjno
0.25	Slabije ka snažno manjem
0.20	Snažno manje značajno/poželjno
0.17	Snažno manje ka veoma snažno manjem
0.14	Izuzetno snažno manje značajno/poželjno
0.13	Veoma snažno ka apsolutno manjem
0.11	Apsolutno najmanje značajno/poželjno

Slajderi se kreću od 0 do 16 ali su mapirani tako da sadrže ove vrednosti. Mapirani su pomoću heš liste.Odnosno liste koja sadrži ključ i vrednost koju nosi taj ključ.

```
valueList = {0: 1/9, 1: 1/8, 2: 1/7, 3:1/6, 4:1/5, 5:1/4, 6:1/3, 7:1/2, 8:1, 9:2,
10:3, 11:4, 12:5, 13:6, 14:7, 15:8, 16:9}
```

Kada se prikupe vrednosti sa svih slajdera i mapiraju da sadrže vrednosti kao u skali gore popunjuje se matrica odnosa. (U matrici su zagradom napisane invertne vrednosti, pa tako (3.0) je zapravo 1/3 odnosno 0.33)

	A_1	A_2	A_3	A_4	A_5	A_6
A_1		(3.0)	3.0	(6.0)	(6.0)	2.0
A_2			4.0	(4.0)	(5.0)	4.0
A_3				(7.0)	(6.0)	(3.0)
A_4					1.0	6.0
A_5						6.0
A_6						

Pošto se poredi samo A1 u odnosu na A2 nema potrebe da se ponovo poredi i A2 u odnosu na A1. Ali matrica mora biti popunjena u celini. Pa tako se matrica popunjuje invertovanim vrednostima, odnosno ako je A1 prema A2 bilo 0.33 onda će A2 prema A1 biti 3.00.Vrednosti gde se poredi A1 i A1 se popunjavaju jedinicom.

	A_1	A_2	A_3	A_4	A_5	A_6
A_1	1.00	0.33	3.00	0.17	0.17	2.00
A_2	3.00	1.00	4.00	0.25	0.20	4.00
A_3	0.33	0.25	1.00	0.14	0.17	0.33
A_4	6.00	4.00	7.00	1.00	1.00	6.00
A_5	6.00	5.00	6.00	1.00	1.00	6.00
A_6	0.50	0.25	3.00	0.17	0.17	1.00

Ovako formatirana matrica se prosledjuje metodi za računanje težina.

```
defkorak1(matrica):
```

Prvi korak u računanju težina je računanje sume za svaku od kolona. Ovo radimo tako što smestimo sumu svake od kolona u odgovarajući element niza suma.

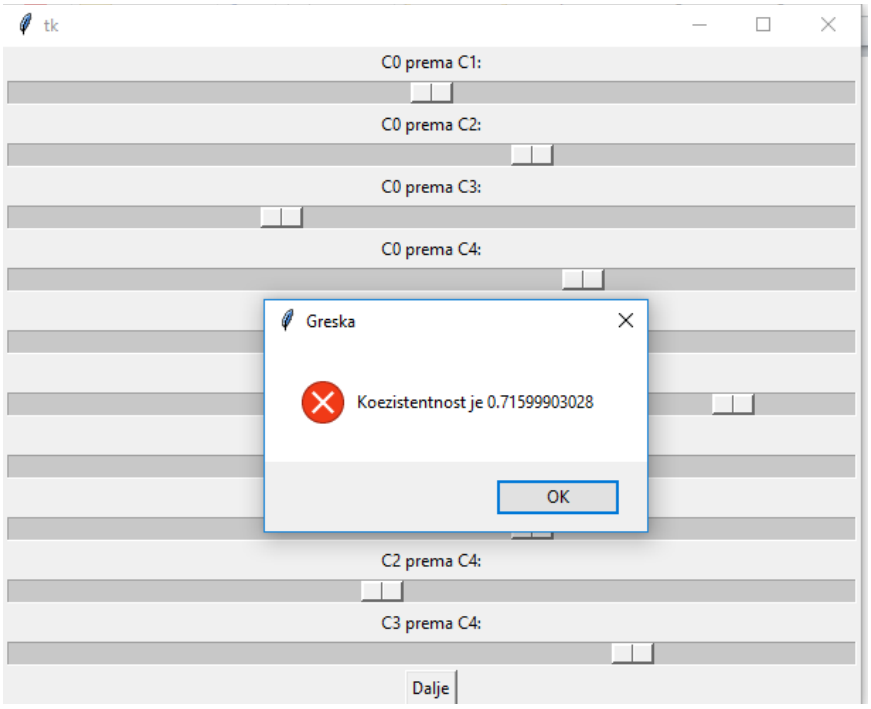
```
#inicijalizacija niz koji ce da cuva sume kolona
sume = np.zeros(len(matrica))
#racunanje sume za svaku kolonu
for i in range(0, len(matrica)):
for j in range(0, len(matrica)):
            sume[i] = sume[i] + matrica[j][i]
```

Sledeći korak je konstruisanje nove matrice. Odnosno matrica se takoreći normalizuje na način gde se svaki element matrice deli sa sumom matrice za tu kolonu.

```
#inicijalizacija nove matrice
matrica_dva = np.zeros((len(matrica), len(matrica)))

#svaki element nove matrice se racuna tako sto se dati element matrice
    #deli sa sumom za tu kolonu
for i in range(0, len(matrica)):
for j in range(0, len(matrica)):
            matrica_dva[j][i] = matrica[j][i]/sume[i]
```

Sume kolona za normalizovanu matricu se dele sa brojem kriterijuma i smeste se u niz zadužen za čuvanje težina, a koji je dužine koliki je i broj kriterijuma. Time se dobijaju konačne težine koje se koriste za računanje u ELECTREImetodi.

```
#inicijalizacija niza zaduzenog za skladistenje tezina
tezine = np.zeros(len(matrica))
#tezine se racunaju za svaku kolonu tako sto suma elemenata u kolonu nove matrice
    #podeli sa brojem kriterijuma
for i in range(0, len(matrica_dva)):
        sum = 0
for j in range(0, len(matrica_dva)):
            sum = sum + matrica_dva[i][j]
        tezine[i]=sum/len(matrica_dva)

#funkcija vraca nove tezine
return tezine
```



Da bi mogli da omogućimo proveru koezistentnosti, potrebno je implementirati i jednu metodu koja računa koezistentnost.Ovo se može reći da je kao procedura validacije unetih podataka.Koezistentnost se računa pomoću dobijenih težina i prosleđenih odnosa.

```
#funkcija za racunanje koezistentnosti tezina
def ci(m, t):
```

Prvi korak u računanju koezistentnosti je to da se ondosi prosledjeni pomnože sa težinama

```
    mat = np.zeros((len(t), len(t)))
for i in range(0, len(mat)):
for j in range(0, len(mat)):
            mat[j][i] = m[j][i]*t[i]
```

Sledeći korak je da se izračuna suma svakog reda u matrici.

```
niz = np.zeros(len(t))
for i in range(0, len(mat)):
        sum = 0
for j in range(0, len(mat)):
            sum = sum + mat[i][j]
        niz[i] = sum
```

Potom se svaka od suma podeli sa odgovarajućom težinom.

```
    niz0 = np.zeros(len(t))
for i in range(0, len(t)):
        niz0[i] = niz[i]/t[i]
```

Sledeći korak je da se izračuna srednja vrednost tih suma.

```
sum = 0
for i in range(0, len(niz0)):
        sum = sum + niz0[i]
    lMax = sum/len(niz0)
```

$$CI = \frac{\lambda_{max} - n}{n - 1};$$

C indeks CI se računa po ovoj formuli. Gde je λ_{max} srednja vrednost koju smo izračunali u predhodnom koraku, a n broj težina.

```
cIndex = (lMax-len(niz0))/(len(niz0)-1)
```

Koezistentnost se onda računa sledećom formulom. Gde se RI uzima iz tabele

$$CR = \frac{CI}{RI}$$

Veličina matrice	1	2	3	4	5	6	7	8	9	10
RI	0	0	0.58	0.9	1.12	1.24	1.32	1.41	1.45	1.49

```
    ri = {1: 0, 2: 0, 3: 0.58, 4: 0.9, 5: 1.12, 6: 1.24, 7: 1.32, 8: 1.41, 9: 1.45,
10: 1.49, 11: 1.51}
    cRatio = cIndex/ri[len(t)]
return cRatio
```

Ovako izračunata koezistentnost se vraća grafičkom delu programa i tu se proverava da li je manja od 0.1. Ako je manja program može da nastavi sa radom, ako nije podaci koji su uneti nisu koezistentni.

Kod ovako izgleda

```
#funckija za racunanje tezina pomocu ahp metode
#funkciji se prosledjuje matrica subjektivnih odnosa kriterijuma
#odnosno, korisnik uporedi svaki kriterijum sa svakim i time daje svoje subjektivno
#misljenje o odnosu izmedju dva kriterijuma.
def korak1(matrica):
#inicijalizacija niz koji ce da cuva sume kolona
sume = np.zeros(len(matrica))

#racunanje sume za svaku kolonu
for i in range(0, len(matrica)):
for j in range(0, len(matrica)):
            sume[i] = sume[i] + matrica[j][i]

#inicijalizacija nove matrice
matrica_dva = np.zeros((len(matrica), len(matrica)))

#svaki element nove matrice se racuna tako sto se dati element matrice
    #deli sa sumom za tu kolonu
for i in range(0, len(matrica)):
for j in range(0, len(matrica)):
            matrica_dva[j][i] = matrica[j][i]/sume[i]

#inicijalizacija niza zaduzenog za skladistenje tezina
tezine = np.zeros(len(matrica))

#tezine se racunaju za svaku kolonu tako sto suma elemenata u kolonu nove matrice
    #podeli sa brojem kriterijuma
for i in range(0, len(matrica_dva)):
        sum = 0
for j in range(0, len(matrica_dva)):
            sum = sum + matrica_dva[i][j]
        tezine[i]=sum/len(matrica_dva)

return tezine

#funkcija za racunanje koezistentnosti tezina
def ci(m, t):
    mat = np.zeros((len(t), len(t)))
for i in range(0, len(mat)):
for j in range(0, len(mat)):
            mat[j][i] = m[j][i]*t[i]

    niz = np.zeros(len(t))
for i in range(0, len(mat)):
        sum = 0
for j in range(0, len(mat)):
            sum = sum + mat[i][j]
        niz[i] = sum

    niz0 = np.zeros(len(t))
for i in range(0, len(t)):
        niz0[i] = niz[i]/t[i]

    sum = 0
for i in range(0, len(niz0)):
        sum = sum + niz0[i]
    lMax = sum/len(niz0)
    cIndex = (lMax-len(niz0))/(len(niz0)-1)
```

```
    ri = {1: 0, 2: 0, 3: 0.58, 4: 0.9, 5: 1.12, 6: 1.24, 7: 1.32, 8: 1.41, 9: 1.45,
10: 1.49, 11: 1.51}
    cRatio = cIndex/ri[len(t)]
return cRatio
```