Univerzitet u Beogradu,

Elektrotehnički fakultet

Složenost algoritama i odbrane metode optimizacije

Seminarski rad

Simplex Metoda

Autori:

Filip Babović 2018/0147
Mina Kaljević 2018/0197
Valentina Ikodinović 2018/0199
Isidora Ćirić 2018/0457

Maj 2020.

Sadržaj

1. Uvod

Tema ovog rada jeste Simplex Metoda. Pre svega ćemo se ukratko upoznati sa temom linearnog programiranja, jer je Simplex metoda jedna od procedura linearnog programiranja kojom se dolazi do optimalnog rešenja. Zatim ćemo preći preko nekoliko zadataka primenom Simplex metode i za kraj, pričaćemo o softverima koji u sebi imaju implementiranu Simplex metodu i to ćemo ilustrovati sa nekoliko primera.

1.1. **Uvod u linearno programiranje**

Linearno programiranje je matematička metodologija za modeliranje i rešavanje problema nalaženja maksimuma ili minimuma linearne funkcije, pod uslovima iskazanim kao linearne jednačine ili nejednačine. Algoritmi koji rešavaju ove probleme (pre svega Simplex algoritam) zahtevaju da problem ima specijalan, odredjen oblik, da bi mogao biti rešen. Postoje četiri različita oblika problema linearnog programiranja:

- opšti
- simetrični
- standardni
- kanonski (u suštini, ovo je interni oblik Simplex metode)

Linearni program se može zapisati u kanonskom obliku na sledeći način:

$$\max c^T x$$

pri ograničenjima $Ax \leq b$, $x \geq 0$, pri čemu su:

- x-vektor promenljivih koje treba odrediti,
- b i c-koeficijenti (poznati),
- A-matrica (poznata).

1.2. Istorijat i primena linearnog programiranja

Sovjetski matematičar Leonid Kantorovič 1939. godine je formulisao problem linearnog programiranja. Prvi modeli su korišćeni u drvnoj proizvodnji. Kantorović je tokom Drugog svetskog rata radio za vojsku na optimizaciji vojnih operacija. Metod rešavanja problema nije bio poznat za javnost sve do 1947. kada je Džordž B. Dancig objavio Simplex algoritam.

Ovakvim matematičkim modelima se mogu predstaviti problemi alokacije resursa, planiranja kretanja vozila, razni problemi proizvodnje u industriji,

problemi u ekonomiji itd. Primenom linearnog programiranja se mogu rešavati svi praktični problemi koji mogu da se iskažu matematičkim modelom linearnog programiranja.

2. Uvod u Simplex Metodu

U ovom delu opisaćemo istorijat Simplex metode, kao i definisati neke opšte pojmove i samu Simplex metodu. Dok ćemo zadatke raditi u sledećem poglavlju.

2.1. Simplex Metod

Simplex metod, koji je 1947. godine predložio J.B. Dancig polazi od ekstremne tačke i u svakoj sledećoj iteraciji prelazi na obližnju ekstremnu tačku sa nižom vrednošću funkcije cilja, sve dok ne dodje do optimalnog rešenja.

Neka je (P) problem linearnog progrmiranja dat u kanoničkom obliku, tj. u obliku

(1) $\min F=c_1x_1 + c_2x_2 + \ldots + c_nx_n,$

$a_{11}x_1 + a_{12}x_2+ \ldots + a_{1n}x_n = b_1,$

$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2,$

$\vdots$

$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_n,$

$x_1,x_2, \ldots x_n \geq 0,$

ili u matričnoj notaciji

(1') $\min F=c^T \cdot x,$

$A \cdot x=b,$

$x \geq 0$

gde je $A= \|a_{ij}\|_{m,n}$, $x^T= \|x_1x_2...x_n\|$, $b^T= \|b_1b_2...b_m\|$, $c^T= \|c_1c_2...c_n\|$. Da bi problem (P) imao smisla, pretpostavimo da je rangA=rang $\|Ab\|$. Pored toga uzimamo da je rangA = m < n. Stavimo sada $x_0= -F$, a dobijeni uslov:

$x_0 + c_1x_1 + c_2x_2 + \ldots + c_nx_n = 0$, priključujemo ostalim uslovima. Tada primenom Gausovog algoritma na uslove (1) izražene jednakostima, dobijamo:

(2) $x_0 + \ldots + \hat{a}_{0,m+1}x_{m+1} + \ldots + \hat{a}_{0n}x_n = \hat{b}_0,$

$x_1 + \ldots + \hat{a}_{1,m+1}x_{m+1} + \ldots + \hat{a}_{1n}x_n = \hat{b}_1,$

$\vdots$

$x_m + \hat{a}_{m,m+1}x_{m+1} + \ldots + \hat{a}_{mn}x_n = \hat{b}_m.$

Odavde, stavljamo $x_{m+1} = x_{m+2} = \ldots = x_n = 0$, dobijamo $x_0 = \hat{b}_0$, $x_1 = \hat{b}_1$, . . . $x_m = \hat{b}_m$, tj. dobili smo jedno rešenje sistema (2). Ovakva rešenja predstavljaju tzv.

bazična rešenja. Promenljive $x_1, x_2, \ldots, x_m$ nazivaju se i *bazične promenljive,* dok su ostale $x_{m+1}, x_{m+2}, \ldots, x_n$ (izjednačene sa nulom) *nebazične promenljive*. Ukoliko se ograničimo samo na promenljive $x_1, x_2, \ldots, x_n$ (bez x_0) koje odgovaraju dopustivom skupu i ukoliko su sve one nenegative, tada one daju jedno *dopustivo rešenje.* Uzimo najlpre da je za pomenuti podsistem (2) dobijeno jedno bazično dopustivo rešenje. Geometrijska interpretacija navedenog je sledeća: tačka$(x_1, x_2,..., x_n)$ koja odgovara bazičnom dopustivom rešenju predstavlja jedno teme konveksnog poliedra koji odgovara dopustivom skupu; funkcija cilja u toj tački uzima vrednost $-x_0$. Osnovna ideja simpleks algoritma sastoji se sada u prelasku iz tog temena poliedra u neko obližnje, u kojem se vrednosti funkcije cilja smanjuje. Algebarski kriterijum ovog prelaska sadržani su u sledećoj analizi:

1. Ako je $\hat{a}_{0j} \geq 0$ za svako j ($j \in \{m + 1, m + 2,... n\}$), tada je x_0 maksimalno ($-x_0$ minimalno) ako je $x_{m+1} = x_{m+2} = \ldots = x_n = 0$, a u tom slučaju je $x_1 = \hat{b}_1, \ldots x_m = \hat{b}_m$, što znači da je dobijeno bazično dopustivo rešenje i rešenje optimizacijonog problema
2. Ako je za neko j ($j \in \{m + 1, m + 2,... n\}$) $\hat{a}_{0j} < 0$ i ako je za to isto j $\hat{a}_{ij} \leq 0$ za svako i ($i \in \{1, 2,...,m\}$), tada se stiču uslovi da x_j postane neograničeno, a takodje i vrednost funkcije cilja. Naime, ako se sada uzme da je $x_j = M$ ($M>0$), a $x_{m+1} = x_{m+2} = \ldots = x_{j-1} = x_{j+1} = ... = x_n = 0$, tada je $x_1 = \hat{b}_1 - \hat{a}_{1j}M,..., x_m = \hat{b}_m - \hat{a}_{mj}M$, te dobijamo dopustivo rečenje za koje je x_0 ($x_0 = \hat{b}_0 - \hat{a}_{0j}M$) proizvoljno veliko kada M teži $+\infty$, što znači da optimalno rešenje u ovom slučaju nije konačno.
3. Ako je za neko j ($j \in \{m + 1, m + 2,... n\}$) $\hat{a}_{0j} < 0$ postoji i ($i \in \{1, 2,...,m\}$) tako da je $\hat{a}_{ij} > 0$, tada se x_0 može povećati, ako se uzme da x_j nije obavezno jednako 0, tj. da promenljiva x_j postaje bazična, a samim tim da se i jedna od bazičnih promenljivih prebacuje u nebazične. Ovo prebacivanje se može realizovati *stožernom transformacijom* u Gausovom algoritmu. U tom cilju neka je $\hat{a}_{sj}$ ($s \in \{1, 2,...,m\}$) *stožerni element*. Tada slobodni članovi iz (2) postaju:

$$\hat{b}'_i = \begin{cases} \dfrac{\tilde{b}_i}{\hat{a}_{sj}}, & i = s \\ \tilde{b}_i - \dfrac{\tilde{b}_i}{\hat{a}_{sj}} * \hat{a}_{ij,} & i \neq s \end{cases}$$

Iz uslova da je $\hat{b}'_I \geq 0$ za svako i ($i \in \{1, 2,...,m\}$) dobijamo najpre da je $\hat{a}_{sj} > 0$, a zatim $\tilde{b}_i - \dfrac{\tilde{b}_i}{\hat{a}_{sj}} * \hat{a}_{ij,} \geq 0$ za svako $i \neq s$. Da bi ovaj uslov uvek važio, dovoljno je uzeti s takvo da je :

$$\frac{\hat{b}_s}{\hat{a}_{sj}} = \min\left\{\frac{\hat{b}_i}{\hat{a}_{sj}} \mid i \in \{1, 2, \ldots, m\}\ \hat{a}_{ij} > 0\right.$$

Dakle, x_0 ne opada tako da se vrednost funkcije cilja skoro po pravilu poboljčava, dok za $\hat{b}_s = 0$, vrednost funkcije cilja se ne menja.

Opisani algoritam se sigurno završava u konačnom broju koraka. Razlog je što je broj bazičnih rešenja sistema kao što je na primer sistem (2) konačan, teorijski najvise $\binom{n}{m}$. Mnogi eksperimentalni rezultati ukazuju da simplex algoritam uglavnom ne treba više od 2(m + n) iteracija.

Sada ćemo sva gornja razmatranja sumirati u formi simpleks tablica u okviru kojih ćemo dati i formalan opis algoritma. Tablica, u oznaci T_k, koja se iz nulte tablice, T_0, dobija primeno k transformacija (preciziranih algoritmom) ima sledeći izgled:

$-d^{(k)}$	$c_1^{(k)}$	$c_2^{(k)}$	.	.	.	$c_n^{(k)}$
$b_1^{(k)}$	$a_{11}^{(k)}$	$a_{12}^{(k)}$	.	.	.	$a_{1n}^{(k)}$
$b_2^{(k)}$	$a_{21}^{(k)}$	$a_{22}^{(k)}$	.	.	.	$a_{2n}^{(k)}$
⋮	⋮	⋮				⋮
$b_m^{(k)}$	$a_{m1}^{(k)}$	$a_{m2}^{(k)}$	.	.	.	$a_{mn}^{(k)}$

ili, u matričnoj notaciji:

$-d^{(k)}$	$c^{(k)}$
$b^{(k)}$	$A^{(k)}$

Da bi ovakva tablica bila stvarno simpleks tablica u njoj mora da postoji m različitih takozvanih *bazičnih kolona*, tj. kolona takvih da je $c_j^{(k)} = 0$ i $a_{ij}^{(k)} = 0$ za svako *i* ($i \in \{1, 2, \ldots, m\}$) sem jednog, recimo i = s, za koje je $a_{sj}^{(k)} = 1$; pored toga zahteva se i da je $b_i^{(k)} \geq 0$ za svako *i* ($i \in \{1, 2, \ldots, m\}$). Za k ≥ 1, to je obezbedjeno algoritmom, a za k = 0 postupa se na sledeći način: najpre se formira tablica T:

0	c
b	A

i ukoliko je ona sama po sebi simpleks tablica tada je 0-ta simpleks tablica; u protivnom, elementarnim transformacijama vrsta ona se „na neki pogodan način“ prevodi u traženu formu.

2.2. Simplex algoritam

Simplex algoritam se sada sastoji od sledećih koraka:

Korak 1 (inicijalizacija): Formirati tablicu T_0 i staviti k=0.

Korak 2 (provera optimalnosti rešenja): Ako je $c_j^{(k)} \geq 0$ za svako j, tada "stop" (nadjeno je optimalno rešenje: $d^{(k)}$ je minimum, bazične promenljive uzimaju vrednosti odgovarajućih elemenata $b_i^{(k)}$ iz prve kolone, a nebazične 0)

Korak 3 (provera konačnosti rešenja): Ako je za neko j $c_j^{(k)} < 0$ i ako je $a_{ij}^{(k)} \leq 0$ za sve elemente j-te kolone tada "stop" (funkcija cilja je neograničena, jednaka $-\infty$)

Korak 4 (modifikacija k-te tabele): Izabrati bilo koje j takvo da je $c_j^{(k)} < 0$; zatim naći takvo s da je

$$\frac{b_s^{(k)}}{a_{sj}^{(k)}} = \min\left\{\frac{b_i^{(k)}}{a_{sj}^{(k)}} \mid i \in \{1, 2, \dots, m\}\ a_{ij}^{(k)} > 0\right.$$

Polazeći od elemenata $a_{sj}^{(k)}$ kao stožernog izvršiti jednu stožernu transformaciju tabele T_k (u novu simpleks tabelu). Najzad zameniti k sa k+1 i preći na korak 2.

Postupak opisan Simplex algoritmom u smislu prelaska od početne tablice u završnu tablicu može se matrično opisati sa sledece dve tablice:

0	c_B^T	c_N^T
b	B	N

$-c_B^T B^{-1} b$	0	$c_N^T - c_B^T B^{-1} b$
$B^{-1} b$	I	$B^{-1} N$

pri čemu su blokovi B, N, c_B^T, c_N^T dobijeni iz tablice:

0	c^T
b	A

tako što je u matrici A izdvojeno m bazičnih kolona koje u završnoj tablici odgovaraju bazičnim promenljivima, zatim su te kolone stavljene u blok B, a preostale u blok N; ovim je ujedno indukovana i odgovarajuća deoba vektora c^T. Primetimo da je minimalna vrednost funkcije cilja sada data sa $c_B^T B^{-1} b$, a da vektor x_B bazičnih promenljivih važi $x_B = B^{-1} b$, dok za vektor x_N nebazičnih promenljivih imamo $x_N = 0$. Pored toga je $c_B^T - c_N^T B^{-1} N \geq 0$, jer je to uslov optimalnosti rešenja.

U prethodnim razmatranjima, problem linearnog programiranja, inače dat u obliku (1), je nekim pogodnim transformacijama preveden u oblik (2) iz kog je dobijeno jedno bazično dopustivo rešenje. Samim tim bilo je mogućno formirati

0-tu simpleks tablicu, a zatim i rešiti problem (1). Međutim, u opštem slučaju, prevođenje problema (1) u oblik (2) ne daje obavezno bazično dopustivo rešenje. Opisaćemo sada jedan postupak kojim se obezbeđuje prevođenje problema (P) u traženi oblik. Jedna pretpostavka koju je sada zgodno uzeti je da su slobodni članovi $b_1,...,b_m$ iz (1) nenegativni, dok se dodatne pretpostavke u vezi sa saglasnosti sistema kao i ranga matrice mogu izostaviti. Problemu (P) pridružićemo problem ($\hat{P}$) dat sa

(3) $\min g = x_{n+1} + x_{n+2} + ... + x_{n+m},$

$a_{11}x_1 + a_{12}x_2 + ...+ a_{1n}x_n + x_{n+1} = b_1,$

.

.

.

$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n + x_{n+m} = b_m,$

$x_1, x_2, ..., x_{m+n} \geq 0.$

Promenljive $x_{n+1}, x_{n+2}, ..., x_{n+m}$ iz (3) se nazivaju *veštačke promenljive.* Između problema (P) i ($\hat{P}$) postoji sledeća veza: ako je dopustivi skup problema (P) neprazan, tada problem ($\hat{P}$) ima optimalno rešenje i to jednako 0; nasuprot tome, ako je dopustivi skup problema (P) prazan, tada optimalno rešenje problema ($\hat{P}$) mora biti pozitivno. Problem ($\hat{P}$) uvek ima bar jedno dopustivo rešenje kod koga je $x_1 = x_2 = ...= x_n = 0$, a $x_{n+1} = b_1$, $x_{n+2} = b_2$,$x_{n+m} = b_m$, a ovo rešenje je istovremeno i bazično rešenje problema ($\hat{P}$). Ove činjenice se mogu iskoristiti za rešavanje problema (P) tzv. metodom *dvofazne modifikacije simpleks algoritma.*

Faza 1 (pomoćni problem)

Korak 1 (inicijalizacija pomoćnog problema): Formirati iz problema (P) datog u obliku (1) kod koga je $b_i \geq$ za svako i, problem ($\hat{P}$)

Korak 2 (rešavanje pomoćnog problema): Primenom simpleks algoritma, rešiti problem ($\hat{P}$). Ako je optimalno rešenje pozitivno (min g > 0), tada **stop**. U protivnom, ako je optimalno rešenje jednako 0, preći na fazu 2.

Faza 2 (prelazak na osnovi problem)

Korak 1 (redukcija poslednje tablice iz faze 1): Ukloniti sve one nebazične kolone iz poslednje simpleks tablice problema($\hat{P}$) koje odgovaraju veštačkim promenljivim

Korak 2 (inicijalizacija početne tablice jednog prelaznog niza problema): U tablici koja je dobijena iz prethodnog koraka, nultu vrstu zameniti sa k, gde je k broj preostalih veštačkih promenljivih.

Ova tablica nije simpleks tablica jedino zbog toga što su neke bazične kolone postale nebazične posle zamene nula vrste. Međutim, elementarnim transformacijama vrsta one se jednostavno prevode u bazične kolone; time opet dobijamo simpleks tablicu. Primetimo da su sada sve komponente bazičnog dopustivog rešenja koje odgovaraju veštačkim promenljivim jednake 0.

Korak 3 (eliminacija veštačkih promenljivih): Ako nema kolona koje odgovaraju veštačkim promenljivim, preći na korak 4. U protivnom, naći kolonu, recimo p-tu, koja odgovara promenljivoj x_p ($p > n$). Neka je upravo element na poziciji (i,p) jednak 1 (ostali elemnti p-te kolone su jednaki 0). Ukoliko su svi preostali elementi i-te vrste jednai 0, izbaciti tu vrstu kao suvišnu. Naime, ova vrsta u problemu (P) indukuje nula vrstu koja je očigledno suvišna. U protivnom, neka je neki element i-te vrste recimo na poziciji j ($j \leq n$) razlicit od 0. Uzmimo sada da je taj element (i,j) stožerni, i izvršimo stožernu transformaciju. Tom prilikom se elementi nulte kolone ne menjaju, jer je element na nultoj poziciji i-te vrste jednak 0. Ovim je dobijena nova simpleks tablica, za koju je značajno da je sada p-ta kolona nebazična, te se stoga, pošto odgovara veštačkoj promenljivoj, može izostaviti. Dakle sada je k smanjeno za 1. Vratiti se na početak koraka 3.

Korak 4 (kraj druge faze): Najzad je dobijena nutla simpleks tablica problema (P), čime je omogućena primena simpleks algoritma.

3. Zadaci Simplex metode

Sada ćemo objasniti metodu kroz nekoliko zadataka.

3.1. Zadatak 1

Preduzeće proizvodi sok i džem od divljih kupina i njihova prdaja se odvija pod sledećim uslovima.

U procesu proizvodnje se angažuju radnici čiji ukupan broj radnih sati ne sme preći 900. Za proizvodnju soka neophodno je 4 sata angažovanja ovih radnika, a za proizvodnju džema 2 sata. U procesu proizvodnje koristi se i šećer, tako da po jednoj ambalaži za sok odnosno džem (flaši, odnosno tegli) treba po 2, odnosno 4 kilograma šećera. Najviše se može nabaviti 1 tona šecera. Prema ugovoru sa potencijalnim kupcima, kupci su spremni da kupe najviše 300 ambalaža ukupno

bez obzira na vrstu. Profit po jedinici ambalaže soka i džema iznosi 10 i 5 eura respektivno. Potrebno je odrediti optimalan program izrade soka i džema ako je cilj maksimalni ukupni profit.

Napomena: Šećer se nabavlja u pakovanjima od po 1 kilogram.

Rešenje:

- Pažljivo čitamo tekst zadatka i izvlačimo relacije.

U procesu proizvodnje se angažuju radnici čiji ukupan broj radnih sati ne sme preći 900. Za proizvodnju soka neophodno je 4 sata angažovanja ovih radnika, a za proizvodnju džema 2 sata.

$$\Rightarrow 4*x_1+2*x_2 \leq 900$$

...treba po 2 odnosno 4 kilograma šećera. Najviše se može nabaviti 1 tona šecera. (1 tona=1000 kilograma)

$$\Rightarrow 2*x_1+4*x_2 \leq 1000$$

kupci su spremni da kupe najviše 300 ambalaža ukupno bez obzira na vrstu.

$$\Rightarrow x_1+x_2 \leq 300$$

Profit po jedinici ambalaže soka i džema iznosi 10 i 5 evra respektivno -> u ovom delu se krije naša funkcija cilja. Označimo je sa z.

$$\Rightarrow z= 10*x_1+5*x_2$$

- Sledeći korak je ispisivanje knonskog oblika ovog modela tako što od nejednačina pravimo jednačine. To radimo uvođenjem po jedne promenljive (x_3, x_4 , x_5) u svakoj releciji tzv. izravnjavajuće/izjednačujuće promenljive. Podrazumevamo da su x_3 , x_4 , x_5 nenegativne veličine.

$$\begin{array}{rcrcrcrcrcr} 4*x_1 & + & 2*x_2 & + & x_3 & & & & & & =900 \\ 2*x_1 & + & 4*x_2 & + & & + & x_4 & & & & =1000 \\ x_1 & + & x_2 & + & & & & + & x_5 & & =300 \end{array}$$

pa funkcija cilja izgleda ovako:

$$\Rightarrow z_{max}= 10*x_1 + 5*x_2 + 0*x_3+ 0*x_4+ 0*x_5$$

- Sada formiramo Simpleks tabelu. Imamo kolene koeficijenata (kolona C), bazna kolona (kolona B) i kolona slobodnih članova (kolona x_0). Pravimo i kolone za sve promenljive (i stvarne i dopunske). Posto imamo 3 jednačine pravimo 3 reda plus još jedan dopunski. U zaglavlju upisujemo koeficijente funkcije cilja, u baznu kolonu izravnjavajuće promenljive, a njihove koeficijente u kolonu C i kolonu slobodnih članova. Ostaje još da popunimo tabelu koeficijentima iz ovog sistema jednačina. Primetimo da u desnom delu tabele imamo jediničnu matricu, tako uvek možemo izvršiti proveru.

			X_1	X_2	X_3	X_4	X_5
C	B	X_0	10	5	0	0	0
0	X_3	900	4	2	1	0	0
0	X_4	1000	2	4	0	1	0
0	X_5	300	1	1	0	0	1
z-c							

Ostatak popunjavamo sledećim algoritmom: počev od kolone x_0, svaki član kolone množimo odgovarajućim koeficijentom iz kolone C, a zatim saberemo sve proizvode po koloni i od tog zbira oduzmemo odgovarajući koeficijent funkcije cilja.

Poslednji član kolone X_0 dobijamo kao:
0*900 + 0*1000 + 0*300 = 0
Poslednji član kolone X_1 dobijamo kao:
0*4 + 0*2 + 0*1 - 10 = -10
Poslednji član kolone X_2 dobijamo kao:
0*2 + 0*4 + 0*1 - 5 = -5

Analogno dobijamo i poslednje članove ostalih kolona, pa tabela sada izgleda ovako:

			X_1	X_2	X_3	X_4	X_5
C	B	X_0	10	5	0	0	0
0			2	2	1	0	0
0				4	0	1	0
0			1	1	0	0	1
z-c		**0**	**-10**	**-5**	**0**	**0**	**0**

- Saledeći korak je traženje **najmanjeg broja u poslednjem redu** (to je pravilo kod određivanja maksimuma) i taj broj nam određuje koju kolonu uzimamo kao vodeću.U ovom slučaju traženi broj je -10, te je vodeća prva kolona.

- Kada smo odredili vodeću kolonu sad bi trebalo sve elemente vodeće kolone podeliti sa elementima kolone x_0 (ishod računa je upisan sa desne strane tabele), pa da vidimo gde je najmanji pozitivan količnik. Time određujemo vodeći red. U preseku vodeće kolone i reda nalazi se **vodeći ili pivot element**. U našem slučaju:

900 / 4 = 225
1000 / 2 = 500
300 / 1 = 300

225 je najmanji pa je prvi red je vodeći, a 4

			X_1	X_2	X_3	X_4	X_5	
C	B	X_0	10	5	0	0	0	
0	X_3	900	**4**	2	1	0	0	***225***
0	X_4	1000	2	4	0	1	0	*500*
0	X_5	300	1	1	0	0	1	*300*
z-c		0	**-10**	-5	0	0	0	

- Od postojeće formiramo drugu simpleks tabelu. Kako je pivot element broj različit od 1 (u našem slučaju 4), sve elemente vodećeg reda podelimo tim projem (brojem 4). Nakon toga pivot red množimo odgovarajućim koeficijentima i dodajemo ostalim redovima tako da u pivot koloni, ispod pivot elementa (koji je postao 1) imamo sve 0.

			X_1	X_2	X_3	X_4	X_5	
C	B	X_0	10	5	0	0	0	
0	X_3	~~900~~ 225	~~4~~ 1	~~2~~ 1/2	~~1~~ 1/4	~~0~~ 0	~~0~~ 0	-2
0	X_4	1000	2	4	0	1	0	
0	X_5	300	1	1	0	0	1	-1
z-c		0	**-10**	-5	0	0	0	

Kako je pivot kolona ona u kojoj je x_1, onda će promenljiva x_1 da pređe u bazu umesto x_3 i sa sobom povlači i svoj koeficijent 10. Popunjavanje poslednjeg reda vršimo na već poznati način. Novonastala tabela izgleda ovako:

			X_1	X_2	X_3	X_4	X_5
C	B	X_0	10	5	0	0	0
10	**X_1**	225	1	1/2	1/4	0	0
0	X_4	550	0	3	-1/2	1	0
0	X_5	75	0	1/2	-1/4	0	1
z-c		2250	0	0	5/2	0	0

Posmatramo samo poslednji red i primećujemo da, ne računajući trenutnu vrednost funkcije cilja (mesto gde je broj 2250), nema više negativnih vrednosti. Međutim u koloni x_2 se nalazi 0, a promenljiva x_2 nije u baznoj koloni. Ovo ukazuje na to da osim ovog postoji i jos jedno rešenje. Dakle, prvo optimalno rešenje je:

$z_{max}= 2250 \quad x_1 =225$

- Tražimo i drugo optimalmno rešenje tako sto x_2 ulazi u bazu i ponavljamo postupak. Pivot kolona je x_2, pa tražimo pivot red tako sto ćemo svaki od koeficijenata iz kolone x_0 da delimo sa odgovarajućim koeficijentom iz x_2 i tražimo najmanji količnik. Vidimo da je 150 najmanji među njima pa označimo žutom pivot red, a pivot element je ½.

			X_1	X_2	X_3	X_4	X_5	
C	B	X_0	10	5	0	0	0	
10	**X_1**	225	1	1/2	1/4	0	0	150
0	X_4	550	0	3	-1/2	1	0	183,3
5	X_5	75	0	**1/2**	-1/4	0	1	150
z-c		2250	0	0	5/2	0		

Kako je pivot element ½ pomnožićemo ceo red sa 2 kako vodeći elemet postao 1, a zatim ceo red množimo odgovarajućim koeficijentima i dodajemo prvom odnosno drugom redu. Imaćemo sledeću tabelu:

			X_1	X_2	X_3	X_4	X_5	
C	B	X_0	10	5	0	0	0	-1/2
10	**X_1**	225	1	1/2	1/4	0	0	
0	X_4	550	0	3	-1/2	1	0	-3
5	X_2	150	0	**1**	-1/2	0	2	
z-c		2250	0	0	5/2	0		

Nakon ovog sređivanja treba popuniti I poslednji red na već objašnjen način. Tabela sada izgleda ovako:

			X_1	X_2	X_3	X_4	X_5
C	B	X_0	10	5	0	0	0
10	X_1	150	1	0	1/2	0	-1
0	X_4	100	0	0	1/2	1	-6
5	X_2	150	0	1	-1/2	0	2
z-c		2250	0	0	5/2	0	0

Primećujemo da sada u poslednjem redu nema negativnih vrednosti, a promenljive i x_1 i x_2 već nalaze u bazi, pa naša funkcija ima vrednost:

$z = 2250$ $x_1 = 150$

$x_2 = 150$

To bi bilo drugo optimalno rešenje.

Dakle, najveći profit iznosi 2250 eura I da bi se to dostiglo preduzeće ima 2 mogućnosti:

1. da proizvede 225 ambalaža soka od divlje kupine ili
2. 150 ambalaža soka i 150 ambalaža džema od divlje kupine.

3.2. Zadatak 2

Preduzeće proizvodi tastature i miševe za računare i njihova prodaja se odvija pod sledećim uslovima.

U procesu proizvodnje se angažuju radnici čiji ukupan broj radnih sati ne sme preći 130. Za proizvodnju tastature neophodno je 3 sata angažovanja ovih radnika, a za proizvodnju miševa 4 sata. U procesu proizvodnje koristi se crna farba, tako da je za tastaturu neophodno jedno pakovanje ove farbe, a za miš 3. Najviše se može nabaviti 90 pakovanja. Prema ugovoru sa potencijalnim kupcima, kupci su spremni da kupe najviše 60 proizvoda ukupno, bez obzira na vrstu. Profit po jednoj tastaturi i mišu iznosi 10 i 7 eura respektivno. Potrebno je odrediti optimalan program izrade tastature i miševe za računare ako je cilj maksimalni ukupni profit.

Rešenje:

- Pažljivo čitamo tekst zadatka i izvlačimo relacije. Neka je x_1 količina tastatura, a x_2 količina miševa. Dobijamo sledeće relacije:

$3*x_1 + 4*x_2 \leq 130$
$1*x_1 + 3*x_2 \leq 90$
$1*x_1 + 1*x_2 \leq 60$
$z = 10*x_1 + 7*x_2$

- Sledeći korak je ispisivanje kanonskog oblika ovog modela tako što od nejednačina pravimo jednačine. To radimo uvođenjem po jedne promenljive (x_3, x_4 , x_5) u svakoj releciji tzv. izravnjavajuće/izjednačujuće promenljive. Podrazumevamo da su x_3 , x_4 , x_5 nenegativne veličine.

$3*x_1 + 4*x_2 + x_3 = 130$
$1*x_1 + 3*x_2 + x_4 = 9$
$x_1 + x_2 + x_5 = 60$

pa funkcija cilja izgleda ovako:

⇨ $z_{max} = 10*x_1 + 7*x_2 + 0*x_3 + 0*x_4 + 0*x_5$

- Sada formiramo Simpleks tabelu na klasičan način i određujemo pivot kolonu i pivot red, pa dobijamo iz preseka pivot element.

			X_1	X_2	X_3	X_4	X_5
C	B	X_0	10	7	0	0	0
0	X_3	130	**3**	4	1	0	0
0	X_4	90	1	3	0	1	0
0	X_5	60	1	1	0	0	1
z-c		0	-10	-7	0	0	0

- Dobijamo da je to 3, pa sledi sređivanje tabele.

			X_1	X_2	X_3	X_4	X_5
C	B	X_0	10	7	0	0	0
10	X_1	130/3	1	4/3	1/3	0	0
0	X_4	140/3	0	5/3	-1/3	1	0
0	X_5	50/3	0	-1/3	-1/3	0	1
z-c		1300/3	0	19/3	1/3	0	0

Primećujemo da nema negativnih brojeva u poslednjem redu. Imamo nulu u koloni x_1, a pošto je x_1 već u bazi naše rešenje je:
$z_{max}=1300/3$ za $x_1=130$

3.2. Zadatak 3

U restoranu *Keba Kraba* služi se isključivo meduzin sok i Kebina pljeska, a njihov tajni sastojak je meduzin sirup. Sunđer Bob, najvredniji radnik tog restorana, na raspolaganju ima 180 teglica meduzinog sirupa. Prema tajnom receptu, za jednu porudžbinu meduzinog soka potrebna je 1 teglica meduzinog sirupa, a za pljesku 2 teglice. Najavljeno je da će restoran posetiti tačno 75 mušterija i da će svako poručiti tačno po jednu Kebinu pljesku ili meduzin sok (nikako oboje). Cena jedne porudžbine meduzinog soka je 35 zlatnih novčića, a Kebine pljeske 55. Keba Krabi, vlasniku restorana, je cilj maksimalna zarada tako da će on zapisati model linearnog programiranja koji odgovara zapisanom problem i rešiti ga simpleks metodom.

Napomena: Sunđer Bob će napraviti tačno onoliko odgovarajućih porudžbina koliko mu Keba Kraba naredi, a mušterije, ukoliko nestane željene porudžbine, poručuju ono što ima.

Keba Krabino rešenje:

- Pažljivo čitamo tekst zadatka i izvlačimo relacije. Neka je broj napravljenih meduzinih sokova x_1, a Kebinih pljeski x_2. Dobijamo sledećea ograničenja:

$$1*x_1 + 2*x_2 \leq 180$$
$$x_1 + x_2 = 75$$
$$z = 35*x_1 + 55*x_2$$

- Sledeći korak je ispisivanje knonskog oblika ovog modela tako što od nejednačina pravimo jednačine. To radimo uvođenjem po jedne veštačke promenljive (x_3, x_4) u svakoj releciji. Podrazumevamo da su x_3 i x_4 nenegativne veličine.

$$1*x_1 + 2*x_2 + x_3 = 180$$
$$1*x_1 + 1*x_2 + x_4 = 75$$

pa funkcija cilja izgleda ovako:

$$\Rightarrow z_{max} = 35*x_1 + 55*x_2 + 0*x_3 - M*x_4$$

Koeficijent M se dodaje uz odredjeni koeficijent u funkciji cilja kada postoje ogranicenja u vidu jednakosti i zapisuje se sa predznakom minus kada tražimo maksimum funkcije. (Koeficijent M se zapisuje sa predznakom plus kada tražimo minimum funkcije). Smatramo da je koeficijent neki dovoljno veliki broj.

- Sada sledi klasičan ispis simpleks tabele kao u prethodnim zadacima. i tražimo pivot kolonu i red i dobijamo pivot element. Dakle, pivot broj je 1, pa odmah prelazimo na množenje pivot reda odgovarajućim koeficijentom (to je -2) i dodavanjem redu iznad. Promenljiva x_2 će otići u bazu i zauzeti mesto x_4.

			X_1	X_2	X_3	X_4	
C	B	X_0	35	55	0	-M	
0	X_3	180	1	2	1	0	-2
-M	X_4	75	1	**1**	0	1	
z-c		-75M	-M-35	-M-55	0	0	

Nakon sređivanja imamo sledeću tabelu:

			X_1	X_2	X_3	X_4
C	B	X_0	35	55	0	-M
0	X_3	30	-1	0	1	-2
55	X_2	75	1	1	0	1
z-c		4125	20	0	0	55+M

- kada pogledamo poslednji red, vidimo da su svi koeficijenti pozitivni što znači da je tu kraj zadatka:

  z_{max}= 4125 za x_2 =75

  x_3 =30

Dakle, zarada je 4125 zlatnih novčića ukoliko Sunđer Bob napravi 75 Kebinih pljeski i ostaće mu 30 teglica meduzinog sirupa neiskorišćeno.

4. Primena Simplex Metode

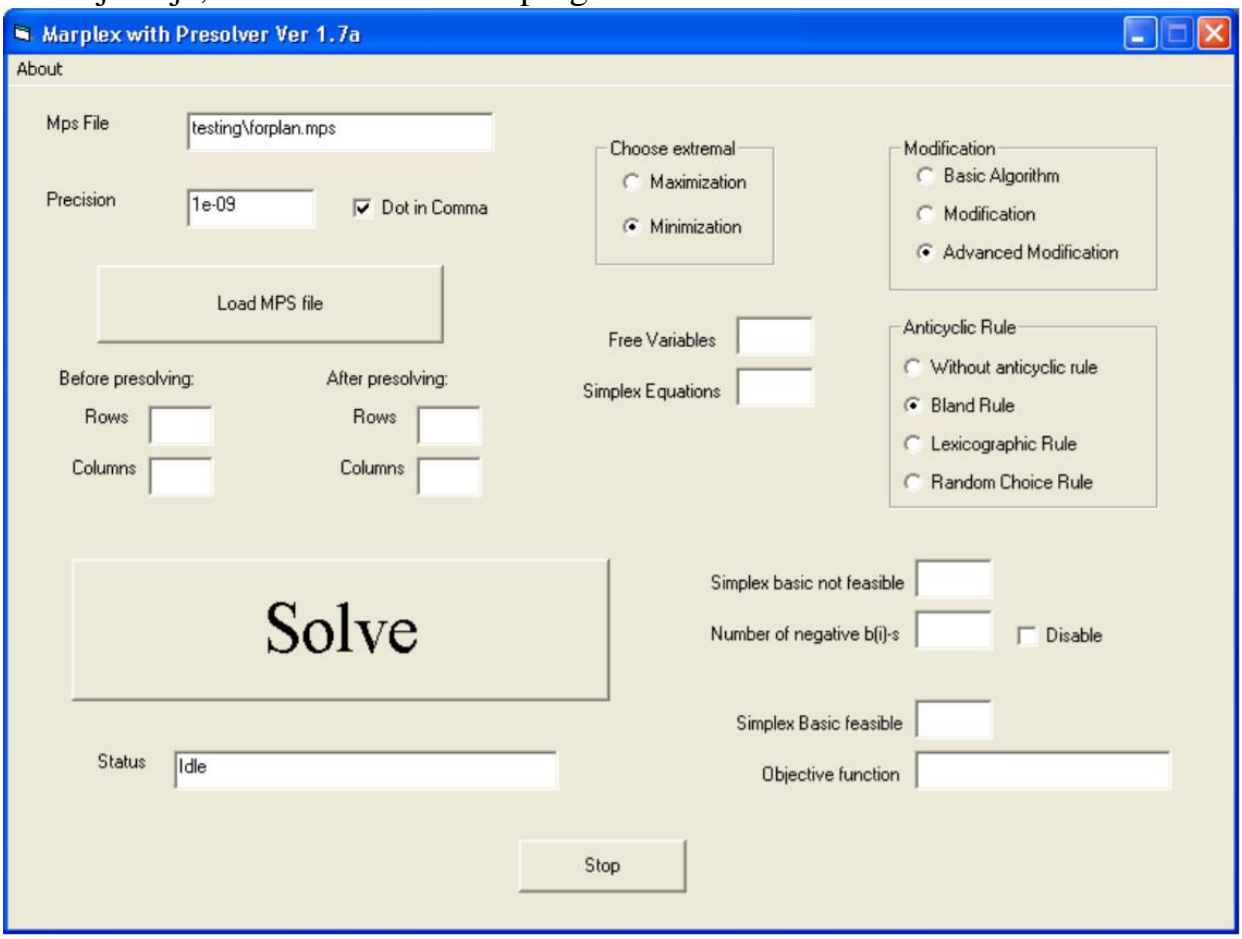
Simpleks algoritam, kao i neke njegove modifikacije, implementirani su u programskom jeziku *Visual Basic 6.0*. Tako je nastao program *MarPlex* koji vrlo uspešno rešava široku klasu problema linearnog programiranja. *MarPlex* je besplatan program i dostupan je na internetu. Pošto je implementiran u *Visual Basic*-u, i koristi simpleks metod, zaostaje po pitanju brzine.
Ulazni podaci se zadaju u obliku MPS fajla. Ovaj tip fajla predstavlja standard za zadavanje problema linearnog programiranja u vidu simpleks matrica (tablica):

```
-----------------------------------------------------------------
Polje:    1          2             3        4        5        6
Kolone :  2-3       5-12          15-22    25-36    40-47    50-61
          NAME   naziv problema
          ROWS
           tip     ime
          COLUMNS
                  ime           ime    vrednost   ime    vrednost
                  kolone        vrste             vrste
          RHS
                  ime           ime    vrednost   ime    vrednost
                  rhs           vrste             vrste
          RANGES
                  ime           ime    vrednost   ime    vrednost
                  oblasti       vrste             vrste
          BOUNDS
            tip   ime           ime
                  ogranicenja   kolone vrednost
          ENDATA
-----------------------------------------------------------------
```

U sekciji ROWS (vrste), svaka vrsta mora da ima tip i simboličko ime.
U sekciji COLUMNS (kolone) data su simbolička imena kolona sa simboličkim imenima vrsta i vrednostima (VALUES).
RHS je deo MPS formata koji opisuje slobodne članove nejednačina posmatranog problema. Postoje još i delovi RANGES i BOUNDS kojima se zadaje opseg vrednosti koji može da uzme svaka promenljiva.

Interface *MarPlex*-a je prilagođen korisniku i za razliku od drugih sličnih programa ne opterećuje korisnika brojnim opcijama koje se mahom ne koriste.
Jedna od njegovih najvažnijih opcija je *preciznost*. Ona omogućava korisniku da zada vrednost preciznosti sa kojom će program raditi. Sve vrednosti koje su po apsolutnoj vrednosti manje od zadatog broja, *MarPlex* će tretirati kao da su jednake nuli. Na taj način se sprečava biranje suviše malog pivot elementa, ali ukoliko je ova uneta vrednost prevelika, može doći do pogrešnih rezultata.
Sledeća opcija koju korisnik može da izabere je da li želi da rešava problem maksimizacije ili minimalizacije funkcije cilja. Takođe, može izabrati algoritam za nalaženje početnog bazičnog rešenja – Osnovni algoritam, Modifikacija ili Poboljšana modifikacija. Opcija "*Anticyclic Rule*" dozvoljava korisniku izbor anticikličnog pravila koje će se koristiti. Kod poslednje od navedenih opcija pivot se bira na slučajan način pri čemu su sve mogućnosti jednako verovatne.

Tokom rada algoritma, MarPlex omogućava praćenje nekoliko parametara samog algoritma: trenutni i ukupan broj iteracija svake faze algoritma, trenutnu vrednost funkcije cilja, kao i trenutni status programa.



Interface programa MarPlex

RevMarPlex predstavlja korigovanu verziju programa *MarPlex* napisanu u programskom jeziku *MATHEMATICA*. Za razliku od *MarPlex*-a koji je implementiran u proceduralnom programskom jeziku, odlučeno je koristiti paket *MATHEMATICA* prvenstveno zbog integrisanog solvera za sisteme linearnih jednačina. Korišćene su prednosti *MATHEMATICA*-e po pitanju ulaznih i izlaznih podataka (ovde se funkcija cilja i ograničenja mogu zadati u svom prirodnom obliku). Glavni nedostatak programa RevMarPlex je brzina rada, jer se programi napisani u *MATHEMATICA*-i izvršavaju znatno sporije od odgovarajućih programa u proceduralnim jezicima.

Na sledećim slikama prikazan je kod *Mathlab* funkcije "*nma_simplex.m*" koji implementira simpleks algoritam za rešavanje standardne forme problema linearnog programiranja. Ova funkcija rešava problem nalaženjem optimalnog rešenja, i kao izlaz sadrži tablicu iz koje se direktno mogu iščitati rešenja.

```
function tab = nma_simplex(A,b,c,debug)
%function [A,b,c]=nma_simplex(A,b,c)
%This function implments the simplex matrix algorithm.
%It accepts A_eq and b_eq and c as defined in standard
%documentation and generates all the simplex tableaus, and
%returns the final tableau which the user can read from it the
%minimum value of the objective function and the optimal x vector
%directly.
%
%It runs both phase one and phase two automatically.
%
%The input is
%
%A: This is the Ax=b matrix. This is for simplex standard
%   form only. The caller must convert all inequalites to
%   equalities first by using slack and suprluse variables. This
%   is what is called the Aeq matrix in Matlab documenation.
%   This function does not support Ax<b form. A has to be in
%   standard form
%
%b: Vector. This is the right hand side of Ax=b.
%
%c: Vector. This is from minimize  J(x) = c'x. As defined in
%   standard Matlab documentations.
%
%debug: flag. Set to true to see lots of internal steps.
%
%Returns:
%
%This function returns the final tableau. It has the form
%
%  [ A | b  ]
%  [ c | J  ]
%
% Version 5/12/2016
% by Nasser M. Abbasi
% Free for use.

validate_input(A,b,c);

[A,b] = make_phase_one(A,b,debug);
tab   = simplex(A,b,c,debug,'phase two');
end
%==========================
```

```
function [A,b] = make_phase_one(A,b,debug)
[m,n]               = size(A);
tab                 = zeros(m+1,n+m+1);
tab(1:m,1:n)        = A;
tab(end,n+1:end-1) = 1;
tab(1:m,end)        = b(:);
tab(1:m,n+1:n+m)    = eye(m);

if debug
    fprintf('>>>>Current tableau [phase one]\n');
    disp(tab);
end

for i = 1:m %now make all entries in bottom row zero
    tab(end,:) = tab(end,:)-tab(i,:);
end

tab = simplex(tab(1:m,1:n+m),tab(1:m,end),tab(end,1:n+m), ...
                                                    debug,'phase one');
%if tab(end,end) ~=0
%   error('artificial J(x) is not zero at end of phase one.');
%end

A = tab(1:m,1:n);
b = tab(1:m,end);

end
%================================
function tab = simplex(A,b,c,debug,phase_name)
[m,n]        = size(A);
tab          = zeros(m+1,n+1);
tab(1:m,1:n) = A;
tab(m+1,1:n) = c(:);
tab(1:m,end) = b(:);

keep_running = true;
while keep_running
    if debug
        fprintf('**********************\n');
        fprintf('Current tableau [%s] \n',phase_name);
        disp(tab);
    end

    if any(tab(end,1:n)<0)%check if there is negative cost coeff.
        [~,J] = min(tab(end,1:n)); %yes, find the most negative
        % now check if corresponding column is unbounded
        if all(tab(1:m,J)<=0)
          error('problem unbounded. All entries <= 0 in column %d',J);
        %do row operations to make all entries in the column 0
        %except pivot
        else
            pivot_row = 0;
            min_found = inf;
            for i = 1:m
                if tab(i,J)>0
                    tmp = tab(i,end)/tab(i,J);
                    if tmp < min_found
                        min_found = tmp;
                        pivot_row = i;
                    end
                end
            end
            if debug
                fprintf('pivot row is %d\n',pivot_row);
            end
```

```
            %normalize
            tab(pivot_row,:) = tab(pivot_row,:)/tab(pivot_row,J);
            %now make all entries in J column zero.
            for i=1:m+1
                if i ~= pivot_row
                    tab(i,:)=tab(i,:)-sign(tab(i,J))*...
                                abs(tab(i,J))*tab(pivot_row,:);
                end
            end
        end
        if debug  %print current basic feasible solution
            fprintf('current basic feasible solution is\n');
            disp(get_current_x());
        end
    else
        keep_running=false;
    end
end

    %internal function, finds current basis vector
    function current_x = get_current_x()
        current_x = zeros(n,1);
        for j=1:n
            if length(find(tab(:,j)==0))==m
                idx= tab(:,j)==1;
                current_x(j)=tab(idx,end);
            end
        end
    end
end
%=========================
function validate_input(A,b,c)
if ~ismatrix(A)
    error('A must be matrix');
end

if ~isvector(b)
    error('b must be vector');
end
if ~isvector(c)
    error('c must be vector');
end

[m,n]=size(A);
if rank(A) <m
    error('Rank A must be equal to number of rows in A');
end

if length(b) ~= m
    error('b must have same size as number of rows of A');
end
if length(c) ~= n
    error('c must have same size as number of columns of A');
end
end
```

1.Primer

Prodavac prodaje prstenje i minđuše. Prsten je sačinjen od tri grama zlata i jednog grama srebra. Minđuše su sačinjene od jednog grama zlata i dva grama srebra. Cena prstena je 4 evra, dok je cena minđuša 5 evra. Inicijalno, na raspolaganju

prodavac ima 8 grama zlata i 9 grama srebra. Koliko prstenja i minđuša je potrebno napraviti da bi se najviše zaradilo? (brojčane vrednosti su izmišljene radi lakšeg računa.)

x_1- broj prstenja;

x_2- broj minđuša; (pod uslovom da su x_1 i x_2 pozitivne konstante)

f(x)- funkcija cilja;

Potrebno je sada da postavimo zadatak na već poznati način. Pošto želimo da imamo najveću vrednost ciljne funkcije, potrebno je da joj promenimo znak:

$f(x)= - 4x_1 - 5x_2$

$3x_1 + x_2 \leq 8$

$x_1 + 2x_2 \leq 9$

Nakon dodavanja dodatnih promenljivih x_3 i x_4 unosimo sledeći kod:

```
A=[3,1,1,0;
1,2,0,1]; -koeficijenti iz novodobijenih jednačina
b=[8,9]; -slobodni koeficijenti
c=[-4,-5,0,0]; -koeficijenti iz ciljne funkcije sa koeficijentima uz dodatne promenljive
format short; - format u kom će se ispisivati brojevi (4 decimale nakon zareza)
nma_simplex(A,b,c,false) – poziv gorenavedene funkcije.
```

Poslednji argument funkcije je *true* ili *false* u zavisnosti od toga da li želimo da vidimo postupno promene u matrici(ako izaberemo true, praktično izgleda kao list na kome bismo to radili ručno).

Rezultat:

```
1.0000         0    0.4000   -0.2000    1.4000
     0    1.0000   -0.2000    0.6000    3.8000
     0         0    0.6000    2.2000   24.6000
```

Iščitavamo rešenje na već poznati način i vidimo da je **x_1=1.4** a **x_2=3.8** (pomoćne promenljive su iz očiglednih razloga jednake nuli). Ubacivanjem dobijenih parametara u ciljnu funkciju, kao rešenje dobijamo **f(x)=24.4 evra**, što znači da bi prodavac za maksimalan profit trebalo da proda 1.4 prstena i 3.8 minđuša.

Pored navedene funkcije koja je pisana od strane korisnika, u okviru *Mathlab*-a postoje unapred definisane funkcije koje uspešno rešavaju problem linearnog programiranja. Klikom na svaki od sledećih linkova možete se dodatno informisati o njihovom korišćenju.

x = linprog(f,A,b)
x = linprog(f,A,b,Aeq,beq)
x = linprog(f,A,b,Aeq,beq,lb,ub)
x = linprog(f,A,b,Aeq,beq,lb,ub,options)

x = linprog(problem)
[x,fval] = linprog(___)
[x,fval,exitflag,output] = linprog(___)
[x,fval,exitflag,output,lambda] = linprog(___)

Prethodni primer smo mogli da rešimo i preko integrisanih *Mathlab* funkcija za linearno programiranje:

```
A=[3,1,1,0;
1,2,0,1];
b=[8,9];
c=[-4,-5,0,0];
format short;
options = optimset('LargeScale','off','Simplex','on');
[X,FVAL,EXITFLAG,OUTPUT]=...
        linprog(c,[],[],A,b,zeros(size(c)),[],[],options)
```

Rezultat:

```
X =
    1.4000
    3.8000
         0
         0

FVAL =
  -24.6000

EXITFLAG =
     1

OUTPUT =
         iterations: 0
          algorithm: 'simplex'
      cgiterations: []
            message: 'Optimization terminated.'
    constrviolation: 8.8818e-16
      firstorderopt: 8.8818e-16
```

Nakon navedenih softvera zanimljivo je pomenuti i aplikaciju za telefon koja može da posluži za rešavanje problema Simpleks metodom, kao i njenu onlajn verziju - https://linprog.com/ .

5.Literatura

- https://sr.wikipedia.org/sr-ec/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B0%D1%80%D0%BD%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B8%D1%80%D0%B0%D1%9A%D0%B5
- https://youtu.be/Bb0iUw00e6w
- https://youtu.be/ldpODs2olTQ
- https://youtu.be/FWjafpqppkw
- D. Cvetković, S. Simić, *Odabrana poglavlja iz diskretne matematike,* 2012
- https://www.12000.org/my_notes/simplex/index_legal.pdf
- http://www1.pmf.ni.ac.rs/pmf/vesti/biblioteka/knjige/matematicko_programiranje.pdf