

ՀԱՅԱՍՏԱՆԻ ՀԱՆՐԱՊԵՏՈՒԹՅԱՆ ԿՐԹՈՒԹՅԱՆ ԵՎ ԳԻՏՈՒԹՅԱՆ
ՆԱԽԱՐԱՐՈՒԹՅՈՒՆ

ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՊՈԼԻՏԵԿՆԻԿԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

ՄԱԳԻՍՏՐՈՍԱԿԱՆ ԹԵԶ

ԹԵՍԱ՝ ԳԾ Խնդիրների լուծման սիմպլեքս և էլիպտիկների մեթոդների ծրագրային
իրականացումը և համեմատական վերլուծությունը

ՄԱԳԻՍՏՐԱՏ

Պապոյան Հովսեփ Սարգսի

ՄԱՍՆԱԳԻՏՈՒԹՅՈՒՆ

«Տեղեկատվական տեխնոլոգիաներ»

ԿՐԹԱԿԱՆ ԾՐԱԳԻՐ

061105.01.7 Տեղեկատվական տեխնոլոգիաներ

ՈՐԱԿԱՎՈՐՄԱՆ ԱՍՏԻՃԱՆ

Ինֆորմատիկայի մագիստրոսի

ԵՐԵՎԱՆ 2018

ՀԱՍՏԱՏՄԱՆ ԹԵՂՄ

ԹԵՍԱ՝ ԳԾ խնդիրների լուծման սիմպլեքս և էլիպտիկների մեթոդների
ծրագրային իրականացումը և համեմատական վերլուծությունը

Թեզի դեկավար՝ Աղամյան Արմեն Գևորգի Առաքել
ԱԱՀ, գիտ.աստիճան, կոչում

Մագիստրանտ՝ Պապոյան Հովհաննես Սարգսի

Գրախոս՝ Աղամյան Արմեն Գևորգի Թամագյան
ԱԱՀ, գիտ.աստիճան, կոչում

Ամբիոնի վարիչ՝ Աղամյան Արմեն Գևորգի Սիմոնյան
ԱԱՀ, գիտ.աստիճան, կոչում

ՏՀՏԵ ինստիտուտի տնօրենի

պաշտոնակատար՝ Աղամյան Արմեն Գևորգի Բարեկամյան
ԱԱՀ, գիտ.աստիճան, կոչում

ՏՎՅԱԼՆԵՐ ՇՐՋԱՆԱԿԱՐՏԻ ՄԱՍԻՆ

Մազիստրանտ՝ Պապոյան Հովսեփ Սարգսի

Մասնագիտություն՝ Ինֆորմատիկայի մազիստրոս

Կրթական ծրագիր՝ 061105.01.7 Տեղեկատվական տեխնոլոգիաներ

Ծննդյան տարեթիվը՝ 1993

Մինչ մազիստրոսական որակավորումը՝ Ինֆորմատիկայի բակալավր

Մասնագիտությունը՝ Տեղեկատվական տեխնոլոգիաներ

Կրթական ծրագիրը՝ 061105.01.6 Տեղեկատվական տեխնոլոգիաներ

Հրատարակված աշխատանքներ՝ _____

ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՊՈԼԻՏԵԽՆԻԿԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ
ՏԵՂԵԿԱՏՎԱԿԱՆ և ՀԵՌԱՀԱԴՐԱԿՑԱԿԱՆ ՏԵԽՆՈԼՈԳԻԱՆԵՐԻ
ՈՒ ԷԼԵԿՏՐՈՆԻԿԱՅԻ ԻՆՍԻՏՈՒԹԵ

ԱՄԲԻՈՆ _____ **Տեղեկատվական տեխնոլոգիաներ և ավտոմատացում**
(ամբիոնի անվանումը)

Մասնագիտություն _____ **Տեղեկատվական տեխնոլոգիաներ** դասից **061105.01.7**
(գրել առանց հապավումների)

Կրթական ծրագիր **061105.01.7 Տեղեկատվական տեխնոլոգիաներ**
(գրել առանց հապավումներ)

Թիվ **ԱՀ621** ակադեմիական խմբի
(խմբի համարը)
Պապոյան Հովհաննեսի Սարգսի

ՄԱԳԻՍՏՐՈՍԱԿԱՆ ԹԵԶԻ ԱՌԱՋԱԴՐԱՆՔ

- Թեման **ԳԾ խնդիրների լուծման սխմալեքս և էլիպտիդների մեթոդների ծրագրային իրականացումը և համեմատական վերլուծությունը**
Հաստատված է **ՀԱՊՀ 2016 թ.-ի 10 «18» թիվ 01-11/887** հրամանով:
2. Նախնական տվյալներ **Ուսումնասիրել ԳԾ խնդիրների լուծման էլիպտիդների մեթոդը,**
կազմել սխմալեքս և էլիպտիդների մեթոդների ալգորիթմը իրականացնող ծրագրեր
C++ լեզվով, ծրագրերի աշխատունակությունը ստուգելու նպատակով լուծել
թեստային օրինակներ, լուծումների հիման վրա կատարել մեթոդների
համեմատական վերլուծություն
- Հաշվեբացատրագրի բովանդակություն (բաժինների և մշակման ենթակա հարցերի
թվարկմամբ)

1. ՆԵՐԱԾՈՒԹՅՈՒՆ.

2. ԳԼՈՒԽ 1. ԳԾԱՅԻՆ ԾՐԱԳՐԱՎՈՐՈՒՄ,

3. ԳԼՈՒԽ 2. ԱԻՄՊԼԵՔՍ ՄԵԹՈԴ,

4. ԳԼՈՒԽ 3. ԷԼԻՊՍՈՒԹԵՐԻ ՄԵԹՈԴ,

5. ԳԼՈՒԽ 4. ԱՐԴՅՈՒՆՔՆԵՐԻ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆ,

6. ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ

4. Թեղի կատարման օրացուցային պլան

Հ/Հ	Թեղի կատարման փուլերը			Ծանոթություն
	Անվանումը	Կատ. ժամկ.	հաշ. ձևը	
1.	Գրականության ակնարկ:			
2.	Տեսական հարցերի ուսումնասիրություն:			
3.				
	I ատեստավորում		20 %	
4.	Ալգորիթմների մշակում:			
5.	Ծրագրերի կազմում:			
6.				
	II ատեստավորում		60 %	
7.	Փորձնական հետազոտություններ:			
8.	Վերլուծություններ:			
9.	Եզրակացություն:			
10.				
	III ատեստավորում		100%	
11.	Աշխատանքի ներկայացումը ամբիոն		Ավ. աշխ.	
12.	Նախնական պաշտպանություն			

5. Աշխատանքի պաշտպանության օրը _____

6. Ամբիոնի վարիչ _____ Սիմոնյան Ա. Հ.
(Ա.Ա.Հ., ստորագրություն, ամսաթիվ)

7. Աշխատանքի դեկանը _____ Արամյան Գ. Վ.
(Ա.Ա.Հ., ստորագրություն, ամսաթիվ)

8. Աշխատանքի առաջադրանքը ստացա
_____ Պապոյան Հ. Ս.
(Ուսանողի Ա.Ա.Հ., ստորագրություն, ամսաթիվ)

ԲՈՎԱՆԴԱԿՈՒԹՅՈՒՆ

ԲՈՎԱՆԴԱԿՈՒԹՅՈՒՆ.....	6
ՆԵՐԱԾՈՒԹՅՈՒՆ.....	8
ԳԼՈՒԽ 1. ԳԾԱՅԻՆ ԾՐԱԳՐԱՎՈՐՈՒՄ	10
1.1.Գծային ծրագրավորման (ԳԾ) խնդրի դրվածքը	10
1.2. ԳԾ խնդրի օրինակ	12
ԳԼՈՒԽ 2. ՍԻՄՊԼԵՔՍ ՄԵԹՈԴ	13
2.1. Սիմպլեքս մեթոդում օգտագործված նշանակումներ.....	13
2.2. ԳԾ կանոնական խնդրում անկյունային կետերը: Անկյունային կետի օպտիմալու- թյան չափանիշ.....	15
2.3. Մոդիֆիկացված և աղյուսակային սիմպլեքս ալգորիթմներ.....	22
2.4. Սիմպլեքս մեթոդի երկրաչափությունը	34
2.5. Վերասերվածություն և ցիլլ: Բլենդի կանոն	36
2.6. Արհեստական բազիսի մեթոդ և երկֆազ սիմպլեքս մեթոդ	38
ԳԼՈՒԽ 3. ԷԼԻՊՍՈՒՐԻՆԵՐԻ ՄԵԹՈԴ	50
3.1. Էլիպսուիդների մեթոդի ստեղծման և հետագա զարգացման մասին.....	50
3.2. Էլիպսուիդների եղանակի երկրաչափությունը	52
3.3. Հիմնական էլիպսուիդների մեթոդի նկարագրությունը.....	59
3.4. Էլիպսուիդների մեթոդի իրականացումը և պոլինոմիալությունը.....	64
3.5. Էլիպսուիդների մեթոդով խնդրի լուծման օրինակ.....	67
3.6. Սահմանափակության և լրիվ չափողականության ենթադրությունների վերա- նայում	68
3.7. Էլիպսուիդների մեթոդը գծային ծրագրավորման համար.....	70
ԳԼՈՒԽ 4. ԱՐԴՅՈՒՆՔՆԵՐԻ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆ	72
4.1. Երկֆազ սիմպլեքս մեթոդով ԳԾ խնդիրների լուծման օրինակներ	72
4.2. Էլիպսուիդների մեթոդով ԳԾ խնդիրների լուծման օրինակներ	75
ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ	82

ՕԳՏԱԳՈՐԾՎԱԾ ԳՐԱԿԱՆՈՒԹՅԱՆ ՑԱՆԿ	84
ՀԱՎԵԼՎԱԾ 1. ԳԾ ՀԵՏ ԿԱՊՎԱԾ ՓԱՍՏԵՐ	89
Հ.1.1. ԳԾ ընդհանուր խնդիրը	90
Հ.1.2. Բազմանիստեր	90
Հ.1.3. ԳԾ երկավիոլյուն	93
Հ.1.4. Ուռուցիկ թաղանթներ և բազմանիստեր	95
ՀԱՎԵԼՎԱԾ 2. ԵՐԿՖԱԶ ՍԻՄՊԼԵՔՍ ԵՎ ԷԼԻՊՍՈՒԴԻՆԵՐԻ ԱԼԳՈՐԻԹՄ-ՆԵՐԻ ԱՇԽԱՏԱՆՔԸ	97
Հ.2.1. Երկֆազ սիմպլեքս ալգորիթմի աշխատանքը	98
Հ.2.2. Էլիպսուդների ալգորիթմի աշխատանքը	116
ՀԱՎԵԼՎԱԾ 3. ԵՐԿՖԱԶ ՍԻՄՊԼԵՔՍ ՄԵԹՈԴԻ ԾՐԱԳՐԱՅԻՆ ԻՐԱԿԱՆԱՑՈՒՄԸ	125
ՀԱՎԵԼՎԱԾ 4. ԷԼԻՊՍՈՒԴԻՆԵՐԻ ՄԵԹՈԴԻ ԾՐԱԳՐԱՅԻՆ ԻՐԱԿԱՆԱՑՈՒՄԸ	140

ՆԵՐԱԾՈՒԹՅՈՒՆ

Գծային օպտիմալացումը, այսինքն գծային ֆունկցիայի էքստրեմումի որոնումը գծային սահմանափակումների առկայության պայմաններում, մաթեմատիկայի կարևորագույն բաժիններից մեկն է՝ ինչպես տեսական հետազոտությունների, այնպես էլ գործնական կիրառությունների տեսակետից: 1947 թ. Զ. Բ. Դանցից(G. B. Dantzig) կողմից մշակված սիմպլեքս մեթոդ՝ իրականացման պարզության և խնդրի լուծման արագության բավական լավ ցուցանիշների առկայության հետևանքով, ենթարկվել է տարատեսակ և մեծաքանակ ձևափոխությունների և դարձել գծային ծրագրավորման խնդիրների լուծման առավել տարածված եղանակը:

Դրա հետ մեկտեղ սիմպլեքս մեթոդն այդ խնդիրների լուծման միակ եղանակը չէ: Նախորդ դարի 80-ական թվականների մի շարք հետազոտությունների արդյունքում ստեղծվել է այդ խնդիրների լուծման այլընտրանքային ուղղություն, որին տրվել է ներքին կետերի ալգորիթմներ անունը: Ի տարբերություն սիմպլեքս-մեթոդի, որն ընտրություն իրականացնում է լուծումների թույլատրելի տիրույթի բազմանիստի գագաթների միջև, ներքին կետերի ալգորիթմներում հաշվողական գործընթացն իրականացվում է թույլատրելի բազմության ներքին կետերի նկատմամբ: Բացի դրանից, մոտարկումների ձևավորվող հաջորդականությունը զուգամիտում է օպտիմալ լուծումների բազմության ներքին կետին:

Այդ մոտեցման նկատմամբ մեծ հետաքրքրությունը պայմանավորված է գծային ծրագրավորման խնդիրների լուծման համար բազմանդամային ալգորիթմների ստեղծման հնարավորություններով: Խնդիրների դասի բազմանդամային լուծելիության հասկացությունը խաղում է հիմնական դեր ալգորիթմների բարդության տեսության մեջ: Այդ տեսությունը հնարավորություն է տալիս գնահատել ալգորիթմի իրականացման համար պահանջվող հաշվողական ռեսուրսները՝ օգտագործելով ժամանակային բարդություն և տարածական բարդություն պարամետրերը: Սովորաբար ալգորիթմները դասակարգվում են ըստ դրանց ժամանակային բարդության: Ալգորիթմի բարդությունը ներկայացվում է մուտքային տվյալներից կախված ֆունկցիայի տեսքով: Եթե ալգորիթմը ընդունակ է լուծել հետազոտվող դասի ցանկացած խնդիր՝ դրա չափողականությունից

կախված ինչ-որ բազմանդամով ներկայացվող ժամանակում, ապա ալգորիթմն ունի բազմանդամային բարդության աստիճան, իսկ ինքը՝ խնդիրների դասը կոչվում է բազմանդամային լուծելիության:

L. G. Խաչիյանը(L. G. Khachiyan) ուռուցիկ ծրագրավորման բնագավառում օգտագործվող՝ ծավալով փոքրացող բազմությունների հաջորդականության տեխնիկայի կիրառման հիման վրա ապացուցել է, որ գծային ծրագրավորման խնդիրը պատկանում է բազմանդամային լուծելիության խնդիրների դասին: Նրա առաջարկած էլիպսուղիների մեթոդը գծային ծրագրավորման խնդիրների լուծման այլընտրանքային մոտեցում է: Չնայած այդ ալգորիթմի գործնական կիրառությունը ցույց է տվել գուգամիտման ոչ բարձր արագություն, սակայն այդ հետազոտության արդյունքները տեսության զարգացման տեսակետից շատ կարևոր են և խթանել են այդ բնագավառում հետազա ուսումնասիրությունները:

Ավարտական թեզում խնդիր է դրվել՝ կատարել գծային ծրագրավորման խնդիրների լուծման սիմպլեքս մեթոդի և էլիպսուղիների մեթոդի համեմատական վերլուծություն:

ԳԼՈՒԽ 1

ԳԾԱՅԻՆ ԾՐԱԳՐԱՎՈՐՈՒՄ

1.1. Գծային ծրագրավորման (ԳԾ) խնդրի դրվածքը

ԳԾ խնդրի է կոչվում գծային տեսքի էքստրեմումի (մաքսիմում կամ մինիմում) որոնման խնդիրը՝ գծային հավասարումներով և անհավասարումներով որոշվող տիրույթի վրա:

ԳԾ ընդհանուր խնդիրն ունի հետևյալ տեսքը.

$$f(x_1, \dots, x_n) = c_1x_1 + \dots + c_nx_n \rightarrow \max(\min); \quad (1.1)$$

$$a_{j1}x_1 + \dots + a_{jn}x_n = b_j, \quad j = 1, \dots, r, \quad (1.2)$$

$$a_{j1}x_1 + \dots + a_{jn}x_n \leq b_j, \quad j = r+1, \dots, m, \quad (1.3)$$

$$x_i \geq 0, \quad i \in I \subseteq \{1, \dots, n\}; \quad (1.4)$$

$f(x_1, \dots, x_n)$ ֆունկցիան կոչվում է նպատակային ֆունկցիա: Նպատակային ֆունկցիայի նշանը փոխելով մաքսիմումի խնդիրը բերվում է մինիմումի խնդրի և հակառակը: Եթե խնդրում կան $a_{j1}x_1 + \dots + a_{jn}x_n \geq b_j$ տեսքի սահմանափակումներ, ապա դրանք կարելի է բերել (1.3) տեսքի՝ բազմապատկելով անհավասարման 2 մասերը (-1)-ով: Փոփոխականների ոչ բացասական լինելու (1.4) պայմանը հանդիսանում է (1.3)-ի մասնավոր դեպքը ($-x_i \leq 0$), սակայն ԳԾ խնդիրներում ընդունված է դրանք դիտարկել առանձին: Եթե խնդրի պայմաններից ելնելով ինչ-որ $x_i \leq 0$, ապա այդ փոփոխականը կարելի է համարել ոչ բացասական՝ փոխելով համապատասխան c_i և a_{ji} նշանները:

ԳԾ ընդհանուր խնդիրը կոչվում է նաև խառը պայմաններով խնդիր՝ (1.2) հավասարումներով և (1.3) անհավասարումներով: ԳԾ խնդիրների հետազոտումը և լուծման մեթոդների կառուցումը հարմար է միատիպ սահմանափակումներով խնդիրների դեպքում:

$$c_1x_1 + \dots + c_nx_n \rightarrow \max; \quad (1.5)$$

$$a_{j1}x_1 + \dots + a_{jn}x_n = b_j, \quad j = 1, \dots, m, \quad (1.6)$$

$$x_i \geq 0, \quad i = 1, \dots, n, \quad (1.7)$$

կամ

$$c_1x_1 + \dots + c_nx_n \rightarrow \max; \quad (1.8)$$

$$a_{j1}x_1 + \dots + a_{jn}x_n \leq b_j, \quad j = 1, \dots, m, \quad (1.9)$$

$$x_i \geq 0, \quad i = 1, \dots, n; \quad (1.10)$$

(1.5) - (1.7) կոչվում է ԳԾ կանոնական խնդիր, (1.8) - (1.10)` ԳԾ ստանդարտ խնդիր: ԳԾ կանոնական խնդիրը հետևում է ԳԾ ընդհանուր խնդրից՝ $r = m$, $I = \{1, \dots, n\}$, իսկ ստանդարտ խնդրի դեպքում՝ $r = 0$, $I = \{1, \dots, n\}$:

Ցանկացած ԳԾ խնդիր հեշտությամբ բերվում է ինչպես կանոնական, այնպես էլ ստանդարտ խնդրի: Կանոնականը բերվում է ստանդարտի, ստանդարտը՝ կանոնականի:

Յուրաքանչյուր (1.6) հավասարում համարժեք է 2 անհավասարման՝

$$a_{j1}x_1 + \dots + a_{jn}x_n \leq b_j,$$

$$-a_{j1}x_1 - \dots - a_{jn}x_n \leq -b_j,$$

հետևաբար (1.5) - (1.7) կանոնական խնդիրը համարժեք է հետևյալ ստանդարտ խնդրին՝

$$c_1x_1 + \dots + c_nx_n \rightarrow max;$$

$$a_{j1}x_1 + \dots + a_{jn}x_n \leq b_j, \quad j = 1, \dots, 2m,$$

$$x_i \geq 0, \quad i = 1, \dots, n,$$

որտեղ $a_{ji} = -a_{j-m,i}$, $b_j = -b_{j-m}$, $j = m+1, \dots, 2m$:

Անհավասարումից հավասարման անցումը իրականացվում է լրացուցիչ փոփոխականների ներմուծմամբ: Յուրաքանչյուր (1.9) սահմանափակման հետ կապվում է լրացուցիչ փոփոխական՝

$$x_{n+j} = b_j - (a_{j1}x_1 + \dots + a_{jn}x_n):$$

Ակնհայտ է, որ $x_{n+j} \geq 0$ այն և միայն այն դեպքում, եթե տեղի ունի (1.9) պայմանը: Հետևաբար (1.9)-ը կարելի է փոխարինել հետևյալ պայմանների զույգով՝

$$a_{j1}x_1 + \dots + a_{jn}x_n + x_{n+j} = b_j,$$

$$x_{n+j} \geq 0:$$

Այսպիսով (1.8)-(1.10) ստանդարտ խնդիրը համարժեք է հետևյալ կանոնական խնդրին՝

$$c_1x_1 + \dots + c_nx_n \rightarrow max; \quad (1.11)$$

$$a_{j1}x_1 + \dots + a_{jn}x_n + x_{n+j} = b_j, \quad j = 1, \dots, m, \quad (1.12)$$

$$x_i \geq 0, \quad i = 1, \dots, n+m, \quad (1.13)$$

Նկատենք, որ ի տարրերություն սկզբնական (1.8) - (1.10) խնդրի, որն ուներ n փոփոխական, (1.11) - (1.13) խնդրում փոփոխականների քանակը $n+m$ է, ընդ որում

լրացուցիչ m փոփոխականները՝ x_{n+1}, \dots, x_{n+m} նպատակային ֆունկցիայում ունեն 0 գործակից՝

$$c_{n+1} = \dots = c_{n+m} = 0:$$

Դիտարկենք այն դեպքը, եթե ԳԾ խնդրում ինչ-որ x_i փոփոխականի նշանի վրա չկա սահմանափակում: Տեղադրենք՝

$$\bar{x}_i = \max\{0, x_i\}, \quad \bar{\bar{x}}_i = m a x\{0, -x_i\};$$

Այդ դեպքում ակնհայտ է, որ $\bar{x}_i \geq 0$, $\bar{\bar{x}}_i \geq 0$, $x_i = \bar{x}_i - \bar{\bar{x}}_i$: Այսինքն դիտարկված օրինակում x_i -ն կարելի է փոխարինել երկու ոչ բացասական փոփոխականների տարբերությունով:

1.2. ԳԾ խնդրի օրինակ

Սննդակարգի խնդիրը: Ենթադրենք կենդանիների կերակրման սննդակարգը կարող է կազմված լինել n տեսակի կերերից՝ K_1, \dots, K_n : Յուրաքանչյուր կերի տեսակ պարունակում է կենդանիների համար անհրաժեշտ m սննդարար նյութերից՝ Q_1, \dots, Q_m մեկ կամ մի քանի տեսակ: Սննդարար նյութերի պարունակությունը յուրաքանչյուր կերի տեսակում հայտնի է և կազմում է a_{ji} միավոր (ենթադրենք զ) Q_j տեսակի սննդարար նյութ K_i -րդ տեսակի կերի միավորի (ենթադրենք կզ) համար: Հայտնի է յուրաքանչյուր K_i տեսակի կերի համար այդ տեսակի միավոր կերի c_i արժեքը, ինչպես նաև հայտնի է Q_j տեսակի սննդարար նյութի համար b_j (միավոր) նվազագույն թույլատրելի չափը սննդակարգում: Խնդրի պահանջն է կազմել մինիմալ արժեք ունեցող սննդակարգ, որը բավարարում է նշված պահանջներին՝ սննդարար նյութեր պարունակելու առումով: x_i -ով նշանակենք i -րդ տեսակի կերի քանակը (միավորների քանակը) սննդակարգում: Կստանանք հետևյալ ԳԾ խնդիրը՝

$$f(x) = c_1 x_1 + \dots + c_n x_n \rightarrow \min;$$

$$a_{j1} x_1 + \dots + a_{jn} x_n \geq b_j, \quad j = 1, \dots, m,$$

$$x_i \geq 0, \quad i = 1, \dots, n:$$

ԳԼՈՒԽ 2

ՍԻՄՊԼԵՔՍ ՄԵԹՈԴ

Գծային Ծրագրավորման խնդիրների լուծման համար ամենահին և ամենահայտնի ալգորիթմը Զ. Բ. Դանցիզի (G. B. Dantzig) [1] սիմպլեքս մեթոդն է: Այս գլխում կանդրադառնանք մեթոդի նկարագրությանը:

2.1. Սիմպլեքս մեթոդում օգտագործված նշանակումներ

\mathbb{R}^n -ով նշանակենք n -չափանի եվկլիդյան տարածությունը: Որպես n -չափանի x, y վեկտորների՝ $x = (x_1, \dots, x_n)$, $y = (y_1, \dots, y_n)$ սկալյար արտադրյալ՝

$$\langle x, y^T \rangle = \sum_{i=1}^n x_i y_i:$$

$M_{m,n}$ -ով նշանակենք $m \times n$ չափանի մատրիցների բազմությունը, M_n -ով՝ n չափանի քառակուսային մատրիցներ բազմությունը: $x \in \mathbb{R}^n$ վեկտորը համարենք վեկտոր-սյուն՝

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}:$$

Այսպիսով՝ $x = (x_1, \dots, x_n)$ գրառումը նշանակում է, որ այն $x \in \mathbb{R}^n$ վեկտոր-սյուն է:

$[x_1, \dots, x_n]$ գրառումը նշանակում է վեկտոր-տող՝ x^T :

Որոշակիության համար դիտարկենք (1.5) - (1.7) կանոնական խնդիրը:
Ներմուծենք՝

$$c = (c_1, \dots, c_n), \quad x = (x_1, \dots, x_n), \quad b = (b_1, \dots, b_n)$$

վեկտորները և

$$A = [a_{ji}]_{j,i=1}^{m,n} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

մատրիցը: Ակնհայտ է, որ (1.5) - (1.7) խնդիրը կարելի է գրել հետևյալ տեսքով՝

$$\langle c, x \rangle \rightarrow \max; \quad Ax = b, \quad x \geq 0: \tag{2.1}$$

c վեկտորը կոչվում է նպատակային վեկտոր, x վեկտորը՝ խնդրի պլան, b վեկտորը՝ սահմանափակումների վեկտոր, իսկ A մատրիցը՝ պայմանների մատրից: A մատրիցի սյուները՝

$$a^1 = \begin{bmatrix} a_{11} \\ \cdots \\ a_{m1} \end{bmatrix}, \dots, a^n = \begin{bmatrix} a_{1n} \\ \cdots \\ a_{mn} \end{bmatrix}$$

կոչվում են պայմանների վեկտորներ:

$Ax = b$ պայմանի ձախ մասը երբեմն հարմար է ներկայացնել պայմանների վեկտորների գծային համակցության տեսքով (x_i գործակիցներով)

$$Ax = \begin{bmatrix} a_{11} \\ \cdots \\ a_{m1} \end{bmatrix} x_1 + \cdots + \begin{bmatrix} a_{1n} \\ \cdots \\ a_{mn} \end{bmatrix} x_n = \sum_{i=1}^n x_i a^i:$$

Խնդրի պայմաններին բավարարող պլանների բազմությունը կոչվում է թույլատրելի բազմություն կամ թույլատրելի պլանների բազմություն: Նշանակենք այդ բազմությունը X -ով, իսկ (2.1)-ը ներկայացնենք հետևյալ կերպ՝

$$f(x) = \langle c, x \rangle \rightarrow \max; \quad x \in X,$$

$$X = \{ x \in \mathbb{R}^n : Ax = b, \quad x \geq 0 \}:$$

$x \in X$ վեկտորները կոչվում են թույլատրելի վեկտորներ կամ թույլատրելի պլաններ: Եթե այլ բան ասված չէ, ապա հետագա շարադրանքում պլանների բազմություն և պլաններ ասելով կհասկանանք համապատասխանաբար թույլատրելի բազմություն և թույլատրելի պլաններ:

ԳԾ խնդրի լուծում կամ օպտիմալ պլան է կոչվում $f(x) = \langle c, x \rangle$ ֆունկցիայի մաքսիմումը X բազմության վրա: Օպտիմալ պլանները նշանակենք x^* , իսկ օպտիմալ պլանների բազմությունը՝ X_* : Էքստրեմումի խնդիրներում $f(x)$ ֆունկցիայի մաքսիմումի կետերի բազմությունը ընդունված է նշանակել $\underset{x}{\operatorname{Argmax}} f(x)$, իսկ կամայական մաքսիմումի կետը՝ $\underset{x}{\operatorname{argmax}} f(x)$: Այսպիսով՝

$$X_* = \underset{x}{\operatorname{Argmax}} f(x), \quad x^* = \underset{x}{\operatorname{argmax}} f(x):$$

(2.1) խնդիրը իրենից ներկայացնում է (1.5) - (1.7) կանոնական խնդրի մատրիցային գրառումը: (1.8) - (1.10) ստանդարտ խնդրի մատրիցային գրառումը ունի հետևյալ տեսքը՝

$$\langle c, x \rangle \rightarrow \max; \quad Ax \leq b, \quad x \geq 0,$$

որտեղ $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in M_{m,n}$: Այս խնդրին համարժեք (1.11) - (1.13) կանոնական խնդիրը կարող է ներկայացվել հետևյալ կերպ՝

$$\langle c, x \rangle \rightarrow \max; \quad Ax + u = b, \quad x \geq 0, \quad u \geq 0, \quad (2.2)$$

որտեղ $u = (u_1, \dots, u_m) = (x_{n+1}, \dots, x_{n+m})$ լրացուցիչ փոփոխականների վեկտորն է: Նշենք, որ (2.2) խնդրում $\bar{c} = (c_1, \dots, c_n, 0, \dots, 0) = (c, 0) \in \mathbb{R}^{n+m}$, պայմանների մատրիցը՝ $\bar{A} = [A \ E]$, որտեղ $E \in M_m$ միավոր մատրից է, իսկ խնդրի պլանը՝

$$z = (x_1, \dots, x_n, x_{n+1}, \dots, x_{n+m}) = (x, u):$$

$$\langle \bar{c}, z \rangle = \langle (c, 0), (x, u) \rangle = \langle c, x \rangle + \langle 0, u \rangle = \langle c, x \rangle,$$

$$\bar{A}z = [A \ E] \cdot \begin{bmatrix} x \\ u \end{bmatrix} = Ax + u:$$

ԳԾ (1.1) - (1.4) ընդհանուր խնդիրը մատրիցային տեսքով կարելի է ներկայացնել հետևյալ կերպ՝

$$\langle c, x \rangle \rightarrow \max; \quad Ax = b, \quad \bar{A}x \leq \bar{b}, \quad x_i \geq 0, \quad i \in I:$$

Այստեղ՝ $c, x \in \mathbb{R}^n$, $b = (b_1, \dots, b_r)$, $\bar{b} = (b_{r+1}, \dots, b_m)$, իսկ A և \bar{A} պայմանի մատրիցները ունեն հետևյալ տեսքը՝

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{r1} & \dots & a_{rn} \end{bmatrix}, \quad \bar{A} = \begin{bmatrix} a_{r+1,1} & \dots & a_{r+1,n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}:$$

2.2. ԳԾ կանոնական խնդրում անկյունային կետերը: Անկյունային կետի օպտիմալության չափանիշ

Դիտարկենք ԳԾ կանոնական խնդրի պլանների բազմությունը.

$$X = \{x \in \mathbb{R}^n: Ax = b, \quad x \geq 0, \quad A \in M_{m,n}\}: \quad (2.3)$$

$Ax = b$ համակարգը կարելի է ներկայացնել A մատրիցի այուների (պայմանների վեկտորներ) գծային համակցության տեսքով.

$$x_1a^1 + x_2a^2 + \dots + x_na^n = b: \quad (2.4)$$

Ըստ (2.4) ներկայացման x վեկտորի յուրաքանչյուր x_i բաղադրիչին համապատասխանում է A մատրիցի իր այունը և հակառակը:

Թեորեմ: Որպեսզի X բազմության $\bar{x} \neq 0$ կետը լինի X բազմության անկյունային կետ, անհրաժեշտ է և բավարար, որ այդ կետի դրական բաղադրիչներին համապատասխան պայմանների վեկտորները լինեն գծորեն անկախ:

Ապացույցը տե՛ս [2]:

Հետևանք: \bar{x} անկյունային կետի դրական բաղադրիչների քանակը չի գերազանցում $r = \text{rank}(A)$:

Սահմանում: Ենթադրենք \bar{x} -ը X բազմության անկյունային կետ է և $r = \text{rank}(A)$: r գծորեն անկախ պայմանների վեկտորների համակարգը, որը ներառում է բոլոր այն a^i -ները, որոնց համար $\bar{x}_i > 0$, կոչվում է \bar{x} անկյունային կետի բազիս: Համակարգը կազմող վեկտորները կոչվում են բազիսային վեկտորներ, իսկ նրանց համապատասխանող \bar{x} վեկտորի բաղադրիչները՝ բազիսային բաղադրիչներ:

Եթե անկյունային կետի դրական բաղադրիչների քանակը հավասար է r , ապա դրանք միարժեքորեն որոշում են բազիսը, հակառակ դեպքում բազիսը կարող է միակը չհնել, բայց միշտ գոյություն ունի: Ենթադրենք \bar{x} անկյունային կետը ունի $s > r$ դրական բաղադրիչ, որոնց համապատասխանում է a^{i_1}, \dots, a^{i_s} պայմանների վեկտորների գծորեն անկախ համակարգ: Մեծացնենք համակարգը մինչև մաքսի-մալ գծորեն անկախ վեկտորների համակարգի՝ մնացած պայմանական վեկտորների հաշվին՝ $a^{i_{s+1}}, \dots, a^{i_n}$: Արդյունքում կստանանք a^{i_1}, \dots, a^{i_k} , $k \geq s$, որը կազմում է a^1, \dots, a^n վեկտորների գծային թաղանթի բազիսը՝ $k = \text{rank}[a^1, \dots, a^n] = r$: Այսպիսով r վեկտորներից կազմված համակարգը հանդիսանում է գծորեն անկախ և ներառում է իր մեջ բոլոր այն a^i -երը, որոնց համար $\bar{x}_i > 0$, այսինքն իրենից ներկայացնում է \bar{x} անկյունային կետի բազիսը: Կառուցումից երևում է, որ բազիսը կարող է միակը չհնել:

Օրինակ 1:

Ենթադրենք X բազմությունը որոշվում է հետրյալ սահմանափակումներով.

$$x_1 + x_2 + 3x_3 + x_4 = 3, \quad (2.5)$$

$$x_1 - x_2 + x_3 + 2x_4 = 1, \quad (2.6)$$

$$2x_1 + 4x_3 + 3x_4 = 4, \quad (2.7)$$

$$x_i \geq 0, \quad i = 1, 2, 3, 4:$$

Նկատենք, որ (2.7)-ը հանդիսանում է (2.5)-ի և (2.6)-ի գումարը: (2.5)-ը և (2.6)-ը գծորեն անկախ են, այսինքն մատրիցի ռանգը 2 է: $x = (2, 1, 0, 0)$ և $y = (0, 5/3, 0, 4/3)$ ունեն 2-ական դրական բաղադրիչներ: Նրանց բազիսները համապատասխանաբար՝ $\{a^1, a^2\}$ և $\{a^2, a^4\}$: $z = (0, 0, 1, 0)$ անկյունային կետը ունի միայն 1 դրական բաղադրիչ: Նրան

համապատասխանում են $\{a^1, a^3\}$, $\{a^2, a^3\}$, $\{a^3, a^4\}$ բազիսները: $v = (1, 1/2, 1/2, 0) \in X$, բայց չի հանդիսանում անկյունային, քանի որ նրա դրական բաղադրիչների քանակը մեծ է r -ից:

Օրինակ 2:

Ենթադրենք X բազմությունը որոշվում է հետրյալ սահմանափակումներով.

$$x_1 + x_2 + 2x_3 = 1,$$

$$x_1 - x_2 + 2x_3 = 1,$$

$$x_i \geq 0, \quad i = 1, 2, 3:$$

Պայմանների մատրիցի ռանգը $r = 2$: X բազմությունը ունի 2 անկյունային կետ՝ $x = (1, 0, 0)$ և $x = (0, 0, 1/2)$: Նրանցից յուրաքանչյուրը ունի միակ բազիս, համապատասխանաբար՝ $\{a^1, a^2\}$ և $\{a^2, a^3\}$: $x = 0$ կետը X բազմության համար հանդիսանում է անկյունային կետ այն և միայն այն դեպքում, եթե $b = 0$: Որպես նրա բազիս կարելի է վերցնել A մատրիցի ցանկացած r գծորեն անկախ այուներ: Ենթադրենք $\sigma = \{a^{i_1}, \dots, a^{i_r}\} \bar{x}$ անկյունային կետի բազիսն է, $B_\sigma = [a^{i_1}, \dots, a^{i_r}]$ -ը բազիսային մատրիցն է, $I_\sigma = \{i_1, \dots, i_r\}$ -ը բազիսային ինդեքսների բազմությունն է:

\bar{x} անկյունային կետի բազիսային բաղադրիչներից կազմված վեկտորը նշանակենք $\bar{x}^\sigma = (\bar{x}_1^\sigma, \dots, \bar{x}_r^\sigma) = (\bar{x}_{i_1}, \dots, \bar{x}_{i_r})$: Քանի որ $\bar{x}_i = 0$, $i \notin I_\sigma$, ապա

$$b = A\bar{x} = \sum_{i=1}^n \bar{x}_i a^i = \sum_{i \in I_\sigma} \bar{x}_i a^i + \sum_{i \notin I_\sigma} \bar{x}_i a^i = \sum_{i \in I_\sigma} \bar{x}_i a^i = B_\sigma \bar{x}^\sigma: \quad (2.8)$$

Քանի որ B_σ -ի ռանգը r է, (2.8)-ից հետևում է, որ \bar{x}^σ -ն $B_\sigma x^\sigma = b$ գծային հանրահաշվական հավասարումների համակարգի (r անհայտներով) միակ լուծումն է, որտեղ $x^\sigma = (x_1^\sigma, \dots, x_r^\sigma) = (x_{i_1}, \dots, x_{i_r})$: Այսպիսով անկյունային կետի բազիսը միարժեքորեն որոշում է բազիսային բաղադրիչները, հետևաբար և անկյունային կետը: Հակառակը, ընդհանրապես ասած, ճիշտ չէ: Այդ իմաստով առաջնային համարվում է անկյունային կետի բազիսը, ոչ թե անկյունային կետը:

Ենթադրենք s -ը \bar{x} անկյունային կետի 0-ից տարբեր բաղադրիչների քանակն է: Այդ դեպքում սահմանափակումների քանակը, որոնք որոշում են X բազմությունը և \bar{x} կետում բավարարվում են հավասարման տեսքով, հավասար է $m + (n - s) \geq m + (n - r)$: Այստեղից հետևում է, որ $r < m$ նշանակում է, որ

$$a_{j1}x_1 + \dots + a_{jn}x_n = b_j, \quad j = 1, \dots, m, \quad (2.9)$$

սահմանափակումների մեջ գոյություն ունի $m - r$ գծորեն կախված սահմանափակումներ: Եթե (2.9) համակարգը համատեղ է, ապա այդ գծորեն կախված սահմանափակումները կարելի է ուղղակի ջնջել համակարգից, առանց պլանների բազմության համար վնասի: Այդ իսկ պատճառով հետագա շարադրանքում կհամարենք $X \neq \emptyset$ և $r = \text{rank}(A) = m$: Բացի այդ կհամարենք, որ $m < n$, հակառակ դեպքում X -ը կարող է բաղկացած լինել 1 կետից, որը, բնական է, հետաքրքրություն չի ներկայացնում:

Ենթադրենք $\text{rank}(A) = m < n$: Եթե \bar{x} անկյունային կետի 0-ից տարբեր բաղադրիչների քանակը m է, ապա սահմանափակումների քանակը, որոնք այդ կետում վերածվում են հավասարման՝ $m + (n - m) = n$: Այսինքն \bar{x} անկյունային կետը չվերասերված է: Հակառակը, եթե \bar{x} անկյունային կետը չվերասերված է, ապա այն X բազմությունը որոշող $n + m$ սահմանափակումներից n հատը վերածում է հավասարման տեսքի, իսկ մնացած սահմանափակումները բավարարվում են ինչպես իմաստ անհավասարումներ: Հետևաբար չվերասերված անկյունային կետը ունի ճիշտ m դրական բաղադրիչ:

Այսպիսով \bar{x} անկյունային կետը համարվում է չվերասերված տեսքի, այն և միայն այն դեպքում, եթե նրա դրական բաղադրիչների քանակը հավասար է m (սահմանափակումների քանակին) և վերասերված տեսքի է, եթե նրա դրական բաղադրիչների քանակը փոքր է m -ից: Չվերասերված տեսքի անկյունային կետը ունի միակ բազիս, վերասերված տեսքի անկյունային կետը կարող է ունենալ մի քանի բազիսներ: Օրինակ 1-ում (2.7) գծորեն կախված սահմանափակման հեռացումից հետո x և y անկյունային կետերը չվերասերված տեսքի են, իսկ z անկյունային կետը վերասերված տեսքի է: Օրինակ 2-ում երկու անկյունային կետերն էլ վերասերված տեսքի են:

Լեմմա: Եթե x վեկտորը հանդիսանում է a^1, \dots, a^k ոչ զրոյական վեկտորների ոչ բացասական գծային համակցությունը, այսինքն

$$x = \sum_{i=1}^k \lambda_i a^i, \quad \lambda_i \geq 0, \quad i = 1, \dots, k,$$

ապա x վեկտորը կարելի է ներկայացնել a^1, \dots, a^k վեկտորների գծորեն անկախ ենթահամակարգի վեկտորների ոչ բացասակ գծային համակցության տեսքով:
Ապացույցը տե՛ս [2]:

Դիցուք X -ը ԳԾ կանոնական խնդրի պլանների բազմությունն է ((1.1) բազմություն): $x \in X$ վեկտորի կրող է կոչվում իր ոչ զրոյական բաղադրիչների ինդեքսների բազմությունը՝ $I(x) = \{ i: x_i > 0 \}$:

Թեորեմ: Եթե X բազմությունը դատարկ չէ, ապա այն ունի անկյունային կետեր: Այդ դեպքում՝ կամայական $x \in X$ համար, գոյություն ունի այնպիսի \bar{x} անկյունային կետ, որ՝ $I(\bar{x}) \subset I(x)$:

Ապացույցը տե՛ս [2]:

Դիտարկենք ԳԾ կանոնական խնդիրը.

$$\langle c, x \rangle \rightarrow \max; \quad x \in X, \quad (2.10)$$

$$X = \{ x \in R^n: Ax = b, \quad x \geq 0 \}, \quad (2.11)$$

որտեղ՝ $A \in M_{m,n}$, $\text{rank}(A) = m < n$:

Դիցուք $\sigma = \{ a^{i_1}, \dots, a^{i_m} \}$ -ն $\bar{x} \in X$ անկյունային կետերի բազիսն է, B_σ -ն՝ բազիսային մատրիցը, I_σ -ն՝ բազիսային ինդեքսների բազմությունը, $\bar{x}^\sigma = [\bar{x}_i]_{i \in I_\sigma}$: (2.8)-ից ունենք՝

$$\bar{x}^\sigma = B_\sigma^{-1}b: \quad (2.12)$$

Ներմուծենք $c^\sigma = [c_i]_{i \in I_\sigma}$ վեկտորը: Այդ դեպքում՝

$$\langle c, \bar{x} \rangle = \sum_{i \in I_\sigma} c_i \bar{x}_i + \sum_{i \notin I_\sigma} c_i \bar{x}_i = \langle c^\sigma, \bar{x}^\sigma \rangle:$$

(2.12)-ից \bar{x}^σ -ն տեղադրելով այստեղ, կստանանք՝

$$\langle c, \bar{x} \rangle = \langle c^\sigma, B_\sigma^{-1}b \rangle = \langle B_\sigma^{-T}c^\sigma, b \rangle, \quad (2.13)$$

որտեղ՝ $B_\sigma^{-T} \triangleq (B_\sigma^{-1})^T = (B_\sigma^T)^{-1}$: Հիմնվելով հետևյալին՝

$$y = y(\sigma) = B_\sigma^{-T}c^\sigma,$$

(2.13) հավասարումը կարելի է գրել այս տեսքով՝

$$\langle c, \bar{x} \rangle = \langle b, y \rangle :$$

Այստեղից ստանում ենք՝ $Ax = b$ համակարգի կամայական լուծման համար, այդ թվում և կամայական $x \in X$ համար

$$\langle c, \bar{x} \rangle = \langle Ax, y \rangle = \langle A^T y, x \rangle :$$

Հետևաբար՝ կամայական $x \in X$ համար

$$\langle c, x \rangle - \langle c, \bar{x} \rangle = \langle c - A^T y, x \rangle : \quad (2.14)$$

Ներմուծենք հետևյալ վեկտորը՝

$$\Delta = \Delta(\sigma) = A^T y - c: \quad (2.15)$$

(2.14) բանաձևը կընդունի այս տեսքը.

$$\langle c, x \rangle - \langle c, \bar{x} \rangle = -\langle \Delta, x \rangle : \quad (2.16)$$

(2.16)-ը կոչվում է նպատակային ֆունկցիայի աճի բանաձև \bar{x} անկյունային կետում (քանի որ Δ վեկտորը կախված է, ընդհանուր առմամբ, բազիսից, (2.16) բանաձևը ավելի ճիշտ կլիներ անվանել σ բազիսում աճի բանաձև):

Թեորեմ: (Անկյունային կետի օպտիմալության չափանիշ) Դիցուք \bar{x} -ն անկյունային կետ է (2.10), (2.11) կանոնական խնդրում: Եթե գոյություն ունի նրա այնպիսի σ բազիս, որ

$$\Delta(\sigma) \geq 0, \quad (2.17)$$

ապա \bar{x} -ը (2.10), (2.11) խնդրի լուծումն է:

Ապացույցը տե՛ս [2]:

Օրինակ 3: Դիցուք ԳԾ խնդիրն ունի հետևյալ տեսքը՝

$$-x_1 - x_2 - 2x_3 - 2x_4 \rightarrow max;$$

$$x_1 + x_2 + 3x_3 + x_4 = 3,$$

$$x_1 - x_2 + x_3 + 2x_4 = 1,$$

$$x_i \geq 0, \quad i = 1,2,3,4:$$

$\bar{x} = (0,0,1,0)$ վերասերված անկյունային կետ է՝ $\sigma_1 = \{a^1, a^3\}$, $\sigma_2 = \{a^2, a^3\}$ և $\sigma_3 = \{a^3, a^4\}$ բազիսներով: Դժվար չէ նկատել, որ $\Delta(\sigma_1) = (0, 1, 0, 1/2) \geq 0$ և հետևաբար \bar{x} կետը դիտարկվող խնդրի լուծումն է: Նկատենք, որ (2.17) պայմանը տեղի ունի նաև σ_2 բազիսում և տեղի չունի՝ σ_3 -ում. $\Delta(\sigma_2) = (1/2, 0, 0, 7/4)$, $\Delta(\sigma_3) = (-1/5, 7/5, 0, 0)$:

Δ վեկտորը կոչվում է գնահատականների վեկտոր (հարաբերական գնահատականների, քանի որ կախված է բազիսից), իսկ նրա Δ_i , $i = 1, \dots, n$ բաղադրիչները՝ գնահատականներ (հարաբերական գնահատականներ): (2.15)-ից բխում է, որ

$$\Delta_i = \langle a^i, y \rangle - c_i, \quad i = 1, \dots, n:$$

(2.13)-ից y -ը տեղադրելով այստեղ, կստանանք՝

$$\Delta_i = \langle a^i, B_\sigma^{-T} c^\sigma \rangle - c_i = \langle B_\sigma^{-1} a^i, c^\sigma \rangle - c_i: \quad (2.18)$$

Տեղադրենք՝

$$\lambda^i = B_\sigma^{-1} a^i: \quad (2.19)$$

Այդ դեպքում (2.18) բանաձևը կընդունի հետևյալ տեսքը.

$$\Delta_i = \langle c^\sigma, \lambda^i \rangle - c_i, \quad i = 1, \dots, n: \quad (2.20)$$

Քանի որ

$$B_\sigma^{-1} B_\sigma = [B_\sigma^{-1} a^{i_1} \dots B_\sigma^{-1} a^{i_m}] = [\lambda^{i_1} \dots \lambda^{i_m}] = E,$$

(որտեղ E -ն միավոր մատրիցն է) ապա $\lambda^{i_j} = e^j$, որտեղ $e^j \in \mathbb{R}^m - j$ -րդ միավոր վեկտորն է: Հետևաբար՝

$$\Delta_{i_j} = \langle c^\sigma, e^j \rangle - c_{i_j} = c_{i_j} - c_{i_j} = 0, \quad j = 1, \dots, m:$$

Այսինքն՝

$$\Delta_i = 0, \quad \forall i \in I_\sigma:$$

Այստեղից հետևում է, որ (2.16) աճի բանաձևը և (2.17) օպտիմալության չափանիշը կարելի է գրել համապատասխանաբար հետևյալ տեսքով՝

$$\langle c, x \rangle - \langle c, \bar{x} \rangle = - \sum_{i \notin I_\sigma} \Delta_i x_i, \quad (2.21)$$

$$\Delta_i \geq 0, \quad \forall i \notin I_\sigma:$$

Դիտողություն: (2.19)-ից հետևում է, որ

$$a^i = B_\sigma \lambda^i = \sum_{j=1}^m \lambda_{ji} a^{ij},$$

որտեղ $\lambda_{ji} = \lambda_j^i$ -ն λ^i վեկտորի j -րդ բաղադրիչն է: Այսպիսով λ^i -ի բաղադրիչները իրենցից ներկայացնում են a^i վեկտորի բազիսային վեկտորով վերլուծության գործակիցները:

2.3. Մողիֆիկացված և աղյուսակային սիմպլեքս ալգորիթմներ

Դիտարկենք կանոնական տեսքի ԳԾ խնդիրը:

$$f(x) = \langle c, x \rangle \rightarrow \max; \quad x \in X, \quad (2.22)$$

$$X = \{ x \in \mathbb{R}^n : Ax = b, \quad x \geq 0 \}, \quad (2.23)$$

ນັບຕໍ່ລວມ $A \in M_{m,n}$, $rank(A) = m < n$:

Այժմ ենթադրենք, որ (2.22), (2.23) խնդրում բոլոր անկյունային կետերը չվերասերված տեսքի են (հետազայում կազատվենք այս ենթադրությունից): Այս դեպքում (2.22), (2.23) խնդիրը կոչվում է չվերասերված տեսքի ԳԾ խնդիր:

Ենթադրենք \bar{x} -ը X բազմության անլյունային կետ է՝ $\sigma = (a^{i_1}, \dots, a^{i_m})$ բազիտով։ Ենթադրենք օպտիմալության չափանիշը \bar{x} կետում տեղի չունի՝ գոյություն ունեն $i \notin I_\sigma$ ինդեքսներ այնպիսին, որ դրանց համապատասխան գնահատականները $\Delta_i < 0$: Այդ դեպքում (2.21) աճի բանաձևից հետևում է, որ $\langle c, x \rangle > \langle c, \bar{x} \rangle$, $\forall x \in X$ վեկտորի համար, այնպիսին, որ

$$\sum_{i \notin I_\sigma} \Delta_i x_i < 0: \quad (2.24)$$

Ակնհայտ է, որ իմաստ ունի փորձել փնտրել այնպիսի վեկտոր, անգամ եթե այն չլինի դիտարկվող խնդրի լուծումը, որը կտա ավելի լավ մոտեցում փնտրվող օպտիմալ արժեքին, քան \bar{x} -ը:

Դիտարկենք (2.24) պայմանին բավարարող $x \in X$ վեկտորի կառուցման հետևյալ հասարակ էղանակը: Վերցնենք ցանկացած ինդեքս $i \notin I_\sigma$, որի համար $\Delta_i < 0$: Թող լինի $i = k$ և տեղադրենք

$$x_i = 0, i \notin I_\sigma, i \neq k, \quad (2.25)$$

$$x_k = \theta, \quad (2.26)$$

որտեղ θ -ն ինչ-որ (այժմ դեռ ցանկացած) դրական թիվ է: (2.21)-ից կստանանք

$$\langle c, x \rangle - \langle c, \bar{x} \rangle = -\theta \Delta_k \quad (2.27)$$

(2.25) և (2.26) բանաձևերը որոշում են x վեկտորի այն x_i բաղադրիչները, որոնց համար $i \notin I_\sigma$: Մնացած բաղադրիչները գտնենք $Ax = b$ համակարգից: Դրա համար նշանակենք $x^\sigma = (x_1^\sigma, \dots, x_m^\sigma) = (x_{i_1}, \dots, x_{i_m})$ և տեղադրենք (2.25)-ը և (2.26)-ը $Ax = b$ համակարգի մեջ՝

$$Ax = \sum_{i \in I_\sigma} x_i a^i + \sum_{i \notin I_\sigma} x_i a^i = B_\sigma x^\sigma + \theta a^k = b:$$

Այստեղից հաշվի առնելով (2.12)-ը և (2.19)-ը՝

$$x^\sigma = \bar{x}^\sigma - \theta \lambda^k \quad (2.28)$$

կամ բաղադրիչային գրառմամբ՝

$$x_{i_j} = \bar{x}_{i_j} - \theta \lambda_{j_k}, \quad j = 1, \dots, m: \quad (2.29)$$

Մնաց ընտրել $\theta > 0$ այնպես, որ x^σ վեկտորի բոլոր x_{i_j} բաղադրիչները լինեն ոչ բացասական: Հնարավոր է 2 դեպք.

Դեպք 1. $\lambda^k \leq 0$:

Այս դեպքում, քանի որ $\bar{x}^\sigma > 0$, ապա կամայական $\theta > 0$ -ի համար՝ $x^\sigma > 0$: Հետևաբար, կամայական $\theta > 0$ -ի համար (2.25), (2.26) և (2.28) բանաձևերով կառուցված $x = x(\theta)$ պատկանում է X բազմությանը: Եթե $\theta \rightarrow +\infty$, (2.27)-ից կստանանք, որ $\langle c, x \rangle \rightarrow +\infty$: Բայց դա նշանակում է, որ դիտարկվող խնդիրը լուծում չունի. $\sup_X \langle c, x \rangle = +\infty$ (նպատակային ֆունկցիան վերևից սահմանափակ չէ պլանների բազմության վրա):

Դեպք 2. Գոյություն ունեն $j \in \{1, \dots, m\}$ ինդեքսներ, այնպիսին որ $\lambda_{j_k} > 0$:

Ներմուծենք $J_k = \{j: \lambda_{j_k} > 0\}$ բազմությունը: (2.8)-ից հետևում է, որ եթե $j \notin J_k$, ապա կամայական $\theta > 0$ -ի համար $x_{i_j} > 0$: $j \in J_k$ -ի համար պետք է տեղի ունենա հետևյալ պայմանը.

$$\bar{x}_{i_j} - \theta \lambda_{j_k} \geq 0:$$

Այստեղից կստանանք θ պարամետրը ընտրելու պայման.

$$0 < \theta \leq \min_{j \in J_k} \frac{\bar{x}_{i_j}}{\lambda_{j_k}} = \min_{j \in J_k} \frac{\bar{x}_j^\sigma}{\lambda_{j_k}}:$$

(2.27) բանաձևից հետևում է, որ որքան մեծ է θ պարամետրի արժեքը, այդքան մեծ է նպատակային ֆունկցիայի արժեքը $x = x(\theta)$ կետում.

$$\langle c, x \rangle = \langle c, \bar{x} \rangle + \theta |\Delta_k|:$$

Հետևաբար, լավագույն դեպքում՝

$$\theta = \theta_k = \min_{j \in J_k} \frac{\bar{x}_j^\sigma}{\lambda_{j_k}}: \quad (2.30)$$

Թեորեմ: (2.25), (2.26), (2.28) և (2.30) բանաձևերով կառուցված $x = x(\theta)$ կետը հանդիսանում է X բազմության անկյունային կետ:

Ապացույցը տե՛ս [2]:

Այսպիսով եթե \bar{x} անկյունային կետում տեղի չի ունենում օպտիմալության չափանիշը, ապա, ընտրելով կամայական k ինդեքս, այնպիսին որ $\Delta_k < 0$, կա՛մ հնարավոր է պնդել նպատակային ֆունկցիայի անսահմանափակությունը պլանների բազմության վրա, կա՛մ օգտագործելով (2.25), (2.26), (2.28) և (2.30) բանաձևերը կարելի է կառուցել x անկյունային կետ, այնպիսին, որ նպատակային ֆունկցիայի արժեքը ավելի մեծ լինի քան \bar{x} կետում: x կետը կարելի է վերցնել որպես նախնական և կրկնել նրա համար ամբողջ պրոցեդուրան. ստուգել օպտիմալության չափանիշը, և եթե այն տեղի չունի, կա՛մ պնդել նպատակային ֆունկցիայի անսահմանափակությունը, կա՛մ կառուցել նոր անկյունային կետ, որում նպատակային ֆունկցիայի արժեքը կլինի ավելի մեծ, քան x կետում: Եվ այդպես շարունակ: Նկարագրված իտերացիոն գործընթացը կոչվում է սիմպլեքս մեթոդ: Քանի որ ամեն մի իտերացիայում նպատակային ֆունկցիայի արժեքը խիստ մոնուոն աճում է, ապա ոչ մի անկյունային կետ չի կարող հանդիպել կրկնակի անգամ և քանի որ անկյունային կետերը վենցավոր քանակի են, ապա վերջավոր թվով իտերացիաների արդյունքում կա՛մ կգտնվի անկյունային կետ, որը կհանդիսանա խնդրի լուծումը, կա՛մ կպարզվի, որ $\sup_x \langle c, x \rangle = +\infty$: Գործնականում սիմպլեքս մեթոդի իտեսացիաների քանակը սովորաբար չի գերազանցում $2m - 4m$ -ը:

Բերված հիմնավորումները միաժամանակ ծառայում են որպես ապացույց հետևյալ թեորեմների համար:

Թեորեմ: Եթե $X_* = \underset{X}{\operatorname{Argmax}} \langle c, x \rangle \neq \emptyset$, ապա այն պարունակում է X բազմության անյունային կետերը: Անկյունային կետ $\bar{x} \in X_*$ այն և միայն այն դեպքում, եթե նրա համար տեղի ունի օպտիմալության չափանիշը:

Թեորեմ: Եթե (2.22), (2.23) խնդրի նպատակային ֆունկցիան վերևից սահմանափակված է պլանների բազմության վրա, ապա խնդիրը ունի լուծում ($X_* \neq \emptyset$):

Նոր x անկյունային կետի բազիսը ձևավորվում է հին \bar{x} անկյունային կետի բազիսից՝ փոխարինելով a^{i_s} վեկտորը՝ a^k վեկտորով: Այս փոխարինումը և կանոնը, որով այն ստացվում է, ըստ Էության կազմում են սիմպլեքս մեթոդի միջուկը, որի ալգորիթմը այժմ կարող է ներկայացվել:

Ալգորիթմ 1:

Քայլ 0: Տալ նպատակային ֆունկցիայի և վեկտորը, պայմանների A մատրիցը, սահմանափակումների b վեկտորը և $I_\sigma = \{i_1, \dots, i_m\}$ բազիսային ինդեքսների բազմությունը: Կազմել $B_\sigma = [a^{i_1}, \dots, a^{i_m}]$ բազիսային մատրիցը և $c^\sigma = (c_{i_1}, \dots, c_{i_m})$ վեկտորը:

Քայլ 1: Հաշվել B_σ^{-1} մատրիցը և $\bar{x}^\sigma = B_\sigma^{-1}b$ վեկտորը:

Քայլ 2: Հաշվել $y = B_\sigma^{-T}c^\sigma$ վեկտորը և $\Delta_i = \langle y, a^i \rangle - c_i$, $i \notin I_\sigma$ գնահատականները:

Քայլ 3: Եթե $\Delta_i \geq 0$ բոլոր $i \notin I_\sigma$ համար, ապա տեղադրել $x_i^* = 0$ բոլոր $i \notin I_\sigma$, $x_{i_j}^* = \bar{x}_j^\sigma$, $j = 1, \dots, m$, հաշվել $f(x^*) = \langle c, x^* \rangle = \langle c^\sigma, \bar{x}^\sigma \rangle$ և կանգնել՝ x^* -ը օպտիմալ պլանն է, հակառակ դեպում անցնել քայլ 4:

Քայլ 4: Ընտրել ցանկացած $k \notin I_\sigma$ ինդեքս, այնպիսին որ $\Delta_k < 0$ և հաշվել $\lambda^k = B_\sigma^{-1}a^k$:

Քայլ 5: Եթե $\lambda_{jk} \leq 0$ բոլոր $j = 1, \dots, m$ համար ապա կանգնել՝ $\sup_x \langle c, x \rangle = +\infty$,

հակառակ դեպքում անցնել քայլ 6:

Քայլ 6: Հաշվել $s = \arg \min_{j: \lambda_{jk} > 0} \frac{\bar{x}_j^\sigma}{\lambda_{jk}}$:

Քայլ 7: I_σ բազմությունում i_s ինդեքսը փոխարինել k ինդեքսով, B_σ մատրիցում a^{i_s} վեկտորը՝ a^k -վեկտորով, c^σ վեկտորում c_{i_s} բաղադրիչը՝ c_k -ով (այսինքն տեղադրել $i_s = k$, $a^{i_s} = a^k$, $c_{i_s} = c_k$): Անցնել քայլ 1:

Ալգորիթմ 1-ը իրենից ներկայացնում է սիմպլեքս մեթոդի ամենաընդհանուր, սկզբունքային, ամենատարածված սխեմաներից մեկը, որը կոչվում է մոդիֆիկացված սիմպլեքս մեթոդ: Քննարկենք նրա իրականացման հետ կապված մի քանի հարցեր:

1) Որպես k -րդ ինդեքս 4-րդ քայլում սովորաբար խորհուրդ է տրվում ընտրել այն ինդեքսը, որի համար համապատասխան Δ_k -ն փոքրագույնն է (քացարձակ արժեքով մեծագույնը): Որպես կանոն այդպիսի ընտրությունը քշացնում է սիմպլեքս մեթոդի խորացիաների քանակը, բայց պահանջում է բոլոր Δ_i , $i \notin I_\sigma$ գնահատականների հաշվարկ: Ակնհայտ է, որ կարելի է չհաշվել բոլոր գնահատականները, եթե որպես k -րդ ինդեքս ընտրվի առաջին 0-ից փոքր գնահատականը:

2) Սիմպլեքս մեթոդի բոլոր խորացիաներում, բացի առաջինից, B_σ բազիսային մատրիցը կարելի է ամեն անգամ սկզբից չհակադարձել, այլ՝ B_σ^{-1} -ը հաշվարկել անդրադարձ առնչություններով: Այդ առնչությունների դուրս բերման համար հին անկյունային կետի բազիսային մատրիցը նշանակենք B_σ , իսկ նոր անկյունային կետինը՝ B_τ : Այսինքն՝

$$B_\sigma = [a^{i_1} \dots a^{i_{s-1}} a^{i_s} a^{i_{s+1}} \dots a^{i_m}],$$

$$B_\tau = [a^{i_1} \dots a^{i_{s-1}} a^k a^{i_{s+1}} \dots a^{i_m}]:$$

Այդ դեպքում, քանի որ $B_\sigma^{-1} a^{ij} = e^j$, $j = 1, \dots, m$, իսկ $B_\sigma^{-1} a^k = \lambda^k$, ապա

$$B_\sigma^{-1} B_\tau = [e^1 \dots e^{s-1} \lambda^k e^{s+1} \dots e^m] = E_s:$$

Այստեղից՝

$$B_\tau^{-1} = E_s^{-1} B_\sigma^{-1}: \quad (2.31)$$

E_s^{-1} մատրիցը ունի հետևյալ տեսքը՝

$$E_s^{-1} = [e^1 \dots e^{s-1} \eta^k e^{s+1} \dots e^m], \quad (2.32)$$

որտեղ η^k -ի բաղադրիչները հաշվարկվում են հետևյալ բանաձևով.

$$\eta_j^k = \begin{cases} -\lambda_{jk}/\lambda_{sk}, & j = 1, \dots, m, \quad j \neq s, \\ 1/\lambda_{sk}, & j = s: \end{cases} \quad (2.33)$$

$B_\sigma^{-1}, B_\tau^{-1}$ մատրիցների էլեմենտները նշանակենք համապատասխանաբար $\bar{\beta}_{ji}$ և β_{ji} :

(2.31) - (2.33)-ից հետևում է, որ

$$\beta_{ji} = \bar{\beta}_{ji} - \frac{\lambda_{jk}}{\lambda_{sk}} \bar{\beta}_{si}, \quad j, i = 1, \dots, m, \quad j \neq s, \quad (2.34)$$

$$\beta_{si} = \frac{1}{\lambda_{sk}} \bar{\beta}_{si}, \quad i = 1, \dots, m: \quad (2.35)$$

Սիմպլեքս մեթոդի իրականացման մյուս մոտեցումը կայանում է նրանում, որ բոլոր խորացիաներում, բացի առաջինից, ընդհանրապես գործ չունենանք հակադարձ բազիսային մատրիցի հետ՝ դրա փոխարեն անդրադարձ կերպով հաշվարկենք \bar{x}^σ վեկտորի բաղադրիչները, նպատակային ֆունկցիայի արժեքը, λ^i վեկտորների

բաղադրիչները և Δ_i գնահատականները: Այս բոլոր պարամետրերը հարմար է միավորել $(m+1) \times (n+1)$ չափանի Հ մատրիցի մեջ:

$$\Lambda = [\lambda_{ji}]_{j,i=0}^{m,n} = \begin{bmatrix} < c, \bar{x} > & \Delta_1 \dots \Delta_n \\ \bar{x}^\sigma & \lambda^1 \dots \lambda^n \end{bmatrix}:$$

Հ մատրիցը կոչվում է \bar{x} անկյունային կետին (ավելի ճիշտ σ բազիսին) համապատասխան սիմպլեքս աղյուսակ:

Միօրինակության համար ներմուծենք հետևյալ նշանակումները.

$$\Delta_0 = < c, \bar{x} >, \quad a^0 = b \text{ և } \text{տեղադրենք } c_0 = 0: \text{Այդ դեպքում } \bar{x}^\sigma = B_\sigma^{-1} a^0 = \lambda^0,$$

$$\Delta_0 = < c^\sigma, \bar{x}^\sigma > = < c^\sigma, \lambda^0 > - c_0, \text{ իսկ } \Lambda\text{-ն } \text{կնդունի } \text{հետևյալ } \text{տեսքը.}$$

$$\Lambda = \begin{bmatrix} \Delta_0 & \Delta_1 \dots \Delta_n \\ \lambda^0 & \lambda^1 \dots \lambda^n \end{bmatrix}:$$

Այսպիսով Հ մատրիցի կեմենտների համար՝

$$\lambda_{ji} = \lambda_j^i, \quad j = 1, \dots, m, \quad i = 0, \dots, n, \quad (2.36)$$

$$\lambda_{0i} = \Delta_i = < c^\sigma, \lambda^i > - c_i = \sum_{j=1}^m c_{ij} \lambda_{ji} - c_i, \quad i = 0, \dots, n:$$

(2.37)

(Այստեղ գնահատականների հաշվարկման համար օգտագործվել է (2.20) բանաձևը):

Նոր x անկյունային կետին համապատասխանում է նոր բազիս՝

$$\tau = \{ a^{i_1}, \dots, a^{i_{s-1}} a^k a^{i_{s+1}}, \dots, a^{i_m} \}$$

և նոր սիմպլեքս աղյուսակ՝

$$\Gamma = [\gamma_{ji}]_{j,i=0}^{m,n} = \begin{bmatrix} \Delta_0(\tau) & \Delta_1(\tau) \dots \Delta_n(\tau) \\ \gamma^0 & \gamma^1 \dots \gamma^n \end{bmatrix},$$

որտեղ $\gamma^i = B_\tau^{-1} a^i$, $\Delta_i(\tau) = < c^\tau, \gamma^i > - c_i$, իսկ $c^\tau = (c_{i_1}, \dots, c_{i_{s-1}}, c_k, c_{i_{s+1}}, \dots, c_{i_m})$: Բնչպես և Հ աղյուսակում, Γ աղյուսակի զրոյական այան կեմենտներն են նպատակային ֆունկցիայի արժեքը և անկյունային կետի բազիսային բաղադրիչները՝ $\Delta_0(\tau) = < c, x >$, $\gamma^0 = x^\tau$:

Ցույց տանք հին և նոր սիմպլեքս աղյուսակների տարրերի միջև կապը: Համաձայն (2.31) - (2.33) առնչությունների ցանկացած $i = 0, \dots, n$ համար

$$\gamma^i = B_\tau^{-1} a^i = E_s^{-1} B_\sigma^{-1} a^i = E_s^{-1} \lambda^i,$$

կամ, համաձայն բաղադրիչային զրառման՝

$$\gamma_{ij} = \gamma_j^i = \lambda_{ij} - \frac{\lambda_{jk}}{\lambda_{sk}} \lambda_{si}, \quad j = 1, \dots, m, \quad j \neq s,$$

$$\gamma_{si} = \gamma_s^i = \frac{\lambda_{si}}{\lambda_{sk}}$$

Այդ դեպքում՝

$$\gamma_{0i} = \Delta_i(\tau) = \langle c^\tau, \gamma^i \rangle - c_i = \sum_{\substack{j=1 \\ j \neq s}}^m c_{ij} \gamma_{ji} + c_k \gamma_{si} - c_i = \sum_{\substack{j=1 \\ j \neq s}}^m c_{ij} \left(\lambda_{ji} - \frac{\lambda_{jk}}{\lambda_{sk}} \lambda_{si} \right) + c_k \frac{\lambda_{si}}{\lambda_{sk}} - c_i:$$

Այստեղից, քանի որ $\lambda_{ji} - \frac{\lambda_{jk}}{\lambda_{sk}} \lambda_{si} = 0$, ապա $j = s$ դեպքում՝

$$\begin{aligned} \gamma_{0i} &= \sum_{j=1}^m c_{ij} \left(\lambda_{ji} - \frac{\lambda_{jk}}{\lambda_{sk}} \lambda_{si} \right) + c_k \frac{\lambda_{si}}{\lambda_{sk}} - c_i = \left(\sum_{j=1}^m c_{ij} \lambda_{ji} - c_i \right) - \frac{\lambda_{si}}{\lambda_{sk}} \left(\sum_{j=1}^m c_{ij} \lambda_{jk} - c_k \right) = \\ &= \Delta_i(\sigma) - \frac{\lambda_{si}}{\lambda_{sk}} \Delta_k(\sigma) = \lambda_{0i} - \frac{\lambda_{si}}{\lambda_{sk}} \lambda_{0k}: \end{aligned}$$

Այսպիսով սիմպլեքս աղյուսակների վերահաշվարկի բանաձևերը Շ անկյունային կետից չ անկյունային կետ անցնելիս ունեն հետևյալ տեսքը.

$$\gamma_{ji} = \lambda_{ji} - \frac{\lambda_{jk}}{\lambda_{sk}} \lambda_{si}, \quad j = 0, \dots, m, \quad j \neq s, \quad i = 0, \dots, n, \quad (2.38)$$

$$\gamma_{si} = \frac{\lambda_{si}}{\lambda_{sk}}, \quad i = 0, \dots, n: \quad (2.39)$$

(2.19), (2.20) բանաձևերում λ_{sk} տարրը կոչվում է տանող տարր: A մատրիցի (հին սիմպլեքս աղյուսակի) s -երրդ տողը և k -երրդ սյունը, որոնց հատման կետում գտնվում է տանող տարրը, կոչվում են տանող տող և տանող սյուն:

Նաև ալգորիթմը կարելի է ներկայացնել սիմպլեքս աղյուսակների տեսքով:

Ալգորիթմ 2:

Քայլ 0: Տալ c նպատակային վեկտորը, պայմանների A մատրիցը, սահմանափակումների b վեկտորը և $I_\sigma = \{i_1, \dots, i_m\}$ բազիսային ինդեքսների բազմությունը:

Քայլ 1: (Սկզբնական A սիմպլեքս աղյուսակի կազմում)

- 1.1) Կազմել B_σ բազիսային մատրիցը, c^σ վեկտորը և հաշվարկել B_σ^{-1} -ը: Տեղադրել $a^0 = b$, $c_0 = 0$:
- 1.2) $i = 0, \dots, n$ համար հաշվարկել $\lambda^i = B_\sigma^{-1} a^i$ (սիմպլեքս աղյուսակի սյուները, առանց զրոյական տողի էլեմենտների): Տեղադրել $\lambda_{ji} = \lambda_j^i$, $j = 1, \dots, m$, $i = 0, \dots, n$:
- 1.3) $i = 0, \dots, n$ համար տեղադրել $\lambda_{0i} = \langle c^\sigma, \lambda^i \rangle - c_i$ (լրացնել սիմպլեքս աղյուսակի զրոյական տողը):

Քայլ 2: (Օպտիմալության պայմանների ստուգում)

Եթե $\lambda_{0i} \geq 0$, $i = 1, \dots, n$, ապա տեղադրել $x_i^* = 0$, $i \notin I_\sigma$, $x_{i_j}^* = \lambda_{j0}$, $j = 1, \dots, m$,

$\langle c, x^* \rangle = \lambda_{00}$ և կանգնել՝ x^* -ը օպտիմալ պլանն է, հակառակ դեպքում անցնել Քայլ 3:

Քայլ 3: (Տանող տարրի ընտրություն)

Ըստրել կամայական ինդեքս $k \in \{1, \dots, n\}$, $\lambda_{0k} < 0$:

Քայլ 4: (Նպատակային ֆունկցիայի անսահմանափակության ստուգում)

Եթե $\lambda_{jk} \leq 0$, բոլոր $j = 1, \dots, m$ համար, ապա կանգնել՝ $\sup_x \langle c, x \rangle = +\infty$, հակառակ դեպքում անցնել Քայլ 5:

Քայլ 5: (Տանող տողի ընտրություն)

Հաշվարկել $s = \operatorname{argmin}_{\lambda_{jk}} \left\{ \frac{\lambda_{j0}}{\lambda_{jk}} : \lambda_{jk} > 0, 1 \leq j \leq m \right\}$:

Քայլ 6: (Միմպլեքս աղյուսակի վերահաշվարկ)

I_σ բազմության մեջ տեղադրել $i_s = k$ (նոր բազիսային ինդեքսների բազմությունը ձևավորվում է հնի տեղը): Հաշվարկել Γ սիմպլեքս աղյուսակի տարրերը.

$$\gamma_{si} = \frac{\lambda_{si}}{\lambda_{sk}}, \quad i = 0, \dots, n,$$

$$\gamma_{ji} = \lambda_{ji} - \gamma_{si} \lambda_{jk}, \quad j = 0, \dots, m, \quad j \neq s, \quad i = 0, \dots, n,$$

Քայլ 7: (Անցում նոր իտերացիայի)

Տեղադրել $\Lambda = \Gamma$ և անցնել Քայլ 2:

Ալգորիթմ 2-ի համակարգչային իրականացման ժամանակ, այն կարելի է մոդիֆիկացնել՝ հանել ոչ պետքական զանգվածները, պայմանի օպերատորները և այլն:

Նկատենք, որ քանի որ $i \in I_\sigma$ համար գնահատականները՝ $\lambda_{0i} = \Delta_i = 0$, իսկ $\lambda^{ij} = e^j$, $j = 1, \dots, m$, ապա յուրաքանչյուր իտերացիայում սիմպլեքս աղյուսակի բազիսային այուները կարելի է լրացնել միանգամից, առանց հաշվարկելու միավոր այուները, որտեղ 1-ը գտնվում է j -րդ տողի և i_j -րդ այան ($j = 1, \dots, m$) հատման կետում: Միմպլեքս աղյուսակի ընդհանուր տեսքը ներկայացված է աղյուսակ 2.1 և 2.2-ում: Հարմարության համար աղյուսակների վերևում նշված են այուների համարները, ձախ կողմում նշված են զրոյական տողը՝ Δ և բազիսային ինդեքսները: Աղյուսակ 2.1-ում ընդգծված են զրոյական տողը (գնահատականների տող) և զրոյական այունը (որը պարունակում է անկյունային կետի բազիսային բաղադրիչները), ինչպես նաև տանող տողը և տանող այունը (ենթադրելով, որ $\Delta_k < 0$ և $\lambda_{sk} > 0$): Աղյուսակ 2.2-ը իրենից ներկայացնում է աղյուսակ

2.1-ի վերահաշվարկից հետո ստացված սիմպլեքս աղյուսակը (որոշակիության համար ենթադրվում է, որ աղյուսակ 2.1-ում $i_s = r$):

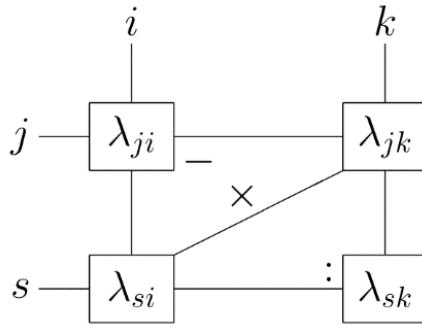
Δ	0	1	2	\cdots	i_1	\cdots	i_s	\cdots	k	\cdots	i_m	\cdots	n
$\langle c, \bar{x} \rangle$	Δ_1	Δ_2	\cdots	0	\cdots	0	\cdots		Δ_k	\cdots	0	\cdots	Δ_n
i_1	\bar{x}_{i_1}	λ_{11}	λ_{12}	\cdots	1	\cdots	0	\cdots	λ_{1k}	\cdots	0	\cdots	λ_{1n}
i_2	\bar{x}_{i_2}	λ_{21}	λ_{22}	\cdots	0	\cdots	0	\cdots	λ_{2k}	\cdots	0	\cdots	λ_{2n}
\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots
i_s	\bar{x}_{i_s}	λ_{s1}	λ_{s2}	\cdots	0	\cdots	1	\cdots	λ_{sk}	\cdots	0	\cdots	λ_{sn}
\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots
i_m	\bar{x}_{i_m}	λ_{m1}	λ_{m2}	\cdots	0	\cdots	0	\cdots	λ_{mk}	\cdots	1	\cdots	λ_{mn}

աղ. 2.1

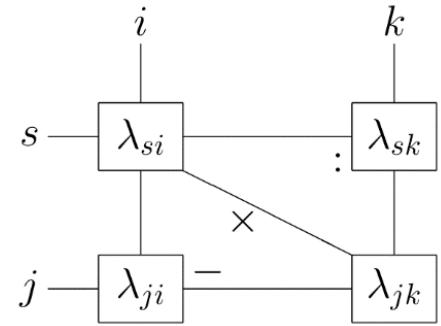
Δ	0	1	2	\cdots	i_1	\cdots	r	\cdots	k	\cdots	i_m	\cdots	n
$\langle c, x \rangle$	Δ_1	Δ_2	\cdots	0	\cdots	Δ_r	\cdots	0	\cdots	0	\cdots	Δ_n	
i_1	x_{i_1}	γ_{11}	γ_{12}	\cdots	1	\cdots	γ_{1r}	\cdots	0	\cdots	0	\cdots	γ_{1n}
i_2	x_{i_2}	γ_{21}	γ_{22}	\cdots	0	\cdots	γ_{2r}	\cdots	0	\cdots	0	\cdots	γ_{2n}
\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots
$i_s = k$	x_k	γ_{s1}	γ_{s2}	\cdots	0	\cdots	γ_{sr}	\cdots	1	\cdots	0	\cdots	γ_{sn}
\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots	\cdots
i_m	x_{i_m}	γ_{m1}	γ_{m2}	\cdots	0	\cdots	γ_{mr}	\cdots	0	\cdots	1	\cdots	γ_{mn}

աղ. 2.2

Սիմպլեքս աղյուսակների վերահաշվարկի բանաձևերը հեշտ է մտապահել հետևյալ վերահաշվակի կանոնի ձևով: Նախկին աղյուսակի տանող տողը բաժանվում է տանող տարրի վրա և մտցվում է նոր աղյուսակում իրեն համապատասխան տողի մեջ (բանաձև (2.39)): Նոր աղյուսակի սյուները լրացվում են այն համարներով, որոնք պատկանում են բազիսային ինդեքսների նոր բազմությանը (ըստ վերոնշյալի այդ սյուները բաղկացած են համապատասխան միավոր վեկտորներից): Նոր աղյուսակի մնացած տարրերը հաշվարկվում են (2.38) բանաձևից բխող ուղղանկյան կանոնի միջոցով. λ_{ji} , λ_{si} , λ_{jk} և λ_{sk} տարրերը գտնվում են ուղղանկյան գագաթներում, որը կազմված է j -րդ ու s -րդ (տանող) տողերի և i -րդ ու k -րդ (տանող) սյուների հատումից (նկ. 2.1, 2.2): Նոր աղյուսակի (j, i) -րդ տարրը որոշվում է հին աղյուսակի (j, i) -րդ տարրի և հակառակ անկյունագծի տարրերի արտադրյալի ու տանող տարրի հարաբերության տարբերություն:



նկ. 2.1



նկ. 2.2

Օրինակ 1: Օգտագործելով աղյուսակային սիմպլեքս մեթոդը լուծենք հետևյալ ԳԾ խնդիրը.

$$\begin{aligned}
 f(x) &= 2x_1 + x_2 + 3x_3 + 5x_4 \rightarrow \max; \\
 2x_1 + 3x_2 + x_3 + 2x_4 + x_5 &= 30, \\
 4x_1 + 2x_2 + x_3 + 2x_4 + x_6 &= 40, \\
 x_1 + 2x_2 + 3x_3 + x_4 + x_7 &= 25, \\
 x_i &\geq 0, \quad i = 1, \dots, 7;
 \end{aligned}$$

Այս խնդրում նախնական անկյունային կետը ակնհայտ է. $\bar{x} = (0,0,0,0,30,40,25)$: $I_\sigma = \{5,6,7\}$ բազմային խնդեքսների բազմությունն է, $B_\sigma = E^*$ բազմային մատրիցը, հետևաբար $\lambda^i = B_\sigma^{-1} a^i = a^i$: Քանի որ $c^\sigma = (0,0,0)$, ապա $\lambda_{00} = \Delta_0 = \langle c^\sigma, \bar{x}^\sigma \rangle = 0$, $\lambda_{0i} = \Delta_i = \langle c^\sigma, \lambda^i \rangle = -c_i = -c_i$: Այսպիսով նախնական սիպլեքս աղյուսակը կունենա հետևյալ տեսքը.

$$\left[\begin{array}{ccccccc} 0 & -c_1 & -c_2 & -c_3 & -c_4 & 0 & 0 & 0 \\ b & a^1 & a^2 & a^3 & a^4 & e^1 & e^2 & e^3 \end{array} \right];$$

Այս աղյուսակի ընդլայնված տեսքը աղյուսակ 2.3-ն է: Սիմպլեքս մեթոդի հաջորդ 2 խոերացիաները ներկայացված են 2.4 և 2.5 աղյուսակներով: 2.3 և 2.4 աղյուսակների աջ կողմից գրված են $\theta_j = \lambda_{j0}/\lambda_{jk}$ արժեքները $j \in \{1, \dots, m\}$, որտեղ $\lambda_{jk} > 0$ (նրանցից փոքրագույնը, ալգորիթմ 2-ի 5-րդ քայլի համաձայն, որոշում է տանող տողը): Աղյուսակներում ընդգծված են տանող այունը և տանող տարրը: k խնդեքսով նշված է տանող այունը, s -ով՝ տանող տողը: Ուղղանկյան կանոնը պատկերավոր ներկայացնելու համար աղյուսակ 2.3-ում նշված է ուղղանկյուն, որի գագաթները օգտագործվում են λ_{32} տարրի վերահաշվարկի համար: Այս կանոնին համաձայն՝ աղյուսակ 2.4-ի γ_{32} տարրը հավասար է $2 - 1 \cdot 3:2 = 1/2$:

Δ	0	1	2	3	4	5	6	7
	0	-2	-1	-3	-5	0	0	0
5	30	2	3	1	2	1	0	0
6	40	4	2	1	2	0	1	0
7	25	1	2	3	1	0	0	1

$\uparrow k$

աղ. 2.3

Δ	0	1	2	3	4	5	6	7
	75	3	13/2	-1/2	0	5/2	0	0
4	15	1	3/2	1/2	1	1/2	0	0
6	10	2	-1	0	0	-1	1	0
7	10	0	1/2	5/2	0	-1/2	0	1

$\uparrow k$

աղ. 2.4

Δ	0	1	2	3	4	5	6	7
	77	3	33/5	0	0	12/5	0	1/5
4	13	1	7/5	0	1	3/5	0	-1/5
6	10	2	-1	0	0	-1	1	0
3	4	0	1/5	1	0	-1/5	0	2/5

աղ. 2.5

Արյուսակ 2.5-ում բոլոր գնահատականների համար ունենք՝ $\Delta_i \geq 0$, $i = 1, \dots, 7$:

Հետևաբար խնդիրը լուծված է. $x^* = (0, 0, 4, 13, 0, 10, 0)$, $f(x^*) = 77$:

Օրինակ 2: Դիտարկենք հետևյալ ԳԾ խնդիրը.

$$f(x) = 5x_1 + x_2 - x_4 \rightarrow \max;$$

$$2x_1 + 2x_2 - x_3 + x_4 = 18,$$

$$6x_1 - 4x_2 - x_3 - x_4 = 14,$$

$$x_i \geq 0, \quad i = 1, 2, 3, 4:$$

Դիցուք տրված է $I_\sigma = \{1, 4\}$ բազմապայմանը ինդեքսների բազմությունը: Այդ դեպքում $c^\sigma = (5, -1)$,

$$B_\sigma = \begin{bmatrix} 2 & 1 \\ 6 & -1 \end{bmatrix}, \quad B_\sigma^{-1} = \begin{bmatrix} 1/8 & 1/8 \\ 3/4 & -1/4 \end{bmatrix}.$$

Հաջորդաբար բազմապատկելով B_σ^{-1} մատրիցը $a^0 = b, a^2$ և a^3 այուներով՝ լրացնենք սիմպլեքս աղյուսակի ներքին մասը (հիշեցնենք, որ նրա բազմապայման այուները հավասար

են համապատասխան միավոր վեկտորներին): Արդյունքը ներկայացված է աղյուսակ 2.6-ում:

	0	1	2	3	4
Δ		0			0
1	4	1	$-1/4$	$-1/4$	0
4	10	0	$5/2$	$-1/2$	1

աղ. 2.6

Մնում է հաշվել Δ_0, Δ_2 և Δ_3 գնահատականները: Համաձայն $\Delta_i = \langle c^\sigma, \lambda^i \rangle = -c_i$ բանաձևի, պետք է սկայար բազմապատկել c^σ -ն աղյուսակ 2.6-ի անհրաժեշտ սյունով, այնուհետև հանել նպատակային վեկտորի համապատասխան գործակիցը (հիշեցնենք՝ $c_0 = 0$): Կստանանք աղյուսակ 2.7-ը:

	0	1	2	3	4
Δ	10	0	$-19/4$	$-3/4$	0
1	4	1	$-1/4$	$-1/4$	0
4	10	0	$5/2$	$-1/2$	1

աղ. 2.7

2.7 աղյուսակի երրորդ սյան բոլոր տարրերը բացասական են: Հետևաբար՝

$$\sup_x f(x) = +\infty:$$

1 և 2 ալգորիթմները իրենցից ներկայացնում են մեթոդի երկու իրականացում: Սկզբունքայինորեն նրանք իրարից ոչնչով չեն տարբերվում. սկսելով միևնույն բազիսից և օգտագործելով տանող սյան ու տանող տողի ընտրման նույն կանոնները այս ալգորիթմները կգեներացնեն նույն բազիսների հերթականությունը և նրանց համապատասխան անկյունային կետերը: Այդ իսկ պատճառով ստորև մենք կհամարենք նրանց ամբողջվին իրավահավասար, կիրառելով այս կամ այն գրելաձևը՝ կախված կոնկրետ պահանջներից (ալգորիթմ 1 կամ 2): Ավելի ճշգրիտ, սիմպլեքս մեթոդի ալգորիթմի հիմնական ձևը կհամարենք 1-ը, իսկ սիմպլեքս աղյուսակները (ալգորիթմ 2) կօգտագործենք հիմնականում արդյունքները պատկերավոր ներկայացնելու համար:

Հաշվողական տեսակետից 1 և 2 ալգորիթմները ոչ մի կերպ համարժեք չեն: Ալգորիթմ 1-ը ավելի նախընտրելի է, քան 2-ը, ինչ միայն նրա համար, որ չի պահանջում ողջ սիմպլեքս աղյուսակի վերահաշվարկ ամեն իտերացիայում, այլ բավարարվում է միայն

իր մեկ այունով՝ λ^k (և, իհարկե, հակադարձ բազիսային մատրիցի վերահաշվարկը): Ներկայումս գոյություն ունեցող գծային ծրագրավորման խնդիրների լուծման համար ստեղծված կոմերցիոն ծրագրային փաթեթները հիմնված է հենց ալգորիթմ 1-ի կամ նրա տարատեսակներից մեկի վրա:

2.4. Սիմպլեքս մեթոդի երկրաչափությունը

Սահմանում: Դիցուք $X \subset \mathbb{R}^n$ ուռուցիկ բազմություն է: $Q \subset X$ բազմությունը կոչվում է X բազմության կող, եթե X բազմությունը որոշող սահմանափակումների մեջ գտնվեն $n - 1$ հատ գծորեն անկախ սահմանափակումներ, որոնք Q բազմության ցանկացած կետում բավարարվում են ինչպես ճիշտ հավասարումներ:

Դիցուք X -ը (2.1),(2.2) խնդրի պլանների բազմությունն է, \bar{x} -ը X բազմության անկյունային կետ է, σ -ն նրա բազիսն է, իսկ $x = x(\theta)$ կառուցված է համաձայն (2.25), (2.26) և (2.28) բանաձևերի՝

$$x_i = 0, \quad i \notin I_\sigma, \quad i \neq k,$$

$$x_k = \theta,$$

$$x^\sigma = \bar{x}^\sigma - \theta \lambda^k,$$

որտեղ $k \notin I_\sigma$, այնպիսին որ $\Delta_k(\sigma) < 0$: x կետը կարելի է ներկայացնել

$$x = \bar{x} + \theta d^k, \tag{2.40}$$

տեսքով, որտեղ d^k -ն որոշվում է հետևյալ կերպ.

$$d_i^k = 0, \quad i \notin I_\sigma, \quad i \neq k, \tag{2.41}$$

$$d_k^k = 1, \tag{2.42}$$

$$(d^k)^\sigma = -\lambda^k: \tag{2.43}$$

(2.40) տեսքով ներկայացվող $x \in X$ կետերի բազմությունը նշանակենք Q_k -ով՝

$$Q_k = \{ x \in X: x = \bar{x} + \theta d^k \}: \tag{2.44}$$

Պարագրաֆ 2.3-ում ցույց է տրված, որ եթե $\lambda^k \leq 0$, ապա $x = x(\theta) \in X, \forall \theta \geq 0$: Հետևաբար այս դեպքում Q_k -ն իրենից ներկայացնում է \bar{x} անկյունային կետից դուրս եկող ճառագայթ d^k վեկտորի ուղղությամբ: Այդ ճառագայթի երկայնքով $x(\theta)$ կետի շարժվելիս նպատակային ֆունկցիայի արժեքը $\langle c, x(\theta) \rangle$ անսահմանափակորեն աճում է:

Եթե $\lambda_j^k > 0$ գոնեւ մեկ $j \in \{1, \dots, m\}$ իդեքսի համար, ապա $x(\theta) \in X$ բոլոր $\theta \in [0, \theta_k]$ համար, որտեղ θ_k -ն որոշվում է (2.30) բանաձևով: $x(\theta_k)$ կետը հանդիսանում է X բազմության անկյունային կետ: Քանի որ ցանկացած $\theta \in [0, \theta_k]$ թիվ կարելի է ներկայացնել $\theta = \alpha\theta_k$, որտեղ $\alpha \in [0, 1]$, ապա ցանկացած $x \in Q_k$ կետ կարելի է ներկայացնել հետևյալ կերպ.

$$x = \bar{x} + \alpha\theta_k d^k = [\alpha + (1 - \alpha)]\bar{x} + \alpha\theta_k d^k = \alpha x(\theta_k) + (1 - \alpha)\bar{x}:$$

Սա նշանակում է, որ Q_k բազմությունը իրենից ներկայացնում է հատված, որը միացնում է \bar{x} և $x(\theta_k)$: $Q_k = [\bar{x}, x(\theta_k)]$ անկյունային կետերը:

Դիցուք $\theta > 0$ (և $\theta < \theta_k$, եթե $Q_k = [\bar{x}, x(\theta_k)]$): Այդ դեպքում կառուցման համաձայն

$$\begin{aligned} Ax(\theta) &= b, \\ x^\sigma(\theta) &> 0, \\ x_i(\theta) &= 0, \quad i \notin I_\sigma, \quad i \neq k, \\ x_k(\theta) &= \theta > 0: \end{aligned}$$

Այստեղից հետևում է, որ $x(\theta)$ կետը X բազմությունը որոշող ($n + m$) հավասարումներից $(n - 1)$ -երորդին բավարարում է ինչպես ճիշտ հավասարման՝

$$Ax = b \text{ (}m \text{ սահմանափակում),} \quad (2.45)$$

$$x_i = 0, \quad i \notin I_\sigma, \quad i \neq k \text{ (}n - m - 1 \text{ սահմանափակում):} \quad (2.46)$$

Դրանք նույն սահմանափակումներն են, որոնք որոշում են \bar{x} անկյունային կետը, հանած $x_k = 0$ սահմանափակումը, հետևաբար (2.45)-ը և (2.46)-ը գծորեն անկախ են: $x(\theta_k)$ անկյունային կետը նույնպես բավարարում է այդ սահմանափակումներին (եթե Q_k -ն հատված է): Այսպիսով Q_k բազմությունը հանդիսանում է X թույլատրելի բազմության կող: d^k վեկտորն անվանենք Q_k կողի ուղղորդ վեկտոր: Եթե Q_k -ն ճառագայթ է, ապա նրան կանվանենք անսահմանափակ կող, իսկ եթե Q_k -ն հատված է, ապա այդ հատվածի ծայրակետերին՝ անկյունային կետեր \bar{x} -ին և $x(\theta_k)$ -ին կանվանենք հարևան անկյունային կետեր: Սիմպլեքս մեթոդը իրականացնում է հարևան կետերի ընտրություն այնքան ժամանակ, մինչև չգտնվի օպտիմալ պլան կամ մինչև չհայտնաբերվի անսահմանափակ կող:

2.5. Վերասերվածություն և ցիկլ: Բլենդի կանոն

Դիտարկենք ԳԾ հետևյալ խնդիրը.

$$\begin{aligned} f(x) &= 2x_1 + 3x_2 - x_3 - 12x_4 \rightarrow max; \\ -2x_1 - 9x_2 + x_3 + 9x_4 + x_5 &= 0, \\ x_1 + 3x_2 - x_3 - 6x_4 + x_6 &= 0, \\ 2x_1 + 3x_2 - x_3 - 12x_4 + x_7 &= 2, \\ x_i &\geq 0, \quad i = 1, \dots, 7; \end{aligned}$$

Ակնհայտ է, որ դիտարկվող խնդրում $\bar{x} = (0,0,0,0,0,0,2)$ կետը անկյունային է: Չնայած, որ \bar{x} կետը վերասերված անկյունային կետ է, փորձենք լուծել խնդիրը սիմպլեքս մեթոդով, ընտրելով \bar{x} -ը որպես սկզբնական անկյունային կետ: Որպես սկզբնական բազիս ընտրենք ակնհայտ $\sigma = \{a^5, a^6, a^7\}$ բազիսը: $B_\sigma = E$ (B_σ -ն բազիսային մատրիցն է), $c^\sigma = (0,0,0)$: Սկզբնական սիմպլեքս աղյուսակը ներկայացված է աղյուսակ 2.8-ում:

Δ	0	1	2	3	4	5	6	7	
0	0	-2	-3	1	12	0	0	0	
5	0	-2	-9	1	9	1	0	0	
6	0	1	3	-1	-6	0	1	0	$0 \leftarrow s$
7	2	2	3	-1	-12	0	0	1	$2/3$

$\uparrow k$

աղ. 2.8

Ընտրենք որպես տանող այուն երկրորդ այունը և իրականացնենք մեկ սիմպլեքս իտերացիա: Արդյունքում կստանանք 2.9 աղյուսակը:

Δ	0	1	2	3	4	5	6	7	
0	0	-1	0	0	6	0	1	0	
5	0	1	0	-2	-9	1	3	0	$0 \leftarrow s$
2	0	$1/3$	1	$-1/3$	-2	0	$1/3$	0	0
7	2	1	0	0	-6	0	-1	1	2

$\uparrow k$

աղ. 2.9

Դժվար չէ նկատել, որ իրականացված սիմպլեքս իտերացիայի արդյունքում սիմպլեքս մեթոդը մնաց նույն \bar{x} անկյունային կետում, բայց գտավ այդ կետի համար նոր բազիս $\{a^5, a^2, a^7\}$: Այս փաստի բացատրությունը բավական ակնհայտ է՝ (2.29) բանաձևից հետևում է, որ «նոր» անկյունային կետի x_6 բաղադրիչը ունի հետևյալ տեսքը՝ $x_6 = 0 - \theta \cdot 3$ և ցանկացած $\theta > 0$ համար $x = x(\theta)$ դառնում է անթույլատրելի: Այդ պատճառով սիմպլեքս մեթոդը արգելում է շարժման փորձը անթույլատրելի ուղղությամբ՝ ընտրելով $\theta_k = 0$, միաժամանակ այն \bar{x} անկյունային կետի համար գտնում է նոր բազիս՝ թողնելով

կրկնել խնդիրը լուծելու փորձը՝ այս անգամ սկսելով նոր բազիսից: Հնարավոր է նոր բազիսը ավելի հաջող լինի և նրանում կա՝ մ տեղի ունենա կանգառի պայմաններից մեկը, կա՝ մ հաջողվի տեղափոխվել նոր անկյունային կետ:

Նախքան նոր իտերացիա կատարելը, նկատենք որ աղյուսակ 2.9-ը ներկայացնում է ևս մեկ իրավիճակ, որը չի հանդիպում չվերասերված խնդիրներում՝ տանող տողը կարող է ընտրվել ոչ միարժեքորեն: Աղյուսակ 2.9-ում այն կարող է լինել կա՝ մ առաջին, կա՝ մ երկրորդ տողը: Քանի որ այս իրավիճակը կարող է հանդիպել նաև հետազայում, պայմանավորվենք այս օրինակի համար տանող տողի և տանող սյան ընտրությունը կատարենք համաձայն հետևյալ բանաձևերի.

$$k = \max\{ i: \Delta_i < 0 \},$$

$$s = \min\{ j: \theta_k = \bar{x}_j^\sigma / \lambda_{jk} \}$$

(այլ կերպ ասած՝ հավակնորդների միջից որպես տանող տող ընտրվում է վերևից առաջինը): Այսպիսով աղյուսակ 2.9-ում, որպես տանող տող ընտրենք առաջին տողը: Վերահաշվարկելով սիմպլեքս աղյուսակը կգանք նույն անկյունային կետին, բայց արդեն նոր բազիսով՝ $\{a^1, a^2, a^7\}$: Շարունակելով սիմպլեքս իտերացիաները նույն \bar{x} անկյունային կետում, փոխելով միայն այդ կետի բազիսները: Արդյունքում ինդեքսների բազմության հաջորդաբար փոխումը սկսած սկզբնականից կունենա հետևյալ տեսքը.

$(5,6,7) \rightarrow (5,2,7) \rightarrow (1,2,7) \rightarrow (1,4,7) \rightarrow (3,4,7) \rightarrow (3,6,7) \rightarrow (5,6,7)$: Մենք վերադարձանք սկզբնական ինդեքսների բազմությանը, այսինքն սիմպլեքս մեթոդը ընկալ ցիկլ: Գոյություն ունեն սիմպլեքս մեթոդի մի քանի մոդիֆիկացիաներ, որոնք թույլ են տալիս խուսափել ցիկլից և ստանալ խնդրի վերջնական լուծումը: Դրանցից պարզագույնը Բլենդի մոդիֆիկացիան է կամ Բլենդի կանոնը, որը ձևակերպվում է հետևյալ կերպ.

$$k = mi n\{ i: \Delta_i < 0 \}, \quad (2.47)$$

$$i_s = \min\{ i_j: \theta_k = \bar{x}_{i_j} / \lambda_{jk} \}: \quad (2.48)$$

Այսինքն, բազիս է մտնում այն a^k վեկտորը, որը համապատասխանում է կարգով առաջին բացասական գնահատականին և բազիսից դուրս է գալիս a^{i_s} վեկտորը, որը համապատասխանում է բազիսային այն ինդեքսների մինիմալին, որոնց հիման վրա (2.30) բանաձևով որոշվում է մինիմումը (որպես տանող սյուն սիմպլեքս աղյուսակից ընտրվում է բացասական գնահատական ունեցող սյուներից առաջինը, իսկ որպես տանող տող՝ այն տողը, որը ունի հավակնություն այդ դերի համար և որին համապա-

տասխանում է բազիսային ինդեքսներից փոքրագույնը): Նկատենք, որ դիտարկված օրինակում որպես տանող սյուն ընտրվել էր «Ճիշտ հակառակը»: Դժվար չէ համոզվել, որ դիտարկված օրինակում՝ Բլենդի կանոնի կիրառման դեպքում սիմպլեքս մեթոդը ցիկլ չի ընկնի (օպտիմալ պլանը՝ $x^* = (0,0,0,0,0,2)$):

Թեորեմ: Սիմպլեքս մեթոդում Բլենդի կանոնի կիրառման դեպքում ցիկլ ընկնելն անհնար է:

Ապացույցը տե՛ս [2]:

Քանի որ սիմպլեքս մեթոդում Բլենդի կանոնի կիրառման դեպքում ցիկլ ընկնելն անհնար է, մեթոդը ընտրելով վերասերված անկյունային կետի համար մի քանի բազիս, վերջիվերջո կգտնի այնպիսի բազիս, որում կա՛մ տեղի ունի օպտիմալության պայմանը, կա՛մ կհաստատվի նպատակային ֆունկցիայի անսահմանափակ լինելը, կա՛մ սիմպլեքս-մեթոդը կկարողանա գալ նոր անկյունային կետի, որտեղ նպատակային ֆունկցիայի արժեքը կլինի ավելի մեծ քան նախորդում:

Այսպիսով Բլենդի կանոնով սիմպլեքս մեթոդը վերջավոր է նաև վերասերված տեսքի ԳԾ խնդիրների համար:

Թեորեմ: Կանոնական ԳԾ խնդրում \bar{x} անկյունային կետի համար (2.17)-պայմանը հանդիսանում է անհրաժեշտ և բավարար օպտիմալության պայման:

Ապացույցը տե՛ս [2]:

2.6. Արհեստական բազիսի մեթոդ և երկֆազ սիմպլեքս մեթոդ

Մինչ այժմ մենք ենթադրել ենք, որ հայտնի է որոշակի \bar{x} անկյունային կետ, որից սիմպլեքս մեթոդը սկսում է աշխատանքը: Որոշ դեպքերում կարելի է հեշտությամբ գտնել այդպիսի անկյունային կետ: Օրինակ, դիտարկենք ԳԾ ստանդարտ խնդիրը.

$$\langle c, x \rangle \rightarrow \max; \quad Ax \leq b, \quad x \geq 0;$$

Այս խնդրին համապատասխանող ԳԾ կանոնական խնդիրը ունի հետևյալ տեսքը.

$$\langle c, x \rangle \rightarrow \max; \quad Ax + u = b, \quad x \geq 0, \quad u \geq 0,$$

որտեղ u -ն լրացուցիչ փոփոխականների վեկտորն է: Եթե $b \geq 0$, ապա ակնհայտ է, որ $(\bar{x}, \bar{u}) = (0, b)$ կետը հանդիսանում է անկյունային կետ, որի բազիսը կազմված է միավոր վեկտորներից:

Սակայն ընդհանուր դեպքում, սկզբնական անկյունային կետի որոնումը իրենից ներկայացնում է ոչ պակաս բարդ խնդիր, քան նախնական ԳԾ խնդիրը:

Դիտարկենք ԳԾ կանոնական խնդիրը՝

$$\langle c, x \rangle \rightarrow \max; \quad x \in X, \quad (2.49)$$

$$X = \{x \in R^n: Ax = b, \quad x \geq 0\}, \quad (2.50)$$

որտեղ $A \in M_{m,n}$: b վեկտորը կարելի է համարել ոչ բացասական. $b \geq 0$ (սրան միշտ կարելի է հասնել՝ անհրաժեշտության դեպքում համապատասխան հավասարման երկու մասերը բազմապատկելով (-1)-ով): Նշենք, որ (2.49), (2.50) խնդրում արդեն չկան այն սահմանափակող ենթադրությունները, որոնք արվել էին ավելի վաղ. $\text{rank}(A) = m < n$ և խնդիրը չվերասերված է: Այսինքն (2.49), (2.50) խնդիրը կարող է լինել վերասերված և X բազմությունը նկարագրող հավասարումների մեջ կարող են լինեն գծորեն կախված հավասարումներ:

Օգտագործելով X բազմությունը նկարագրող պայմանները՝ կազմենք հետևյալ ԳԾ խնդիրը.

$$\begin{aligned} -x_{n+1} - x_{n+2} - \cdots - x_{n+m} &\rightarrow \max; \\ a_{11}x_1 + \cdots + a_{1n}x_n + x_{n+1} &= b_1, \\ a_{21}x_1 + \cdots + a_{2n}x_n + x_{n+2} &= b_2, \\ \dots & \\ a_{m1}x_1 + \cdots + a_{mn}x_n + x_{n+m} &= b_m, \\ x_i \geq 0, \quad i = 1, \dots, n+m, & \end{aligned}$$

կամ՝ մատրիցային տեսքով.

$$-\sum_{j=1}^m x_{n+j} \rightarrow \max; \quad Ax + v = b, \quad x \geq 0, \quad v \geq 0, \quad (2.51)$$

որտեղ՝ $v = (x_{n+1}, \dots, x_{n+m})$: x_{n+j} փոփոխականները կոչվում են արհեստական փոփոխականներ, իսկ (2.51) խնդիրը՝ արհեստական խնդիր:

(2.51) խնդրում պայմանների մատրիցի ռանգը հավասար է $m < n + m$ և, բացի այդ, կա ակնհայտ անկյունային կետ $(x^0, v^0) = (0, b)$, որի բազիսը կազմում են (e^1, \dots, e^m)

միավոր վեկտորները: Շնորհիվ փոփոխականների ոչ բացասական լինելու պայմանի, (2.51) խնդրի նպատակային ֆունկցիան սահմանափակ է վերևսից. $-\sum_{j=1}^m x_{n+j} \leq 0$: Հետևաբար (2.51) խնդիրը ունի լուծում: Գտնենք այդ լուծումը սիմպլեքս մեթոդով՝ ընտրելով որպես սկզբնական անկյունային կետ (x^0, v^0) կետը: Ենթադրենք (2.51) խնդրում (\bar{x}, \bar{v}) կետը սիմպլեքս մեթոդով գտած օպտիմալ անկյունային կետն է, μ -ն՝ նպատակային ֆունկցիայի օպտիմալ արժեքը, այսինքն՝ $\mu = -\sum_{j=1}^m \bar{x}_{n+j} \leq 0$:

Թեորեմ: Եթե $\mu < 0$, ապա (2.49), (2.50) խնդրի պլանների բազմությունը դատարկ է: Եթե $\mu = 0$, ապա \bar{x} -ը X բազմության անկյունային կետ է:
Ապացույցը տե՛ս [2]:

Այսպիսով, արհեստական խնդրի լուծումը սիմպլեքս մեթոդով թույլ է տալիս ստուգել նախնական խնդրի պայմանների համատեղելիությունը և, եթե $X \neq \emptyset$, գտնել նրա անկյունային կետը: (2.49), (2.50) խնդրում անկյունային կետի գտնելու մեթոդը, որը հիմնված է (2.51) խնդրի սիմպլեքս մեթոդով լուծման վրա, կոչվում է արհեստական բազիսի մեթոդ, իսկ գործընթացը, որը բաղկացած է արհեստական բազիսի սկզբնական անկյունային կետի կառուցումից և այնուհետև սիմպլեքս մեթոդով (2.49), (2.50) խնդրի լուծումից, կոչվում է երկֆազ սիմպլեքս մեթոդ (առաջին ֆազը սիմպլեքս մեթոդի կիրառումն է արհեստական խնդրի վրա, երկրորդ ֆազը կիրառումն է նախնական խնդրի վրա):

Եթե արհեստական բազիսի մեթոդով ստացված $\bar{x} \in X$ անկյունային կետը չվերասերված է, ապա նրա բազիսը, ապա նրա բազիսը, որը կառուցվել է առաջին ֆազի վերջին իտերացիայում, ակնհայտ է, որ բաղկացած է միայն (2.49), (2.50) խնդրի պայմանների վեկտորներից և կարելի է միանգամից անցնել սիմպլեքս մեթոդի երկրորդ ֆազին: Եթե արհեստական խնդիրը լուծվել է սիմպլեքս աղյուսակների միջոցով, ապա սիմպլեքս մեթոդի առաջին իտերացիայում նախնական խնդրի համար կարելի է չհաշվել $\lambda^i = B_\sigma^{-1} a^i$, $i = 0, \dots, n$: Այս վեկտորները հաշվել են արհեստական բազիսի մեթոդի վերջին իտերացիայում և պարունակվում են արհեստական խնդրի վերջին սիմպլեքս աղյուսակի 0-րդից մինչև ո-րդ այուներում (առանց զրոյական տողի կեմենտների): Այդ իսկ պատճառով, նախնական խնդրի սկզբնական աղյուսակը

լրացնելիս, նրանք ուղղակի արհեստական խնդրի վերջին աղյուսակից տեղափոխվում են և նախնական խնդրի սկզբնական աղյուսակում մնում է միայն հաշվել և լրացնել նրա գրոյական տողը: Նկատենք, որ արհեստական խնդրի համար էլ սկզբնական աղյուսակի լրացնելը տրիվիալ է և բերվում է խնդրի պայմանները արտագրելուն: Մասնավորապես, քանի որ արհեստական խնդրի առաջին իտերացիայում $B_\sigma = E$ և $c^\sigma = (-1, \dots, -1)$, ապա $\lambda^i = B_\sigma^{-1} a^i = a^i$, իսկ $\Delta_i = \langle c^\sigma, \lambda^i \rangle - c_i = \langle c^\sigma, \lambda^i \rangle = -\sum_{j=1}^m a_{ji}$, $i = 0, \dots, n$: Այսպիսով արհեստական խնդրի համար սկզբնական սիմպլեքս աղյուսակը ունի հետևյալ տեսքը.

$$\begin{bmatrix} -\sum_{j=1}^m b_j & -\sum_{j=1}^m a_{j1} & \dots & -\sum_{j=1}^m a_{jn} & 0 & \dots & 0 \\ b & a^1 & \dots & a^n & e^1 & \dots & e^m \end{bmatrix}.$$

Օրինակ 1:

Դիտարկենք հետևյալ ԳԾ խնդիրը.

$$f(x) = x_1 + 2x_2 + 3x_3 - 4x_4 \rightarrow max; \quad (2.52)$$

$$x_1 + x_2 - x_3 + x_4 = 2, \quad (2.53)$$

$$x_1 + 14x_2 + 10x_3 - 10x_4 = 24, \quad (2.54)$$

$$x_i \geq 0, \quad i = 1, 2, 3, 4: \quad (2.55)$$

(2.52) - (2.55) խնդրի համար արհեստական խնդիրը կունենա այս տեսքը.

$$-x_5 - x_6 \rightarrow max; \quad (2.56)$$

$$x_1 + x_2 - x_3 + x_4 + x_5 = 2, \quad (2.57)$$

$$x_1 + 14x_2 + 10x_3 - 10x_4 + x_6 = 24, \quad (2.58)$$

$$x_i \geq 0, \quad i = 1, \dots, 6: \quad (2.59)$$

(2.56) - (2.59) խնդրի համար սիմպլեքս մեթոդի իտերացիաները ներկայացված են աղյուսակ 2.10 - 2.12-ում:

	0	1	2	3	4	5	6	
Δ	-26	-2	-15	-9	9	0	0	
5	2	1	1	-1	1	1	0	$2 \leftarrow s$
6	24	1	14	10	-10	0	1	24
			$\uparrow k$					

աղ. 2.10

	0	1	2	3	4	5	6	
Δ	-22	0	-13	-11	11	2	0	
1	2	1	1	-1	1	1	0	
6	22	0	13	11	-11	-1	1	
				↑ k				← s

աղ. 2.11

	0	1	2	3	4	5	6	
Δ	0	0	0	0	1	1		
1	4	1	24/11	0	0	10/11	1/11	
3	2	0	13/11	1	-1	-1/11	1/11	

աղ. 2.12

Աղյուսակ 2.12-ում գնահատականներ՝ $\Delta_i \geq 0$ բոլոր $i = 1, \dots, 6$ -ի համար: Հետևաբար արհեստական խնդիրը լուծված է: Քանի որ $\Delta_0 = 0$, ապա $\bar{x} = (4, 0, 2, 0)$ կետը (2.52) - (2.55) խնդրի անկյունային կետ է: Սկզբնական խնդրի սիմպլեքս աղյուսակի ներքեւ մասը այս կետում ամբողջովին տեղափոխվում է 2.12 աղյուսակից: (2.52)-(2.55) խնդրի սկզբնական սիմպլեքս աղյուսակը ներկայացված է աղյուսակ 2.13-ում:

	0	1	2	3	4	
Δ	10	0	41/11	0	1	
1	4	1	24/11	0	0	
3	2	0	13/11	1	-1	

աղ. 2.13

2.13 աղյուսակից հետևում է, որ նախնական $\bar{x} = (4, 0, 2, 0)$ ակյունային կետը (2.49), (2.50) խնդրի լուծումն է: Նպատակային ֆունկցիայի առավելագույն արժեքը՝ $f(\bar{x}) = 10$:

Դիցուք $m \leq n$ և \bar{x} -ը (2.49), (2.50) խնդրի անկյունային կետն է, որը ստացվել է արհեստական բազիսի մեթոդով, $\bar{x}_{i_1}, \dots, \bar{x}_{i_r}$ -ը \bar{x} վեկտորի ոչ զրոյական բաղադրիչներն են, a^{i_1}, \dots, a^{i_r} -ը ($1 \leq i_1, \dots, i_r \leq n$) նրանց համապատասխանող պայմանների վեկտորներն են: Եթե նախնական խնդիրը չվերասերված է, ապա $r = m$ և a^{i_1}, \dots, a^{i_r} վեկտորները կազմում են \bar{x} անկյունային կետի բազիսը: Եթե նախնական խնդիրը վերասերված է, ապա կարող է պարզվել, որ $r < m$ և a^{i_1}, \dots, a^{i_r} վեկտորների հետ միասին՝ \bar{x} անկյունային կետի «բազիսի» մեջ մտնեն ևս $m - r$ պայմանների վեկտորներ: Ընդ որում, $m - r$ վեկտորների շարքում կարող են լինել ինչպես նախնական խնդրի պայմանների վեկտորներ, այպես էլ՝ արհեստական փոփոխականներին համատասխանող պայմանների վեկտորներ: Վերջինը նշանակում է, որ սիմպլեքս մեթոդի ալգորիթմին անհրա-

Ժեշտ \bar{x} անկյունային կետի բազիսը դեռևս գտնված չէ: Դրա կառուցման համար պետք է \bar{x} անկյունային կետի «բազիսից» բացառել արհեստական փոփոխականներին համապատասխանող բոլոր պայմանների վեկտորները՝ դրանք փոխարինելով նախնական խնդրի գծորեն անկախ պայմանների վեկտորներով:

Արհեստական խնդրի պայմանների մատրիցը նշանակենք \bar{A} -ով.

$$\bar{A} = [a^1 \dots a^n a^{n+1} \dots a^{n+m}], \quad a^{n+j} = e^j, \quad j = 1, \dots, m:$$

Դիցուք $\sigma = \{a^{i_1}, \dots, a^{i_m}\}$ -ը (\bar{x}, \bar{v}) = ($\bar{x}, 0$) անկյունային կետի բազիսն է, որը ստացվել է արհեստական բազիսի մեթոդի վերջին իտերացիայում. $I_\sigma = \{i_1, \dots, i_m\}$: Ենթադրենք, որ σ բազիսը պարունակում է պայմանների վեկտորներ, որոնք համապատասխանում են ինչպես նախնական փոփոխականներին, այնպես էլ արհեստականներին: Այսինքն բազիսային խնդեքսների շարքում կան ինչպես $i_j \leq n$ խնդեքսներ, այնպես էլ $i_j > n$:

Ներմուծենք հետևյալ բազմությունները.

$$J_1 = \{j: i_j \in I_\sigma, 1 \leq i_j \leq n\},$$

$$J_2 = \{j: i_j \in I_\sigma, n+1 \leq i_j \leq n+m\},$$

$$I_1 = \{i \in I_\sigma: 1 \leq i \leq n\} = \{i_j: j \in J_1\},$$

$$I_2 = \{i \notin I_\sigma: 1 \leq i \leq n\}:$$

Սիմպլեքս աղյուսակների լեզվով՝ J_1 բազմությունը կազմված է սիմպլեքս աղյուսակի տողերի համարներից, որոնք համապատասխանում են $i_j \leq n$ համարներով բազիսային փոփոխականներին (նախնական փոփոխականներ), իսկ J_2 -ը կազմված է այն տողերի համարներից, որոնք համապատասխանում են $i_j > n$ համարներով բազիսային փոփոխականներին (արհեստական փոփոխականներ): I_1 և I_2 բազմությունները համապատասխանաբար կազմված են նախնական խնդրի բազիսային և ոչ բազիսային պայմանների վեկտորներից: Ակնհայտ է, որ

$$J_1 \neq \emptyset, \quad J_2 \neq \emptyset, \quad J_1 \cap J_2 = \emptyset, \quad J_1 \cup J_2 = \{1, \dots, m\},$$

$$I_1 \neq \emptyset, \quad I_2 \neq \emptyset, \quad I_1 \cap I_2 = \emptyset, \quad I_1 \cup I_2 = \{1, \dots, n\}:$$

Դիցուք $|I_1| = p$, որտեղ $r \leq p < m$ (նախնական խնդրի σ բազիս մտնող պայմանների վեկտորների քանակն է, որը փոքր չէ \bar{x} անկյունային կետի դրական բաղադրիչների քանակից): Այդ դեպքում

$$|J_1| = p, \quad |J_2| = m - p, \quad |I_2| = n - p:$$

Այսպիսով պահաջընում է σ բազիսում արհեստական փոփոխականներին համապատասխանող $m - p$ a^{ij} ($j \in J_2$) պայմանների վեկտորները փոխարինել $\{a^i, i \in I_2\}$ բազմությունից գծորեն անկախ վեկտորներով: Դրա համար դիտարկենք նախնական խնդրի պայմանների վեկտորների վերլուծությունը բազիսային վեկտորներով.

$$a^i = \sum_{j=1}^m \lambda_{ji} a^{ij}, \quad i = 1, \dots, n,$$

(2.60)

որտեղ $(\lambda_{1i}, \dots, \lambda_{ni}) = \lambda^i = B_\sigma^{-1} a^i$: Հնարավոր է երկու դեպք:

Դեպք 1. Գոյություն ունեն այնպիսի $k \in I_2$ և $s \in J_2$ ինդեքսներ, որ $\lambda_{sk} \neq 0$: Այդ դեպքում, եթե $i = k$, (2.60)-ից հետևում է, որ

$$a^k = \sum_{\substack{j=1 \\ j \neq s}}^m \lambda_{jk} a^{ij} + \lambda_{sk} a^{is}:$$

(2.61)

Քանի որ (2.61) վերլուծության մեջ $\lambda_{sk} \neq 0$, ապա σ բազիսում a^{is} վեկտորը կարելի է փոխարինել a^k վեկտորով (բազիսային ինդեքսների բազմությունում i_s ինդեքսը հավասարեցնել k -ին, s ինդեքսը ներմուծել J_1 բազմություն, J_2 -ից հանել s ինդեքսը, ներմուծել k ինդեքսը I_1 բազմություն, I_2 -ից հանել k ինդեքսը): Եթե արդյունքում պարզվի, որ $J_2 = \emptyset$ ($|J_1| = |I_1| = m$), ապա որոնելի բազիսը գտնված է: Հակառակ դեպքում, ամբողջ պլոցեղուրան անհրաժեշտ է կրկնել նոր բազիսում, ընդ որում, սիպլեքս աղյուսակներ օգտագործելու դեպքում բազիսի յուրաքանչյուր փոփոխություն պետք է ուղեկցվի $\lambda^i = B_\sigma^{-1} a^i$, $i = 1, \dots, n$, վեկտորների վերահաշվարկով (ակնհայտ է, որ $\lambda^0 = \bar{x}^\sigma$ վեկտորը չի փոփոխվում, իսկ արհեստական փոփոխականներին համապատասխանող պայմանների վեկտորների վերլուծության գործակիցները այս դեպքում էական չեն): Նկարագրված գործընթացի դեպքում՝ ոչ ավել, քան $m - p$ քայլից հետո կամ կգտնվի \bar{x} անկյունային կետի բազիսը, կամ կհանգենք 2-րդ դեպքին:

Դեպք 2. Բոլոր $i \in I_2$, $j \in J_2$ համար $\lambda_{ji} = 0$: Ամենից առաջ նկատենք, որ իրականում դիտարկվող դեպքում

$$\lambda_{ji} = 0 \quad \forall i \in 0, 1, \dots, n, \quad j \in J_2 \quad (2.62)$$

(այսինքն արհեստական փոփոխականներին համապատասխանող սիմպլեքս աղյուսակի տողերը $0, 1, \dots, n$ այուներում պարունակում են զրոներ): Իրոք, եթե $i \in I_1$, ապա $i = i_t$, ինչոր $t \in J_1$ -ի համար, և հետևաբար՝ $\lambda^i = \lambda^{i_t} = e^t$: Այստեղից՝ $\lambda_{ji} = e_j^t = 0$, եթե $i \in I_1, j \in J_2$: Բացի այդ, քանի որ $\lambda_{j0} = \bar{x}_{i_j}$, ապա $j \in J_2$ դեպքում $i_j = n + q$, ինչոր $q \in \{1, \dots, m\}$ համար և $\lambda_{j0} = \bar{x}_{n+q} = 0$:

(2.62)-ի տեղի ունենալու դեպքում (2.60)-ից հետևում է, որ կամայական $i = 1, \dots, n$ համար՝

$$a^i = \sum_{j \in J_1} \lambda_{ji} a^{i_j} + \sum_{j \in J_2} \lambda_{ji} a^{i_j} = \sum_{j \in J_1} \lambda_{ji} a^{i_j} :$$

Դա նշանակում է, որ նախնական խնդրի պայմանների վեկտորներից կամայականը հանդիսանում է p հատ $a^{i_j}, j \in J_1$ բազմային վեկտորների գծային համակցություն: Հետևաբար A մատրիցի ռանգը հավասար է p և խնդրի պայմանների շարքում կան $m - p$ գծորեն կախված պայմաններ: Ցույց տանք, որ այդպիսի գծորեն կախված պայմաններից են հանդիսանում $[Ax]_q = b_q$ -ն, $q = i_j - n, j \in J_2$ համարներով:

Սկզբում դիտարկենք այն դեպքը, եթե $p = m - 1$, այսինքն J_2 բազմությունը պարունակում է միայն մեկ տարր և հետևաբար խնդրի պայմաններից միայն մեկն է գծորեն կախված մնացածից: Դիցուք $J_2 = \{s\}$: Այս դեպքում՝ $i_s = n + q$ (այնպես, որ $q = i_s - n$), $a^{i_s} = a^{n+q} = e^q$: Հետևաբար B_σ բազմային մատրիցը կունենա այս տեսքը. $B_\sigma = [a^{i_1} \dots a^{i_{s-1}} e^q a^{i_{s+1}} \dots a^{i_m}]$ կամ ավելի մանրամասն գրելածենով՝

$$B_\sigma = \left[\begin{array}{ccc|c|ccc} a_{1,i_1} & \dots & a_{1,i_{s-1}} & 0 & a_{1,i_{s+1}} & \dots & a_{1,i_m} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{q-1,i_1} & \dots & a_{q-1,i_{s-1}} & 0 & a_{q-1,i_{s+1}} & \dots & a_{q-1,i_m} \\ \hline a_{q,i_1} & \dots & a_{q,i_{s-1}} & 1 & a_{q,i_{s+1}} & \dots & a_{q,i_m} \\ a_{q+1,i_1} & \dots & a_{q+1,i_{s-1}} & 0 & a_{q+1,i_{s+1}} & \dots & a_{q+1,i_m} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m,i_1} & \dots & a_{m,i_{s-1}} & 0 & a_{m,i_{s+1}} & \dots & a_{m,i_m} \end{array} \right] :$$

B_σ մատրիցից ջնջենք q -րդ տողը և s -րդ սյունը: Ստացված մատրիցը նշանակենք \widehat{B}_σ -ով: Նրա այուներն են հանդիսանում հետևյալ վեկտորները՝

$$\hat{a}^{i_1}, \dots, \hat{a}^{i_{s-1}}, \hat{a}^{i_{s+1}}, \dots, \hat{a}^{i_m}, \quad (2.63)$$

որոնք միաժամանակ պատկանում են հետևյալ մատրիցին.

$$\widehat{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{q-1,1} & a_{q-1,2} & \dots & a_{q-1,n} \\ a_{q+1,1} & a_{q+1,2} & \dots & a_{q+1,n} \\ \dots & \dots & \dots & \dots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix};$$

\widehat{A} մատրիցը ստացվել է A մատրիցից q -րդ տողը ջնջելով: \widehat{A} մատրիցը անվանենք պայմանների կրծատված մատրից:

Քանի որ $\det B_\sigma \neq 0$, ապա վերլուծելով ըստ s -րդ այան տարրերի, կստանանք՝ $\det \widehat{B}_\sigma \neq 0$: Դա նշանակում է, որ \widehat{A} մատրիցի (2.63) սյուները գծորեն անկախ են և հետևաբար՝ $\text{rank}(\widehat{A}) = m - 1$: Այստեղից իր հերթին հետևում է, որ A մատրիցի տողերը, որոնք կազմում են կրծատված \widehat{A} մատրիցը, գծորեն անկախ են, իսկ A մատրիցի q -րդ տողը հանդիսանում է նրանց գծային համակցությունը: Խնդրի պայմաններից ջնջելով q -րդ պայմանը՝ $[Ax]_q = b_q$ և նշանակելով՝ $\widehat{b} = (b_1, \dots, b_{q-1}, b_{q+1}, \dots, b_m)$, կստանանք հետևյալ խնդիրը՝

$$< c, x > \rightarrow \max; \quad \widehat{A}x = \widehat{b}, \quad x \geq 0, \quad (2.64)$$

որը համարժեք է (2.49), (2.50) խնդրին: Ակնհայտ է, որ \bar{x} -ը (5.16) խնդրի անկյունային կետ է և նրա \widehat{s} բազիսն են կազմում (2.63) վեկտորները:

(2.64) խնդրի համար \bar{x} կետում սիմպլեքս աղյուսակի $\widehat{\lambda}^0, \widehat{\lambda}^1, \dots, \widehat{\lambda}^n$ սյուները ստացվում են արհեստական բազիսի մեթոդի վերջին աղյուսակի $\lambda^0, \lambda^1, \dots, \lambda^n$ սյուներից՝ ջնջելով s -րդ բաղադրիչը, որը համարժեք է այդ աղյուսակի s -րդ տողը ջնջելուն (0 -ից մինչև n -րդ սյուներում զրոներ պարունակող): Իրականում քանի որ դիտարկվող դեպքում $J_1 = \{1, \dots, m\} \setminus \{s\}$, ապա (2.61) վերլուծությունը ունի հետևյալ տեսքը.

$$a^i = \sum_{j \in J_1} \lambda_{ji} a^{ij} = \sum_{\substack{j=1 \\ j \neq s}}^m \lambda_{ji} a^{ij};$$

Ջնջելով յուրաքանչյուր a^i և a^{ij} , $j \in J_1$, վեկտորների q -րդ բաղադրիչը, կստանանք՝

$$\widehat{a}^i = \sum_{\substack{j=1 \\ j \neq s}}^m \lambda_{ji} \widehat{a}^{ij} = \widehat{B}_\sigma \widehat{\lambda}^i,$$

որտեղ $\widehat{\lambda}^i = (\lambda_{1i}, \dots, \lambda_{s-1,i}, \lambda_{s+1,i}, \dots, \lambda_{mi}) = \widehat{B}_\sigma^{-1} \widehat{a}^i$:

Ընդհանուր առմամբ B_σ մատրիցը պարունակում է $m - p$ միավոր վեկտորներ, արհեստական փոփոխականներին համապատասխանող, $q = i_j - n$, $j \in J_2$ համարնե-

րով պայմանների վեկտորներ: Վերլուծելով B_σ մատրիցի որոշիչը այդ միավոր սյուների տարրերով և հետո քննարկելով ինչպես նախորդ դեպքում, դժվար չէ ստանալ, որ A մատրիցի $q = i_j - n$, $j \in J_2$ համարներով տողերը գծորեն կախված են մնացած p տողերից: Զնշելով A մատրիցից գծորեն կախված տողերը և խնդրի պայմաններից հեռացնելով համապատասխան պայմանները, կստանանք հետևյալ խնդիրը՝

$$\langle c, x \rangle \rightarrow \max; \quad \hat{A}x = \hat{b}, \quad x \geq 0, \quad (2.65)$$

որտեղ \hat{A} -ն կրծատված պայմանների մատրիցն է, իսկ \hat{b} -ն կրծատված սահմանափակումների վեկտորը, $\hat{A} \in M_{p,n}$, $\hat{b} \in R^p$: (2.65) խնդիրը համարժեք է (2.49), (2.50) խնդրին, իսկ արհեստական բազիսի մեթոդով ստացված \bar{x} կետը (2.65) խնդրում հանդիսանում է անկյունային կետ: Այդ անկյունային կետի բազիսն են կազմում \hat{A} մատրիցի a^{ij} , $j \in J_1$ սյունները: Դժվար չէ նկատել, որ (2.65) խնդրի համար սիմպլեքս աղյուսակի $\hat{\lambda}^0, \hat{\lambda}^1, \dots, \hat{\lambda}^n$ սյունները \bar{x} կետում ստացվում են արհեստական բազիսի մեթոդի վերջին աղյուսակի $\lambda^0, \lambda^1, \dots, \lambda^n$ սյուններից՝ զնշելով $j \in J_2$ համարներով բաղադրիչները, որը համարժեք է 0-րդից մինչև n -րդ սյուններում զրոներ պարունակող տողերի զնշելուն:

Դիտողություն: Վերևում մենք ենթադրեցինք, որ $m \leq n$: Եթե $m > n$, ապա հնարավոր է, որ $I_1 = \{1, \dots, n\}$, իսկ $I_2 = \emptyset$ ($|J_1| = n$, $|J_2| = m - n$): Այդ դեպքում, դատելով այնպես, ինչպես 2-րդ դեպքում, դժվար չէ նկատել, որ տեղի ունի (5.14)-ը և հետևաբար, $[Ax]_q = b_q$ պայմանները, $q = i_j - n$, $j \in J_2$ համարներով, գծորեն կախված են մնացած n պայմաններից: Այսպիսով, դիտարկվող դեպքում, նախնական խնդրի պլանների բազմությունը կազմված է մեկ կետից:

Օրինակ 2: Դիտարկենք հետևյալ ԳԾ խնդիրը.

$$\begin{aligned} 5x_1 + 4x_2 + 3x_3 + 2x_4 - 3x_5 &\rightarrow \max; \\ 2x_1 + x_2 + x_3 + x_4 - x_5 &= 3, \\ x_1 - x_2 + x_4 + x_5 &= 1, \\ -2x_1 - x_2 - x_3 + x_4 + x_5 &= 1, \\ -x_1 - 2x_2 - x_3 + 2x_4 + x_5 &= 2, \\ x_i &\geq 0, \quad i = 1, \dots, 5: \end{aligned}$$

Աղյուսակ 2.14-2.16-ում բերված են արհեստական խնդրի համար առաջին երեք խտերացիաները:

Δ	0	1	2	3	4	5	6	7	8	9
-7	0	3	1	-5	-1	0	0	0	0	0
6	3	2	1	1	1	-1	1	0	0	0
7	1	1	-1	0	1	1	0	1	0	0
8	1	-2	-1	-1	1	0	0	0	1	0
9	2	-1	-2	-1	2	1	0	0	0	1

աղ. 2.14

Δ	0	1	2	3	4	5	6	7	8	9
-2	5	-2	1	0	4	0	5	0	0	0
6	2	1	2	1	0	-2	1	-1	0	0
4	1	1	-1	0	1	1	0	1	0	0
8	0	-3	0	-1	0	-1	0	-1	1	0
9	0	-3	0	-1	0	-1	0	-2	0	1

աղ. 2.15

Δ	0	1	2	3	4	5	6	7	8	9
0	0	6	0	2	0	2	1	4	0	0
2	1	1/2	1	1/2	0	-1	1/2	-1/2	0	0
4	2	3/2	0	1/2	1	0	1/2	1/2	0	0
8	0	-3	0	-1	0	-1	0	-1	1	0
9	0	-3	0	-1	0	-1	0	-2	0	1

աղ. 2.16

2.16 աղյուսակում բոլոր $\Delta_i \geq 0$, $i = 1, \dots, 9$, $\Delta_0 = 0$: Հետևաբար, $\bar{x} = (0, 1, 0, 2, 0)$ կետը դիտարկվող խնդրի անկյունային կետ է: Նրա բազիսը գտնելու համար $\{a^2, a^4, a^8, a^9\}$ վեկտորների համակարգից պետք է դուրս բերել a^8 և a^9 վեկտորները, փոխարինելով դրանք նախնական խնդրի պայմանների վեկտորներով: Դիտարկվող օրինակում՝

$$J_1 = \{1, 2\}, \quad J_2 = \{3, 4\}, \quad I_1 = \{2, 4\}, \quad I_2 = \{1, 3, 5\},$$

ընդ որում $\lambda_{sk} \neq 0$ բոլոր $k \in I_2$, $s \in J_2$ համար: Վերջինս նշանակում է, որ a^8 և a^9 վեկտորներից ցանկացածը հնարավոր է փոխարինել a^1, a^3, a^5 վեկտորներից ցանկացածով, λ_{sk} , $k \in I_2$, $s \in J_2$ տարրերից ցանկացածը կարող է ընտրվել որպես տանող: Դիցուք դա λ_{35} -ն է: Վերահաշվարկելով 2.16 աղյուսակը անցնենք հաջորդ՝ 2.17 աղյուսակին, որում բաց են թողնված արհեստական փոփոխականներին համապատասխանող սյուները. անկյունային կետը գտնելուց հետո արհեստական փոփոխականները այլևս պետք չեն:

	0	1	2	3	4	5
Δ	0	0	0	0	0	0
2	1	7/2	1	3/2	0	0
4	2	3/2	0	1/2	1	0
5	0	3	0	1	0	1
9	0	0	0	0	0	0

աղ. 2.17

2.17 աղյուսակի վերջին տողը բաղկացած է միայն զրոներից: Հետևաբար այդ տողը կարելի է ջնջել աղյուսակից, իսկ խնդրի պայմաններից ջնջել $9 - 5 = 4$ համարն ունեցող պայմանը (հավասարման տեսք ունեցող պայմաններից վերջինը): Կրծատված խնդրում \bar{x} անկյունային կետի բազիսը կազմված է a^2, a^4 և a^5 վեկտորներից, իսկ նրա համար նախնական սիմպլեքս աղյուսակը տարբերվում է 2.17 աղյուսակից միայն զրոյական տողով և, փաստորեն, վերջին տողով (տե՛ս աղյուսակ 2.18):

	0	1	2	3	4	5
Δ	8	3	0	1	0	0
2	1	7/2	1	3/2	0	0
4	2	3/2	0	1/2	1	0
5	0	3	0	1	0	1

աղ. 2.18

2.18 աղյուսակում բոլոր գնահատականները՝ $\Delta_i \geq 0$, $i = 1, \dots, 5$ -ի համար: Հետևաբար $\bar{x} = (0, 1, 0, 2, 0)$ օպտիմալ պլանն է:

ԳԼՈՒԽ 3

ԷԼԻՊՍՈՒԴՆԵՐԻ ՄԵԹՈԴ

3.1. Էլիպսուդների մեթոդի ստեղծման և հետագա զարգացման մասին

1979 թվականին Լ. Գ. Խաչիյանի (L. G. Khachiyan) գրառումը [3] ցույց տվեց, թե ինչպես այսպես կոչված էլիպսուդների մեթոդը, որն ի սկզբանե մշակված էր ոչ գծային ոչ դիֆերենցիալ օպտիմալացման համար, կարող է փոփոխվել՝ ստուգելու համար գծային անհավասարումների համակարգի համատեղելիությունը պոլինոմիալ ժամանակային բարդությունով: Այս արդյունքը մաթեմատիկական ծրագրավորման աշխարհում մեծ ոգևորություն էր առաջացրել, քանի որ ենթադրում էր գծային ծրագրավորման խնդիրների համար պոլինոմիալ ժամանակային բարդության լուծելիություն:

Այդ ոգևորությունը ուներ մի քանի պատճառ: Ամենից առաջ՝ աշխարհի բազմաթիվ գիտնականներ երկար ժամանակ, առանց հաջողության, քրտնաջան աշխատել էին գծային ծրագրավորման համար պոլինոմիալ ժամանակային բարդությամբ ալգորիթմ գտնելու խնդրի վրա: Այսպիսով՝ իսկապես զլիավոր բաց խնդիրը լուծվել էր:

Շատ մարդիկ կարծում էին, որ $\mathcal{P} = \mathcal{NP} \cap co\text{-}\mathcal{NP}$ և գծային ծրագրավորման խնդիրը այն մի քանի խնդիրներից էր, որը հայտնի էր, որ պատկանում է՝ $\mathcal{NP} \cap co\text{-}\mathcal{NP}$, բայց ցույց տրված չէր, որ պատկանում է \mathcal{P} -ին: Այսպիսով, ստացվեցին այս վարկածի ճշգրտության վերաբերյալ հետագա ապացույցները:

Էլիպսուդների մեթոդը լրացուցիչ թվով տեսական «հնարքների» հետ միասին շատ տարբեր էր բոլոր գծային ծրագրավորման ալգորիթմներից: Մեթոդը և ապացույցի ճշտությունը իսկապես անակնկալ էր:

Չնայած Էլիպսուդների մեթոդի «տեսականորեն արդյունավետության», մինչ այժմ չի ապացուցվել «գործնականորեն արդյունավետությունը»: Հետևաբար, ալգորիթմների բարդության տեսությունում ալգորիթմի պոլինոմիալ ժամանակային բարդության արժեքի, կոդավորման երկարության ու աշխատանքի ժամանակը չափելու ձևի մասին հակասությունները ուշադրության կենտրոնում էին:

Սիմպլեքս մեթոդի ներկայիս գրեթե բոլոր հայտնի տարբերակների համար գոյություն ունի մի քանի (արհեստական) օրինակներ, որոնց համար սիմպլեքս

ալգորիթմը ունի էքսպոնենցիալ բարդություն: Այս տիպի առաջին օրինակը հայտնաբերվել է Վ. Քլիի(V. Klee) և Գ. Ջ. Մինտի(G. J. Minty) կողմից [4]: Այդպիսի վատ օրինակներ գոյություն չունեն էլիպսուղիների մեթոդի համար, բայց պարզվեց, որ էլիպսուղիների մեթոդը ավելի դանդաղ է միջին գործնական հաշվարկում, քան՝ սիմպլեքս ալգորիթմը:

Ինչպես նշվեց՝ ոչ գծային օպտիմալացման էլիպսուղիների մեթոդի արմատներից մեկն է: Մեթոդն աճել է ուռուցիկ ոչ դիֆերենցելի օպտիմալացման հետ աշխատելիս (ռելաքսացիա, սուբգրադիենտ, տարածության կիսման մեթոդներ, կենտրոնական հատույթների մեթոդներ), ինչպես նաև ուռուցիկ ծրագրավորման խնդիրների հաշվողական բարդության ուսումնասիրություններից: Էլիպսուղիների մեթոդի և իր նախատիպերի պատմությունը մեծապես կերտվել է Ռ. Գ. Բլենդի(R. G. Bland), Դ. Գոլդֆարբի(D. Goldfarb) և Մ. Ջ. Տոդդի(M. J. Todd) [5], ինչպես նաև Ռ. Շրեդրի(R. Schrader) [6] կողմից:

Հիմնվելով իր հին աշխատանքի վրա, Ն. Զ. Շոր(N. Z. Shor) [7],[8] ուռուցիկ ոչ դիֆերենցելի ծրագրավորման համար նկարագրեց գրադիենտի պրոեկցիայի մի նոր ալգորիթմ՝ տարածության ընդլայնմամբ: Դ. Բ. Յուդին(D. B. Yudin) և Ա. Ս. Նեմիրովսկին(A. S. Nemirovski) [9],[10] նկատեցին, որ Ն. Զ. Շոր(N. Z. Shor) ալգորիթմը Ա. Յու. Լևինի(A. Yu. Levin) [11] կողմից քննարկված մի խնդրի պատասխանն է տալիս և ինչ-որ կերպ տրվեց էլիպսուղիների մեթոդի ուրվագիծը: Էլիպսուղիների մեթոդի առաջին հստակ ձևակերպումը, ինչպես մենք այսօր գիտենք, շնորհիվ Ն. Զ. Շոր(N. Z. Shor) է [12]: Այս մեթոդը Լ. Գ. Խաչիյանի(L. G. Khachiyan) [3] կողմից աղապտացվել է գծային ծրագրավորման խնդիրների դասի համար պոլինոմիալ ժամանակային բարդությամբ լուծելիություն ստանալու նպատակով: Ապացուցները հայտնվեցին [13]-ում: Լ. Գ. Խաչիյանի(L. G. Khachiyan) 1979թ.-ի հոդվածը խթանեց հետազոտությունների տարափի, որոնց նպատակն էր արագացնել մեթոդը և դարձնել այն թվերի նկատմամբ ավելի կայուն: Էլիպսուղիների մեթոդի կիրառելիությունը կոմբինատորային օպտիմալացման մեջ իրարից անկախ հայտնաբերվել է Ռ. Մ. Կարպի(R. M. Karp) ու Ք. Հ. Պապադիմիտրիուի(C. H. Papadimitriou) [14], Մ. Վ. Պադբերգի(M. W. Padberg) ու Մ. Ռ. Ռաոյի(M. R. Rao) [15], Մ. Գրուտչելի(M. Grötschel), Լ. Լովասի(L. Lovász) ու Ա. Սխրիյվերի(A. Schrijver) [16] կողմից:

Լ. Գ. Խաչիյանի (L. G. Khachiyan) ձեռքբերումը գրավեց ոչ գիտական մամուլի ուշադրությունը: Թերթերը և ամսագրերը, ինչպիսիք էին «The Guardian», «Der Spiegel», «Nieuwe Rotterdamsche Courant», «Népszabadság», «The Daily Yomiuri», զրեցին «Իրական աշխարհի խնդիրների լուծման մեջ զլիավոր, խոշոր նվաճման» մասին: Էլիպսուիդների մեթոդը նույնիսկ հայտնվեց «The New York Times»-ում. «Խորհրդային Միության հայտնագործությունը ցնցեց մաթեմատիկայի աշխարհը» (Նոյեմբեր 7, 1979):

Վերջին տարիների՝ գծային ծրագրավորման տեսական և գործնական անսպասելի զարգացումները դրդել են հետազոտությունների զարթոնքի այս ոլորտում: Ոչ գծային ծրագրավորման մոտեցումը գծային ծրագրավորմանը՝ օգտագործելով այնպիսի եղանակներ, ինչպիսիք են Նյուտոնի մեթոդը և հարակից վայրէջքի պրոցեդուրաները արժանացան հատուկ ուշադրության: Գծային ծրագրավորման խնդիրների համար առաջարկվել են մի շարք լրացուցիչ պոլինոմիալ բարդությամբ մեթոդներ (տե՛ս օրինակ՝ [17], [18], [19], [20]) և ուսումնասիրությունների են ենթարկվում՝ իրենց տեսական և գործնական վարքագծի առումով: Հասկանալի է, որ այս մեթոդների մանրակրկիտ իրականացումը և հնարավոր համարումները սիմպլեքս ալգորիթմի հետ կհանգեցնեն գծային ծրագրավորման համար լավ ալգորիթմների: Այդպիսի ալգորիթմները հնարավոր են դառնան լուրջ մրցակիցներ սիմպլեքս ալգորիթմների համար, որոնք այս պահին գերիշխում են «Գծային ծրագրավորման շուկան»:

3.2. Էլիպսուիդների եղանակի երկրաչափությունը

$E \subseteq \mathbb{R}^n$ բազմությունը էլիպսուիդ է, եթե գոյություն ունի $a \subseteq \mathbb{R}^n$ վեկտոր և $(n \times n)$ չափի դրական որոշված A մատրից, այնպիսին որ՝

$$E = E(A, a) = \{x \in \mathbb{R}^n : (x - a)^T A^{-1} (x - a) \leq 1\}: \quad (3.1)$$

Օգտագործելով $\| \cdot \|_A$ էլիպսուիդալ նորմը ($\| x \|_A = \sqrt{x^T A^{-1} x}$) համարժեքորեն կարող ենք գրել՝

$$E = E(A, a) = \{x \in \mathbb{R}^n : \|x - a\|_A \leq 1\}, \quad (3.2)$$

այսինքն $E(A, a)$ էլիպսուիդը \mathbb{R}^n էվկլիդյան տարածությունում a վեկտորի շուրջ միավոր գունդ է՝ օժտված $\| \cdot \|_A$ նորմով: Այսպիսով, մասնավորապես զրո կետի շուրջ $S(0,1)$ միավոր գունդը (էվկլիդյան նորմում) իրենից ներկայացնում է $E(I, 0)$ էլիպսուիդ: Նկատենք,

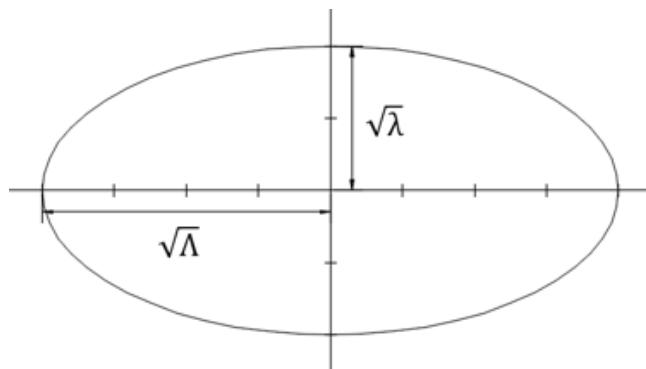
որ E -ն որոշում է A -ն և a -ն միարժեքորեն: a վեկտորը կոչվում է E -ի կենտրոն, իսկ $E(A, a)$ -ն՝ A -ի և a -ի հետ կապված էլիպսություն:

Յուրաքանչյուր դրական որոշված A մատրիցի համար գոյություն ունի միակ դրական որոշված $A^{1/2}$ մատրից, այնպիսին որ $A = A^{1/2}A^{1/2}$: Այստեղից հետևում է, որ

$$E(A, a) = A^{1/2}S(0, 1) + a: \quad (3.3)$$

Այսպիսով յուրաքանչյուր էլիպսություն իրենից ներկայացնում է միավոր գնդի պատկեր՝ բիեկտիվ աֆինյան ձևափոխության ներքո:

Գոյություն ունեն մի շարք հետաքրքիր կապեր $E = E(A, a)$ էլիպսությունի երկրաչափական հատկությունների և A մատրիցի հանրահաշվական հատկությունների միջև: A իրական տարրերով մատրիցի դրական որոշվածությունից հետևում է, որ նրա սեփական արժեքները դրական իրական թվեր են: E -ի տրամագիծ է կոչվում E -ի մեծագույն առանցքի երկարությունը, որը հավասար է $2\sqrt{\Lambda}$, որտեղ Λ -ն A -ի մեծագույն սեփական արժեքն է: E -ի մեծագույն առանցքը համապատասխանում է Λ -ին համապատասխան սեփական վեկտորներին: E -ի լայնություն է կոչվում E -ի կարճագույն առանցքի երկարությունը, որը հավասար է $2\sqrt{\lambda}$, որտեղ λ -ն A -ի փոքրագույն սեփական արժեքն է: Այս դիտարկումները ենթադրում են, որ $S(a, \sqrt{\Lambda})$ -ն $E(A, a)$ -ում պարունակվող մեծագույն գունդն է, իսկ $S(a, \sqrt{\Lambda})$ -ն փոքրագույն գունդն է, որը պարունակում է $E(A, a)$ -ն: Ավելին E -ի սիմետրիկության առանցքը համապատասխանում է A -ի սեփական վեկտորներին: Գրաֆիկորեն էլիպսությունը պատկերված է նկար 3.1-ում, որտեղ $E(A, 0) \subseteq \mathbb{R}^2$ և $A = \text{diag}((16, 4)^T)$: A մատրիցի մեծագույն և փոքրագույն սեփական արժեքներն են համապատասխանաբար՝ $\Lambda = 16$ և $\lambda = 4$, իսկ դրանց համապատասխանող սեփական վեկտորները՝ $e_1 = (1, 0)^T$ և $e_2 = (0, 1)^T$: Այսպիսով, $E(A, 0)$ -ի տրամագիծը՝ $2\sqrt{\Lambda} = 8$, իսկ լայնությունը՝ $2\sqrt{\lambda} = 4$:



Նկ. 3.1

$E = E(A, a)$ էլիպսոիդի ծավալը նշանակենք՝ $vol(E)$: Այն կախված է միայն A մատրիցի դետերմինանտից և տարածության չափողականությունից: Այսինքն՝

$$vol(E(A, a)) = \sqrt{\det A} \cdot V_n, \quad (3.4)$$

որտեղ V_n -ը $S(0,1)$ միավոր գնդի ծավալն է \mathbb{R}^n տարածությունում: Հայտնի է որ

$$V_n = \frac{\pi^{n/2}}{\Gamma(n/2 + 1)} \sim \frac{1}{\sqrt{\pi n}} \left(\frac{2e\pi}{n}\right)^{n/2}, \quad (3.5)$$

որտեղ

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt, \quad x > 0$$

գամմա-ֆունկցիան է: Գամմա ֆունկցիան բավարարում է՝

$$\Gamma(n) = (n - 1)!, \quad \forall n \in \mathbb{N}:$$

Պարզվում է, որ էլիպսոիդների եղանակի նկարագրման համար V_n -ի ստորին և վերին սահմանների համար բավարար է օգտագործել հետևյալ կոպիտ գնահատականը՝

$$n^{-n} \leq V_n \leq 2^n, \quad (3.6)$$

որը ստացվում է այն փաստերից, որ

$S(0,1)$ -ը պարունակում է $\{x \in \mathbb{R}^n : 0 \leq x_i \leq 1/n, i = 1, \dots, n\}$ -ը և $S(0,1)$ -ը պարունակում է $\{x \in \mathbb{R}^n : \|x\|_\infty \leq 1\}$ -ում, որտեղ $\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$ առավելագույն նորմն է:

Եթե $x \mapsto Dx + d$ բիեկտիվ աֆինյան T ձևափոխություն է, ապա

$$vol(T(E(A, a))) = \det D \sqrt{\det A} \cdot V_n:$$

Սա մասնավորապես ցույց է տալիս, որ

$$\frac{vol(E(A, a))}{vol(E(B, b))} = \frac{vol(T(E(A, a)))}{vol(T(E(B, b)))},$$

այսինքն բիեկտիվ աֆինյան ձևափոխության տակ երկու էլիպսոիդների ծավալների հարաբերությունը ինվարիանտ է:

Անհրաժեշտ է օպտիմիզացնել զծային նպատակային ֆունկցիաները էլիպսոիդների վրա: Սա բարդ չէ և կարող է ստացվել այն փաստից, որ եթե $c \neq 0$, $c^T x$ -ը հասնում է իր առավելագույն արժեքին՝ $a + c/\|c\|$ վեկտորի վրա. $\max c^T x$, $x \in S(a, 1)$: Այսինքն ենթադրենք, որ $E(A, a) \subseteq \mathbb{R}^n$ էլիպսոիդ է և դիցուք՝ $c \in \mathbb{R}^n \setminus \{0\}$: Տեղադրենք $Q = A^{1/2}$: Համաձայն (3.3)-ի $Q^{-1}E(A, a) = S(0,1) + Q^{-1}a = S(Q^{-1}a, 1)$ և այսպիսով՝

$$\begin{aligned}
\max\{ c^T x: x \in E(A, a) \} &= \max\{ c^T Q Q^{-1} x: Q^{-1} x \in Q^{-1} E(A, a) \} = \\
&= \max\{ c^T Q y: y \in S(Q^{-1} a, 1) \} = \\
&= c^T Q \frac{1}{\|Qc\|} Qc + c^T Q Q^{-1} a = \\
&= c^T \frac{1}{\sqrt{c^T A c}} A c + c^T a = \\
&= c^T a + \sqrt{c^T A c}:
\end{aligned}$$

Տեղադրելով՝

$$b = \frac{1}{\sqrt{c^T A c}} A c, \quad (3.7)$$

$$z_{max} = a + b,$$

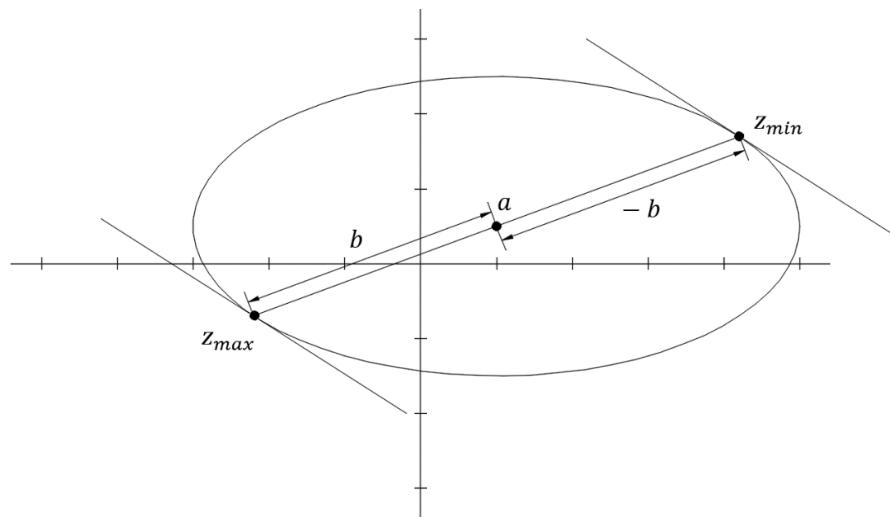
$$z_{min} = a - b,$$

կստանանք՝

$$c^T z_{max} = \max\{ c^T x: x \in E(A, a) \} = c^T a + \sqrt{c^T A c} = c^T a + \|c\|_{A^{-1}}, \quad (3.8)$$

$$c^T z_{min} = \min\{ c^T x: x \in E(A, a) \} = c^T a - \sqrt{c^T A c} = c^T a - \|c\|_{A^{-1}}:$$

Սա նշանակում է, որ z_{max} -ը մաքսիմալացնում է $c^T x$ -ը $E(A, a)$ -ի վրա, իսկ z_{min} -ը՝ մինիմալացնում: z_{max} -ը և z_{min} -ը միացնող զիջը անցնում է $E(A, a)$ -ի a կենտրոնով և ունի b ուղղություն: Նկար 3.2-ում պատկերված է $E(A, a)$ էլիպսով, որտեղ $A = diag((16, 4)^T)$, $a^T = (1, 0.5)$ և նպատակային ֆունկցիան՝ $c^T = (-2, -3)$: (3.7)-ից՝ $b^T = (-3.2, -1.2)$, $z_{max}^T = (1, 0.5) + (-3.2, -1.2) = (-2.2, -0.7)$ և $z_{min}^T = (1, 0.5) - (-3.2, -1.2) = (4.2, 1.7)$:



Նկ. 3.2

Հայտնի փաստ է, որ յուրաքանչյուր ուռուցիկ մարմին պարունակվում է միակ փոքրագույն ծավալի էլիպսուիդի մեջ և պարունակում է միակ մեծագույն ծավալ ունեցող էլիպսուիդ: Այս երկու արդյունքները բացահայտվել են մի քանի մաթեմատիկոսների կողմից՝ իրարից անկախ [21]: Մասնավորապես այս հեղանիկները առաջին արդյունքը վերագրել են Կ. Լոուներին (K. Löwner): Ֆ. Ջոն (F. John) [22] ապացուցեց ավելի ընդհանուր թեորեմ՝

Թեորեմ: Յուրաքանչյուր $K \subseteq \mathbb{R}^n$ ուռուցիկ մարմնի համար գոյություն ունի միակ փոքրագույն ծավալի E էլիպսուիդ, որը պարունակում է K -ն: Ավելին, K -ն պարունակում է էլիպսուիդ, որը ստացվում է E էլիպսուիդից՝ կրծատելով E -ն իր կենտրոնից n գործոնով:

Փոքրագուն ծավալով էլիպսուիդը, որը պարունակում է K ուռուցիկ մարմինը անվանենք K -ի համար Լոուներ-Ջոն (Löwner-John) էլիպսուիդ: Համաձայն վերոնշյալ թեորեմի երկրորդ մասի՝ եթե $E(A, a)$ -ն K -ի համար Լոուներ-Ջոնի էլիպսուիդն է, ապա K -ն պարունակում է $E(n^{-2}A, a)$ էլիպսուիդը:

Կանոնավոր S սիմպլեքսի համար Լոուներ-Ջոնի էլիպսուիդը իրենից ներկայացնում է $E(R^2I, a)$ գունդ՝ S սիմպլեքսի a ծանրության կենտրոնի շուրջ՝ համապատասխան R շառավղով: Համակենտրոն $E(n^{-2}R^2I, a)$ գունդը մեծագույն էլիպսուիդն է, որը պարունակվում է S -ում: Սա ցույց է տալիս, որ վերոնշյալ թեորեմում n պարամետրը հնարավոր լավագույնն է:

Ընդհանուր առմամբ բարդ է հաշվարկել K ուռուցիկ մարմնի համար Լոուներ-Ջոնի էլիպսուիդը: Էլիպսուիդների եղանակի կամ դրա տարատեսակների դեպքում էլիպսուիդալ հատույթների համար օգտագործվում են Լոուներ-Ջոնի էլիպսուիդները: Այս դեպքում հայտնի են Լոուներ-Ջոնի էլիպսուիդների կառուցման բացահայտ բանաձևերը:

Ենթադրենք՝ $E(A, a)$ -ն էլիպսուիդ է և $c \in \mathbb{R}^n \setminus \{0\}$: Այդ դեպքում՝

$$E'(A, a, c) = E(A, a) \cap \{x \in \mathbb{R}^n : c^T x \leq c^T a\}: \quad (3.9)$$

Այսպիսով՝ $E'(A, a, c)$ -ն $E(A, a)$ էլիպսուիդի այն կեսն է, որը ստացվում է $E(A, a)$ -ը $\{x \in \mathbb{R}^n : c^T x = c^T a\}$ հիպերհարթությամբ a կենտրոնով կտրելիս: $E'(A, a, c)$ -ի համար Լոուներ-Ջոնի էլիպսուիդը՝ $E(A', a')$ ստացվում է հետևյալ բանաձևերով.

$$a' = a - \frac{1}{n+1} b,$$

(3.10)

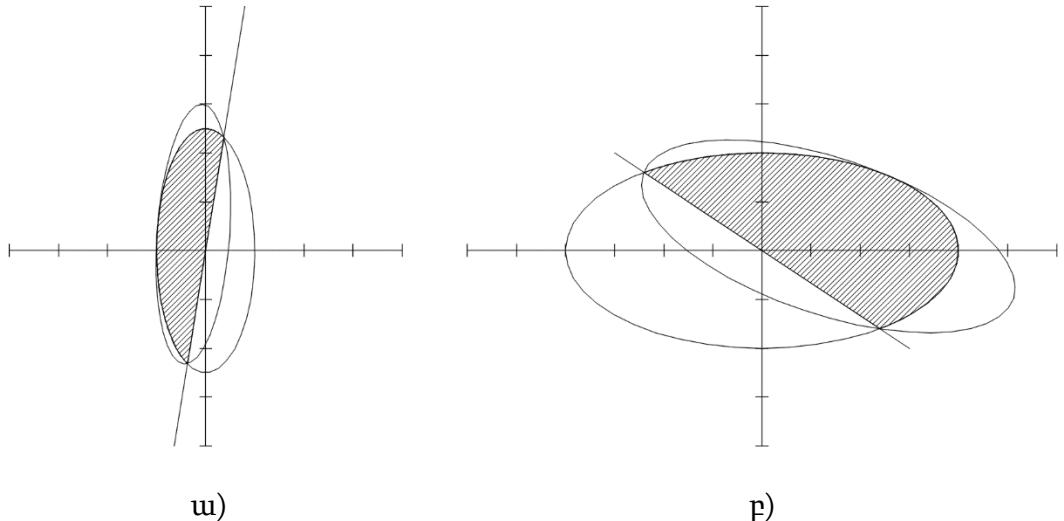
$$A' = \frac{n^2}{n^2-1} \left(A - \frac{2}{n+1} b b^T \right),$$

(3.11)

որտեղ b -ն (3.7)-ով որոշվող վեկտորն է:

Նկար 3.3-ում պատկերված է $E'(A, a, c)$ կիսաէլիպսուիդների (ստվերագծված մասերը) համար $E(A', a')$ Լոռուներ-Զոնի էլիպսուիդները.

- ա) դեպքում՝ $A = diag((1, 25/4)^T)$, $a^T = (0, 0)$, $c^T = (25, -4)$,
- բ) դեպքում՝ $A = diag((16, 4)^T)$, $a^T = (0, 0)$, $c^T = (-2, -3)$:



Նկ. 3.3

Նկատենք, որ $E'(A, a, c)$ կիսաէլիպսուիդի համար կառուցված $E(A', a')$ Լոռուներ-Զոնի էլիպսուիդի a' կենտրոնը ընկած է z_{min} -ը և z_{max} -ը միացնող գծի վրա (տե՛ս (3.7)): Ավելի ճշգրիտ a -ից a' -ին կարելի է անցնել կատարելով $\frac{1}{n+1} \| z_{min} - a \|$ երկարությամբ քայլ a -ից՝ z_{min} -ի ուղղությամբ: Ավելին՝ $E(A', a')$ -ի սահմանը հատում է $E'(A, a, c)$ -ը z_{min} կետում և $\{x: \|x - a\|_A = 1\} \cap \{x: c^T x = c^T a\}$ բազմությունում: \mathbb{R}^2 տարածությունում վերոնշյալ բազմությունը բաղկացած է միայն երկու կետից (տե՛ս նկ.3.3), մինչդեռ \mathbb{R}^3 -ում այն իրենից ներկայացնում է էլիպս:

Դիցուք $E(A, a)$ -ն էլիպսուիդ է, $c \in \mathbb{R}^n \setminus \{0\}$, $\gamma \in \mathbb{R}$: Այդ դեպքում (3.8)-ից հետևում է, որ $H = \{x \in \mathbb{R}^n: c^T x = \gamma\}$ հիպերհարթությունը $E(A, a)$ -ի հետ ունի ոչ դատարկ հատում

այն և միայն այն դեպքում, եթե $c^T z_{min} \leq \gamma \leq c^T z_{max}$, այսինքն այն և միայն այն դեպքում, եթե $|c^T a - \gamma| \leq \sqrt{c^T A c}$:

Ելնելով հարմարության տեսանկյունից՝ կատարենք հետևյալ նշանակումը.

$$\alpha = \frac{c^T a - \gamma}{\sqrt{c^T A c}} : \quad (3.12)$$

Այսպիսով H -ը $E(A, a)$ -ն հատում է այն և միայն այն դեպքում, եթե՝

$$-1 \leq \alpha \leq 1:$$

α -ն կարող է մեկնաբանվել ինչպես \mathbb{R}^n տարածությունում a կենտրոնի «նշանով հեռավորություն» $\{x: c^T x \leq \gamma\}$ կիսատարածության սահմանից՝ օժտված $\|\cdot\|_{A-1}$ նորմով:

(Հեռավորությունը ոչ դրական է, եթե a -ն պարունակվում է կիսատարածությունում):

Անհրաժեշտ է $E(A, a)$ -ն բաժանել 2 մասի օգտագործելով H -ը, այնուհետև հաշվարկել Լոուներ-Ջոնի էլիպտիկը այն մասի համար, որը պարունակվում է $\{x: c^T x \leq \gamma\}$ կիսատարածությունում:

$$E'(A, a, c, \gamma) = E(A, a) \cap \{x \in \mathbb{R}^n: c^T x \leq \gamma\} \quad (3.13)$$

բազմության համար Լոուներ-Ջոնի $E(A', a')$ էլիպտիկը կարող է որոշվել հետևյալ կերպ.

Եթե $-1 \leq \alpha \leq -1/n$, ապա $E(A', a') = E(A, a)$:

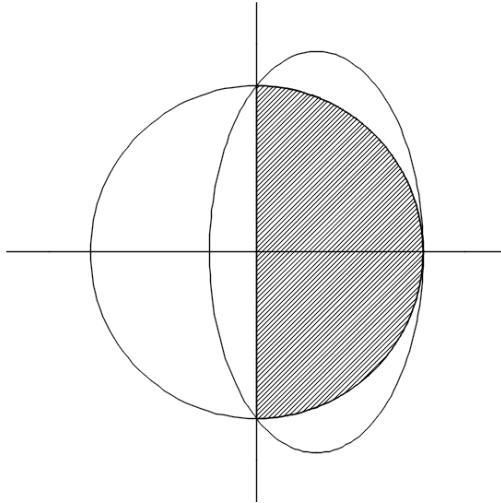
Եթե $-1/n \leq \alpha \leq 1$, ապա $E(A', a')$ -ը որոշվում է հետևյալ կերպ.

$$a' = a - \frac{1 + n\alpha}{n + 1} b, \quad (3.14)$$

$$A' = \frac{n^2}{n^2 - 1} (1 - \alpha^2) \left(A - \frac{2(1 + n\alpha)}{(n + 1)(1 + \alpha)} b b^T \right), \quad (3.15)$$

որտեղ b -ն (3.7)-ով որոշվող վեկտորն է:

Նկատենք, որ եթե $\gamma = c^T a$, ապա $E'(A, a, c, \gamma) = E'(A, a, c)$ և (3.14), (3.15) բանաձևերը ընդունում են համապատասխանաբար (3.10) և (3.11) տեսքերը: Այսպիսով (3.10) և (3.11) բանաձևերով հաշվարկվում է $E(A, a)$ էլիպտիկի $E(A, a, c)$ հատույթի համար (որը ստացվել է $E(A, a)$ -ն a կենտրոնով անցնող H հիպերհարթությամբ բաժանելիս) Լոուներ-Ջոնի էլիպտիկ: Սա կանվանենք կենտրոնական կտրում:



նկ. 3.4

Նկար 3.4-ում պատկերված է Էլիպսուղղների եղանակի դեպքում կենտրոնական կտրումը: Ենթադրվում է, որ $E(A, a)$ -ն $S(0,1)$ միավոր գունդն է, $c^T = (-1, 0)$ վեկտորն օգտագործվում է $E(A, a)$ -ն կտրելու համար: Ստվերագծված մակերեսի համար՝

$$E'(A, a, c) = S(0,1) \cap \{ x: x_1 \geq 0 \}$$

կառուցված է Լոուներ-Ջոնի Էլիպսուղղը:

3.3. Հիմնական Էլիպսուղղների մեթոդի նկարագրությունը

Էլիպսուղղների մեթոդի հիմնական գաղափարը ցույց տալու համար, նկարագրենք այն որպես խիստ ոչ դատարկության խնդիրը լուծող մեթոդ՝ բացահայտորեն տրված պոլիտոպների համար, որոնք կամ դատարկ են, կամ՝ լրիվ չափողականության: Ալգորիթմի մուտք է հանդիսանում հետևյալ անհավասարումների համակարգը՝ $c_i^T x \leq \gamma_i, \quad i = 1, \dots, m$, ամբողջաթիվ գործակիցներով և n փոփոխականներով: Արդյո՞ք,

$$P = \{ x \in \mathbb{R}^n: c_i^T x \leq \gamma_i, \quad i = 1, \dots, m \} = \{ x: Cx \leq d \} \quad (3.16)$$

բազմությունը դատարկ է, թե՞ ոչ՝ ոչ, և եթե այն դատարկ չէ, անհրաժեշտ է P -ում գտնել մի կետ: Ճիշտ պատճենաբանը ստանալու համար մուտքը պետք է բավարարի հետևյալ պայմաններին.

P -ն սահմանափակ է: (3.17)

Եթե P -ն դատարկ չէ, ապա P -ն լրիվ չափայնության է: (3.18)

Հետագայում ցույց կտրվի, որ (3.17) և (3.18) պայմանները անհրաժեշտ չեն, այսինքն՝ փոփոխված մեթոդը աշխատում է նաև ոչ սահմանափակ բազմանիստերի համար,

որոնք լրիվ չափողականության չեն: Ավելին՝ մեթոդը կարող է աշխատել այնպիսի բազմանիստների հետ, որոնք որոշվում են բաժանից օրաքիլի կողմից ներկայացվող անհավասարումներով, այսինքն անհավասարումները նախօրոք տրման կարիք չունեն:

Հիշենք Սահարահում առյուծ որսալու հայտնի մեթոդը: Այն աշխատում է հետևյալ կերպ. ցանկապատել Սահարան և բաժանել երկու մասի, ստուգել՝ ո՞ր մասը չի պարունակում առյուծ, այնուհետև ցանկապատել մյուս մասը և այդպես շարունակել: Վերջավոր թվով քայլերից հետո, որսված կլինի առյուծը (եթե իհարկե կար առյուծ), որովհետև ցանկապատված տարածքը կլինի այնքան փոքր, որ առյուծը այլևս չի կարողանա շարժվել, կամ կհանգենք այն գաղափարին, որ ցանկապատված տարածքը այնքան փոքր է, որ այն չի կարող պարունակել ոչ մի առյուծ, այսինքն՝ ընդհանրապես առյուծ չեր էլ եղել: Ծեր որսորդի այս պատմության միջոցով՝ Էլիպսուղիների մեթոդը ներկայացնելու համար ստիպված ենք նկարագրել, թե ո՞րն է Սահարան, ինչպես բաժանել այն երկու մասի, ինչպես ցանկապատել և ե՞րբ կարելի է հայտարարել որսված առյուծի կամ նրա գոյություն չունենալու մասին:

Սահարայի համար կընտրենք մի գունդ՝ կոռորդինատային սկզբնակետը որպես նրա կենտրոն, ասնեք՝ $S(0, R)$, որը պարունակում է մեր P պոլիտոպը՝ առյուծը: Եթե (3.16) անհավասարումների համակարգը իր փոփոխականների վրա պարունակում է բացահայտորեն տրված վերին և ստորին սահմաններ, ենթադրենք՝ $l_i \leq x_i \leq u_i, i = 1, \dots, n$, ապա R շառավիղը, որի դեպքում $P \subseteq S(0, R)$, հեշտորեն կգտնվի: Օրինակ կարելի է վերցնել

$$R = \sqrt{\sum_{i=1}^n \max \{ u_i^2, l_i^2 \}}: \quad (3.19)$$

Եթե փոփոխականների վրա սահմանները բացահայտորեն տրված չեն, կարելի է օգտագործել այն ինֆորմացիան, որ C -ն և d -ն ամբողջաթիվ են և այն, որ P -ն պոլիտոպ է: R շառավիղը հաշվելու համար ունենք հետևյալը.

Լեմմա:

$$P \subseteq S(0, R), R = \sqrt{n} 2^{< C, d > - n^2}, \quad (3.20)$$

որտեղ $< C, d >$ -ն նշանակում է C -ի և d -ի կողավորման երկարությունը:

Ապացույցը տե՛ս [23]:

Այժմ էլիպսոիդների մեթոդի մեկնարկի կետը կարելի է համարել տրված: Հայտնի է, որ P -ն պարունակում է $S(0, R)$ գնդում, որի R շառավիղը, որոշվում է համաձայն (3.19)-ի կամ (3.20)-ի: Որպես կենտրոն՝ կոորդինատային սկզբնակետով այս գունդը կլինի առաջին էլիպսոիդը՝ $E(A_0, a_0)$ (որի համար հստակ տրված է $A_0 = R^2 I$ և $a_0 = 0$):

Այժմ նկարագրենք պրոցեդուրայի k -րդ քայլը, $k \geq 0$: Համաձայն կառուցման, ընթացիկ

$$E_k(A_k, a_k) \quad (3.21)$$

Էլիպսոիդը պարունակում է P -ն: E_k էլիպսոիդը ունի մեկ բացահայտորեն ընդգծված կետ՝ իր կենտրոն a_k -ն, և ուրեմն a_k -ն պետք է տեղադրել (3.16) անհավասարումների համակարգում և ստուգել՝ արդյո՞ք բոլոր անհավասարումներն են բավարարվում: Եթե բոլոր անհավասարումները բավարարվում են, ավարտումն ենք գործընթացը՝ արձանագրելով որ գտնվել է թույլատրելի լուծումը: Հակառակ դեպքում՝ եթե կենտրոնը թույլատրելի չէ, ուրեմն (3.16) համակարգի առնվազն մեկ անհավասարում խախտվում է, ենթադրենք՝ $c^T x \leq \gamma$, այսինքն ունենում ենք՝ $c^T a_k > \gamma$: Հետևյալ $\{x: c^T x = c^T a_k\}$ հիպերիարթությունը a_k կենտրոնով կտրում է E_k -ն երկու «կեսի» և կառուցումից հայտնի է, որ P պոլիտոպը պարունակում է հետևյալ կետում

$$E'(A_k, a_k, c) = \{x \in E(A_k, a_k): c^T x < c^T a_k\}: \quad (3.22)$$

Հետևաբար էլիպսոիդների հաջորդականության համար որպես հաջորդ E_{k+1} էլիպսոիդ, պետք է ընտրել Լոուներ-Չոնի $E'(A_k, a_k, c)$ էլիպսոիդը, որը տրված է (3.10) և (3.11) բանաձևերով: Պետք է շարունակել այս գործընթացը՝ հաջորդաբար ներառելով P -ն՝ փոքր և ավելի փոքր էլիպսոիդների մեջ:

Հարց է ծագում. ե՞րբ կարելի է ավարտել այս գործընթացը: Պարզ է՝ եթե P -ում կետ գտնվի գործընթացը պետք է ավարտել, բայց որքա՞ն պետք է շարունակել իտերացիաները, եթե ոչ մի հնարավոր կետ չի գտնվել: Ավարտելու չափանիշը որոշվում է էլիպսոիդի ծավալի գործոնից: Քանի որ հայտնի է նախնական $S(0, R)$ էլիպսոիդը,

հետևաբար կարելի է գնահատել նրա ծավալը, օրինակ՝ (3.6)-ի միջոցով: Յուրաքանչյուր k քայլում կառուցվում է E_{k+1} էլիպսոիդը, որի ծավալը խստորեն փոքր է, քան E_k էլիպսոիդի ծավալը: Ավելի ճշգրիտ կարելի է ցույց տալ.

Լեմմա: (3.23)

$$\frac{\text{vol}(E_{k+1})}{\text{vol}(E_k)} = \left(\left(\frac{n}{n+1} \right)^{n+1} \left(\frac{n}{n-1} \right)^{n-1} \right)^{1/2} < e^{-1/(2n)} < 1:$$

Ապացույցը տե՛ս [23]:

(3.17) և (3.18) պայմաններից ելնելով կարելի է վստահ լինել, որ P պոլիտոպը ունի դրական ծավալ, եթե այն դատարկ չէ: Օգտագործելով տվյալների ամբողջաթիվ լինելը և այն փաստը, որ P -ն պոլիտոպ է, կարելի է ապացույցել հետևյալ լեմման.

Լեմմա: Եթե P պոլիտոպը լրիվ չափողականության է, ապա

$$\text{vol}(P) \geq 2^{-(n+1)< C > + n^3}: \quad (3.24)$$

Ապացույցը տե՛ս [23]:

Այժմ կարելի է ավարտել վերլուծությունները: Այսպիսով կարող ենք վերևույթ գնահատել նախնական էլիպսոիդի ծավալը և P -ի ծավալը ներքեւ, եթե P -ն դատարկ չէ, ինչպես նաև հայտնի է ծավալների կրծատման արագությունը: Այդ իսկ պատճառով պետք է շարունակել գործընթացը միչև ընթացիկ էլիպսոիդը ունենա ավելի փոքր ծավալ քան P -ի ծավալի (3.24) ստորին սահմանը: Եթե այս իրավիճակը տեղի ունենա ենթադրենք N -րդ քայլում, ապա պետք է դադարեցնել գործընթացը, քանի որ ստացել ենք հետևյալ հակասությունը՝

$$P \subseteq E_N, \quad (3.25)$$

$$\text{vol}(E_N) < \text{vol}(P):$$

Այս դեպքում կարելի է եզրակացնել, որ P -ն դատարկ է: Վատագույն դեպքում իտերացիաների N քանակը ընտրելով հետևյալ կերպ

$$N = 2n((2n+1)< C > + n < d > - n^3) \quad (3.26)$$

բարդ չէ նկատել, որ $\text{vol}(E_N) < 2^{-(n+1)<\mathcal{C}>+n^3}$ (ստորև ներկայացված լեմման հաստատում է իտերացիաների N քանակի ընտրությունը) Համադրելով (3.24)-ի հետ, կարելի է նկատել, որ (3.25)-ը այսպիսի N -ի դեպքում տեղի ունի:

Լեմմա:

Ենթադրենք $P = \{x \in \mathbb{R}^n : Cx \leq d\}$ լրիվ չափողականության պոլիտոպ է և $Cx \leq d$ ամբողջաթիվ անհավասարումների համակարգ է: Դիցուք՝ $E_0 = E(A_0, a_0)$, որտեղ $a_0 = 0$, $A_0 = R^2 I$ և $R = \sqrt{n} 2^{<\mathcal{C}, d>-n^2}$, սկզբնական էլիպսոիդն է: Եթե էլիպսոիդների հիմնական մեթոդի գլխավոր քայլը կիրառվի $N := 2n((2n+1)<\mathcal{C}> + n<\mathcal{d}>-n^3)$ անգամ, ապա

$$\text{vol}(E_N) < 2^{-(n+1)<\mathcal{C}>+n^3} \leq \text{vol}(P):$$

Ապացույցը տե՛ս [23]:

(3.27) Հիմնական էլիպսոիդների մեթոդը (իսկաւ ոչ դատարկության խնդրի համար՝ լրիվ չափողականության կամ դատարկ պոլիտոպերի դեպքում):

Մուտք. $m \times n$ անհավասարումների համակարգ $Cx \leq d$ ՝ ամբողջաթիվ գործակիցներով: Ենթադրվում է, որ տվյալները բավարարում են հետևյալ պայմաններին՝ $P = \{x \in \mathbb{R}^n : Cx \leq d\}$ -ը սահմանափակ է և կա'մ դատարկ է, կա'մ է՝ լրիվ չափայնության:

Սկզբանարձեքավորում. տեղադրել՝

(ա) $k = 0$,

(բ) $N = 2n((2n+1)<\mathcal{C}> + n<\mathcal{d}>-n^3)$,

(շ) $A_0 = R^2 I$, որտեղ $R = \sqrt{n} 2^{<\mathcal{C}, d>-n^2}$ (կամ օգտագործել հնարավորինս ավելի փոքր R , ինչպես, օրինակ տրված է (3.19)-ում),

$a_0 = 0$ (այսպիսով $E_0 = E(A_0, a_0)$ -ն նախնական էլիպսոիդն է):

Գլխավոր քայլ.

(դ) եթե $k = N$, ավարտել: (Հայտարարել՝ P -ն դատարկ):

(ե) եթե $a_k \in P$, ավարտել: (Թույլատրելի լուծումը գտնված է:)

(q) եթե $a_k \notin P$, ուրեմն $\exists n \text{տրել } Cx \leq d$ անհավասարումների համակարգից, մի անհավասարում, ենթադրենք՝ $c^T x \leq \gamma$, որը խախտված է a_k -ի կողմից:

Տեղադրել՝

$$(է) \quad b = \frac{1}{\sqrt{c^T A_k c}} A_k c \quad (\text{տե՛ս (3.7)}),$$

$$(լ) \quad a_{k+1} = a_k - \frac{1}{n+1} b \quad (\text{տե՛ս (3.10)}),$$

$$(թ) \quad A_{k+1} = \frac{n^2}{n^2 - 1} (A_k - \frac{2}{n+1} b b^T) \quad (\text{տե՛ս (3.11)}),$$

և անցնել քայլ (դ):

(3.27) ալգորիթմը անվանում ենք հիմնական Էլիպսոիդների մեթոդ, քանի որ այն պարունակում է պրոցեդուրայի բոլոր հիմնարար գաղափարները: Ալգորիթմը պոլինոմիալ ժամանակային բարդության կամ ավելի արդյունավետ դարձնելու համար պետք ավելացնել որոշակի տեխնիկական մանրամասներ: Օրինակ՝ պետք է հստակեցնել, թե (ը) քայլում a_{k+1} վեկտորը ինչպես պետք է հաշվարկել: (ը) քայլում a_{k+1} -ի աջ մասը ընդհանուր դեպքում հանգեցնում է իռացիոնալ թվերի, քանի որ (է) քայլում b -ի համար հաշվարկվում է քառակուսի արմատ:

3.4. Էլիպսոիդների մեթոդի իրականացումը և պոլինոմիալությունը

Հիմնական Էլիպսոիդների մեթոդի վատագույն դեպքում արագագործությունը զնահատելու համար յուրաքանչյուր թվաբանական գործողություն համարենք ալգորիթմի մեկ տարրական քայլ: Սկզբնարժեքավորման (ա), (թ), (զ) կետերը կարող են կատարվել $O(mn)$ տարրական քայլերով: Ամեն անգամ ալգորիթմի գլխավոր քայլի կատարման ժամանակ պետք է a_k -ն տեղադրել $Cx \leq d$ անհավասարումների համակարգի մեջ, որը պահանջում է $O(mn)$ տարրական քայլեր, և (ը), (թ) քայլերում անհրաժեշտ է թարմացնել a_k -ն և A_k -ն: (ը) քայլը պահանջում է $O(n)$ տարրական քայլեր, իսկ (թ) քայլը՝ $O(n^2)$: Ալգորիթմի գլխավոր քայլի իրագործվելիության մեծագույն քանակը վերևից սահմանափակ է $N := 2n((2n + 1) < C > + n < d > - n^3) = O(n^2 < C, d >)$ թվով: Քանի որ P -ն պոլիտոպ է՝ $m \geq n$ և այսպիսով հիմնական Էլիպսոիդների մեթոդի տարրական քայլերի գումարային քանակը $O(mn^3 < C, d >)$: Այսինքն պարզվեց տարրական քայլերի քանակի պոլինոմիալ վերին սահմանը:

Որոշ տարրական քայլերի իրականացման հետ կապված գոյություն ունեն որոշակի խնդիրներ: (է) քայլում հաշվարկվող b վեկտորը ընդհանուր առմամբ իռացիոնալ է և այսպիսով a_{k+1} -ն էլ: Քանի որ իռացիոնալ թվերը չունեն վերջավոր ներկայացում երկուական կոդավորման դեպքում, a_{k+1} կենտրոնը հնարավոր չէ ճշտորեն հաշվարկել: Հետևաբար էլիպսոիդների եղանակի ցանկացած իրականացման դեպքում պետք է կլորացնել ամեն հաջորդ էլիպսոիդի կենտրոնը: Այս երևույթը առաջացնում է մի շարք բարդություններ: Կլորացնելով a_{k+1} -ը ստացվում է $\tilde{a}_{k+1} \in \mathbb{Q}^n$ վեկտորը, այսինքն E_{k+1} էլիպսոիդի փոխարեն ստացվում է մի փոքր տեղափոխված $\tilde{E}_{k+1} = E(A_{k+1}, \tilde{a}_{k+1})$ էլիպսոիդը: Ըստ ալգորիթմի E_{k+1} -ը մինիմալ ծավալ ունեցող էլիպսոիդն է, որը պարունակում է $E'(A_k, a_k, c)$ կիսաէլիպսոիդը: Հնարավոր է կլորացման արդյունքում ստացված \tilde{E}_{k+1} էլիպսոիդը այլևս չպարունակի $E'(A_k, a_k, c)$ կիսաէլիպսոիդը: Այսինքն հնարավոր է, որ $P \subseteq E'(A_k, a_k, c)$ պոլիտոպը այլևս չպարունակվի \tilde{E}_{k+1} -ում: Հետևաբար հնարավոր է, որ լինի շեղում ալգորիթմի կենտրանական գաղափարից, այսինքն կառուցվող էլիպսոիդների հաջորդականության մեջ կարող է գտնվել էլիպսոիդ, որը այլևս չի պարունակում P պոլիտոպը:

Անդրադառնանք մյուս հարցին: Դիտարկենք (թ) քայլում կառուցված նոր A_{k+1} մատրիցը, որը ռացիոնալ է, քանի որ A_k -ն ու c -ն նույնպես ռացիոնալ են: Այստեղից պարզ չէ, որ A_{k+1} մատրիցի տարրերը չեն աճի չափազանց արագ:

Փաստորեն պարզ դառձավ որ a_{k+1} կենտրոնի հաշվարկում կլորացումը անխուսափելի է: Կլորացումը կարելի է կատարել հետևյալ կերպ. a_{k+1} կենտրոնի յուրաքանչյուր բաղադրիչ կարելի է ներկայացնել երկուական տեսքով, այնուհետև պահպանել ստորակետից հետո p նիշ: Հետագա շարադրանքում կնշվի p թիվը: Այլ կերպ ասած, կարելի է ֆիքսել 2^p հայտարարը և յուրաքանչյուր թիվ մոտարկել այս հայտարարը ունեցող ռացիոնալ թվով: A_{k+1} մատրիցների կոդավորման երկարությունները կառավարման տակ պահելու համար, կարելի է կիրառել նույն կլորացման մեթոդը A_{k+1} մատրիցի տարրերի նկատմամբ: Ինչպես նշվել էր վերևում կլորացնելով կենտրոնի բաղադրիչները ստացվում է մեկ այլ էլիպսոիդ: A_{k+1} մատրիցի տարրերի կլորացումը առաջացնում է էլիպսոիդի ձևի փոփոխություն: Երկու կլորացումների արդյունքում հնարավոր է նոր էլիպսոիդը չպարունակի P -ն: Ավելին՝ շատ կոպիտ կլորացման դեպքում մատրիցը կարող է կորցնել դրական որոշվածությունը: Այսպիսով պետք է նաև հոգալ այն մասին, որ

կլորացված մատրիցը դեռևս լինի դրական որոշված: p թիվը պետք է ընտրել բավականաչափ մեծ, որպեսզի ալգորիթմի իրականացման ընթացքում ի հայտ եկած բոլոր թվերը ունենան պոլինոմիալ կողավորման երկարություն:

Քանի որ հնարավոր չէ տեղաշարժել P պոլիտոպը, P -ն նոր հաշվարկված էլիպտիկում պահելու համար ինչ-որ հնարք պետք է կիրառվի նոր կառուցված էլիպտիկի համար: Գաղափարներից մեկը, որը աշխատում է կայանում է հետևյալում. ֆիքսել կլորացված կենտրոնը և փշել էլիպտիկը (որը ստացվել էր հաշվի առնելով A_{k+1} մատրիցի տարրերի կլորացումը) այնպես որ փշելու գործընթացը կոմպենսացնի փոքր տեղափոխությունը և Լոուներ-Զոնի էլիպտիկի ձևի՝ կլորացումից առաջացած փոփոխությունները: Փշելու գործակիցը պետք է լինի այնքան մեծ, որ փշելուց հետո էլիպտիկը պարունակի P -ն: Իրականում փշելու գործընթացը պետք է կազմակերպել շատ ուշադիր, քանի որ ըստ (3.1.26)-ի կրճատման գործակիցը փոքր է 1-ից, բայց այն շատ մոտ է 1-ին: Որպեսզի ստանալ պոլինոմիալ ժամանակային կանգառ պետք է ընտրել փշելու գործոնը այնպես, որ մի կողմից ստացվի բավարար կրճատման գործակից, մյուս կողմից որպեսզի երաշխիք լինի, որ P -ն պարունակվում է փշված էլիպտիկում: Սա ցույց է տալիս, որ փշելու գործակիցը և p թիվը, որով որոշում է թվաբանական գործողությունների ձշությունը ազդում են միմյանց վրա:

Պարզվում է, որ N, p և ξ պարամետրերի հետևյալ ընտրության դեպքում՝

$$N = 50(n + 1)^2 < C, d >,$$

$$p = 8N,$$

$$\xi = 1 + \frac{1}{4(n + 1)^2},$$

կարելի հաղթահարել վերը նշված բոլոր դժվարությունները: Այս ձևափոխությունները հիմնական էլիպտիկների ալգորիթմը ձևափոխում են այնպիսի ալգորիթմի, որը ունի պոլինոմիալ ժամանակային բարդություն:

Դրակտիկ տեսանկյունից այս դիտարկումները և ձևափոխությունները թվում են բավականին անհարմար: Նկատենք, որ էլիպտիկների մեթոդը պահանջում է յուրաքանչյուր խնդրի համար այդ խնդրից կախված ձշություն, բայց սովորաբար համակարգիչները ունեն ֆիքսված ձշություն: Մեր խորին համոզմամբ, անգամ օգտագործելով, հատուկ ծրագիր, որը թույլ է տալիս ստանալ փոփոխական ձշություն, անհնարին է թվում

Էլիպսուղիների մեթոդի այս տարբերակի իրականացումը, քանի որ պահանջվում է շատ մեծ ձշություն:

Էլիպսուղիների մեթոդի ֆիքսված ձշությամբ համակարգչային իրականացման դեպքում զաղափարներից մյուսը կայանում է նրանում, որ հնարավոր է եզրակացնել, որ հաշվարկված կենտրոններից մեկը ընկած է P պոլիտոպում, բայց N իտերացիաներից հետո հնարավոր չէ միարժեքորեն պնդել, որ P -ն դատարկ է: P -ի դատարկ լինելը ապացուցելու համար անհրաժեշտ է կիրառել լրացուցիչ ստուգումներ, հիմնված օրինակ Farkas-ի լեմմայի վրա:

3.5. Էլիպսուղիների մեթոդով խնդրի լուծման օրինակ

Օրինակը վերցված է [23]-ից: Դիտարկենք $P \subseteq \mathbb{R}^2$ պոլիտոպը, որը որոշվում է հետևյալ անհավասարումներով.

$$-x_1 - x_2 \leq -2 \quad (3.28)$$

$$3x_1 \leq 4 \quad (3.29)$$

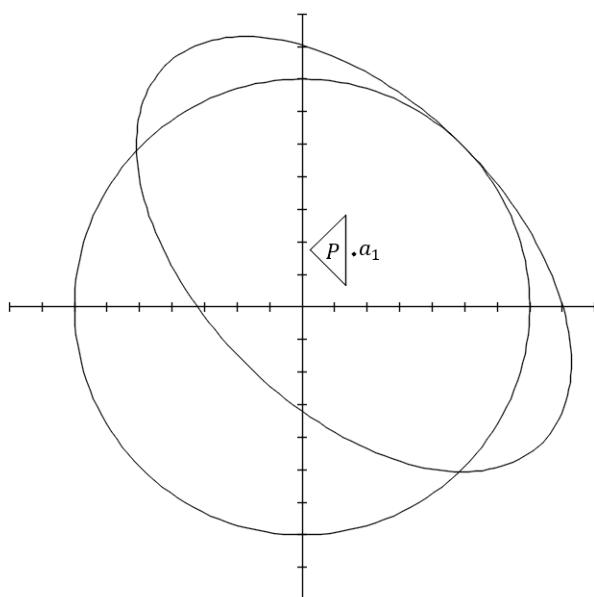
$$-2x_1 + 2x_2 \leq 3 \quad (3.30)$$

Այս պոլիտոպը պարունակվում է 0 կետի շուրջ 7 շառավղով գնդում: P -ում կետ գտնելու համար սկսենք հիմնական էլիպսուղիների մեթոդը $E = (A_0, a_0)$ -ով, որտեղ

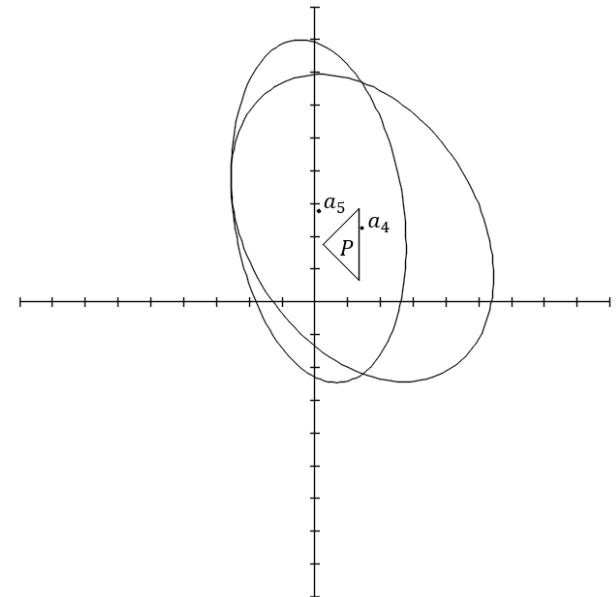
$$A_0 = \begin{pmatrix} 49 & 0 \\ 0 & 49 \end{pmatrix}, \quad a_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

(տե՛ս նկ.3.5):

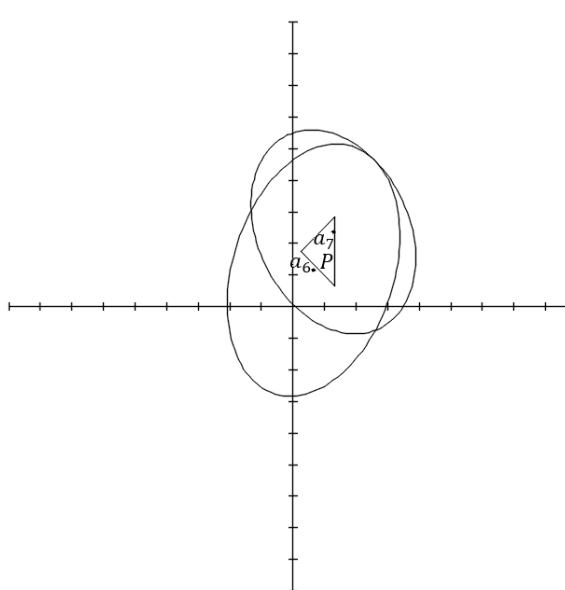
a_0 կենտրոնը խախտում է (3.28) անհավասարումը: Ալգորիթմի առաջին իտերացիայում կառուցում է $E = (A_1, a_1)$ էլիպսուղի, որը նույնպես պատկերված է նկ.3.5-ում: a_1 կենտրոնը խախտում է (3.29) անհավասարումը: 5-րդ իտերացիայում կառուցվում է $E = (A_5, a_5)$ էլիպսուղի, որը պատկերված է նկ.3.6-ում: 7-րդ իտերացիայում կառուցվում է $E = (A_7, a_7)$ էլիպսուղի, որը պատկերված է նկ.3.7-ում: Վերջինիս $a_7^T = (1.2661, 2.3217)$ կենտրոնը պարունակվում է P -ում: Այս օրինակի համար կառուցված բոլոր էլիպսուղիները պատկերված են նկ.3.8-ում:



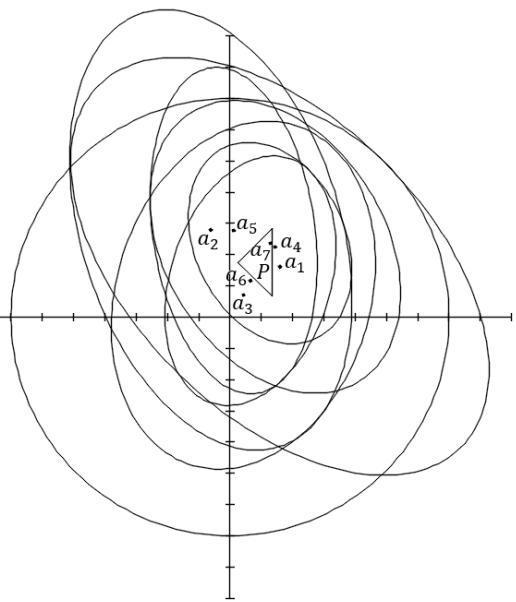
նկ. 3.5



նկ. 3.6



նկ. 3.7



նկ. 3.8

3.6. Սահմանափակության և լրիվ չափողականության ենթադրությունների վերանայում

Վերևում դիտարկված էլիպտիկների եղանակում ենթադրվել էր, որ P բազմանիստը սահմանափակ է (3.17) և կամ դատարկ է, կամ լրիվ չափողականության

(3.18): Այժմ ցույց տանք թե ինչպես կարելի է ազատվել (3.17), (3.18) ենթադրություններից:

Լեմմա: Դիցուք A -ն $m \times n$ ամբողջաթիվ մատրից է, b -ն՝ վեկտոր է \mathbb{R}^m -ում, U -ն՝ A -ի և b -ի տարրերի բացարձակ արժեքներից մեծագույնն է: Այդ դեպքում՝

ա) $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ բազմանիստի յուրաքանչյուր էքստրեմալ կետ բավարարում է հետևյալին՝

$$-(nU)^n \leq x_j \leq (nU)^n, \quad j = 1, \dots, n,$$

բ) $P = \{x \in \mathbb{R}^n : Ax = b\}$ ստանդարտ տեսքի բազմանիստի յուրաքանչյուր էքստրեմալ կետ բավարարում է հետևյալին՝

$$-(mU)^m \leq x_j \leq (mU)^m, \quad j = 1, \dots, n:$$

Ապացույցը տե՛ս [24]:

Համաձայն վերոնշյալ լեմմայի՝ $P = \{x \in \mathbb{R}^n : Ax \geq b\}$ բազմանիստի բոլոր էքստրեմալ կետերը պարունակվում են $P_B = \{x \in P : |x_j| \leq (nU)^n, j = 1, \dots, n\}$ սահմանափակ բազմանիստում: Ենթադրվում է, որ A -ի տողերը ընդգրկում են \mathbb{R}^n -ը և P -ն դատարկ չէ այն և միայն այն դեպքում, եթե այն ունի էքստրեմալ կետ: Սա տեղի ունի այն և միայն այն դեպքում, եթե P_B -ն դատարկ չէ, և ալգորիթմը կարող է կիրառվել P_B -ի վրա՝ P -ի փոխարեն: Նկատենք, որ P_B -ն սահմանափակ բազմանիստ է, որը պարունակվում է $E(0, n(nU)^{2n}I)$ գնդում: Ճետևաբար որպես սկզբնական էլիպսիդ կարելի է ընտրել $E_0 = E(0, n(nU)^{2n}I)$ էլիպսիդը: Նկատենք, որ E_0 -ի ծավալը ավելի փոքր է քան $V = (2n(nU)^n)^n = (2n)^n(nU)^{n^2}$:

Այժմ քննարկենք այն ենթադրությունը, որ եթե P բազմանիստը դատարկ չէ, ապա այն լրիվ չափողականության է, այսինքն ունի դրական ծավալ: Ինչու՞ է անհրաժեշտ այս ենթադրությունը: Եթե P -ն դատարկ չէ, բայց ունի n -ից փոքր չափողականություն, այդ դեպքում $vol(P) = 0$: Օրինակ դիտարկենք $P = \{(x_1, x_2) : x_1 + x_2 = 1, x_1, x_2 \geq 0\}$ բազմանիստը: Ակնհայտ է, որ նրա ծավալը 0 է, թեև բազմանիստը դատարկ չէ: Քանի որ բազմանիստը ունի զրոյական ծավալ էլիպսիդների մեթոդը N իտերացիա կատարելուց հետո կհաստատի, որ P -ն դատարկ է, սակայն իրականում այն դատարկ չէ:

Հետևյալ լեմման ցույց է տալիս, որ ոչ դատարկ բազմանիստի փոքր չափով գրգռումից ստացվում է լրիվ չափողականության բազմանիստ.

Լեմմա: Դիցուք $P = \{x \in \mathbb{R}^n : Ax \geq b\}$: Ենթադրենք՝ A -ն և b -ն ամբողջաթիվ են և բոլոր տարրերի բացարձակ արժեքները սահմանափակված են U -ով: Դիցուք

$$\epsilon = \frac{1}{2(n+1)} ((n+1)U)^{-(n+1)}, \quad P_\epsilon = \{x \in \mathbb{R}^n : Ax \geq b - \epsilon 1\}:$$

Այս դեպքում՝

ա) եթե P -ն դատարկ է, ապա P_ϵ -ը դատարկ է:

բ) եթե P -ն դատարկ չէ, ապա P_ϵ -ը լրիվ չափողականության է:

Ապացույցը տե՛ս [24]:

3.7. Էլիպտիկների մեթոդը գծային ծրագրավորման համար

Դիտարկենք հետևյալ ուղիղ-երկակի գծային ծրագրավորման խնդիրների զույգը.

$$\max c^T x;$$

$$Ax \leq b; \quad x \geq 0$$

և

$$\min b^T y;$$

$$A^T y \geq c, \quad y \geq 0:$$

Համաձայն խիստ երկակիության՝ ուղիղ և երկակի խնդիրներն ունեն օպտիմալ լուծումներ, այն և միայն այն դեպքում եթե գծային անհավասարումների հետևյալ

$$c^T x = b^T y; \tag{3.31}$$

$$Ax \leq b;$$

$$x \geq 0;$$

$$A^T y \geq c;$$

$$y \geq 0$$

համակարգը համատեղ է: Ենթադրենք Q -ն (3.31) անհավասարումների համակարգի թույլատրելի բազմությունն է: Կարելի է կիրառել էլիպտիկների եղանակը՝ պարզելու համար Q -ն դատարկ է, թէ ոչ: Եթե իրոք Q -ն դատարկ չէ և ստացվել է (x, μ) թույլատրելի լուծումը, ապա x -ը հանդիսանում է ուղիղ խնդրի լուծումը, իսկ μ -ն՝ երկակի խնդրի: Եթե Q -ն լրիվ չափողականության չէ, ապա վերը նկարագրված լեմմայի համաձայն այն կարելի է ϵ չափով գրգռել, այնուհետև կիրառել էլիպտիկների մեթոդը Q_ϵ -ի վրա: Արդյունքում ստացված $(x_\epsilon, \mu_\epsilon) \in Q_\epsilon$ կետը հնարավոր է, որ ընկած չլինի Q -ում: Բայց հաշվի առնելով, որ ϵ -ը բավականաշահ փոքր է՝ կիրառելով համապատասխան կլորացման գործընթաց կարելի է ստանալ Q -ում ընկած կետը:

ԳԼՈՒԽ 4

ԱՐԴՅՈՒՆՔՆԵՐԻ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆ

4.1. Երկֆազ սիմպլեքս մեթոդով ԳԾ խնդիրների լուծման օրինակներ

Մագիստրոսական ատենախոսության կատարման ընթացքում մշակվել են երկֆազ սիմպլեքս և Էլիպսուիդների մեթոդների ծրագրային իրականացումները և դրանց միջոցով լուծվել են մի շաբթ մոդելային և կիրառական օրինակներ: Երկֆազ սիմպլեքս մեթոդի ծրագրային իրականացումը (տե՛ս Հավելված 3) հիմնված [24]-ի և [25]-ի վրա, իսկ Էլիպսուիդների մեթոդի ծրագրային իրականացման համար աղբուր են հանդիսացել [5]-ը, [23]-ը, [24]-ը և [26]-ը: Ներկայացնենք օրինակներից մի քանիսը.

Դիտարկենք Քլիի և Մինթի (Klee-Minty) օրինակը n փոփոխականի դեպքում.

$$\begin{aligned} & \max \sum_{j=1}^n 10^{n-j} x_j; \\ & 2 \sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \leq 100^{i-1}; \quad i = 1, \dots, n; \\ & x_j \geq 0; \end{aligned}$$

$n = 3$ փոփոխականի դեպքում Քլիի և Մինթի օրինակը ունի հետևյալ տեսքը.

$$\begin{aligned} & \max 100x_1 + 10x_2 + x_3; \tag{4.1} \\ & x_1 \leq 1; \\ & 20x_1 + x_2 \leq 100; \\ & 200x_1 + 20x_2 + x_3 \leq 10000; \\ & x_1, x_2, x_3 \geq 0; \end{aligned}$$

(4.1) խնդրի լուծումը ամենաբացասական տարրի ընտրման կանոնով (most-negative-entry pivoting rule) աշխատող երկֆազ սիմպլեքս եղանակով տե՛ս Հավելված 2-ում: Ինչպես հայտնի է Քլիի և Մինթի օրինակի դեպքում սիմպլեքս մեթոդը իրականացնում է $2^n - 1$ խտերացիա: Վերոնշյալ դեպքում՝ 7 խտերացիա: Քլիի և Մինթի օրինակը ցույց է տալիս, որ գոյություն ունեն խնդիրներ, որոնց դեպքում սիմպլեքս մեթոդը ունի էքսպոնենցիալ ժամանակային բարդություն:

Դիտարկենք հետևյալ օրինակը.

$$\begin{aligned}
 & \max 10x_1 - 57x_2 - 9x_3 - 24x_4; \\
 & 0.5x_1 - 5.5x_2 - 2.5x_3 + 9x_4 \leq 0; \\
 & 0.5x_1 - 1.5x_2 - 0.5x_3 + x_4 \leq 0; \\
 & x_1 \leq 1; \\
 & x_1, x_2, x_3, x_4 \geq 0;
 \end{aligned} \tag{4.2}$$

Այս օրինակի դեպքում, օգտագործելով ամենաբացասական տարրի ընտրման կանոնը (most-negative-entry pivoting rule), երկֆազ սիմպլեքս եղանակը ընկնում է ցիկլ, իսկ Բլենդի կանոնով աշխատող երկֆազ սիմպլեքս եղանակի կիրառման դեպքում բացառվում է ցիկլը և ստացվում է օպտիմալ լուծումը: Օրինակը վերցված է [25]-ից: Ամենաբացասական տարրի ընտրման կանոնով աշխատանքը, այսինքն ցիկլ ընկնելը, ինչպես նաև Բլենդի կանոնի կիրառման դեպքում օպտիմալ լուծման ստացումը տե՛ս Հավելված 2-ում՝ համապատասխանաբար 4.2 ա) և բ) դեպքեր:

(4.3) օրինակում նպատակային ֆունկցիան սահմանափակ չէ:

$$\begin{aligned}
 & \max 2x_1 + x_3; \\
 & x_1 - x_2 + x_3 \leq 5; \\
 & -2x_1 + x_2 \leq 3; \\
 & x_2 - 2x_3 \leq 5; \\
 & x_1, x_2, x_3 \geq 0;
 \end{aligned} \tag{4.3}$$

Երկֆազ սիմպլեքս եղանակի աշխատանքը ցույց է տրված Հավելված 2-ում:

(4.4) օրինակը ունի անվերջ քանակով օպտիմալ լուծումներ:

$$\begin{aligned}
 & \max 30x_1 + 20x_2 + 10x_4; \\
 & x_1 + 2x_2 + 3x_3 + 4x_4 \leq 80; \\
 & 3x_1 + 2x_2 + 12x_3 + 32x_4 \leq 120; \\
 & x_1, x_2, x_3, x_4 \geq 0;
 \end{aligned} \tag{4.4}$$

Երկֆազ սիմպլեքս եղանակի աշխատանքը ցույց է տրված Հավելված 2-ում:

(4.5) օրինակը պարունակում է ավելորդ սահմանափակում: Երկֆազ սիմպլեքս եղանակը առաջին ֆազով հեռացնում է ավելորդ սահմանափակումը, այնուհետև երկրորդ ֆազով լուծում է խնդիրը:

$$\begin{aligned} & \min x_1 + x_2 + x_3; \\ & x_1 + 2x_2 + 3x_3 = 3; \\ & -x_1 + 2x_2 + 6x_3 = 2; \\ & 4x_2 + 9x_3 = 5; \\ & 3x_3 \leq 1; \\ & x_1, x_2, x_3 \geq 0: \end{aligned} \tag{4.5}$$

Երկֆազ սիմպլեքս եղանակի աշխատանքը ցույց է տրված Հավելված 2-ում:

(4.6) օրինակը անհամատել է:

$$\begin{aligned} & \max x_1 - x_2 + x_3; \\ & 2x_1 - x_2 - 2x_3 \leq 4; \\ & 2x_1 - 3x_2 - x_3 \leq -5; \\ & -x_1 + x_2 + x_3 \leq -1; \\ & x_1, x_2, x_3 \geq 0: \end{aligned} \tag{4.6}$$

Երկֆազ սիմպլեքս եղանակի աշխատանքը ցույց է տրված Հավելված 2-ում:

4.2. Էլիպսուիդների մեթոդով ԳԾ խնդիրների լուծման օրինակներ

Դիտարկենք հետևյալ խնդիրը.

$$\begin{aligned} & \max x_1 + x_2; \\ & 2x_1 + x_2 \leq 2; \\ & x_1 \geq 0; \\ & x_2 \geq 0: \end{aligned} \tag{4.7}$$

Այս խնդիրը էլիպսուիդների մեթոդով լուծելու համար նախ պետք է ստուգել սահմանափակումների համատեղելիությունը: Թեև օրինակը շատ պարզ է, բայց այնուամենայնիվ ենթադրենք, որ խնդիրի սահմանափակումներով որոշվող բազմանիստի չափողականության և սահմանափակության մասին գաղափար չունենք: Էլիպսուիդների մեթոդով խնդիրի սահմանափակումների համատեղելիու-

թյունը պարզելու համար նախ և առաջ ենթադրյալ ոչ սահմանափակ P բազմանիստը անհրաժեշտ է սահմանափակել՝ ավելացնելով բոլոր փոփոխականների վրա ավելացնելով հետևյալ սահմանափակումը՝

$$-(nU)^n \leq x_j \leq (nU)^n, \quad j = 1, \dots, n,$$

որտեղ U -ն խնդրի A մատրիցի և b ազատ անդամների վեկտորի տարրերից մեծագույնն է՝ բացարձակ արժեքով, n -ը՝ փոփոխականների քանակը: Քանի որ $x_1, x_2 \geq 0$ կարելի է ավելացնել միայն

$$x_j \leq (nU)^n, \quad j = 1, \dots, n,$$

տեսակի սահմանափակումները: Կստանանք հետևյալ սահմանափակումներով որոշվող P_b բազմանիստը.

$$\begin{aligned} 2x_1 + x_2 &\leq 2; \\ x_1 &\geq 0; \\ x_2 &\geq 0; \\ x_1 &\leq 16; \\ x_2 &\leq 16: \end{aligned} \tag{4.8}$$

(4.8) սահմանափակումների համատեղելիությունը էլիպտիդների մեթոդով ստուգելու համար անհրաժեշտ է, որ այդ սահմանափակումներով որոշվող բազմանիստը E^2 էվկլիդյան տարածությունում լինի լրիվ չափողականության: Ներկայացնենք (4.8) սահմանափակումներով որոշվող բազմանիստը էլիպտիդների եղանակի համար հարմար տեսքի, այսինքն՝ մեր ծրագրային իրականացման դեպքում, ապահովենք բոլոր անհավասարումների \leq տեսքի լինելը.

$$\begin{aligned} 2x_1 + x_2 &\leq 2; \\ -x_1 &\leq 0; \\ -x_2 &\leq 0; \\ x_1 &\leq 16; \\ x_2 &\leq 16: \end{aligned} \tag{4.9}$$

Բազմանիստի լրիվ չափողականությունը ապահովելու համար անհրաժեշտ է անհավասարումների (4.9) համակարգը գրգռել

$$\epsilon = \frac{1}{2(n+1)} ((n+1)U)^{-(n+1)},$$

չափով: Այս խնդրի դեպքում՝ $\epsilon = 1/663552(n = 2, U = 16)$: Արդյունքում կստանանք $P_{b,\epsilon}$ բազմանիստը.

$$\begin{aligned} 2x_1 + x_2 &\leq 2 + \epsilon; \\ -x_1 &\leq 0 + \epsilon; \\ -x_2 &\leq 0 + \epsilon; \\ x_1 &\leq 16 + \epsilon; \\ x_2 &\leq 16 + \epsilon; \end{aligned} \tag{4.10}$$

որը սահմանափակ է և որի համար տեղի ունի ներքոհիշյալ պայմաններից կամ ա)-ն, կամ բ)-ն:

ա) Եթե P -ն դատարկ է, ապա $P_{b,\epsilon}$ -ը դատարկ է:

բ) Եթե P -ն դատարկ չէ, ապա $P_{b,\epsilon}$ -ը լրիվ չափողականության է:

Սահմանափակումների (4.10) համակարգի համատեղելիությունը ստուգենք էլիպտիդ-ների մեթոդով, այսինքն փորձենք գտնել այդ սահմանափակումներին բավարարող որևէ կետ: Եթե այդպիսի կետ չգտնվի կարող ենք արձանագրել, որ P -ն դատարկ է, այսինքն (4.7) ԳԾ խնդիրը լուծում չունի: Հավելված 2-ում ցույց է տրված, որ (4.10) սահմանափակումների համակարգը համատեղ է, այսինքն գտնվել է կետ, որը բավարարում է բոլոր անհավասարումներին: Հակառակ դեպքում՝ եթե (4.10) համակարգը լիներ անհամատեղ, ապա նրա (4.12) երկակի խնդրի կամ անսահմանափակությունը, կամ անհամատեղելիությունը պարզելու համար անհրաժեշտ կլիներ (4.12) խնդրի հետ վարվել այնպես ինչպես (4.7) խնդրի հետ և եթե էլիպտիդների մեթոդը կիրառելուց հետո պարզվեր, որ (4.12) անհավասարումների համակարգը համատեղ է, ապա դրանից կհետևեր, որ (4.12) խնդիրը սահմանափակ չէ, հակառակ դեպքում՝ անհամատեղ է:

Հաջորդ քայլում անհրաժեշտ է կազմել (4.7) ուղիղ խնդրի երկակի խնդիրը.

$$\begin{aligned} \min 2y_1; \\ 2y_1 &\geq 1; \\ y_1 &\geq 1; \\ y_1 &\geq 0; \end{aligned} \tag{4.11}$$

(4.11) խնդիրը համարժեք է հետևյալ խնդրին.

$$\min 2y_1; \quad (4.12)$$

$$y_1 \geq 1:$$

Համաձայն խիստ երկակիության՝ (4.7) և (4.12) ուղիղ և երկակի խնդիրներն ունեն օպտիմալ լուծումներ, այն և միայն այն դեպքում, եթե գծային անհավասարումների հետևյալ համակարգը՝

$$2x_1 + x_2 \leq 2; \quad (4.13)$$

$$x_1 \geq 0;$$

$$x_2 \geq 0;$$

$$y_1 \geq 1;$$

$$x_1 + x_2 = 2y_1,$$

համատեղ է:

(4.13) համակարգի համատեղելիությունը էլիպսոիդների մեթոդով պարզելու համար՝ մեր ծրագրային իրականացման դեպքում, անհրաժեշտ է վերջին հավասարումը ներկայացնել 2 անհավասարումների տեսքով, այնուհետև բոլոր անհավասարումները բերել \leq տեսքի: Արդյունքում կստանանք հետևյալ անհավասարումների համակարգը.

$$2x_1 + x_2 \leq 2; \quad (4.14)$$

$$-x_1 \leq 0;$$

$$-x_2 \leq 0;$$

$$-y_1 \leq -1;$$

$$x_1 + x_2 - 2y_1 \leq 0;$$

$$-x_1 - x_2 + 2y_1 \leq 0:$$

Լրիվ չափողականություն ապահովելու համար անհրաժեշտ է անհավասարումների (4.14) համակարգը գրգռել $\epsilon = 1/32768$ չափով ($U = 2$, $n = 3$): Արդյունքում կստանանք

$$2x_1 + x_2 \leq 2 + \epsilon; \quad (4.15)$$

$$-x_1 \leq 0 + \epsilon;$$

$$-x_2 \leq 0 + \epsilon;$$

$$-y_1 \leq -1 + \epsilon;$$

$$x_1 + x_2 - 2y_1 \leq 0 + \epsilon;$$

$$-x_1 - x_2 + 2y_1 \leq 0 + \epsilon:$$

համակարգը:

Հավելված 2-ում ցույց է տրված, որ (4.15) սահմանափակումների համակարգը համատեղ է, այսինքն գտնվել է կետ, որը բավարարում է բոլոր անհավասարումներին: Այդ կետը հետևյալն է

$$x_1 = 0.000056075173312666376800238656,$$

$$x_2 = 1.999918033209372197590416633064,$$

$$y_1 = 0.999998522115067279328531376385:$$

Այսինքն (x_1, x_2) -ը հանդիսանում է (4.7) ուղիղ խնդրի $\epsilon = 1/32768$ ճշտությամբ լուծումը, նպատակային ֆունկցիայի՝ $f^* = 1.999974108382684863967216871720$ արժեքով, իսկ y_1 -ը՝ (4.12) երկակի խնդրի $\epsilon = 1/32768$ ճշտությամբ լուծումը, նպատակային ֆունկցիայի՝ $f_* = 1.999997044230134558657062752770$ արժեքով: Հիշեցնենք, որ եթե (4.15) անհավասարումների համակարգը լիներ անհամատեղ, ապա կարձանագրեինք, որ (4.7) ԳԾ խնդիրը սահմանափակ չէ:

Թեև (4.16) օրինակում առանց որևէ լուծման երևում է, որ խնդիրը անհամատեղ է, այնուամենայնտիվ ցույց տանք անհամատեղելիության արձանագրումը էլիպտիկ-ների մեթոդով:

$$\min x_1 + x_2; \quad (4.16)$$

$$x_1 + x_2 \geq 1;$$

$$x_1 + x_2 \leq -1;$$

$$x_1 \geq 0;$$

$$x_2 \geq 0:$$

Ինչպես նախորդ դեպում սահմանափակումներին ավելացնենք սահմանափակության պայմանները՝ x_1, x_2 փոփոխականների վրա, այնուհետև համակարգը ենթարկենք ϵ գրգռման՝ լրիվ չափողականություն ապահովելու համար: Սահմանափակության պայմանների ավելացումից հետո այս խնդրի համար $\epsilon = 1/10368(n = 2, U = 4)$: Կստանանք հետևյալ սահմանափակումների համակարգը.

$$-x_1 - x_2 \leq -1 + \epsilon; \quad (4.17)$$

$$x_1 + x_2 \leq -1 + \epsilon;$$

$$-x_1 \leq 0 + \epsilon;$$

$$-x_2 \leq 0 + \epsilon;$$

$$x_1 \leq 4 + \epsilon; \quad x_2 \leq 4 + \epsilon;$$

Կատարենք (4.17) անհավասարումների համակարգի համատեղելիության ստուգում. տե՛ս Հավելված 2: Ստացանք, որ (4.17)-ը անհամատեղ է, այսինքն (4.16)-ը չունի լուծում: Պարզ է, որ (4.16)-ի երկակի խնդիրը չունի վերջավոր օպտիմում, սակայն անհամատեղելիության կամ ոչ սահմանափակության մասին ոչինչ չենք կարող ասել, առանց երկակի խնդիրը վերոնշյալ ձևով ուսումնասիրելու:

Դիտարկենք հետևյալ ԳԾ խնդիրը.

$$\begin{aligned} & \max x_1 + x_2; \\ & x_1 - x_2 \leq 1; \\ & x_1 \geq 0; \\ & x_2 \geq 0; \end{aligned} \tag{4.18}$$

Թեև պարզության պատճառով ակնհայտ է այս խնդիրի ոչ սահմանափակությունը, այնուամենայնիվ ոչ սահմանափակ լինելը ստանանք էլիպտիզմների մեթոդով:

$$\begin{aligned} & x_1 - x_2 \leq 1; \\ & x_1 \geq 0; \\ & x_2 \geq 0; \end{aligned} \tag{4.19}$$

(4.19) անահավասարումների համակարգին պետք է ավելացվեն սահմանափակության պայմանները՝ x_1, x_2 փոփոխականների վրա, որից հետո պետք է բերել ծրագրային իրականացմանը հարմար տեսքի(բոլոր սահմանափակումները պետք է լինեն \leq տեսքի), վերջում անհրաժեշտ է համակարգը զրգուել $\epsilon = 1/10368(n = 2, U = 4)$ չափով:

Արդյունքում կստանանք հետևյալ անհավասարումների համակարգը.

$$\begin{aligned} & x_1 - x_2 \leq 1 + \epsilon; \\ & -x_1 \leq 0 + \epsilon; \\ & -x_2 \leq 0 + \epsilon; \\ & x_1 \leq 4 + \epsilon; \\ & x_2 \leq 4 + \epsilon; \end{aligned} \tag{4.20}$$

(4.20) անհավասարումների համակարգը համատեղ է, այսինքն գտնվել է կետ, որը բավարարում է բոլոր անհավասարումներին(տե՛ս Հավելված 2):

(4.18) ուղիղ խնդիրի երկակի խնդիրը ունի հետևյալ տեսքը.

$$\begin{aligned} & \min y_1; \\ & y_1 \geq 1; \\ & -y_1 \geq 1; \quad y_1 \geq 0; \end{aligned} \tag{4.21}$$

(4.21) խնդիրը համարժեք է հետևյալ խնդրին.

$$\min y_1;$$

$$y_1 \geq 1;$$

$$-y_1 \geq 1:$$

Համաձայն խիստ երկակիության՝ (4.18) և (4.21) ուղիղ և երկակի խնդիրներն ունեն օպտիմալ լուծումներ, այն և միայն այն դեպքում, եթե գծային անհավասարումների հետևյալ համակարգը՝

$$x_1 - x_2 \leq 1; \quad (4.22)$$

$$x_1 \geq 0;$$

$$x_2 \geq 0;$$

$$y_1 \geq 1;$$

$$-y_1 \geq 1;$$

$$x_1 + x_2 - y_1 = 0;$$

համատեղ է:

(4.22) անահավասարումների համակարգը պետք է բերել ծրագրային իրականացմանը հարմար տեսքի այնուհետև լրիվ չափողականությունը ապահովելու համար անհրաժեշտ է զրգուել $\epsilon = 1/2048(n = 3, U = 1)$ չափով:

Արդյունքում կստանանք հետևյալ համակարգը.

$$x_1 - x_2 \leq 1 + \epsilon; \quad (4.23)$$

$$-x_1 \leq 0 + \epsilon;$$

$$-x_2 \leq 0 + \epsilon;$$

$$-y_1 \leq -1 + \epsilon;$$

$$y_1 \leq -1 + \epsilon;$$

$$x_1 + x_2 - y_1 \leq 0 + \epsilon;$$

$$-x_1 - x_2 + y_1 \leq 0 + \epsilon;$$

Անհավասարումների (4.23) համակարգը անհամատեղ է (լուծման լայնածավալությունից ելնելով այն չենք ներկայացնի): (4.23) համակարգի անհամատեղելիությունից և (4.20)-ի համատեղելիությունից հետևում է, որ (4.18) խնդիրը սահմանափակ չէ: Այստեղից հետևում է, որ (4.21) երկակի խնդիրը անհամատեղ է:

ԵԶՐԱԿԱՑՈՒԹՅՈՒՆ

Մեր տեղեկությունների համաձայն մինչ այժմ հայտնի չէ բազմանդամային (պոլինոմիալ) բարդություն ունեցող սիմպլեքս մեթոդի տարբերակ, սակայն էլիպտիդների մեթոդը ԳԾ խնդիրների համար ապահովում է պոլինոմիալ բարդությամբ լուծելիություն։ Ըստհանրապես էլիպտիդների մեթոդը ունի ոչ խիստ պոլինոմիալ ժամանակային բարդություն, այսինքն մեթոդի ժամանակային բարդությունը կախված է նաև ԳԾ խնդրի գործակիցների թվային արժեքների մեծությունից, մինչդեռ համաձայն խիստ պոլինոմիալ ալգորիթմի սահմանման՝ ժամանակային բարդությունը պետք է կախված լինի միայն փոփոխականների n քանակից և/կամ սահմանափակումների m քանակից, բայց ոչ ԳԾ խնդրի նկարագրման համար պահանջվող L բիթային երկարությունից։ Էլիպտիդների եղանակը տեսականորեն արդյունավետ է (այսինքն պոլինոմիալ ժամանակային է), սակայն գործնականում՝ մեր կատարած փորձերի արդյունքները արձանագրել են բավականաշափ դանդաղ զուգամիտություն։ Մեթոդը անգամ տրիվիալ խնդիրների դեպքում պահանջում է շատ մեծ հաշվողական ճշտություն։ Նույնիսկ մեթոդում կիրառված կոպիտ գնահատականների լավարկման դեպքում կառուցվող էլիպտիդների քանակը և պահանջվող հաշվարկային ճշտությունը թվում են շատ մեծ՝ ոչ տրիվիալ խնդիրների լուծման դեպքում։ Մեր համոզմամբ էլիպտիդների եղանակը ներկա դրությամբ չի կարող համարվել սիմպլեքս եղանակին մրցակից՝ գործնական արդյունավետության տեսանկյունից (չնայած գոյություն չունի ապացույց այն քանի մասին, որ հնարավոր չէ էլիպտիդների մեթոդը լավարկել այնքան, որ այն համարվի գործնականում շատ արդյունավետ)։ Սիմպլեքս մեթոդը գործնականում համարվում է շատ արդյունավետ մեթոդ ԳԾ դասի խնդիրների լուծման հարցում, սակայն ինչպես քաջ հայտնի է տեսականորեն սիմպլեքս ալգորիթմը ունի էքսպոնենցիալ ժամանակային բարդություն՝ համաձայն մեր տեղեկությունների, տանող տարրի ընտրման ցանկացած կանոնի դեպքում սիմպլեքս ալգորիթմի համար գոյություն ունեն այսպես կոչված Քլիի և Մինթիի խորանարդներ, որոնց լուծումը պահանջում է $2^n - 1$ քանակով իտերացիաներ, որտեղ n -ը փոփոխականների թիվն է։ Էլիպտիդների մեթոդի դեպքում նման օրինակներ գոյություն չունեն։ Համաձայն [25]-ի՝ սիմպլեքս մեթոդով, համապատասխան տանող տարրի ընտրման կանոնի դեպքում, $n = 50$ փոփոխականով Քլիի և Մինթիի օրինակը լուծելու համար կպահանջվի մոտավորապես

300000 տարի, եթե համարենք, որ 1 վայրկյանում մեթոդը կատարում է 100 իտերացիա: Մեր կարծիքով, նույն օրինակը, նույն համակարգչի դեպքում, Էլիպսուիդների մեթոդի համապատասխան մոդիֆիկացիայով լուծելիս, կպահանջվի համեմատաբար խելամիտ ժամանակահատված: Համաձայն [27]-ի՝ մոդիֆիկացված Էլիպսուիդների մեթոդով լուծվել են 50 փոփոխականից կազմված ԳԾ խնդիրներ, որոնցից որոշները պահանջել են ավելի քան 24000 իտերացիա: Որպես առավելություն կարելի է համարել այն փաստը, որ ի տարբերություն սիմպլեքս մեթոդի էլիպսուիդների եղանակը չի պահանջում բացահայտ կերպով ԳԾ խնդրի սահմանափակումների նկարագրություն: Այն միայն պահանջում է պոլինոմիալ ժամանակային բարդությամբ բաժանիչ օրաք, որը ստուգում է արդյո՞ք կետը պատկանում է տիրույթին, թե ոչ, և եթե ոչ, ապա վերադարձնում է բաժանիչ հիպերհարթությունը:

Մեր խորին համոզմամբ, թեև մինչ այժմ Լ. Գ. Խաչիյանի կողմից էլիպսուիդների մեթոդի հարմարեցումը ԳԾ դասի խնդիրներում չի արդարացրել իր գործնական արդյունավետությունը, սակայն տեսության մեջ անգնահատելի մեծ դեր է ունեցել՝ տալով մի խոշոր հարցի պատասխան ԳԾ դասի խնդիրների պոլինոմիալ ժամանակային բարդությամբ լուծելիության վերաբերյալ՝ նրանց դասակարգելով \mathcal{P} դասի խնդիրների շարքին: Փաստորեն էլիպսուիդների մեթոդը հանդիսացավ ապացույցը այն բանի, որ ալգորիթմների տեսականորեն ապացուցված պոլինոմիալ բարդության և գործնական կիրառության միջև կա արժեքների վերանայման անհրաժեշտություն:

Մեր կողմից մշակված էլիպսուիդների մեթոդի տարբերակի համար հետազա հետազոտման առարկա են հանդիսանում հետևյալ խնդիրները.

- 1) Սկզբնական էլիպսուիդի շառավղի համար տրված գնահատականի լավարկումը,
- 2) Տրված խնդրի համար օպտիմալ հաշվողական ճշտության (այս դեպքում մեր կողմից մշակված Fixed_Point_Arithmetic տիպի համար պահանջվող ամբողջ և տասնորդական մասերի համար թվանշանների քանակի) ընտրումը,
- 3) Վլորացումների կոմպենսացման համար ընտրված այսպես կոչված Էլիպսուիդների փշման տեխնիկայի վերանայումը՝ ընտրելով տրված խնդրի համար պահանջվող հաշվողական ճշտությունից և փոփոխանների քանակից կախված փշման օպտիմալ գործակցի արժեք:

Օգտագործված Գրականության ցանկ

- [1] G.B. Dantzig (1951): Maximization of a linear function of variables subject to linear inequalities. In: Activity Analysis of Production and Allocation (T.C. Koopmans, ed.), Wiley, New York 1951, pp. 359–373
- [2] А.И. Беников (2005), Линейное программирование: Учебное пособие.-Иркутск: Иркутский университет, 2005.-147с.
- [3] L. G. Khachiyan (1979), "A polynomial algorithm in linear programming" (in Russian), Doklady Akademii Nauk SSSR 244 (1979) 1093-1096 (English translation: Soviet Mathematics Doklady 20 (1979) 191-194).
- [4] V. Klee and G. J. Minty (1972), "How good is the simplex algorithm?", in: O. Shisha (ed.), Inequalities III, Academic Press, New York, 1972, 159-175.
- [5] R. G. Bland, D. Goldfarb and M. J. Todd (1981), "The ellipsoid method: a survey", Operations Research 29 (1981) 1039-1091.
- [6] R. Schrader (1982), "Ellipsoid methods", in: B. Korte (ed.), Modern Applied Mathematics - Optimization and Operations Research, North-Holland, Amsterdam, 1982, 265-311.
- [7] N. Z. Shor (1970a), "Utilization of the operation of space dilatation in the minimization of convex functions" (in Russian), Kibernetika (Kiev) 1970 (1) (1970) 6-12 (English translation: Cybernetics 6 (1970) 7-15).
- [8] N. Z. Shor (1970b), "Convergence rate of the gradient descent method with dilatation of the space" (in Russian), Kibernetika (Kiev) 1970 (2) (1970) 80-85 (English translation: Cybernetics 6 (1970) 102-108).

- [9] D. B. Yudin and A. S. Nemirovskii (1976a), "Evaluation of the informational complexity of mathematical programming problems" (in Russian), *Ekonomika i Matematicheskie Metody* 12 (1976) 128-142 (English translation: *Matekon* 13 (2) (1976-7) 3-25).
- [10] D. B. Yudin and A. S. Nemirovskii (1976b), "Informational complexity and efficient methods for the solution of convex extremal problems" (in Russian), *Ekonomika i Matematicheskie Metody* 12 (1976) 357-369 (English translation: *Matekon* 13 (3) (1977) 25- 45).
- [11] A. Yu. Levin (1965), "On an algorithm for the minimization of convex functions" (in Russian), *Doklady Akademii Nauk SSSR* 160 (1965) 1244-1247 (English translation: Soviet Mathematics *Doklady* 6 (1965) 286-290).
- [12] N. Z. Shor (1977), "Cut-off method with space extension in convex programming problems" (in Russian), *Kibernetika* (Kiev) 1977 (1) (1977) 94-95 (English translation: *Cybernetics* 13 (1977) 94-96).
- [13] L. G. Khachiyan (1980), "Polynomial algorithms in linear programming" (in Russian), *Zhurnal Vychislitelnoi Matematiki i Matematicheskoi Fiziki* 20 (1980) 51-68 (English translation: USSR Computational Mathematics and Mathematical Physics 20 (1980) 53-72).
- [14] R. M. Karp and C. H. Papadimitriou (1980), "On linear characterization of combinatorial optimization problems", in: 21th Annual Symposium on Foundations of Computer Science (Syracuse, New York, 1980), IEEE, New York, 1980, 1-9 (final publication: SI AM Journal on Computing 11 (1982) 620-632).
- [15] M. W. Padberg and M. R. Rao (1981), "The Russian method for linear programming III: Bounded integer programming", Research Report 81-39, New York University, Graduate School of Business Administration, New York, 1981.

- [16] M. Grötschel, L. Lovasz and A. Schrijver (1981), "The ellipsoid method and its consequences in combinatorial optimization", *Combinatorica* 1 (1981) 169-197.
- [17] M. Iri and H. Imai (1986), "A multiplicative barrier function method for linear programming", *Algorithmica* 1 (1986) 455-482.
- [18] G. de Ghellinck and J.-R Vial (1986), "A polynomial Newton method for linear programming", *Algorithmica* 1 (1986) 425-453.
- [19] U. Betke and R Gritzmann (1986), "Projection algorithms for linear programming", *Mathematischer Forschungsbericht* 176/1986, Universitat Gesamthochschule Siegen, Siegen, 1986.
- [20] G. Sonnevend (1986), "An "analytical centre" for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming", in: A. Prekopa, J. Szelezsan and B. Strazicky (eds.), *System Modelling and Optimization (Proceedings of the 12-th IFIP Conference, Budapest, 1985)*, Lecture Notes in Control and Information Sciences 84, Springer, Berlin 1986, 866-875.
- [21] L. Danzer, B. Grunbaum and V. Klee (1963), "Helly's theorem and its relatives", in: V. L. Klee (ed.), *Convexity*, American Mathematical Society, Providence, Rhode Island, 1963, 101-180.
- [22] F. John (1948), "Extremum problems with inequalities as subsidiary conditions", in: *Studies and Essays, presented to R. Courant on his 60th Birthday, January 8, 1948*, Interscience, New York, 1948, 187-204 (reprinted in: J. Moser (ed.), *Fritz John, Collected Papers Volume 2*, Birkhauser, Boston, Massachusetts, 1985, 543-560).

[23] M. Grötschel, L. Lovász, and A. Schrijver (1988), Geometric Algorithms and Combinatorial Optimization (Springer-Verlag, New York, 1988).

[24] D. Bertsimas and J.N. Tsitsiklis (1997): Introduction to Linear Optimization, Athena Scientific, Belmont, Massachusetts 1997.

[25] V. Chvátal (1983), Linear Programming. Freeman, New York 1983.

[26] C. H. Papadimitriou and K. Steiglitz (1982), Combinatorial Optimization: Algorithms and Complexity Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[27] S. Halfin (1983), The sphere method and the robustness of the ellipsoid algorithm, Mathematical Programming 26 (1983) 109–116.

[28] B. Korte and J. Vygen (2005), Combinatorial Optimization: Theory and Algorithms, third edition. Springer, 2005.

[29] A.J. Hoffman and J.B. Kruskal (1956): Integral boundary points of convex polyhedra. In: Linear Inequalities and Related Systems; Annals of Mathematical Study 38 (H.W. Kuhn, A.W. Tucker, eds.), Princeton University Press, Princeton 1956, pp. 223–246.

[30] J. von Neumann (1947): Discussion of a maximum problem. Working paper. Published in: John von Neumann, Collected Works; Vol. VI (A.H. Taub, ed.), Pergamon Press, Oxford 1963, pp. 27–28.

[31] D. Gale, H.W. Kuhn and A.W. Tucker (1951): Linear programming and the theory of games. In: Activity Analysis of Production and Allocation (T.C. Koopmans, ed.), Wiley, New York 1951, pp. 317–329.

[32] G. Farkas (1894): A Fourier-féle mechanikai elv alkalmazásai. Matematikai és Természettudományi Értesítő 12 (1894), 457–472.

[33] H. Minkowski (1896): Geometrie der Zahlen. Teubner, Leipzig 1896.

- [34] E. Steinitz (1916): Bedingt konvergente Reihen und konvexe Systeme. *Journal für die reine und angewandte Mathematik* 146 (1916), 1–52.
- [35] H. Weyl (1935): Elementare Theorie der konvexen Polyeder. *Commentarii Mathematici Helvetici* 7 (1935), 290–306.
- [36] A. Schrijver (1986): *Theory of Linear and Integer Programming*. Wiley, Chichester 1986.

ՀԱՎԵԼՎԱԾ 1

ԳԾ ՀԵՏ ԿԱՊՎԱԾ ՓԱՍՏԵՐ

Հ.1.1. ԳԾ ընդհանուր խնդիրը

Այստեղ անդրադարձել ենք Գծային Ծրագրավորման հետ կապված մի քանի շատ կարևոր փաստերի:

Ընդհանուր խնդիրը հետևյալն է.

Տրված են $A \in \mathbb{R}^{m \times n}$ մատրիցը և $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ վեկտոր սյուները: Խնդիրը կայանում է հետևյալում.

1. Կա՞մ գտնել այնպիսի $x \in \mathbb{R}^n$ վեկտոր-սյուն, որ $Ax \leq b$ և $c^T x$ -ը առավելագույնն է,
2. Կա՞մ ցույց տալ, որ $\{x \in \mathbb{R}^n : Ax \leq b\}$ բազմությունը դատարկ է,
3. Կա՞մ ցույց տալ, որ $\forall \alpha \in \mathbb{R}$ համար $\exists x \in \mathbb{R}^n$, այնպիսին որ $Ax \leq b$ և $c^T x > \alpha$:

$\max\{c^T x : Ax \leq b\}$ ԳԾ խնդրի թույլատրելի լուծում է կոչվում $x \in \mathbb{R}^n$ վեկտորը, որի համար $Ax \leq b$: Թույլատրելի լուծումը կոչվում է օպտիմալ, եթե $c^T x$ -ը առավելագույնն է: x և y վեկտորների համար $x \leq y$ նշանակում է, որ անհավասարությունը տեղի ունի այդ վեկտորների յուրաքանչյուր բաղադրիչի համար (ենթադրվում է, որ x և y վեկտորները նույն չափի են): Հետագա շարադրանքում, եթե նշված չեն մատրիցների և վեկտորների չափերը, ենթադրվում է, որ նրանք չափերի առումով համապատասխանում են:

Ինչպես նշվեց գոյություն ունի 2 հնարավոր դեպք, երբ ԳԾ խնդիրը լուծում չունի.

- 1) Խնդիրը կարող է լինել ոչ համատեղ, այսինքն $P = \{x \in \mathbb{R}^n : Ax \leq b\} = \emptyset$,
- 2) Խնդիրը կարող է լինել անսահմանափակ, այսինքն $\forall \alpha \in \mathbb{R}$ համար $\exists x \in \mathbb{R}^n$, այնպիսին որ $Ax \leq b$ և $c^T x > \alpha$:

Եթե տեղի չունեն $n \geq 1$, $n \geq 2$ դեպքերը, ապա ԳԾ խնդիրը ունի օպտիմալ լուծում: Այսինքն $\sup\{c^T x : Ax \leq b\}$ նշանակման փոխարեն արդարացի է $\max\{c^T x : Ax \leq b\}$ նշանակումը:

Հ.1.2 Բազմանիստեր

ԳԾ խնդիրն է վերջավոր քանակով փոփոխականներից կազմված նպատակային ֆունկցիայի մաքսիմիզացիան կամ մինիմիզացիան վերջավոր քանակով սահմանափակումների դեպքում: Թույլատրելի լուծումների բազմությունը իրենից ներկայացնում է վերջավոր քանակով կիսատարածությունների հատում: Այդպիսի բազմությունը կոչվում է բազմանիստ:

Սահմանում: $\mathbb{R}^n - n$ չափանի Էվկլիդյան տարածությունում բազմանիստ է կոչվում հետևյալ բազմությունը $P = \{x \in \mathbb{R}^n: Ax \leq b\}$, որտեղ $A \in \mathbb{R}^{m \times n}$ և $b \in \mathbb{R}^m$: Եթե A -ն և b -ն ուղիղության են, ապա P -ն ուղիղության բազմանիստ է: Սահմանափակ բազմանիստին անվանում են պոլիտոպ: A մատրիցի n ռանգը նշանակենք $rank(A)$: $X \subseteq \mathbb{R}^n$ ոչ դատարկ բազմության չափանություն է կոչվում

$$dim(X) = n - max(rank(A): A \in \mathbb{R}^{n \times n} \text{ և } \forall x, y \in X: Ax = Ay)$$

$P \in \mathbb{R}^n$ բազմանիստը կոչվում է լրիվ-չափողականության եթե $dim(P) = n$: Համարժեքորեն կարող ենք պնդել, որ բազմանիստը լրիվ-չափողականության է այն և միայն այն դեպքում, եթե գոյություն ունի կետ, որը բազմանիստի համար ներքին է:

Հիմնականում տվյալ ենթագլխի սահմանումներում էական չէ թե ինչպիսի տարածության մասին է խոսքը՝ իրական թե ուղիղության մասին:

Սահմանում: Դիցուք $P = \{x: Ax \leq b\}$ ոչ դատարկ բազմանիստ է: Եթե c ոչ զրոյական վեկտորի համար $\delta = max\{c^T x: Ax \leq b\}$ բազմությունը վերջավոր է, ապա $\{x: c^T x = \delta\}$ բազմությունը կոչվում է P բազմանիստի համար աջակցող հիպերիարթություն: P բազմանիստի դեմք է կոչվում P -ն ինքնին կամ P -ի և աջակցող հիպերիարթության հատումը ($P = \{x: Ax \leq b\} = \emptyset$ բազմանիստի համար \emptyset բազմությունը համարվում է դեմք, ի տարբերություն մնացած դեմքերի, որոնք համարվում են պատշաճ, \emptyset և P դեմքերը համարվում են ոչ պատշաճ): x կետը, որի համար $\{x\}$ -ը դեմք է, կոչվում է P -ի գագաթ, ինչպես նաև $Ax \leq b$ համակարգի բազմային լուծում:

Պնդում: Դիցուք $P = \{x: Ax \leq b\}$ բազմանիստ է և $F \subseteq P$: Այդ դեպքում հետևյալ 3 պնդումները համարժեք են՝

- 1) F -ի դեմքն է,
- 2) $\exists c$ վեկտոր այնպիսին, որ $\delta = max\{c^T x: x \in P\}$ -ը վերջավոր է և $F = \{x \in P: c^T x = \delta\}$
- 3) $Ax \leq b$ համակարգի որոշ $A'x \leq b'$ ենթահամակարգի համար $F = \{x \in P: A'x = B'\} \neq \emptyset$

Ապացույցը տե՛ս [28]:

Հետևանք: Եթե $\max\{c^T x: x \in P\}$ -ը սահմանափակ է ոչ դատարկ P բազմանիստի և c վեկտորի համար, ապա կետերի բազմությունը, որտեղ $\{c^T x: x \in P\}$ հասնում է առավելագույն արժեքի, P բազմանիստի դեմքն է:

Հետևանք: Դիցուք P -ն բազմանիստ է և F -ը P -ի դեմքն է: Այդ դեպքում F -ը կրկին բազմանիստ է: Ավելին $F' \subseteq F$ բազմությունը P -ի դեմքն է, այն և միայն այն դեպքում, եթե այն F -ի դեմքն է:

Սահմանում: Դիցուք P -ն բազմանիստ է: Առավելագույն P -ից տարբերվող դեմքը կոչվում է P -ի նիստ: $c^T x \leq \delta$ անհավասարումը P -ի համար նիստ-որոշող է, եթե $\forall x \in P, c^T x \leq \delta$ և $\{x \in P: c^T x = \delta\}$ բազմությունը P -ի նիստն է:

Պնդում: Դիցուք $P \subseteq \{x \in \mathbb{R}^n: Ax = b\}$ $\dim(P) = n - \text{rank}(A)$ չափայնության ոչ դատարկ բազմանիստ է, $A'x \leq b'$ նվազագույն անհավասարումների համակարգ է, այնպիսին որ $P = \{x: Ax = b, A'x \leq b'\}$: Այդ դեպքում $A'x \leq b'$ անհավասարումների համակարգի յուրաքանչյուր անհավասարում P -ի համար նիստ-որոշող է, և յուրաքանչյուր նիստ որոշվում է $A'x \leq b'$ համակարգի անհավասարումներից մեկով: Ապացույցը տե՛ս [28]:

Պնդում: [29] Դիցուք $P = \{x: Ax \leq b\}$ բազմանիստ է: $F \subseteq P$ ոչ դատարկ ենթաբազմությունը P -ի համար մինիմալ դեմք է այն և միայն այն դեպքում, եթե $F = \{x: A'x = b'\}, Ax \leq b$ համակարգի $A'x \leq b'$ ենթահամակարգի համար:

Ապացույցը տե՛ս [28]:

ԳԾ խնդիրը կարող է լուծվել վերջավոր ժամանակում՝ $Ax \leq b$ համակարգի յուրաքանչյուր $A'x \leq b'$ ենթահամակարգի համար լուծելով $A'x = b'$ գծային հավասարումների համակարգը:

Հետևանք: Դիցուք $P = \{x \in \mathbb{R}^n: Ax \leq b\}$ բազմանիստ է: Այդ դեպքում P -ի բոլոր նվազագույն դեմքերը ունեն $n - \text{rank}(A)$ չափայնություն: Բազմանիստի նվազագույն դեմքերը գագաթներն են:

Այս է պատճառը, որ $\{x \in \mathbb{R}^n : Ax \leq b\}$, $rank(A) = n$ բազմանիստերը կոչվում են սրածայր. նրանց փոքրագույն դեմքերը կետեր են:

Սահմանում: \mathbb{R}^n էվկլիում տարածության P ենթաբազմությունը կոչվում է k -սիմպլեքս ($k = 0, 1, 2, \dots$) եթե այն $k + 1$ աֆինյան անկախ կետերի ուրուցիկ թաղանթն է: Այն ունի $k + 1$ հատ գագաթ և $k + 1$ հատ նիստ:

Հ.1.3. ԳԾ երկակիություն

Սահմանում: Տրված է $\max\{c^T x : Ax \leq b\}$ ԳԾ խնդիրը: Այս խնդրի համար երկակի խնդիր է կոչվում $\min\{b^T y : A^T y = c, y \geq 0\}$: Սկզբնական ԳԾ խնդրին հաճախ անվանում են ուղիղ խնդիր:

Պնդում: ԳԾ երկակի խնդրի երկակին սկզբնական ԳԾ խնդրն է (ուղիղ խնդիր): Ապացույցը տե՛ս [28]:

Երկակիության թեորեմ: [30], [31]

Եթե $P = \{x : Ax \leq b\}$ և $D = \{y : A^T y = c, y \geq 0\}$ բազմանիստները դատարկ չեն, ապա $\max\{c^T x : x \in P\} = \min\{b^T y : y \in D\}$:

Կարող ենք ավելին ասել ԳԾ ուղիղ և երկակի խնդրների օպտիմալ լուծումների մասին:

Հետևանք: Դիցուք $\max\{c^T x : Ax \leq b\}$ և $\min\{b^T y : A^T y = c, y \geq 0\}$ ԳԾ ուղիղ-երկակի գույգ են, x -ը և y -ը թույլատրելի լուծումներ են, այսինքն $Ax \leq b$, $A^T y = c$ և $y \geq 0$: Այդ դեպքում հետևյալ պնդումները համարժեք են.

1. x -ը և y -ը օպտիմալ լուծումներ են,
 2. $c^T x = b^T y$,
 3. $y^T(b - Ax) = 0$: (հաճախ անվանում են լրացնուցիչ ոչ կոշտություն)
- Ապացույցը տե՛ս [28]:

Հետևանք: Դիցուք $\min\{c^T x: Ax \geq b, x \geq 0\}$ և $\max\{b^T y: A^T y \leq c, y \geq 0\}$ ԳԾ ուղիղ-երկակի զույգ են, x -ը և y -ը թույլատրելի լուծումներ են, այսինքն $Ax \geq b, A^T y \leq c, x \geq 0, y \geq 0$: Այդ դեպքում հետևյալ պնդումները համարժեք են.

1. x -ը և y -ը օպտիմալ լուծումներ են,
2. $c^T x = b^T y$,
3. $(c - A^T y) x^T = 0$ և $y^T (b - Ax) = 0$:

Ապացույցը տե՛ս [28]:

3-րդ պնդմանը հաճախ անվանում են ուղիղ և երկակի լրացուցիչ ոչ կոշտության պայմաններ:

Երկակիության թեորեմը ունի շատ կիրառություններ: Թեորեմի կարևորություններից մեկը կայանում է նրանում, որ ուղիղ խնդրի լուծման օպտիմալությունը կարող է ապացուցվել՝ նույն նպատակային ֆունկցիայի դեպքում տալով երկակի խնդրի համար թույլատրելի լուծում:

Թեորեմ: $\exists x$ վեկտոր, այնպիսին որ $Ax \leq b$, այն և միայն այն դեպքում, եթե յուրաքանչյուր $y \geq 0$, $y^T A = 0$ համար $y^T b \geq 0$:

Ապացույցը տե՛ս [28]:

Հետևանք: $\exists x \geq 0$ վեկտոր, այնպիսին որ $Ax \leq b$, այն և միայն այն դեպքում եթե յուրաքանչյուր $y \geq 0$, $y^T A \geq 0$ համար $y^T b \geq 0$:

Ապացույցը տե՛ս [28]:

Հետևանք: [32] $\exists x \geq 0$ վեկտոր, այնպիսին որ $Ax = b$, այն և միայն այն դեպքում եթե յուրաքանչյուր y , $y^T A \geq 0$ համար $y^T b \geq 0$:

Այս հետևանքը հայտնի է որպես Ֆարկաշի լեմմա:

Թեորեմ: Եթե ԳԾ խնդիրը սահմանափակ չէ, ապա երկակի խնդիրը անհամատելի է: Եթե ԳԾ խնդիրը ունի օպտիմալ լուծում, ապա երկակին նույնպես ունի օպտիմալ լուծում:

Ապացույցը տե՛ս [28]:

Գոյություն ունի 4 դեպք ուղիղ-երկակի ԳԾ խնդրի գույզի համար (տե՛ս աղյուսակ 1)։

- 1) Կա՛մ երկուսն էլ ունեն օպտիմալ լուծում (նպատակային ֆունկցիայի արժեքները համընկնում են),
- 2) Կա՛մ երկուսն էլ անհամատեղ են,
- 3) Կա՛մ ուղիղ խնդիրը անհամատեղ է և երկակի խնդիրը սահմանափակ չէ,
- 4) Կա՛մ երկակի խնդիրը անհամատեղ է և ուղիղ խնդիրը սահմանափակ չէ։

	Վերջավոր օպտիմում	Ոչ սահմանափակ	Անհամատեղ
Վերջավոր օպտիմում	Հնարավոր է	Անհնար է	Անհնար է
Ոչ սահմանափակ	Անհնար է	Անհնար է	Հնարավոր է
Անհամատեղ	Անհնար է	Հնարավոր է	Հնարավոր է

աղ. 1: ԳԾ ուղիղ-երկակի խնդիրների գույզի համար հնարավոր և անհնար դեպքերը։

Թեորեմ: Դիցուք $P = \{ x \in \mathbb{R}^n : Ax \leq b \}$ բազմանիստ է և $z \notin P$: Այդ դեպքում գոյություն ունի բաժանիչ հիպերհարթություն, այսինքն $\exists c \in \mathbb{R}^n$, որի համար $c^T z > \max\{c^T x : Ax \leq b\}$:

Ապացույցը տե՛ս [28]:

Հ.1.4. Ուռուցիկ թաղանթներ և բազմանիստեր

Սահմանում: Տրված են $x_1, \dots, x_k \in \mathbb{R}^n$ և $\lambda_1, \dots, \lambda_k \geq 0$, $\sum_{i=1}^k \lambda_i = 1$, $x = \sum_{i=1}^k \lambda_i x_i$ անվանում ենք x_1, \dots, x_k ուռուցիկ համակցություն: $X \subseteq \mathbb{R}^n$ բազմությունը ուռուցիկ է, եթե $\lambda x + (1 - \lambda)y \in X$ բոլոր $x, y \in X$ և $\lambda \in [0, 1]$ համար: X բազմության ուռուցիկ թաղանթը $conv(X)$ սահմանվում է որպես X բազմության կետերի բոլոր հնարավոր ուռուցիկ համակցությունների բազմություն: X բազմության ծայրակետ է կոչվում $x \in X$, $x \notin conv(X \setminus \{x\})$:

Այսպիսով X բազմությունը ուռուցիկ է այն և միայն այն դեպքում, եթե այդ բազմության կետերի բոլոր ուռուցիկ համակցությունները պատկանում են այդ բազմությանը: X բազմության ուռուցիկ թաղանթը X բազմությունը պարունակող փոքրագույն բազմությունն է: Ուռուցիկ բազմությունների հատումը նույնպես ուռուցիկ բազմություն է: Այդ պատճառով բազմանիստները ուռուցիկ են:

Թեորեմ: [33], [34], [35] P բազմությունը պոլիտոպ է այն և միայն այն դեպքում, եթե այն P -ի կետերի վերջավոր բազմության ուռուցիկ թաղանթն է:

Ապացույցը տե՛ս [36]:

Հետևանք: Բազմանիստը իր զագարների ուռուցիկ թաղանթն է:
Ապացույցը տե՛ս [28]:

ՀԱՎԵԼՎԱԾ 2

ԵՐԿՅԱԶ ՍԻՄՊԼԵՔՍ ԵՎ ԷԼԻՊՍՈՒԴՆԵՐԻ ԱԼԳՈՐԻԹՄՆԵՐԻ
ԱՇԽԱՏԱՆՔԸ

Thu May 3 15:51:29 2018

Preprocessing Linear Programming Problem:

max: 100 10 1
Subject to:
1 0 0 <= 1
20 1 0 <= 100
200 20 1 <= 10000

Changing Objective Function to Minimum Form:

min: -100 -10 -1
Subject to:
1 0 0 <= 1
20 1 0 <= 100
200 20 1 <= 10000

Linear Programming Problem in Standard Form:

min: -100 -10 -1 0 0 0
Subject to:
1 0 0 1 0 0 = 1
20 1 0 0 1 0 0 = 100
200 20 1 0 0 1 0 = 10000

Starting Simplex Algorithm:
Precision of Calculations: 1e-06

STARTING SIMPLEX ALGORITHM:
(Initial Basic Feasible Solution Available without Solving First Phase)

[Table: 1 (Initial Table)]

	x1	x2	x3	x4	x5	x6	RHS	Basis
	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	4
	20.000000	1.000000	0.000000	0.000000	1.000000	0.000000	100.000000	5
	200.000000	20.000000	1.000000	0.000000	0.000000	1.000000	10000.000000	6
	-100.000000	-10.000000	-1.000000	0.000000	0.000000	0.000000	-0.000000	(-Z)

Հ.2.1. Երկֆազ սիմպլեքս ալգորիթմի աշխատանքը

(4.1) խնդիրի լուծումը:

[Iteration: 1] After Entering: x_1 and Leaving: x_4

[Table: 2]

x_1	x_2	x_3	x_4	x_5	x_6	RHS	Basis
1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	1
0.000000	1.000000	0.000000	-20.000000	1.000000	0.000000	80.000000	5
0.000000	20.000000	1.000000	-200.000000	0.000000	1.000000	9800.000000	6
0.000000	-10.000000	-1.000000	100.000000	0.000000	0.000000	100.000000	(-Z)

[Iteration: 2] After Entering: x_2 and Leaving: x_5

[Table: 3]

x_1	x_2	x_3	x_4	x_5	x_6	RHS	Basis
1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	1
0.000000	1.000000	0.000000	-20.000000	1.000000	0.000000	80.000000	2
0.000000	0.000000	1.000000	-20.000000	1.000000	0.000000	8200.000000	6
0.000000	0.000000	-1.000000	10.000000	0.000000	0.000000	900.000000	(-Z)

[Iteration: 3] After Entering: x_4 and Leaving: x_1

[Table: 4]

x_1	x_2	x_3	x_4	x_5	x_6	RHS	Basis
1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	4
20.000000	1.000000	0.000000	0.000000	1.000000	0.000000	100.000000	2
-200.000000	0.000000	1.000000	-20.000000	1.000000	0.000000	8000.000000	6
100.000000	0.000000	-1.000000	10.000000	0.000000	0.000000	1000.000000	(-Z)

[Iteration: 4] After Entering: x_3 and Leaving: x_6

[Table: 5]

x_1	x_2	x_3	x_4	x_5	x_6	RHS	Basis
1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	4
20.000000	1.000000	0.000000	0.000000	1.000000	0.000000	100.000000	2
-200.000000	0.000000	1.000000	-20.000000	1.000000	0.000000	8000.000000	3
-100.000000	0.000000	0.000000	-10.000000	1.000000	0.000000	9000.000000	(-Z)

[Iteration: 5] After Entering: x1 and Leaving: x4
[Table: 6]

x1	x2	x3	x4	x5	x6	RHS	Basis
1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	1
0.000000	1.000000	0.000000	-20.000000	1.000000	0.000000	80.000000	2
0.000000	0.000000	1.000000	200.000000	-20.000000	1.000000	8200.000000	3
0.000000	0.000000	0.000000	100.000000	-10.000000	1.000000	9100.000000	(-Z)

[Iteration: 6] After Entering: x5 and Leaving: x2
[Table: 7]

x1	x2	x3	x4	x5	x6	RHS	Basis
1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	1
0.000000	1.000000	0.000000	-20.000000	1.000000	0.000000	80.000000	5
0.000000	20.000000	1.000000	-200.000000	0.000000	1.000000	9800.000000	3
0.000000	10.000000	0.000000	-100.000000	0.000000	1.000000	9900.000000	(-Z)

[Iteration: 7] After Entering: x4 and Leaving: x1
[Table: 8]

x1	x2	x3	x4	x5	x6	RHS	Basis
1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000	4
20.000000	1.000000	0.000000	0.000000	1.000000	0.000000	100.000000	5
200.000000	20.000000	1.000000	0.000000	1.000000	1.000000	10000.000000	3
100.000000	10.000000	0.000000	0.000000	1.000000	1.000000	10000.000000	(-Z)

Optimal Solution:
max Z = -10000.000000

x1* = 0.000000
x2* = 0.000000
x3* = 10000.000000

Thu May 3 23:17:59 2018

Preprocessing Linear Programming Problem:

max:	10	-57	-9	-24			
Subject to:							
	0.5	-5.5	-2.5	9	<=	0	
	0.5	-1.5	-0.5	1	<=	0	
	1	0	0	0	<=	1	

Changing Objective Function to Minimum Form:

min:	-10	57	9	24			
Subject to:							
	0.5	-5.5	-2.5	9	<=	0	
	0.5	-1.5	-0.5	1	<=	0	
	1	0	0	0	<=	1	

Linear Programming Problem in Standard Form:

min:	-10	57	9	24	0	0	0	
Subject to:								
	0.5	-5.5	-2.5	9	1	0	0	= 0
	0.5	-1.5	-0.5	1	0	1	0	= 0
	1	0	0	0	0	0	1	= 1

Starting Simplex Algorithm:

Precision of Calculations: 1e-06

STARTING SIMPLEX ALGORITHM:

(Initial Basic Feasible Solution Available without Solving First Phase)

[Table: 1 (Initial Table)]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
0.500000	-5.500000	-2.500000	9.000000	1.000000	0.000000	0.000000	0.000000	5
0.500000	-1.500000	-0.500000	1.000000	0.000000	1.000000	0.000000	0.000000	6
1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	7
-10.000000	57.000000	24.000000	0.000000	0.000000	0.000000	-0.000000	(-Z)	

[Iteration: 1] After Entering: x1 and Leaving: x5
[Table: 2]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
1.000000	-11.000000	-5.000000	18.000000	2.000000	0.000000	0.000000	0.000000	1
0.000000	4.000000	2.000000	-8.000000	-1.000000	1.000000	0.000000	0.000000	6
0.000000	11.000000	5.000000	-18.000000	-2.000000	0.000000	1.000000	1.000000	7
0.000000	-53.000000	-41.000000	204.000000	20.000000	0.000000	0.000000	0.000000	(-Z)

[Iteration: 2] After Entering: x2 and Leaving: x6
[Table: 3]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
1.000000	0.000000	0.500000	-4.000000	-0.750000	2.750000	0.000000	0.000000	1
0.000000	1.000000	0.500000	-2.000000	-0.250000	0.250000	0.000000	0.000000	2
0.000000	0.000000	-0.500000	4.000000	0.750000	-2.750000	1.000000	1.000000	7
0.000000	0.000000	-14.500000	98.000000	6.750000	13.250000	0.000000	0.000000	(-Z)

[Iteration: 3] After Entering: x3 and Leaving: x1
[Table: 4]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
2.000000	0.000000	1.000000	-8.000000	-1.500000	5.500000	0.000000	0.000000	3
-1.000000	1.000000	0.000000	2.000000	0.500000	-2.500000	0.000000	0.000000	2
1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	7
29.000000	0.000000	0.000000	-18.000000	-15.000000	93.000000	0.000000	0.000000	(-Z)

[Iteration: 4] After Entering: x4 and Leaving: x2
[Table: 5]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
-2.000000	4.000000	1.000000	0.000000	0.500000	-4.500000	0.000000	0.000000	3
-0.500000	0.500000	0.000000	1.000000	0.250000	-1.250000	0.000000	0.000000	4
1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	7
20.000000	9.000000	0.000000	-10.500000	70.500000	0.000000	0.000000	0.000000	(-Z)

[Iteration: 5] After Entering: x5 and Leaving: x3

[Table: 6]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
-4.000000	8.000000	2.000000	0.000000	1.000000	-9.000000	0.000000	0.000000	5
0.500000	-1.500000	-0.500000	1.000000	0.000000	1.000000	0.000000	0.000000	4
1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	7
-22.000000	93.000000	21.000000	0.000000	0.000000	-24.000000	0.000000	0.000000	(-Z)

[Iteration: 6] After Entering: x6 and Leaving: x4

[Table: 7]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
0.500000	-5.500000	-2.500000	9.000000	1.000000	0.000000	0.000000	0.000000	5
0.500000	-1.500000	-0.500000	1.000000	0.000000	1.000000	0.000000	0.000000	6
1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	7
-10.000000	57.000000	9.000000	24.000000	0.000000	0.000000	0.000000	0.000000	(-Z)

Fri May 4 00:21:54 2018

Preprocessing Linear Programming Problem:

max:	10	-57	-9	-24
Subject to:				
0.5	-5.5	-2.5	9	<= 0
0.5	-1.5	-0.5	1	<= 0
1	0	0	0	<= 1

Changing Objective Function to Minimum Form:

min:	-10	57	9	24
Subject to:				
0.5	-5.5	-2.5	9	<= 0
0.5	-1.5	-0.5	1	<= 0
1	0	0	0	<= 1

Linear Programming Problem in Standard Form:					
min:	-10	57	9	24	0
Subject to:					
0.5	-5.5	-2.5	9	1	0
0.5	-1.5	-0.5	1	0	0
1	0	0	0	0	= 1

Starting Simplex Algorithm:

Precision of Calculations: 1e-06

STARTING SIMPLEX ALGORITHM:
(Initial Basic Feasible Solution Available without Solving First Phase.)

[Table: 1 (Initial Table)]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
0.500000	-5.500000	-2.500000	9.000000	1.000000	0.000000	0.000000	0.000000	5
0.500000	-1.500000	-0.500000	1.000000	0.000000	1.000000	0.000000	0.000000	6
1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	7
-10.000000	57.000000	9.000000	24.000000	0.000000	0.000000	-0.000000	(-Z)	

(4.2)-ի բ) դեպք. Բլենդի կանոնի կիրառման դեպքում լուծման ստացումը:

[Iteration: 1] After Entering: x1 and Leaving: x5
[Table: 2]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
1.000000	-11.000000	-5.000000	18.000000	2.000000	0.000000	0.000000	0.000000	1
0.000000	4.000000	2.000000	-8.000000	-1.000000	1.000000	0.000000	0.000000	6
0.000000	11.000000	5.000000	-18.000000	-2.000000	1.000000	1.000000	1.000000	7
0.000000	-53.000000	-41.000000	204.000000	20.000000	0.000000	0.000000	0.000000	(-Z)

[Iteration: 2] After Entering: x2 and Leaving: x6
[Table: 3]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
1.000000	0.000000	0.500000	-4.000000	-0.750000	2.750000	0.000000	0.000000	1
0.000000	1.000000	0.500000	-2.000000	0.250000	0.250000	0.000000	0.000000	2
0.000000	0.000000	-0.500000	4.000000	0.750000	-2.750000	1.000000	1.000000	7
0.000000	0.000000	-14.500000	98.000000	6.750000	13.250000	0.000000	0.000000	(-Z)

[Iteration: 3] After Entering: x3 and Leaving: x1
[Table: 4]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
2.000000	0.000000	1.000000	-8.000000	-1.500000	5.500000	0.000000	0.000000	3
-1.000000	1.000000	0.000000	2.000000	0.500000	-2.500000	0.000000	0.000000	2
1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	7
29.000000	0.000000	0.000000	-18.000000	-15.000000	93.000000	0.000000	0.000000	(-Z)

[Iteration: 4] After Entering: x4 and Leaving: x2
[Table: 5]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
-2.000000	4.000000	1.000000	0.000000	0.500000	-4.500000	0.000000	0.000000	3
-0.500000	0.500000	0.000000	1.000000	0.250000	-1.250000	0.000000	0.000000	4
1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	7
20.000000	9.000000	0.000000	-10.500000	70.500000	0.000000	0.000000	0.000000	(-Z)

[Iteration: 5] After Entering: x5 and Leaving: x3
[Table: 6]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
-4.000000	8.000000	2.000000	0.000000	1.000000	-9.000000	0.000000	0.000000	5
0.500000	-1.500000	-0.500000	1.000000	0.000000	1.000000	0.000000	0.000000	4
1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	7
-22.000000	93.000000	21.000000	0.000000	0.000000	-24.000000	0.000000	0.000000	(-Z)

[Iteration: 6] After Entering: x1 and Leaving: x4
[Table: 7]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
0.000000	-4.000000	-2.000000	8.000000	1.000000	-1.000000	0.000000	0.000000	5
1.000000	-3.000000	-1.000000	2.000000	0.000000	2.000000	0.000000	0.000000	1
0.000000	3.000000	1.000000	-2.000000	0.000000	-2.000000	1.000000	1.000000	7
0.000000	27.000000	-1.000000	44.000000	0.000000	20.000000	0.000000	0.000000	(-Z)

[Iteration: 7] After Entering: x3 and Leaving: x7
[Table: 8]

x1	x2	x3	x4	x5	x6	x7	RHS	Basis
0.000000	2.000000	0.000000	4.000000	1.000000	-5.000000	2.000000	2.000000	5
1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	1
0.000000	3.000000	1.000000	-2.000000	0.000000	-2.000000	1.000000	1.000000	3
0.000000	30.000000	0.000000	42.000000	0.000000	18.000000	1.000000	1.000000	(-Z)

Optimal Solution:
max Z = -1.000000
x1* = 1.000000
x2* = 0.000000
x3* = 1.000000
x4* = 0.000000

Fri May 4 00:30:52 2018

Preprocessing Linear Programming Problem:

max: 2 0 1
Subject to:
1 -1 1 <= 5
-2 1 0 <= 3
0 1 -2 <= 5

Changing Objective Function to Minimum Form:

min: -2 -0 -1
Subject to:
1 -1 1 <= 5
-2 1 0 <= 3
0 1 -2 <= 5

Linear Programming Problem in Standard Form:

min: -2 -0 -1 0 0 0
Subject to:
1 -1 1 1 0 0 0
-2 1 0 0 1 0 0
0 1 -2 0 0 0 1
= = = = = = 5

(4.3) խնդրի դեպքում երկֆազ սխմալեքս եղանակի աշխատանքը:

Starting Simplex Algorithm:
Precision of Calculations: 1e-06

STARTING SIMPLEX ALGORITHM:
(Initial Basic Feasible Solution Available without Solving First Phase)

[Table: 1 (Initial Table)]

x1	x2	x3	x4	x5	x6	RHS	Basis
1.000000	-1.000000	1.000000	1.000000	0.000000	0.000000	5.000000	4
-2.000000	1.000000	0.000000	0.000000	1.000000	0.000000	3.000000	5
0.000000	1.000000	-2.000000	0.000000	0.000000	1.000000	5.000000	6
-2.000000	-0.000000	-1.000000	0.000000	0.000000	-0.000000	(-Z)	

[Iteration: 1] After Entering: x1 and Leaving: x4
[Table: 2]

x1	x2	x3	x4	x5	x6	RHS	Basis
1.000000	-1.000000	1.000000	1.000000	0.000000	0.000000	5.000000	1
0.000000	-1.000000	2.000000	2.000000	1.000000	0.000000	13.000000	5
0.000000	1.000000	-2.000000	0.000000	0.000000	1.000000	5.000000	6
0.000000	-2.000000	1.000000	2.000000	0.000000	0.000000	10.000000	(-Z)

[Iteration: 2] After Entering: x2 and Leaving: x6
[Table: 3]

x1	x2	x3	x4	x5	x6	RHS	Basis
1.000000	0.000000	-1.000000	1.000000	0.000000	1.000000	10.000000	1
0.000000	0.000000	0.000000	2.000000	1.000000	1.000000	18.000000	5
0.000000	1.000000	-2.000000	0.000000	1.000000	5.000000	2.000000	2
0.000000	0.000000	-3.000000	2.000000	0.000000	2.000000	(-Z)	

The Problem is Unbounded

Fri May 4 00:44:23 2018

Preprocessing Linear Programming Problem:

max:	30	20	0	10
Subject to:				
1	2	3	4	\leq
3	2	12	32	\leq
				80 120

Changing Objective Function to Minimum Form:

min:	-30	-20	-0	-10
Subject to:				
1	2	3	4	\leq
3	2	12	32	\leq
				80 120

Linear Programming Problem in Standard Form:

min:	-30	-20	-0	-10	0	0
Subject to:						
1	2	3	4	1	0	=
3	2	12	32	0	1	=
						80 120

Starting Simplex Algorithm:

Precision of Calculations: 1e-06

STARTING SIMPLEX ALGORITHM:
(Initial Basic Feasible Solution Available without Solving First Phase)

[Table: 1 (Initial Table)]

x1	x2	x3	x4	x5	x6	RHS	Basis
1.000000	2.000000	3.000000	4.000000	1.000000	0.000000	80.000000	5
3.000000	2.000000	12.000000	32.000000	0.000000	1.000000	120.000000	6
-30.000000	-20.000000	-0.000000	-10.000000	0.000000	0.000000	-0.000000	(-Z)

(4.4) խնդրի դեպքում երկֆազ սիմպլեքս եղանակի աշխատանքը:

[Iteration: 1] After Entering: x1 and Leaving: x6
 [Table: 2]

x1	x2	x3	x4	x5	x6	RHS	Basis
0.000000	1.333333	-1.000000	-6.666667	1.000000	-0.333333	40.000000	5
1.000000	0.666667	4.000000	10.666667	0.000000	0.333333	40.000000	1
0.000000	0.000000	120.000000	310.000000	0.000000	10.000000	1200.000000	(-Z)

Multiple Optima:
 max

$$Z = -1200.000000$$

$$\begin{aligned}x1^* &= 40.000000 * t + 20.000000 * (1-t) \\x2^* &= 0.000000 * t + 30.000000 * (1-t) \\x4^* &= 0.000000 * t + 0.000000 * (1-t)\end{aligned}$$

Where ($t \geq 0$ and $t \leq 1$)

(4.5) խնդրի դեպքում երկֆազ սխմական եղանակի աշխատանքը:

Fri May 4 00:54:49 2018

Preprocessing Linear Programming Problem:

min: 1 1 1

Auxiliary Problem Before Normalizing:

min: 0 0 0 0 1 1 1

Subject to:

1	2	3	0	1	0	0
-1	2	6	0	0	1	0
0	4	9	0	0	0	1
0	0	3	1	0	0	0

= = = =

3 2 5 1

Starting Simplex Algorithm:
Precision of Calculations: 1e-06

Solving Auxiliary Problem:

[Table: 1 (Initial Table for Normalized Auxiliary Problem)]

x1	x2	x3	x4	a1	a2	a3	RHS	Basis
1.000000	2.000000	3.000000	0.000000	1.000000	0.000000	0.000000	3.000000	5
-1.000000	2.000000	6.000000	0.000000	0.000000	1.000000	0.000000	2.000000	6
0.000000	4.000000	9.000000	0.000000	0.000000	0.000000	1.000000	5.000000	7
0.000000	0.000000	3.000000	1.000000	0.000000	0.000000	0.000000	1.000000	4
0.000000	-8.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-10.000000	(-Z)

STARTING SIMPLEX FIRST PHASE: (Finding Basic Feasible Solution or Manifest Infeasibility)

[Iteration: 1] After Entering: x2 and Leaving: a2
[Table: 2]

x1	x2	x3	x4	a1	a2	a3	RHS	Basis
2.000000	0.000000	-3.000000	0.000000	1.000000	-1.000000	0.000000	1.000000	5
-0.500000	1.000000	3.000000	0.000000	0.000000	0.500000	0.000000	1.000000	2
2.000000	0.000000	-3.000000	0.000000	0.000000	-2.000000	1.000000	1.000000	7
0.000000	3.000000	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	4
-4.000000	0.000000	6.000000	0.000000	4.000000	0.000000	-2.000000	(-Z)	

[Iteration: 2] After Entering: x_1 and Leaving: a_1
 [Table: 3]

	x_1	x_2	x_3	x_4	a_1	a_2	a_3	RHS	Basis
1.	1.000000	0.000000	-1.500000	0.000000	0.500000	-0.500000	0.000000	0.500000	1
0.	0.000000	1.000000	2.250000	0.000000	0.250000	0.250000	0.000000	1.250000	2
0.	0.000000	0.000000	0.000000	0.000000	-1.000000	-1.000000	1.000000	0.000000	7
0.	0.000000	0.000000	3.000000	1.000000	0.000000	0.000000	0.000000	1.000000	4
0.	0.000000	0.000000	0.000000	2.000000	0.000000	0.000000	0.000000	0.000000	(-Z)

Removing Artificial Variables, their Columns and Redundant Constraints by Using Partial Elimination:
 Changing Auxiliary Objective Function to Original Objective Function:
 [Table: 5]

x_1	x_2	x_3	x_4	RHS	Basis
1.000000	0.000000	-1.500000	0.000000	0.500000	1
0.000000	1.000000	2.250000	0.000000	1.250000	2
0.000000	0.000000	3.000000	1.000000	1.000000	4
0.000000	0.000000	0.250000	0.000000	-1.750000	(-Z)

STARTING SIMPLEX SECOND PHASE: (Using the Basic Feasible Solution from First Phase)

Optimal Solution:
 $\min Z = 1.750000$

$x_1^* = 0.500000$
 $x_2^* = 1.250000$
 $x_3^* = 0.000000$

Preprocessing Linear Programming Problem:

max: 1 -1 1
 Subject to:
 2 -1 -2 <= 4
 2 -3 -1 <= -5
 -1 1 1 <= -1

Changing Objective Function to Minimum Form:

min: -1 1 -1
 Subject to:
 2 -1 -2 <= 4
 2 -3 -1 <= -5
 -1 1 1 <= -1

Handling Negative Right Hand Sides:

min: -1 1 -1
 Subject to:
 2 -1 -2 <= 4
 -2 3 1 >= 5
 1 -1 -1 >= 1

min: 0 0 0 0 0 0 0 1 1
 Subject to:
 2 -1 -2 1 0 0 0 0 0 = 4
 -2 3 1 0 -1 0 1 0 1 = 5
 1 -1 0 0 0 -1 0 1 0 1 = 1

Auxiliary Problem Before Normalizing:

(4.6) խնդրի դեպքում երկֆազ սխմալեքս եղանակի աշխատանքը:

Starting Simplex Algorithm:
Precision of Calculations: 1e-06

Solving Auxiliary Problem:

[Table: 1 (Initial Table for Normalized Auxiliary Problem)]

x1	x2	x3	x4	x5	x6	a1	a2	RHS	Basis
2.000000	-1.000000	-2.000000	1.000000	0.000000	0.000000	0.000000	0.000000	4.000000	4
-2.000000	3.000000	1.000000	0.000000	-1.000000	0.000000	1.000000	0.000000	5.000000	7
1.000000	-1.000000	-1.000000	0.000000	0.000000	-1.000000	0.000000	1.000000	1.000000	8
1.000000	-2.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	-6.000000	(-Z)

STARTING SIMPLEX FIRST PHASE: (Finding Basic Feasible Solution or Manifest Infeasibility)

[Iteration: 1] After Entering: x2 and Leaving: a1
[Table: 2]

x1	x2	x3	x4	x5	x6	a1	a2	RHS	Basis
1.333333	0.000000	-1.666667	1.000000	-0.333333	0.000000	0.333333	0.000000	5.666667	4
-0.666667	1.000000	0.333333	0.000000	-0.333333	0.000000	0.333333	0.000000	1.666667	2
0.333333	0.000000	-0.666667	0.000000	-1.000000	0.333333	1.000000	2.666667	8	
-0.333333	0.000000	0.666667	0.000000	0.333333	1.000000	0.666667	0.000000	-2.666667	(-Z)

[Iteration: 2] After Entering: x1 and Leaving: x4
[Table: 3]

x1	x2	x3	x4	x5	x6	a1	a2	RHS	Basis
1.000000	0.000000	-1.250000	0.750000	-0.250000	0.000000	0.250000	0.000000	4.250000	1
0.000000	1.000000	-0.500000	0.500000	-0.500000	0.000000	0.500000	0.000000	4.500000	2
0.000000	0.000000	-0.250000	-0.250000	-1.000000	0.250000	1.000000	1.000000	1.250000	8
0.000000	0.250000	0.250000	1.000000	0.750000	0.000000	-1.250000	(-Z)		

Main Problem is Infeasible: Z = 1.25000000000022204

Հ.2.2. Էլիպտիդների ալգորիթմի աշխատանքը

Անհավասարումների (4.10) համակարգի համատեղելիության ստուգումը կիապ-սոխների մեթոդով:

Mon May 14 17:56:49 2018

Ellipsoid Algorithm for Linear Inequalities with Cholesky Decomposition (with Fixed_Point_Number Type):

Number of variables: $n = 2$
 Number of required iterations: $K = 358$
 Integer part precision: 10
 Fractional part precision: 30
 Perturbation: epsilon = 00000000.000001507040895061728395061728
 Blow up parameter: delta = 00000001.027777777777777777777777

System of inequalities:

000000002.00000000000000000000000000000000
 $-0000000001.00000000000000000000000000000000$
 $0000000000.00000000000000000000000000000000$
 $0000000001.00000000000000000000000000000000$
 $0000000000.00000000000000000000000000000000$
 $0000000000.00000000000000000000000000000000$
 $0000000001.00000000000000000000000000000000$

0000000002.0000001507040895061728395061728
 $000000000.000001507040895061728395061728$
 $000000000.000001507040895061728395061728$
 $00000000016.0000001507040895061728395061728$
 $00000000016.0000001507040895061728395061728$

Center of Initial Ellipsoid:

000000000.0000000000000000
 $000000000.0000000000000000$

Initial Ellipsoid:

0002097152.000000000000000000000000
 $0000000000.0000000000000000$

0002097152.000000000000000000000000
 $0000000000.0000000000000000$

Iteration: 1

Feasible solution:

000000000.0000000000000000
 $000000000.0000000000000000$

Անհավասարումների (4.15) համակարգի համատեղելիության ստուգումը կիապ-սոխների մեթոդով:

Mon May 14 17:09:39 2018

Ellipsoid Algorithm for Linear Inequalities with Cholesky Decomposition (with Fixed_Point_Number Type):

Number of variables: n = 3
Number of required iterations: K = 715
Integer part precision: 10
Fractional part precision: 30
Perturbation: epsilon = 0000000000.00003051757812500000000000000000
Blow up parameter: delta = 0000000001.01562500000000000000000000000000

System of inequalities:

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000
0000000000. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000
0000000000. 00000000000000000000000000000000
0000000001. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000001. 00000000000000000000000000000000
0000000000. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000
0000000000. 00000000000000000000000000000000
0000000001. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000000. 00000000000000000000000000000000
0000000000. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000
0000000000. 00000000000000000000000000000000
0000000001. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

0000000002. 00000000000000000000000000000000
-0000000001. 00000000000000000000000000000000

Iteration: 9
Next ellipsoid's center:
-000000041.990301932538871613558471418098
-0000000098.258079584912220160078308347104
-0000000107.503204403796673242207323597253

Next ellipsoid:
-0000015944.381096855751094438581277000497
-0000018031.83949540694316605820703350095
-0000000347.9097309186534526953994391608

After blowing up:
-0000016193.512051494122205289184109453629
-0000018131.586987522676653025442901839940
-0000000353.345826714257412893764645397726

Iteration: 10
Next ellipsoid's center:
-000000027.423493238723262190864980295632
-0000000019.674242503581498952498828852269
-0000000028.171375801715858539688321659314

Next ellipsoid:
-0000016307.973818230889290988828511331246
-0000010301.392680841505617326811632285083
-00000001001.096856291563945610336146587701

After blowing up:
-0000016562.785909140746936160528956820796
-0000010462.351941140291425975430644839537
-000001016.738994671119632260497648796883

Iteration: 11
Next ellipsoid's center:
-000000007.495876899027315962462007006225
-0000000050.898827292155379306483604293545
-0000000019.646009697004220775957030284200

Next ellipsoid:
-0000015059.145110944285847107842710880836
-0000017370.22904098187813746208318860219
-00000000385.18065006264939472761329984

After blowing up:
-0000015294.4425330279305656402752328349
-0000017641.6388697472203036724282842409
-0000000391.199102740873782911847334510683

Iteration: 12
Next ellipsoid's center:
-0000000011.213420542624477515243549306690
-0000000008.034572905778185283051852589751
-0000000026.350026703051182627928635299263

Next ellipsoid:
-0000014055.90948812387086479682589774207
-0000009923.421864232811267081574088411510
-0000000688.747937315174308233088035640448

After blowing up:
-0000014275.5330975923455927630239428
-0000010078.475330861448943129723683542939
-0000000699.50962385723901721033991116080

Iteration: 13
Next ellipsoid's center:
-0000000018.656645026724998254765152001898
-0000000013.05358633623820713010610863571
-0000000027.813677078042427590253053275140

Next ellipsoid:
-0000008029.98735405513198818959292009740
-0000005669.421864232811267081574088411510
-0000000393.47416340759469718081620002840

After blowing up:
-0000008155.455906462244655050505303947392
-0000005757.722723197214484112195659655352
-0000000399.622197210838361823051645315384

Iteration: 14
Next ellipsoid's center:
-0000000005.752173205853940069544232484792
-0000000038.35671695807082567832751816545454
-0000000021.445743388383178547672138223808

Next ellipsoid:
-0000007676.159357988128400893243506163596
-0000009416.14985300666471205977834229905
-0000000289.998415836256580332395344

After blowing up:
-0000007796.099347957853210236685947402
-0000009563.27719445983848185712384920997
-0000000294.529641083697552680559283162248

Iteration: 15
Next ellipsoid's center:
-0000000007.838152788330179440164762076637
-0000000005.62345136236434856580332395344
-0000000026.510717108197926187567056471869

Next ellipsoid:
-0000007108.3391207985291143243856797668
-0000005379.343421883690289604463216518076
-0000002288.165949619160436971328440046664

After blowing up:
-0000007219.406919561131863173565635810131
-0000005463.395662850622950379532954276170
-0000000292.668542785084818779900544692393

Iteration: 16
Next ellipsoid's center:
-0000000013.4043620500935732067635953440452
-00000000010.451582688616450312088182942849
-0000000027.371840314578778292551921835018

Next ellipsoid:
-0000004060.916392253136673035130670143219
-0000003073.160060353347540958487286708365
-000000164.626055316610210574440563893860

After blowing up:
-0000004124.368210882091933551304586864206
-0000003121.178186296498462863307400636308
-0000000167.19833743093224511466617704701

Iteration: 17
Next ellipsoid's center:
-0000000004.73188269633718925861860553010
-00000000030.05406641974752525917424012381
-0000000022.657803391957715288774160508869

Next ellipsoid:
-0000003963.12298027702143021171076878700
-0000005041.994309103469820157734906393046
-0000000179.811900137741398356093971585689

After blowing up:
-0000004025.04678037188498865251874954929
-00000005120.775470183211536097699514385437
-0000000182.621461077339607705407939891715

-00000000347.809733091865345269539904391608
-0000013205.826917905488610327664171171437
-0000154599.94968126113795772956333516146

-00000000353.345822671425741289376465397726
-0000013412.167963497761869864033923845990
-0000157015.57389856209323838307213727335

-00000000347.909733091865345269539904391608
-0000013205.826917905488610327664171171437
-0000154599.94968126113795772956333516146

-00000000353.345822671425741289376465397726
-0000013412.167963497761869864033923845990
-0000157015.57389856209323838307213727335

-000000000101.096856291563945610336146507701
-0000007544.344794674918051798519082163429
-000175618.231707547306631445156546762712

-0000001016.738994671119632260497648796883
-0000007662.224861959170599482878942822232
-000178362.26657777733297561487117805879

-00000000385.1806550062649394397276813229904
-0000006224.194163869572927381629962889387
-0000200803.13095250655205961775123530371

-00000000391.19910274073782911847334510683
-0000006321.44719768003504371967931064205
-0000203128.46642486394669355492789835533

-0000000688.747937315174303233018083560448
-000000355.81404869501968959231961223775
-0000228115.03207839187475421305281577516

-000000699.5096238572390172183399116080
-0000003611.373643205879372614844960617896
-0000231679.32755483674779724763176602164

-0000000393.474163407594694718081620002840
-0000004340.586701735967394086587338866113
-0000260619.963038811552194719826096705475

18-280 խոերացիաները բաց են թողնված:

```

Iteration: 281
Next ellipsoid's center:
  000000000.000073960470836398633504526867
  000000001.999897071878242375034575591139
  000000001.000002725579288636708243176321
Next ellipsoid:
  000000000.0000025413949961756546478554      -000000000.00000398512800369655933245489
  -000000000.00000398512800369655933245489      000000000.0000056404250128928921263080
  -000000000.0000035755779191712721955377      000000000.00000140060212347485773813614
After blowing up:
  000000000.00000258110429648658992517281      -000000000.00000404739562875431807202449
  -000000000.00000404739562875431807202449      000000000.00000979105665371934356204690
  -000000000.0000036314463241583233235929      000000000.00000142248653165415239029451
  -000000000.0000035755779191712721955377
  000000000.00000140060212347485773813614
  000000000.0000074903183458195122066503

Iteration: 282
Next ellipsoid's center:
  000000000.000029479640836257002914216371
  000000001.999829391204312610210585929264
  000000000.999974947437783265098589037676
Next ellipsoid:
  000000000.00000272567335217227869350154      -000000000.00000482426441197144763669742
  -000000000.00000482426441197144763669742      000000000.00001060267810931141819601870
  -000000000.0000051974124256963953404300      000000000.00000143109344772692841372248
After blowing up:
  000000000.00000276826199829997054808750      -000000000.0000048996435430850150602081
  -000000000.0000048996435430850150602081      000000000.00001076834495476940910533149
  -000000000.000005278621994879015176242      000000000.00000145345428284766167018689
  -000000000.0000051974124256963953404300
  000000000.00000143109344772692841372248
  000000000.00000078638112602763558209634

Iteration: 283
Next ellipsoid's center:
  -000000000.000017839607358267948601379520
  000000001.999959683473803768804592865918
  000000000.999945396712846901975975340722
Next ellipsoid:
  000000000.00000291277473561491210706105      -000000000.00000495721908492539243908280
  -000000000.00000495721908492539243908280      000000000.00001058654128009167017529224
  -000000000.0000071969360230366029445849      000000000.00000198165685973969329527872
After blowing up:
  000000000.00000295828684085889510873387      -000000000.00000503467563312735169594346
  -000000000.00000503467563312735169594346      000000000.00001075195598759310252178118
  -000000000.0000073093881483965498655940      000000000.0000021262024817312600301745
  -000000000.0000071969360230366029445849
  000000000.00000198165685973969329527872
  000000000.00000081990979152823116664069

Iteration: 284
Next ellipsoid's center:
  -000000000.000081164039740697860618948829
  000000002.000134045584871711849553941769
  000000001.000017538971701406757945105777
Next ellipsoid:
  000000000.00000296717415967613257166481      -000000000.00000467028573315405264705712
  -000000000.00000467028573315405264705712      000000000.000003595736619571567065583
  -000000000.0000041115308334730592993967      000000000.0000011320988859738337669732
After blowing up:
  000000000.00000301353625592107214309707      -000000000.00000474325894773458471966738
  -000000000.00000474325894773458471966738      000000000.00000950600357504252372800982
  -000000000.0000041757735027460758509497      000000000.00000114978793474734249195821
  -000000000.0000041757735027460758509497
  000000000.00000114978793474734249195821
  000000000.00000047572433201387008837146

Iteration: 285
Next ellipsoid's center:
  000000000.000056075173312666376800238656
  000000001.999918033209372197590416633064
  000000000.999998522115067279328531376385
Next ellipsoid:
  000000000.00000169511414395560308049211      -000000000.00000266808315810070390481291
  -000000000.00000266808315810070390481291      000000000.00000649473284871795585828395
  -000000000.0000023488725952946676661593      000000000.0000092380255274431962148109
After blowing up:
  000000000.00000172160030245490937862479      -000000000.00000270977195744602740332561
  -000000000.00000270977195744602740332561      000000000.00000659621304947917391856963
  -000000000.0000023855737295961468484430      000000000.0000093823696763094961556673
  -000000000.0000023855737295961468484430
  000000000.00000649473284871795585828395
  000000000.0000092380255274431962148109

Iteration: 286
Feasible solution:

  000000000.000056075173312666376800238656
  000000001.999918033209372197590416633064
  000000000.999998522115067279328531376385

```

Անհավասարումների (4.17) համակարգի համատեղելիության ստուգումը կիապ-ստիդների մեթոդով(տվյալների լայնածավալության պատճառով անհավասարումների համակարգի b վեկտորը ներկայացված է ներքոհիշյալ ձևով)։

Անհավասարումների (4.20) համակարգի համատեղելիության ստուգումը կիապ-սոխների մեթոդով:

Wed May 16 05:05:48 2018

Ellipsoid Algorithm for Linear Inequalities with Cholesky Decomposition (with Fixed_Point_Number Type):

Number of variables: n = 2

Number of required iterations: K = 225

Integer part precision: 10

Fractional part precision: 30

Perturbation: epsilon = 0000000000.000096450617283950617

Blow up parameter: delta = 0000000001.02777777777777777777777777

System of inequalities:

$$\begin{aligned} & 0000000001.00000000000000000000000000000000 \\ & - 0000000001.00000000000000000000000000000000 \\ & 0000000000.00000000000000000000000000000000 \\ & 0000000001.00000000000000000000000000000000 \\ & 0000000000.00000000000000000000000000000000 \end{aligned}$$

$$\begin{aligned} & - 0000000001.00000000000000000000000000000000 \\ & 0000000000.00000000000000000000000000000000 \\ & - 0000000001.00000000000000000000000000000000 \\ & 0000000000.00000000000000000000000000000000 \\ & 0000000001.00000000000000000000000000000000 \end{aligned}$$

$$\begin{aligned} & 0000000001.000096450617283950617283950617 \\ & 0000000000.000096450617283950617283950617 \\ & 0000000000.000096450617283950617283950617 \\ & 0000000004.000096450617283950617283950617 \\ & 0000000004.000096450617283950617283950617 \end{aligned}$$

Center of Initial Ellipsoid:

$$0000000000.00000000000000000000000000000000$$

$$0000000000.00000000000000000000000000000000$$

Initial Ellipsoid:

$$\begin{aligned} & 0000000000.00000000000000000000000000000000 \\ & 0000000000.00000000000000000000000000000000 \end{aligned}$$

Iteration: 1

Feasible solution:

$$\begin{aligned} & 0000000000.00000000000000000000000000000000 \\ & 0000000000.00000000000000000000000000000000 \end{aligned}$$

ՀԱՎԵԼՎԱԾ 3

**ԵՐԿԻՆ ՍԻՄՊԼԵՔՍ ՄԵԹՈԴԻ ԾՐԱԳՐԱՅԻՆ
ԻՐԱԿԱՆԱՑՈՒՄԸ**

```

/*
Thu Feb 22 2018
H.S. Papoyan
Class System is Implementation of Linear Programming Problem Preprocessing
Class Simplex is Implementation of Two Phase Simplex Algorithm With Bland's Pivoting Rule

1.Input Example:

max:
30,20,0,10,=,0;
1,2,3,4,<=,80;
3,2,12,32,<=,120;
[NRHS]:0;
[BASIS]:-;

2.Every Entry Number and Constraint Sign Must be Separated With Comma ','

3.Every Row (Except First) Must have ';' at the End

4.Second Row is Objective Function
4.1 In Case of LP Problem Like This max: 2x1 + 3x2 + 7
The Input File Should be
max:
2,3,-,-7
4.2 In Case of LP Problem Like This max: 2x1 + 3x2
The Input File Should be
max:
2,3,=,0

5.In Constraints Following Signs Are Allowed
5.1. less or equal: "<=" ( no comma between < and = )
5.2. greater or equal: ">=" ( no comma between > and = )
5.3. less: '<'
5.4. greater: '>'
5.5. equal: '='

6.In Case of Minimization Problem Change "max" to "min" (case sensitive)

7.If There is a Negative RHS In Problem Then Set [NRHS] Parameter 1, Otherwise 0
(it is important to write exactly 0 or 1 and don't add any symbols)
example [NRHS]:0;
7.1 The Answer Will not be Correct if [NRHS]:0 and System Consist at least 1 Negative RHS
example [NRHS]:1;
7.2 It is Allowed to Have [NRHS]:1; even All RHSs are Positive (programm will perform
additional operations for handling RHSs)

8.If Initial Basis For LP Problem Unknown Set '-', Otherwise The Initial Basis Should be Mentioned
example [BASIS]:-; ( it is important to write exactly '-' and don't add any symbols)
example
max:
4,1,=,0;
3,1,1,0,0,=,10;
1,1,0,0,1,=,5;
1,0,0,1,0,=,3;
[NRHS]:0;
[BASIS]:2,4,3;

(
8.1 It is Important to Write ',' After each Index in "[BASIS]:2,5,3;" row and Don't Add any Symbols
8.2 The Basis Must be Located in Last Columns of LP Problem
For Example in This Row ( [BASIS]:2,4,3; )
2 Indicates that in the First Row the Element of Basis Vector is Located at the Index = 2 (Indexing Started from 0)
4 Indicates that in the Second Row the Element of Basis Vector is Located at the Index = 4 (Indexing Started from 0)
3 Indicates that in the Third Row the Element of Basis Vector is Located at the Index = 3 (Indexing Started from 0)
)

9.Adding Symbols (Except for those Described above) is Strictly Prohibited

10.Before Running This Code The Following Changes Must be Done:
1. Setting in Preprocessor Definitions: _CRT_SECURE_NO_WARNINGS
2. Setting Path for input file
3. Setting Path for output file
*/

```

```

#include "stdafx.h"
#include <iostream>
#include <string>
#include <fstream>

```

```

#include <vector>
#include <algorithm>
#include <iomanip>
#include <chrono>
#include <atlstr.h>

using namespace std;

string Path_for_input = "C:\\\\Users\\\\HP\\\\Desktop\\\\Two_Phase_Simplex\\\\Input\\\\example.txt";
string Path_for_output = "C:\\\\Users\\\\HP\\\\Desktop\\\\Two_Phase_Simplex\\\\Output\\\\";

template<typename T1, typename T2>
struct compare_by_key
{
    bool operator() (const pair<T1, T2> & a, const pair<T1, T2> & b) const
    {
        return a.first < b.first;
    }
};

class LP_System
{
private:

    vector<string> initial_problem;
    bool type_of_problem; // 0 -> min
    bool negative_rhs;
    unsigned int number_of_rows_in_initial_problem;
    vector<unsigned int> final_vars_indices;
    vector<vector<double>> mat; // Constraints(including RHS)
    vector<double> objective_function;
    vector<double> auxilliary_objective_function;
    vector<unsigned int> signs; // [0:=[1:<= ][2:<][3:>= ][4:>]
    vector<unsigned int> basis; // init. in to_equal_form() function
    vector<unsigned int> artificial_indices; // init. in to_equal_form() function
    fstream output_file;
    string full_name_of_output_file;

public:

    LP_System()
    {
        make_output_file();
        reading_from_file();
        number_of_rows_in_initial_problem = initial_problem.size();
        initializing_mat_and_objective_function();
        initializing_signs();
        init_final_vars_indices();

        if (initial_problem[0].find("max") == -1)
        {
            type_of_problem = 0; print_objective_function(objective_function); print_constraints();
            objective_function_to_output_file(objective_function);
        } // 0 -> min
        else
        {
            type_of_problem = 1;
            print_objective_function(objective_function, "max");
            print_constraints();
            objective_function_to_output_file(objective_function, "max");
            constraints_to_output_file();

            cout << " Changing Objective Function to Minimum Form:\n";
            output_file << " Changing Objective Function to Minimum Form:\n";

            changing_objective_function_to_minimum_form();
            print_objective_function(objective_function);
            objective_function_to_output_file(objective_function);
            print_constraints();
            constraints_to_output_file();
        }
        // 0 -> undefined

        if (initial_problem[number_of_rows_in_initial_problem - 1].find('-') != -1)
        {
            negative_rhs = stoi(
                initial_problem[number_of_rows_in_initial_problem - 2],
                substr(initial_problem[number_of_rows_in_initial_problem - 2].find(':') + 1, 1));
        }
    }
};

```

```

// 1 -> negative rhs exist
    transforming_to_standart_form();
}
else
{
    unsigned int index_of_colon = initial_problem[number_of_rows_in_initial_problem - 1].find(':');
    initial_problem[number_of_rows_in_initial_problem - 1].erase(0, index_of_colon + 1);
    int comma_index;

    while ((comma_index = initial_problem[number_of_rows_in_initial_problem - 1].find_first_of(',')) != -1)
    {
        basis.push_back(stoi(initial_problem[number_of_rows_in_initial_problem - 1].substr(0, comma_index)));
        initial_problem[number_of_rows_in_initial_problem - 1].erase(0, comma_index + 1);
    }
    comma_index = initial_problem[number_of_rows_in_initial_problem - 1].find(';');
    basis.push_back(stoi(initial_problem[number_of_rows_in_initial_problem - 1].substr(0, comma_index + 1)));
    for (unsigned int i = 0; i < basis.size(); i++)
    {
        objective_function.insert(objective_function.end() - 1, 0.0);
    }

    output_file.close();
}
~LP_System()
{
}

bool get_type_of_problem() const
{
    return type_of_problem;
}
vector<vector<double>> get_mat() const
{
    return mat;
}
vector<unsigned int> get_basis() const
{
    return basis;
}
vector<unsigned int> get_artificial_indices() const
{
    return artificial_indices;
}
vector<unsigned int> get_final_vars_indices() const
{
    return final_vars_indices;
}
vector<double> get_objective_function() const
{
    return objective_function;
}
vector<double> get_auxiliary_objective_function() const
{
    return auxiliary_objective_function;
}
string get_output_file_full_name() const
{
    return full_name_of_output_file;
}

private:
    void make_output_file()
    {
        chrono::system_clock::time_point today = chrono::system_clock::now();
        std::time_t time_date = chrono::system_clock::to_time_t(today);
        string solution_starting_time = ctime(&time_date);

        vector<string> list_of_files_in_directory;
        CString directory = Path_for_output.c_str();
        HANDLE dir;
        WIN32_FIND_DATA file_data;
        CString file_name;
        if ((dir = FindFirstFile((directory + "/*"), &file_data)) == INVALID_HANDLE_VALUE)
        {
            cout << " Invalid directory\n";

```

```

    }
    while (FindNextFile(dir, &file_data))
    {
        file_name = file_data.cFileName;

        if (strcmp(file_data.cFileName, ".") != 0 && strcmp(file_data.cFileName, "..") != 0)
        {
            list_of_files_in_directory.push_back(file_name.GetString());
        }
    }

    string name_of_output_file; unsigned int h = 0;
    string solution = "solution"; string dot_txt = ".txt";
    vector<string>::iterator it;
    name_of_output_file = solution + dot_txt;
    while ((it = find(list_of_files_in_directory.begin(), list_of_files_in_directory.end(), name_of_output_file)) != list_of_files_in_directory.end())
    {
        name_of_output_file = solution + to_string(h) + dot_txt;
        h++;
    }

    full_name_of_output_file = Path_for_output + name_of_output_file;

    output_file.open(full_name_of_output_file, ios::out);
    if (!output_file) { cout << "Output File doesn't open\n"; exit(1); }
    output_file << "\n" << solution_starting_time << "\n Preprocessing Linear Programming Problem:\n";
}

void init_final_vars_indices()
{
    for (unsigned int i = 0; i < objective_function.size(); i++)
    {
        if (objective_function[i] != 0.0) final_vars_indices.push_back(i);
    }
}

void reading_from_file()
{
    fstream file; string word;
    file.open(Path_for_input, ios::in);
    if (!file)
    {
        cout << " Input file doesn't open\n";
        output_file << " Input file doesn't open\n";
        exit(1);
    }
    while (file >> word) initial_problem.push_back(word);
    file.close();
}

//helper function for initializing mat and objective function
void from_initial_string_to_vec(unsigned int number_of_row, vector<double> & vec)
{
    unsigned int number_of_commas = std::count(initial_problem[number_of_row].begin(),
                                                initial_problem[number_of_row].end(), ',');
    int comma_index; int semicolon_index;

    for (unsigned int j = 0; j < number_of_commas - 1; j++)
    {
        comma_index = initial_problem[number_of_row].find_first_of(',');
        vec.push_back(stod(initial_problem[number_of_row].substr(0, comma_index)));
        initial_problem[number_of_row].erase(0, comma_index + 1);
    }
    comma_index = initial_problem[number_of_row].find(',');
    semicolon_index = initial_problem[number_of_row].find('\'');
    vec.push_back(stod(initial_problem[number_of_row].substr(comma_index + 1, semicolon_index - comma_index - 1)));
}

void initializing_mat_and_objective_function()
{
    vector<double> current_vec;
    for (unsigned int i = 2; i < number_of_rows_in_initial_problem - 2; i++)
    {
        from_initial_string_to_vec(i, current_vec);
        mat.push_back(current_vec);
        current_vec.clear();
    }
    from_initial_string_to_vec(1, objective_function);
}

```

```

void changing_objective_function_to_minimum_form()
{
    for (unsigned int i = 0; i < objective_function.size(); i++)
    {
        objective_function[i] *= -1.0;
    }
}
void initializing_signs()
{
    //[[0:=[1:<= ][2:<][3:>= ][4:>]
    for (unsigned int i = 2; i < number_of_rows_in_initial_problem - 2; i++)
    {
        if (initial_problem[i].find(">=") != -1) { signs.push_back(3); continue; }
        if (initial_problem[i].find("<=") != -1) { signs.push_back(1); continue; }
        if (initial_problem[i].find(">") != -1) { signs.push_back(4); continue; }
        if (initial_problem[i].find("<") != -1) { signs.push_back(2); continue; }
        signs.push_back(0);
    }
}
void transforming_to_standart_form()
{
    if (negative_rhs == true)
    {
        cout << " Handling Negative Right Hand Sides:\n";
        output_file << " Handling Negative Right Hand Sides:\n";
        handling_negative_rhs();
        print_objective_function(objective_function);
        objective_function_to_output_file(objective_function);
        print_constraints();
        constraints_to_output_file();
    }

    to_equal_form();
    adding_surplus_and_slack_zeros_in_objective();

    if (artificial_indices.size() != 0)
    {
        create_auxiliary_objective_function();
        cout << " Auxiliary Problem Before Normalizing:\n";
        output_file << " Auxiliary Problem Before Normalizing:\n";
        print_objective_function(auxiliary_objective_function);
        objective_function_to_output_file(auxiliary_objective_function);
    }
    else
    {
        cout << " Linear Programming Problem in Standart Form:\n";
        output_file << " Linear Programming Problem in Standart Form:\n";
        print_objective_function(objective_function);
        objective_function_to_output_file(objective_function);
    }
    print_constraints();
    constraints_to_output_file();
}
void create_auxiliary_objective_function()
{
    for (unsigned int i = 0; i < mat[0].size(); i++)
    {
        auxiliary_objective_function.push_back(0.0);
    }
    for (unsigned int i = 0; i < artificial_indices.size(); i++)
    {
        auxiliary_objective_function[artificial_indices[i]] = 1.0;
    }
}
void adding_surplus_and_slack_zeros_in_objective()
{
    unsigned int sz = objective_function.size();
    for (unsigned int i = 0; i < mat[0].size() - sz - artificial_indices.size(); i++)
    {
        objective_function.insert(objective_function.end() - 1, 0.0);
    }
}
void handling_negative_rhs()
{

```

```

unsigned int sz = mat[0].size();
for (unsigned int i = 0; i < mat.size(); i++)
{
    if (mat[i][sz - 1] < 0.0)
    {
        for (unsigned int j = 0; j < sz; j++)
            mat[i][j] *= -1.0;
        switch (signs[i])
        {
            case 1: signs[i] = 3; break;
            case 2: signs[i] = 4; break;
            case 3: signs[i] = 1; break;
            case 4: signs[i] = 2; break;
            default: break;
        }
    }
}
void to_equal_form() // [0:=[1:<= ][2:<][3:>= ][4:>]
{
    unsigned int current_basis_number = 0; unsigned int sz = mat[0].size();
    basis.resize(mat.size());
    vector<unsigned int> vec_for_art_indices;

    for (unsigned int i = 0; i < mat.size(); i++)
    {
        switch (signs[i])
        {
            case 0: signs[i] = 0; vec_for_art_indices.push_back(i); break;
            case 1: adding_var(i); signs[i] = 0; basis[i] = (sz - 1 + current_basis_number++); break;
            case 2: adding_var(i); signs[i] = 0; basis[i] = (sz - 1 + current_basis_number++); break;
            case 3: adding_var(i, -1); vec_for_art_indices.push_back(i); current_basis_number++; signs[i] = 0; break;
            case 4: adding_var(i, -1); vec_for_art_indices.push_back(i); current_basis_number++; signs[i] = 0; break;
            default: break;
        }
    }

    unsigned int index; sz = mat[0].size();
    for (unsigned int i = 0; i < vec_for_art_indices.size(); i++)
    {
        index = vec_for_art_indices[i];
        adding_var(index);
        basis[index] = sz - 1 + i;
        artificial_indices.push_back(sz - 1 + i);
    }
}
void adding_var(unsigned int number_of_row, double value = 1.0)
{
    for (unsigned int i = 0; i < mat.size(); i++)
    {
        mat[i].insert(mat[i].end() - 1, 0.0);
    }
    unsigned int sz = mat[number_of_row].size();
    mat[number_of_row][sz - 2] = value;
}
void print_objective_function(const vector<double> & vec, const string & str = "min") const
{
    cout << "\n" << str << ": ";
    for (unsigned int i = 0; i < vec.size() - 1; i++)
    {
        cout << '\t' << vec[i];
    }
    cout << '\n';
}
void print_constraints() const
{
    cout << " Subject to:\n";
    for (unsigned int i = 0; i < mat.size(); i++)
    {
        for (unsigned int j = 0; j < mat[i].size() - 1; j++)
        {
            cout << '\t' << mat[i][j];
        }
        cout << '\t';
        switch (signs[i])
        {

```

```

        case 0: cout << " ="; break;
        case 1: cout << "<="; break;
        case 2: cout << "=<"; break;
        case 3: cout << ">="; break;
        case 4: cout << ">"; break;
    }
    cout << '\t' << mat[i][mat[i].size() - 1];
    cout << '\n';
}
cout << '\n';
}
void objective_function_to_output_file(const vector<double> & vec, const string & str = "min")
{
    output_file << "\n " << str << ":" ;
    for (unsigned int i = 0; i < vec.size() - 1; i++)
    {
        output_file << '\t' << vec[i];
    }
    output_file << '\n';
}
void constraints_to_output_file()
{
    output_file << " Subject to:\n";
    for (unsigned int i = 0; i < mat.size(); i++)
    {
        for (unsigned int j = 0; j < mat[i].size() - 1; j++)
        {
            output_file << '\t' << mat[i][j];
        }
        output_file << '\t';
        switch (signs[i])
        {
            case 0: output_file << " ="; break;
            case 1: output_file << "<="; break;
            case 2: output_file << "=<"; break;
            case 3: output_file << ">="; break;
            case 4: output_file << ">"; break;
        }
        output_file << '\t' << mat[i][mat[i].size() - 1];
        output_file << '\n';
    }
    output_file << '\n';
}
};

class Simplex
{
private:

    double precision;
    bool type_of_problem; // 0 -> min
    bool is_artificial; // 1 -> artificial
    vector<vector<double>> mat; // Constraints(including RHS)
    vector<unsigned int> basis; // first 0 number for objective function
    vector<unsigned int> artificial_indices;
    vector<unsigned int> final_vars_indices;
    vector<double> objective_function;
    vector<double> auxiliary_objective_function;
    unsigned int table_number;
    unsigned int number_of_current_mat_rows;
    unsigned int number_of_current_mat_cols;
    int entering_variable_index;
    int leaving_variable_index;
    double Z_optimal_value;
    vector<double> first_optimal_solution;
    vector<double> second_optimal_solution; // multiple optima
    fstream output_file;
    string full_name_of_output_file;

public:

    Simplex(const LP_System & object)
    {
        full_name_of_output_file = object.get_output_file_full_name();
        open_output_file();
        precision = 0.000001;
    }
};

```

```

cout << fixed << setprecision(1) << scientific;
cout << " Precision of Calculations: " << precision << '\n';
output_file << " Precision of Calculations: " << precision << '\n';
type_of_problem = object.get_type_of_problem();
mat = object.get_mat();
basis = object.get_basis();
artificial_indices = object.get_artificial_indices();
is_artificial = (artificial_indices.size() == 0) ? false : true;

final_vars_indices = object.get_final_vars_indices();

number_of_current_mat_rows = mat.size() + 1; // including objective_function row
number_of_current_mat_cols = mat[0].size();

objective_function = object.get_objective_function();

if (is_artificial)
{
    auxiliary_objective_function = object.get_auxiliary_objective_function();
    mat.push_back(auxiliary_objective_function);
    normalizing_auxiliary_problem();
}
else mat.push_back(objective_function);

main_algorithm();
}

~Simplex()
{
}

private:

void open_output_file()
{
    output_file.open(full_name_of_output_file, ios::app);
    if (!output_file) { cout << "Simplex class: Ouput File doesn't open\n"; exit(1); }
    output_file << "\n Starting Simplex Algorithm:\n";
}

void checking_precision_of_problem()
{
    for (unsigned int i = 0; i < number_of_current_mat_rows; i++)
    {
        for (unsigned int j = 0; j < number_of_current_mat_cols; j++)
        {
            if (abs(mat[i][j]) < precision) mat[i][j] = 0.0;
        }
    }
}

void normalizing_auxiliary_problem() // extending mat to mat.push_back(auxiliary_objective_function)
{
    unsigned int row_index;
    for (unsigned int i = 0; i < artificial_indices.size(); i++)
    {
        row_index = get_row_number(artificial_indices[i]);
        for (unsigned int j = 0; j < number_of_current_mat_cols; j++)
            mat[number_of_current_mat_rows - 1][j] += (-mat[row_index][j]);
    }
}

int get_row_number(unsigned int number_of_col) // getting row number where 1 is located in basis column
{
    for (unsigned int i = 0; i < number_of_current_mat_rows - 1; i++)
    {
        if (number_of_col == basis[i]) return i;
    }
    return -1;
}

bool is_col_artificial(unsigned int col_number)
{
    for (unsigned int i = 0; i < artificial_indices.size(); i++)
    {
        if (col_number == artificial_indices[i]) return true;
    }
    return false;
}

void table_to_output_file()
{
}

```

```

        output_file << "\n\t"; unsigned int artificial_current_number = 1;
        for (unsigned int j = 0; j < number_of_current_mat_cols - 1; j++)
        {
            if (is_col_artificial(j) == true)
                output_file << 'a' << artificial_current_number++ << "\t\t";
            else
                output_file << 'x' << j + 1 << "\t\t";
        }
        output_file << "RHS\t\t" << "Basis\n\n";
        output_file << fixed << setprecision(6);

        for (unsigned int i = 0; i < number_of_current_mat_rows; i++)
        {
            for (unsigned int j = 0; j < number_of_current_mat_cols; j++)
            {
                output_file << '\t' << mat[i][j];
            }
            output_file << "\t\t";
            if (i == number_of_current_mat_rows - 1) output_file << "(-Z)";
            else output_file << basis[i] + 1;
            output_file << '\n';
        }
        output_file << '\n';
    }
    void print_table()
    {
        cout << "\n\t"; unsigned int artificial_current_number = 1;
        for (unsigned int j = 0; j < number_of_current_mat_cols - 1; j++)
        {
            if (is_col_artificial(j) == true)
                cout << 'a' << artificial_current_number++ << '\t';
            else
                cout << 'x' << j + 1 << '\t';
        }
        cout << "RHS\t" << "Basis\n\n";
        cout << fixed << setprecision(2);

        for (unsigned int i = 0; i < number_of_current_mat_rows; i++)
        {
            for (unsigned int j = 0; j < number_of_current_mat_cols; j++)
            {
                cout << '\t' << mat[i][j];
            }
            cout << '\t';
            if (i == number_of_current_mat_rows - 1) cout << "(-Z)";
            else cout << basis[i] + 1;
            cout << '\n';
        }
        cout << endl;
    }

    int get_entering_variable_index_by_Bland()
    {
        int index = -1;
        for (unsigned int j = 0; j < number_of_current_mat_cols - 1; j++)
            if (mat[number_of_current_mat_rows - 1][j] < 0.0)
            {
                index = j; return index;
            }
        return index;
    }

/*
//most-negative-entry pivoting rule
int get_entering_variable_index_by_Bland()
{
    int index = -1;
    vector<double> candidates;
    vector<int> indices;

    for (unsigned int j = 0; j < number_of_current_mat_cols - 1; j++)
        if (mat[number_of_current_mat_rows - 1][j] < 0.0)

```

```

    {
        candidates.push_back(mat[number_of_current_mat_rows - 1][j]);
        indices.push_back(j);
    }
    if (candidates.size() != 0)
    {
        index = min_element(candidates.begin(), candidates.end()) - candidates.begin();
        return indices[index];
    }
    return index;
}
*/
int get_leaving_variable_index_by_Bland()
{
    vector<pair<double, unsigned int>> leaving_candidates_ratio;
    for (unsigned int i = 0; i < number_of_current_mat_rows - 1; i++)
    {
        if (mat[i][entering_variable_index] > 0.0)
            leaving_candidates_ratio.push_back(make_pair((mat[i][number_of_current_mat_cols - 1] /
                mat[i][entering_variable_index]), i));
    }
    if (leaving_candidates_ratio.size() == 0) return -1; // unboundedness
    sort(leaving_candidates_ratio.begin(), leaving_candidates_ratio.end(),
        compare_by_key<double, unsigned int>());
    return leaving_candidates_ratio[0].second;
}

void dividing_by_pivot(unsigned int number_of_cols)
{
    double pivot;
    if ((pivot = mat[leaving_variable_index][entering_variable_index]) == 1.0) return;
    for (unsigned int j = 0; j < number_of_cols; j++)
    {
        mat[leaving_variable_index][j] /= pivot;
    }
}
void standart_elimination(unsigned int number_of_rows, unsigned int number_of_cols)
{
    double fix;
    for (unsigned int i = 0; i < number_of_current_mat_rows; i++)
    {
        if (i == leaving_variable_index) continue;
        fix = -mat[i][entering_variable_index];
        if (fix == 0.0) continue;
        for (unsigned int j = 0; j < number_of_current_mat_cols; j++)
        {
            mat[i][j] += (fix * mat[leaving_variable_index][j]);
        }
    }
}
void start_simplex_algorithm()
{
    unsigned int iteration_number = 1;
    while (true)
    {
        entering_variable_index = get_entering_variable_index_by_Bland();
        if (entering_variable_index == -1)
        { Z_optimal_value = -mat[number_of_current_mat_rows - 1][number_of_current_mat_cols - 1]; return; }
        leaving_variable_index = get_leaving_variable_index_by_Bland();
        if (leaving_variable_index == -1) { cout << "\n The Problem is Unbounded\n\n";
            output_file << "\n The Problem is Unbounded\n\n";
            output_file.close(); exit(1); }
        cout << "\n [Iteration: "
            << iteration_number << "] After Entering: x" << entering_variable_index + 1 << " and Leaving: ";
        output_file << "\n [Iteration: "
            << iteration_number << "] After Entering: x" << entering_variable_index + 1 << " and Leaving: ";
        iteration_number++;

        if (basis[leaving_variable_index] >= number_of_current_mat_cols - 1 - artificial_indices.size())
        {
            cout << 'a' << basis[leaving_variable_index]
                - (number_of_current_mat_cols - 1 - artificial_indices.size()) + 1 << '\n';
        }
    }
}

```

```

        else
        {
            print_multiple_optima();
            multiple_optima_to_output_file();
        }
    }
else
{
    cout << "\n STARTING SIMPLEX ALGORITHM:\n" <<
        " (Initial Basic Feasible Solution Available without Solving First Phase)\n";
    output_file << "\n STARTING SIMPLEX ALGORITHM:\n" <<
        " (Initial Basic Feasible Solution Available without Solving First Phase)\n";
    cout << "\n [Table: " << table_number++ << " (Initial Table)]\n";
    output_file << "\n [Table: " << table_number++ << " (Initial Table)]\n";

    print_table();
    table_to_output_file();

    start_simplex_algorithm();
    init_solution(first_optimal_solution);
    if (is_optimal_solution_multiple() == false)
    {
        print_optimal_solution();
        optimal_solution_to_output_file();
    }
    else
    {
        print_multiple_optima();
        multiple_optima_to_output_file();
    }
}
output_file.close();
}
bool is_optimal_solution_multiple()
{
    for (unsigned int j = 0; j < number_of_current_mat_cols - 1; j++)
    {
        if (is_col_in_basis(j) == false)
        {
            if (mat[number_of_current_mat_rows - 1][j] == 0.0)
            {
                entering_variable_index = j;
                leaving_variable_index = get_leaving_variable_index_by_Bland();
                dividing_by_pivot(number_of_current_mat_cols);
                standart_elimination(number_of_current_mat_rows, number_of_current_mat_cols);
                basis[leaving_variable_index] = entering_variable_index; // update basis column
                init_solution(second_optimal_solution);
                if (first_optimal_solution != second_optimal_solution)
                {
                    //cout << "\n Multiple Optima Exist: Additional Pivoting for Finding Multiple Optima";
                    //cout << "\n After Entering: x" << entering_variable_index + 1 <<
                    // " and Leaving: x" << basis[leaving_variable_index] << endl;
                    //cout << "[Table: " << table_number++ << "]\n";
                    //print_table();
                    return true; // multiple optima exist
                }
            }
        }
    }
    second_optimal_solution.erase(second_optimal_solution.begin(), second_optimal_solution.end());
}
return false;
}
int get_entering_index_for_multiple_optima()
{
    for (unsigned int j = 0; j < number_of_current_mat_cols - 1; j++)
    {
        if (is_col_in_basis(j) == false)
        {
            if (mat[number_of_current_mat_rows - 1][j] == 0.0) return j;
        }
    }
    return -1;
}
void init_solution(vector<double>& vec)
{
    unsigned int index;
    for (unsigned int k = 0; k < final_vars_indices.size(); k++)

```

```

        output_file << 'a' << basis[leaving_variable_index]
        - (number_of_current_mat_cols - 1 - artificial_indices.size()) + 1 << '\n';
    }
    else
    {
        cout << 'x' << basis[leaving_variable_index] + 1 << '\n';
        output_file << 'x' << basis[leaving_variable_index] + 1 << '\n';
    }
    basis[leaving_variable_index] = entering_variable_index; // update basis
    dividing_by_pivot(number_of_current_mat_cols);
    standart_elimination(number_of_current_mat_rows, number_of_current_mat_cols);

    cout << " [Table: " << table_number << "]\n";
    output_file << " [Table: " << table_number << "]\n";
    table_number++;
    print_table();
    table_to_output_file();
    checking_precision_of_problem();
}
bool is_col_in_basis(unsigned int number_of_col)
{
    for (unsigned int i = 0; i < basis.size(); i++)
    {
        if (number_of_col == basis[i]) return true;
    }
    return false;
}
void del_art_column_from_mat(unsigned int number_of_col)
{
    for (unsigned int i = 0; i < number_of_current_mat_rows; i++)
    {
        mat[i].erase(mat[i].begin() + number_of_col);
    }
    number_of_current_mat_cols = mat[0].size();
}
void del_art_row_from_mat(unsigned int number_of_row)
{
    mat.erase(mat.begin() + number_of_row);
    number_of_current_mat_rows = mat.size();
}
void removing_artificial_columns()
{
    unsigned int current_art_var_col_num; unsigned int sz = artificial_indices.size();
    unsigned int number_of_col = number_of_current_mat_cols - sz;
    for (int i = sz - 1; i >= 0; i--)
    {
        current_art_var_col_num = artificial_indices[i];
        if (is_col_in_basis(current_art_var_col_num) == false)
        {
            del_art_column_from_mat(current_art_var_col_num);
        }
        else
        {
            entering_variable_index = -1;
            leaving_variable_index = get_row_number(current_art_var_col_num);
            for (unsigned int j = 0; j < number_of_current_mat_cols - sz; j++)
            {
                if (is_col_in_basis(j) == false)
                {
                    if (mat[leaving_variable_index][j] != 0.0) { entering_variable_index = j; break; }
                }
            }
            if (entering_variable_index == -1)
            {
                del_art_row_from_mat(leaving_variable_index); del_art_column_from_mat(current_art_var_col_num);
                basis.erase(basis.begin() + leaving_variable_index);
            }
            else
            {
                basis[leaving_variable_index] = entering_variable_index;
                dividing_by_pivot(number_of_col); // partial pivoting
                standart_elimination(number_of_current_mat_rows - 1, number_of_col); // partial elimination
                del_art_column_from_mat(current_art_var_col_num);
            }
        }
    }
}

```

```

        }
    }

    void initializing_final_vars_indices()
    {
        for (unsigned int i = 0; i < objective_function.size(); i++)
        {
            if (objective_function[i] != 0.0) final_vars_indices.push_back(i);
        }
    }

    void changing_aux_to_original_objective_function()
    {
        mat[number_of_current_mat_rows - 1] = objective_function; double fix;
        unsigned int col_index_in_basis;
        for (unsigned int j = 0; j < basis.size(); j++)
        {
            col_index_in_basis = basis[j];
            if ((fix = mat[number_of_current_mat_rows - 1][col_index_in_basis]) != 0)
            {
                for (unsigned int k = 0; k < number_of_current_mat_cols; k++)
                {
                    mat[number_of_current_mat_rows - 1][k] += (-fix * mat[j][k]);
                }
            }
        }
    }

    void main_algorithm()
    {
        if (is_artificial)
        {
            cout << "\n Solving Auxiliary Problem: \n";
            output_file << "\n Solving Auxiliary Problem: \n";

            cout << "\n [Table: " << table_number++ << " (Initial Table for Normalized Auxiliary Problem)]\n";
            output_file << "\n [Table: " << table_number++ << " (Initial Table for Normalized Auxiliary Problem)]\n";

            print_table();
            table_to_output_file();

            cout << "\n STARTING SIMPLEX FIRST PHASE: (Finding Basic Feasible Solution or Manifest Infeasibility)\n";
            output_file << "\n STARTING SIMPLEX FIRST PHASE: (Finding Basic Feasible Solution or Manifest Infeasibility)\n";

            start_simplex_algorithm();

            if (fabs(Z_optimal_value) > precision)
            {
                cout << setprecision(20) << "\n Main Problem is Infeasible: Z = " << Z_optimal_value << "\n\n";
                output_file << setprecision(20) << "\n Main Problem is Infeasible: Z = " << Z_optimal_value << "\n\n";
                return;
            }

            cout << "\n Removing Artificial Variables, their Columns " <<
                 "and Redundant Constraints by Using Partial Elimination:\n";
            output_file << "\n Removing Artificial Variables," <<
                 " their Columns and Redundant Constraints by Using Partial Elimination:\n";
            removing_artificial_columns();
            cout << " Changing Auxiliary Objective Function to Original Objective Function:\n";
            output_file << " Changing Auxiliary Objective Function to Original Objective Function:\n";

            changing_aux_to_original_objective_function();

            cout << " [Table: " << table_number++ << "]\n";
            output_file << " [Table: " << table_number++ << "]\n";

            print_table();
            table_to_output_file();

            cout << "\n STARTING SIMPLEX SECOND PHASE: (Using the Basic Feasible Solution from First Phase)\n\n";
            output_file << "\n STARTING SIMPLEX SECOND PHASE: (Using the Basic Feasible Solution from First Phase)\n\n";

            start_simplex_algorithm();
            init_solution(first_optimal_solution);

            if (is_optimal_solution_multiple() == false)
            {
                print_optimal_solution();
                optimal_solution_to_output_file();
            }
        }
    }
}

```

```

    {
        index = final_vars_indices[k];
        if (is_col_in_basis(index) == false) vec.push_back(0.0);
        else
            for (unsigned int i = 0; i < number_of_current_mat_rows - 1; i++)
            {
                if (index == basis[i])
                {
                    vec.push_back(mat[i][number_of_current_mat_cols - 1]); break;
                }
            }
    }
}

void optimal_solution_to_output_file()
{
    optimal_value_to_output_file();
    for (unsigned int k = 0; k < first_optimal_solution.size(); k++)
    {
        output_file << " x" << final_vars_indices[k] + 1 << "* = " << first_optimal_solution[k] << endl;
    }
    output_file << endl;
}

void optimal_value_to_output_file(const string str = " Optimal Solution:")
{
    output_file << setprecision(6);
    output_file << str << "\n";
    if (type_of_problem == 0) output_file << " min ";
    else { output_file << " max "; Z_optimal_value *= -1.0; }
    output_file << " Z = " << Z_optimal_value << "\n\n";
}

void multiple_optima_to_output_file()
{
    optimal_value_to_output_file("\n Multiple Optima:");
    for (unsigned int k = 0; k < first_optimal_solution.size(); k++)
    {
        output_file << " x" << final_vars_indices[k] + 1 << "* = " << first_optimal_solution[k] << " * t + "
<< second_optimal_solution[k] << " * (1-t)" << endl;
    }
    output_file << " Where (t >= 0 and t <= 1)\n\n";
}

void print_optimal_solution()
{
    print_optimal_value();
    for (unsigned int k = 0; k < first_optimal_solution.size(); k++)
    {
        cout << " x" << final_vars_indices[k] + 1 << "* = " << first_optimal_solution[k] << endl;
    }
    cout << endl;
}

void print_optimal_value(const string str = " Optimal Solution:")
{
    cout << setprecision(6);
    cout << str << "\n";
    if (type_of_problem == 0) cout << " min ";
    else { cout << " max "; Z_optimal_value *= -1.0; }
    cout << " Z = " << Z_optimal_value << "\n\n";
}

void print_multiple_optima()
{
    print_optimal_value(" Multiple Optima:");
    for (unsigned int k = 0; k < first_optimal_solution.size(); k++)
    {
        cout << " x" << final_vars_indices[k] + 1 << "* = " << first_optimal_solution[k] << " * t + "
<< second_optimal_solution[k] << " * (1-t)" << endl;
    }
    cout << " Where (t >= 0 and t <= 1)\n\n";
}

};

int main()
{
    LP_System object;
    Simplex Problem(object);
    return 0;
}

```

ՀԱՎԵԼՎԱԾ 4

**ԷԼԻՊՍՈՒՆԵՐԻ ՄԵԹՈԴԻ ԾՐԱԳՐԱՅԻՆ
ԻՐԱԿԱՆԱՑՈՒՄԸ**

```

#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <algorithm>
#include <chrono>
#include <atlstr.h>
using namespace std;

////////////////////////////////////////////////////////////////// class Fixed Point Number //////////////////////////////////////////////////////////////////
class Fixed_Point_Number
{
private:
    bool sign;
    int size_of_integer_part;
    int size_of_fractional_part;
    vector<int> num_vec;

////////////////////////////////////////////////////////////////// Member Functions //////////////////////////////////////////////////////////////////

public:
    //constructors/destructor
    Fixed_Point_Number(int sz_of_int, int sz_of_frac, const string& str);
    Fixed_Point_Number(int sz_of_int, int sz_of_frac, int int_part_value);
    Fixed_Point_Number(const Fixed_Point_Number& rhs);
    Fixed_Point_Number();
    ~Fixed_Point_Number();

    //conversions
    operator int() const;

    //assignment operators
    Fixed_Point_Number& operator=(const Fixed_Point_Number& rhs);
    Fixed_Point_Number& operator=(const int number);

    //getter functions
    inline int get_size_of_integer_part() const;
    inline int get_size_of_fractional_part() const;

    //arithmetic operators
    Fixed_Point_Number& operator+=(const Fixed_Point_Number& rhs);
    Fixed_Point_Number& operator+=(int value);
    Fixed_Point_Number& operator-=(const Fixed_Point_Number& rhs);
    Fixed_Point_Number& operator-=(int value);
    Fixed_Point_Number& operator*=(const Fixed_Point_Number& rhs);
    Fixed_Point_Number& operator*=(int value);
    Fixed_Point_Number& operator/=(const Fixed_Point_Number& rhs);
    Fixed_Point_Number& operator/=(int value);
    Fixed_Point_Number& operator++();
    Fixed_Point_Number operator++(int);
    Fixed_Point_Number& operator--();
    Fixed_Point_Number operator--(int);
    Fixed_Point_Number operator-() const;
    Fixed_Point_Number get_abs() const;
    //only for integer powers
    Fixed_Point_Number& operator^=(const Fixed_Point_Number& pow);
    Fixed_Point_Number& operator^=(int pow);

    //helper functions
    void init_integer_part_with_int_value(int int_part_value);
    void resize_vector(int sz_of_int, int sz_of_frac); //by default sign == true(because of all values in vector are 0)

////////////////////////////////////////////////////////////////// Non-Member Functions //////////////////////////////////////////////////////////////////

//output
    friend ostream& operator<<(ostream & os, const Fixed_Point_Number& rhs);

    //arithmetic operators
    friend Fixed_Point_Number operator+(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
    friend Fixed_Point_Number operator+(const Fixed_Point_Number & lhs, int value);
    friend Fixed_Point_Number operator+(int value, const Fixed_Point_Number & rhs);
    friend Fixed_Point_Number operator-(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
    friend Fixed_Point_Number operator-(const Fixed_Point_Number & lhs, int value);

```

```

friend Fixed_Point_Number operator-(int value, const Fixed_Point_Number & rhs);
friend Fixed_Point_Number operator*(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend Fixed_Point_Number operator*(const Fixed_Point_Number & lhs, int value);
friend Fixed_Point_Number operator*(int value, const Fixed_Point_Number & rhs);
friend Fixed_Point_Number operator/(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend Fixed_Point_Number operator/(const Fixed_Point_Number & lhs, int value);
friend Fixed_Point_Number operator/(int value, const Fixed_Point_Number & rhs);
friend Fixed_Point_Number sqroot(const Fixed_Point_Number& a);
//only for integer powers
friend Fixed_Point_Number operator^(const Fixed_Point_Number& lhs, const Fixed_Point_Number& pow);
friend Fixed_Point_Number operator^(const Fixed_Point_Number& lhs, int pow);

//comparators
friend int comparator_for_moduls(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend bool operator==(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend bool operator==(const Fixed_Point_Number & lhs, int value);
friend bool operator==(int value, const Fixed_Point_Number & rhs);
friend bool operator!=(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend bool operator!=(const Fixed_Point_Number & lhs, int value);
friend bool operator<=(int value, const Fixed_Point_Number & rhs);
friend bool operator<(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend bool operator<(const Fixed_Point_Number & lhs, int value);
friend bool operator<(int value, const Fixed_Point_Number & rhs);
friend bool operator>=(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend bool operator>(const Fixed_Point_Number & lhs, int value);
friend bool operator>(int value, const Fixed_Point_Number & rhs);
friend bool operator<=(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend bool operator<=(const Fixed_Point_Number & lhs, int value);
friend bool operator<=(int value, const Fixed_Point_Number & rhs);
friend bool operator>=(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend bool operator>=(const Fixed_Point_Number & lhs, int value);
friend bool operator>=(int value, const Fixed_Point_Number & rhs);

private:

//helper functions
friend Fixed_Point_Number add_moduls(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend Fixed_Point_Number sub_moduls(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend Fixed_Point_Number mul_moduls(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend Fixed_Point_Number div_moduls(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs);
friend int get_size_of_integer_part(const Fixed_Point_Number& object);
friend int get_size_of_fractional_part(const Fixed_Point_Number& object);
friend void init_new_num(Fixed_Point_Number& new_object,
                           const Fixed_Point_Number& object, int index, int number_of_digits);

};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////// Implementation of Member Functions //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//constructors/destructor
Fixed_Point_Number::Fixed_Point_Number(int sz_of_int, int sz_of_frac, const string& str)
    :size_of_integer_part(sz_of_int), size_of_fractional_part(sz_of_frac)
{
    string number = str; int i;
    if (number[0] == '-') { sign = false; number.erase(0, 1); }
    else sign = true;
    int dot_position = number.find('.');
    if (dot_position > size_of_integer_part) { cout << " integer part: overflow\n"; exit(1); }
    if (size_of_fractional_part < (int)number.length() - dot_position - 1)
    { cout << " fraction part: overflow\n"; exit(1); }
    //integer part
    for (i = 0; i < size_of_integer_part - (dot_position); i++) { num_vec.push_back(0); }
    for (i = 0; i < dot_position; i++) { num_vec.push_back(stoi(number.substr(i, 1))); }
    //fraction part
    for (i = dot_position + 1; i < (int)number.length(); i++) { num_vec.push_back(stoi(number.substr(i, 1))); }
    for (i = 0; i < size_of_fractional_part - ((int)number.length() - dot_position - 1); i++) { num_vec.push_back(0); }
}
Fixed_Point_Number::Fixed_Point_Number(int sz_of_int, int sz_of_frac, int int_part_value)
{
    if (sz_of_frac < 1) { cout << " fraction part: overflow\n"; exit(1); }
    resize_vector(sz_of_int, sz_of_frac);

    if (int_part_value != 0)
    {
        sign = (int_part_value > 0) ? true : false;
    }
}

```

```

        if (int_part_value > 0) sign = true;
        else { sign = false; int_part_value *= (-1); }
        init_integer_part_with_int_value(int_part_value);
    }
}
Fixed_Point_Number::Fixed_Point_Number(const Fixed_Point_Number& rhs)
{
    :sign(rhs.sign), size_of_integer_part(rhs.size_of_integer_part), size_of_fraction_part(rhs.size_of_fraction_part)
{
    int size = size_of_integer_part + size_of_fraction_part;
    num_vec.resize(size);
    for (int i = 0; i < size; i++)
    {
        num_vec[i] = rhs.num_vec[i];
    }
}
Fixed_Point_Number::Fixed_Point_Number()
{
}
Fixed_Point_Number::~Fixed_Point_Number()
{
}

//converions
Fixed_Point_Number::operator int() const
{
    __int64 ret_value = 0; int counter = 0;

    for (int i = size_of_integer_part - 1; i >= 0; i--)
    {
        if (num_vec[i] != 0)
        {
            if (counter + 1 > log10(INT_MAX) + 1) { cout << "can't convert to int\n"; exit(1); }
            ret_value += (__int64)(num_vec[i]) * pow(10, counter));
        }
        counter++;
    }
    if (sign == true)
    {
        if (ret_value > INT_MAX) { cout << "can't convert to int\n"; exit(1); }
        return int(ret_value);
    }
    __int64 num = INT_MIN;
    if (ret_value > (-num)) { cout << "can't convert to int\n"; exit(1); }
    return (int)(-ret_value);
}
//assignment operators
Fixed_Point_Number& Fixed_Point_Number::operator=(const Fixed_Point_Number& rhs)
{
    if (this == &rhs) { return *this; }
    if ((size_of_integer_part != rhs.size_of_integer_part) || (size_of_fraction_part != rhs.size_of_fraction_part))
    {
        cout << " error: different sizes\n"; exit(1);
    }

    sign = rhs.sign;
    size_of_integer_part = rhs.size_of_integer_part;
    size_of_fraction_part = rhs.size_of_fraction_part;
    for (int i = 0; i < size_of_integer_part + size_of_fraction_part; i++)
    {
        num_vec[i] = rhs.num_vec[i];
    }

    return *this;
}
Fixed_Point_Number& Fixed_Point_Number::operator=(const int number)
{
    Fixed_Point_Number temp(this->size_of_integer_part, this->size_of_fraction_part, number);
    *this = temp;
    return *this;
}

//getter functions
int Fixed_Point_Number::get_size_of_integer_part() const
{
    return size_of_integer_part;
}

```

```

int Fixed_Point_Number::get_size_of_fractional_part() const
{
    return size_of_fractional_part;
}

//arithmetic operators
Fixed_Point_Number& Fixed_Point_Number::operator+=(const Fixed_Point_Number& rhs)
{
    *this = (*this) + rhs;
    return (*this);
}

Fixed_Point_Number& Fixed_Point_Number::operator+=(int value)
{
    Fixed_Point_Number temp(this->size_of_integer_part, this->size_of_fractional_part, value);
    *this += temp;
    return *this;
}

Fixed_Point_Number& Fixed_Point_Number::operator-=(const Fixed_Point_Number& rhs)
{
    *this = *this - rhs;
    return (*this);
}

Fixed_Point_Number& Fixed_Point_Number::operator-=(int value)
{
    Fixed_Point_Number temp(this->size_of_integer_part, this->size_of_fractional_part, value);
    *this -= temp;
    return *this;
}

Fixed_Point_Number& Fixed_Point_Number::operator*=(const Fixed_Point_Number& rhs)
{
    *this = (*this) * rhs;
    return *this;
}

Fixed_Point_Number& Fixed_Point_Number::operator*=(int value)
{
    Fixed_Point_Number temp(this->size_of_integer_part, this->size_of_fractional_part, value);
    *this *= temp;
    return *this;
}

Fixed_Point_Number& Fixed_Point_Number::operator/=(const Fixed_Point_Number& rhs)
{
    *this = (*this) / rhs;
    return *this;
}

Fixed_Point_Number& Fixed_Point_Number::operator/=(int value)
{
    Fixed_Point_Number temp(this->size_of_integer_part, this->size_of_fractional_part, value);
    *this /= temp;
    return *this;
}

Fixed_Point_Number& Fixed_Point_Number::operator++()
{
    *this += 1;
    return *this;
}

Fixed_Point_Number Fixed_Point_Number::operator++(int)
{
    Fixed_Point_Number temp(*this);
    *this += 1;
    return temp;
}

Fixed_Point_Number& Fixed_Point_Number::operator--()
{
    *this -= 1;
    return *this;
}

Fixed_Point_Number Fixed_Point_Number::operator--(int)
{
    Fixed_Point_Number temp(*this);
    *this -= 1;
    return temp;
}

Fixed_Point_Number Fixed_Point_Number::operator-() const
{
    Fixed_Point_Number temp(*this);
    temp.sign = !(this->sign);
    return temp;
}

```

```

Fixed_Point_Number Fixed_Point_Number::get_abs() const
{
    Fixed_Point_Number res(*this);
    res.sign = true;
    return res;
}
//only for integer powers
Fixed_Point_Number& Fixed_Point_Number::operator^=(const Fixed_Point_Number& pow)
{
    *this = (*this) ^ pow;
    return *this;
}
Fixed_Point_Number& Fixed_Point_Number::operator^=(int pow)
{
    *this = (*this) ^ pow;
    return *this;
}

//helper functions
void Fixed_Point_Number::init_integer_part_with_int_value(int int_part_value)
{
    vector<int> current_vec;
    int sz = (int)log10(int_part_value) + 1;
    if (sz > size_of_integer_part) { cout << " integer part: overflow\n"; exit(1); }

    for (int i = 0; i < sz; i++)
    {
        current_vec.push_back(int_part_value % 10);
        int_part_value /= 10;
    }

    reverse(current_vec.begin(), current_vec.end());
    copy(current_vec.begin(), current_vec.end(), num_vec.begin() + size_of_integer_part - sz);
}
void Fixed_Point_Number::resize_vector(int sz_of_int, int sz_of_frac)
//by default sign == true(because of all values in vector are 0)
{
//  this function called after default constructor
    if (sz_of_int < 1) { cout << "error: size of integer part < 1\n"; exit(1); }
    if (sz_of_frac < 1) { cout << "error: size of fractional part < 1\n"; exit(1); }
    sign = true;
    size_of_integer_part = sz_of_int;
    size_of_fraction_part = sz_of_frac;
    num_vec.resize(size_of_integer_part + size_of_fraction_part);
}

/////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////// Implementation of Non-Member Functions /////
/////////////////////////////////////////////////////////////////////////

//output
ostream& operator<<(ostream & os, const Fixed_Point_Number& rhs)
{
    int i;
    if (rhs.sign == false) os << '-';
    for (i = 0; i < rhs.size_of_integer_part; i++) { os << rhs.num_vec[i]; }
    os << '.';
    for (i = rhs.size_of_integer_part; i < (int)rhs.num_vec.size(); i++) { os << rhs.num_vec[i]; }
    return os;
}

//arithmetic operators
Fixed_Point_Number operator+(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    if ((lhs.size_of_integer_part != rhs.size_of_integer_part) ||
        (lhs.size_of_fraction_part != rhs.size_of_fraction_part))
    {
        cout << " error: different sizes\n"; exit(1);
    }

    Fixed_Point_Number res;
    res.resize_vector(lhs.size_of_integer_part, lhs.size_of_fraction_part);

    if ((lhs.sign == true & rhs.sign == true) || (lhs.sign == false & rhs.sign == false))
    {
        res = add_moduls(lhs, rhs);
        res.sign = lhs.sign;
        return res;
    }
}

```

```

    }
else
{
    int comparator_result = comparator_for_moduls(lhs, rhs);
    switch (comparator_result)
    {
        //if lhs == rhs
        case 0: { return res; }
                  //if lhs < rhs
        case 1: { res = sub_moduls(rhs, lhs); res.sign = rhs.sign; return res; }
                  //if lhs > rhs
        case -1: { res = sub_moduls(lhs, rhs); res.sign = lhs.sign; return res; }
        default: { exit(1); }
    }
}
}

Fixed_Point_Number operator+(const Fixed_Point_Number & lhs, int value)
{
    Fixed_Point_Number temp(lhs.size_of_integer_part, lhs.size_of_fraction_part, value);
    return (lhs + temp);
}

Fixed_Point_Number operator+(int value, const Fixed_Point_Number & rhs)
{
    return (rhs + value);
}

Fixed_Point_Number operator-(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    if ((lhs.size_of_integer_part != rhs.size_of_integer_part) ||
        (lhs.size_of_fraction_part != rhs.size_of_fraction_part))
    {
        cout << " error: different sizes\n"; exit(1);
    }
    Fixed_Point_Number res;
    res.resize_vector(lhs.size_of_integer_part, lhs.size_of_fraction_part);
    //tested
    if ((lhs.sign == true && rhs.sign == true) || (lhs.sign == false && rhs.sign == false))
    {
        int comparator_result = comparator_for_moduls(lhs, rhs);
        switch (comparator_result)
        {
            //if lhs == rhs
            case 0: { return res; }
                      //if lhs < rhs
            case 1: { res = sub_moduls(rhs, lhs); res.sign = !rhs.sign; return res; }
                      //if lhs > rhs
            case -1: { res = sub_moduls(lhs, rhs); res.sign = lhs.sign; return res; }
        }
    }
    // tested
    if ((lhs.sign == true && rhs.sign == false) || (lhs.sign == false && rhs.sign == true))
    {
        res = add_moduls(lhs, rhs);
        res.sign = lhs.sign;
        return res;
    }
    exit(1);
}

Fixed_Point_Number operator-(const Fixed_Point_Number & lhs, int value)
{
    Fixed_Point_Number temp(lhs.size_of_integer_part, lhs.size_of_fraction_part, value);
    return (lhs - temp);
}

Fixed_Point_Number operator-(int value, const Fixed_Point_Number & rhs)
{
    return (-rhs + value);
}

Fixed_Point_Number operator*(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    if ((lhs.size_of_integer_part != rhs.size_of_integer_part) ||
        (lhs.size_of_fraction_part != rhs.size_of_fraction_part))
    {
        cout << " error: different sizes\n"; exit(1);
    }
    Fixed_Point_Number res = mul_moduls(lhs, rhs);
    if ((lhs.sign == false && rhs.sign == true) || (lhs.sign == true && rhs.sign == false))
    {
        if (res != 0) res.sign = false;
    }
}

```

```

        }
        return res;
    }
    Fixed_Point_Number operator*(const Fixed_Point_Number & lhs, int value)
    {
        Fixed_Point_Number temp(lhs.size_of_integer_part, lhs.size_of_fraction_part, value);
        return (lhs * temp);
    }
    Fixed_Point_Number operator*(int value, const Fixed_Point_Number & rhs)
    {
        return (rhs * value);
    }
    Fixed_Point_Number operator/(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
    {
        if ((lhs.size_of_integer_part != rhs.size_of_integer_part) ||
            (lhs.size_of_fraction_part != rhs.size_of_fraction_part))
        {
            cout << " error: different sizes\n"; exit(1);
        }
        if (rhs == 0) { cout << " error: division by zero\n"; exit(1); }
        Fixed_Point_Number new_lhs(lhs);
        Fixed_Point_Number new_rhs(rhs);
        if (new_lhs.sign == false) new_lhs.sign = true;
        if (new_rhs.sign == false) new_rhs.sign = true;
        Fixed_Point_Number res = div_moduls(new_lhs, new_rhs);
        if ((lhs.sign == false & rhs.sign == true) || (lhs.sign == true & rhs.sign == false))
        {
            if (res != 0) res.sign = false;
        }
        return res;
    }
    Fixed_Point_Number operator/(const Fixed_Point_Number & lhs, int value)
    {
        Fixed_Point_Number temp(lhs.size_of_integer_part, lhs.size_of_fraction_part, value);
        return (lhs / temp);
    }
    Fixed_Point_Number operator/(int value, const Fixed_Point_Number & rhs)
    {
        Fixed_Point_Number temp(rhs.size_of_integer_part, rhs.size_of_fraction_part, value);
        return (temp / rhs);
    }

    Fixed_Point_Number sqroot(const Fixed_Point_Number& object)
    {
        Fixed_Point_Number zero(object.size_of_integer_part, object.size_of_fraction_part, 0);
        if (object < 0) { cout << " error: division by zero\n"; exit(1); }
        Fixed_Point_Number x = object;
        Fixed_Point_Number y; y.resize_vector(object.size_of_integer_part, object.size_of_fraction_part);
        Fixed_Point_Number two(object.size_of_integer_part, object.size_of_fraction_part, 2);
        while ((x - y) != zero)
        {
            y = (x + object / x) / two;
            x = (y + object / y) / two;
        }
        return x;
    }
    //only for integer powers
    Fixed_Point_Number operator^(const Fixed_Point_Number& lhs, const Fixed_Point_Number& pow)
    {
        Fixed_Point_Number num(lhs);
        if (pow == 0)
        {
            return (num = 1);
        }
        Fixed_Point_Number factor(lhs);
        for (int i = 1; i < pow.get_abs(); i++)
        {
            num *= factor;
        }
        if (pow > 0)
            return num;
        return (1 / num);
    }
    Fixed_Point_Number operator^(const Fixed_Point_Number& lhs, int pow)
    {
        Fixed_Point_Number num(lhs);

```

```

        if (pow == 0)
        {
            return (num = 1);
        }
        Fixed_Point_Number factor(lhs);
        for (int i = 1; i < abs(pow); i++)
        {
            num *= factor;
        }
        if (pow > 0)
            return num;
        return (1 / num);
    }

//comparators
int comparator_for_moduls(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    for (int i = 0; i < (int)lhs.num_vec.size(); i++)
    {
        if (lhs.num_vec[i] == rhs.num_vec[i]) continue;
        if (lhs.num_vec[i] > rhs.num_vec[i]) return -1;
        return 1;
    }
    return 0; // equal
}
bool operator==(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    if (lhs.sign == rhs.sign)
    {
        if (comparator_for_moduls(lhs, rhs) == 0) return true;
        return false;
    }
    return false;
}
bool operator==(const Fixed_Point_Number & lhs, int value)
{
    Fixed_Point_Number temp(lhs.size_of_integer_part, lhs.size_of_fraction_part, value);
    return (lhs == temp);
}
bool operator==(int value, const Fixed_Point_Number & rhs)
{
    return (rhs == value);
}
bool operator!=(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    if (lhs == rhs) return false;
    return true;
}
bool operator!=(const Fixed_Point_Number & lhs, int value)
{
    Fixed_Point_Number temp(lhs.size_of_integer_part, lhs.size_of_fraction_part, value);
    return (lhs != temp);
}

bool operator!=(int value, const Fixed_Point_Number & rhs)
{
    return (rhs != value);
}
bool operator<(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    if (lhs == rhs) return false;
    if (lhs.sign < rhs.sign) return true;
    if (lhs.sign > rhs.sign) return false;
    else
    {
        if (lhs.sign == true)
        {
            if (comparator_for_moduls(lhs, rhs) == 1) return true;
            return false;
        }
        else
        {
            if (comparator_for_moduls(lhs, rhs) == -1) return true;
            return false;
        }
    }
}

```

```

bool operator<(const Fixed_Point_Number & lhs, int value)
{
    Fixed_Point_Number temp(lhs.size_of_integer_part, lhs.size_of_fraction_part, value);
    return (lhs < temp);
}
bool operator<(int value, const Fixed_Point_Number & rhs)
{
    return (rhs > value);
}
bool operator>(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    if (lhs < rhs || lhs == rhs) return false;
    return true;
}
bool operator>(const Fixed_Point_Number & lhs, int value)
{
    Fixed_Point_Number temp(lhs.size_of_integer_part, lhs.size_of_fraction_part, value);
    return (lhs > temp);
}
bool operator>(int value, const Fixed_Point_Number & rhs)
{
    return (rhs < value);
}
bool operator<=(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    if (lhs > rhs) return false;
    return true;
}
bool operator<=(const Fixed_Point_Number & lhs, int value)
{
    Fixed_Point_Number temp(lhs.size_of_integer_part, lhs.size_of_fraction_part, value);
    return (lhs <= temp);
}
bool operator<=(int value, const Fixed_Point_Number & rhs)
{
    return (rhs >= value);
}
bool operator>=(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    if (lhs < rhs) return false;
    return true;
}
bool operator>=(const Fixed_Point_Number & lhs, int value)
{
    Fixed_Point_Number temp(lhs.size_of_integer_part, lhs.size_of_fraction_part, value);
    return (lhs >= temp);
}
bool operator>=(int value, const Fixed_Point_Number & rhs)
{
    return (rhs <= value);
}

//helper functions
Fixed_Point_Number add_moduls(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    Fixed_Point_Number res;
    res.resize_vector(lhs.size_of_integer_part, lhs.size_of_fraction_part);
    int carry = 0; int sum;
    for (int i = lhs.num_vec.size() - 1; i >= 0; i--)
    {
        sum = lhs.num_vec[i] + rhs.num_vec[i] + carry;
        res.num_vec[i] = sum % 10;
        carry = sum / 10;
    }

    if (carry == 1) { cout << "overflow\n"; exit(1); }
    return res;
}
Fixed_Point_Number sub_moduls(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    Fixed_Point_Number res;
    res.resize_vector(lhs.size_of_integer_part, lhs.size_of_fraction_part);

    int carry = 0;
    for (int i = lhs.num_vec.size() - 1; i >= 0; i--)
    {
        res.num_vec[i] = lhs.num_vec[i] - rhs.num_vec[i] - carry;
        if (res.num_vec[i] < 0) { res.num_vec[i] += 10; carry = 1; }
        else { carry = 0; }
    }
}

```

```

    }
    return res;
}
Fixed_Point_Number mul_moduls(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    Fixed_Point_Number res;
    res.resize_vector(2 * lhs.size_of_integer_part, 2 * lhs.size_of_fraction_part);
    Fixed_Point_Number current_res;
    current_res.resize_vector(2 * lhs.size_of_integer_part, 2 * lhs.size_of_fraction_part);

    int carry; int j; int i; int current_value; int d;
    int rhs_counter = 0;
    for (i = rhs.num_vec.size() - 1; i >= 0; i--)
    {
        if (rhs.num_vec[i] == 0) { rhs_counter++; continue; }
        carry = 0; d = 0;
        for (j = lhs.num_vec.size() - 1; j >= 0; j--)
        {
            if (lhs.num_vec[j] == 0) { current_res.num_vec[current_res.num_vec.size() - 1 - d - rhs_counter] = carry; carry = 0; d++; continue; }
            current_value = rhs.num_vec[i] * lhs.num_vec[j] + carry;
            current_res.num_vec[current_res.num_vec.size() - 1 - d - rhs_counter] = (current_value % 10);
            carry = current_value / 10;
            d++;
        }
        current_res.num_vec[current_res.num_vec.size() - 1 - d - rhs_counter] = carry;
        res += current_res;
        fill(current_res.num_vec.begin(), current_res.num_vec.end(), 0);
        rhs_counter++;
    }

    Fixed_Point_Number temp;
    temp.resize_vector(lhs.size_of_integer_part, lhs.size_of_fraction_part);

    int int_part_without_initial_zeros;

    for (i = 0; i < res.size_of_integer_part; i++)
    {
        if (res.num_vec[i] != 0)
        {
            if ((int_part_without_initial_zeros = res.size_of_integer_part - i) > temp.size_of_integer_part)
            {
                cout << "overflow\n"; exit(1);
            }
            break;
        }
    }
    int index = temp.size_of_integer_part;
    for (int i = 0; i < (int)temp.num_vec.size(); i++)
    {
        temp.num_vec[i] = res.num_vec[index];
        index++;
    }

    return temp;
}
Fixed_Point_Number div_moduls(const Fixed_Point_Number & lhs, const Fixed_Point_Number & rhs)
{
    if (lhs == rhs)
    {
        Fixed_Point_Number result;
        result.resize_vector(lhs.size_of_integer_part, lhs.size_of_fraction_part);
        result = 1;
        return result;
    }
    int frac_size, int_size, lhs_int_size, rhs_int_size, lhs_frac_size, rhs_frac_size, size, pow;

    lhs_int_size = get_size_of_integer_part(lhs); rhs_int_size = get_size_of_integer_part(rhs);
    int_size = (lhs_int_size >= rhs_int_size) ? lhs_int_size : rhs_int_size;
    lhs_frac_size = get_size_of_fractional_part(lhs); rhs_frac_size = get_size_of_fractional_part(rhs);
    frac_size = (lhs_frac_size >= rhs_frac_size) ? lhs_frac_size : rhs_frac_size;

    Fixed_Point_Number new_lhs, new_rhs;
    new_lhs.resize_vector(int_size + frac_size + 2, 1);
    new_rhs.resize_vector(int_size + frac_size + 2, 1);

    init_new_num(new_lhs, lhs, lhs.size_of_integer_part - int_size, int_size + frac_size);
    init_new_num(new_rhs, rhs, rhs.size_of_integer_part - int_size, int_size + frac_size);
}

```

```

Fixed_Point_Number result;
if (lhs > rhs)
{
    pow = abs(get_size_of_integer_part(new_lhs) - get_size_of_integer_part(new_rhs));
    for (int i = 0; i < pow + 1; i++) { new_rhs *= 10; }
    result.resize_vector(lhs.size_of_integer_part, lhs.size_of_fraction_part + pow + 1);
    size = lhs.size_of_fraction_part + pow + 1;
}
else
{
    result.resize_vector(lhs.size_of_integer_part, lhs.size_of_fraction_part);
    size = result.size_of_fraction_part;
}
new_lhs *= 10;
Fixed_Point_Number current_result;
current_result.resize_vector(new_lhs.size_of_integer_part, new_lhs.size_of_fraction_part);

for (int i = 0; i < size; i++)
{
    if (new_lhs < new_rhs) { result.num_vec[result.size_of_integer_part + i] = 0; new_lhs *= 10; }
    else if (new_lhs >= new_rhs)
    {
        for (int j = 9; j > 0; j--)
        {
            if ((current_result = new_lhs - new_rhs * j) >= 0)
            {
                result.num_vec[result.size_of_integer_part + i] = j;
                new_lhs = current_result * 10;
                break;
            }
        }
    }
}
if (lhs > rhs)
{
    for (int i = 0; i < pow + 1; i++) { result *= 10; }
    result.resize_vector(lhs.size_of_integer_part, lhs.size_of_fraction_part);
}
return result;
}
int get_size_of_fractional_part(const Fixed_Point_Number& object)
{
    int object_frac_size = 0;
    for (int i = object.size_of_fraction_part - 1; i >= 0; i--)
    {
        if (object.num_vec[object.size_of_integer_part + i] != 0) { return (i + 1); }
    }
    return object_frac_size;
}
int get_size_of_integer_part(const Fixed_Point_Number& object)
{
    int object_int_size = 0;
    for (int i = 0; i < object.size_of_integer_part; i++)
    {
        if (object.num_vec[i] != 0) { return (object.size_of_integer_part - i); }
    }
    return object_int_size;
}
void init_new_num(Fixed_Point_Number& new_object, const Fixed_Point_Number& object, int index, int number_of_digits)
{
    for (int i = 0; i < number_of_digits; i++)
    {
        new_object.num_vec[i + 2] = object.num_vec[index + i];
    }
}

////////////////////////////////////////////////////////////////////////// class matrix //////////////////////////////
template<class T>
class matrix
{
private:
    int rows;
    int cols;
    vector<vector<T>> mat;

```

```

/////////////////////////////// Member Functions //////////////////
public:
    //constructors/destructor
    matrix();
    matrix(int, int);
    matrix(const matrix<T>& rhs);
    ~matrix();

    //assignment
    matrix<T>& operator=(const matrix<T>& rhs);

    //indexing subscript
    inline T& operator() (int row, int col);
    inline T operator() (int row, int col) const;

    //different functions for matrices
    matrix<T>& making_diagonal_matrix(const T & val);
    matrix<T>& transpose();
    matrix<T> get_transpose_matrix();
    T get_num() const;

    //arithmetic operations
    matrix<T>& operator+=(const matrix<T>& rhs);
    matrix<T>& operator-=(const matrix<T>& rhs);
    matrix<T>& operator*=(const matrix<T>& rhs);
    matrix<T>& operator*=(const T & val);
    matrix<T>& operator/=(const T & val);

    //Helper Functions
    void alloc_space();
    friend void resize_matrix_elements(matrix<Fixed_Point_Number>& object,
                                      int size_of_int_part, int size_of_frac_part);
    void resize_matrix(int row, int col)
    {
        rows = row;
        cols = col;
        alloc_space();
    } // for Fixed_Point_Number just alloc

    //getters
    inline int get_rows() const;
    inline int get_cols() const;

/////////////////////////////// Non-Member Functions //////////////////
//input/output
template<class T> friend ostream& operator<<(ostream & os, const matrix<T>& rhs);
template<class T> friend istream& operator>>(istream & is, matrix<T>& rhs);

//arithmetic operations
template<class T> friend matrix<T> operator+(const matrix<T>& lhs, const matrix<T>& rhs);
template<class T> friend matrix<T> operator-(const matrix<T>& lhs, const matrix<T>& rhs);
template<class T> friend matrix<T> operator*(const matrix<T>& lhs, const matrix<T>& rhs);
template<class T> friend matrix<T> operator*(const matrix<T>& lhs, const T & val);
template<class T> friend matrix<T> operator*(const T & val, const matrix<T>& rhs);
template<class T> friend matrix<T> operator/(const matrix<T>& lhs, const T & val);

};

/////////////////////////////// Positive definiteness test for square matrices //////////////////

template<class T> bool swap_sqr_mat_rows(matrix<T>& mat, int swap_candidate_row_index, int size)
{
    T num(mat(0, 0));
    for (int i = swap_candidate_row_index + 1; i < size; i++)
    {
        if (mat(i, i) != 0)
        {
            for (int j = 0; j < size; j++)
            {

```

```

        num = mat(swap_candidate_row_index, j);
        mat(swap_candidate_row_index, j) = mat(i, j);
        mat(i, j) = num;
    }
    return true;
}
return false;
}
template<class T> void to_nullify_column(matrix<T>& mat, int index, int size)
{
    T num(mat(0, 0));
    for (int i = index + 1; i < size; i++)
    {
        if (mat(i, index) != 0)
        {
            num = -mat(i, index) / mat(index, index);

            for (int j = 0; j < size; j++)
            {
                mat(i, j) += (num * mat(index, j));
            }
        }
    }
}
template<class T> bool Positive_def_test_for_sqr_mat(matrix<T>& mat)
{
    matrix<T> copy_matrix(mat);
    int size = copy_matrix.get_rows();
    for (int i = 0; i < size - 1; i++)
    {
        if (copy_matrix(i, i) == 0)
        {
            if (swap_sqr_mat_rows(copy_matrix, i, size) == false) return false;
        }
        if (copy_matrix(i, i) <= 0) return false;
        to_nullify_column(copy_matrix, i, size);
    }
    if (copy_matrix(size - 1, size - 1) <= 0) return false;
    return true;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////// Implementation of Member Functions////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Constructors/Destructor
template<class T> matrix<T>::matrix() : rows(1), cols(1)
{
    alloc_space();
}
template<> matrix<Fixed_Point_Number>::matrix() : rows(1), cols(1)
{
    alloc_space();
    mat[0][0].resize_vector(1, 1);
}
template<class T> matrix<T>::matrix(int row, int col) : rows(row), cols(col)
{
    if (row < 1 || col < 1) { cout << " error: wrong sizes for matrix\n"; exit(1); }
    alloc_space();
}
template<class T> matrix<T>::matrix(const matrix<T>& rhs) : rows(rhs.rows), cols(rhs.cols)
{
    alloc_space();
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols; ++j)
        {
            mat[i][j] = rhs.mat[i][j];
        }
    }
}
template<> matrix<Fixed_Point_Number>::matrix(const matrix<Fixed_Point_Number>& rhs) : rows(rhs.rows), cols(rhs.cols)
{
    alloc_space();
    int sz_of_int_part = rhs.mat[0][0].get_size_of_integer_part();
    int sz_of_frac_part = rhs.mat[0][0].get_size_of_fractional_part();
}

```

```

    {
        for (int j = 0; j < cols; ++j)
        {
            mat[i][j].resize_vector(sz_of_int_part, sz_of_frac_part);
        }
    }
    for (int i = 0; i < rows; ++i)
    {
        for (int j = 0; j < cols; ++j)
        {
            mat[i][j] = rhs.mat[i][j];
        }
    }
}
template<class T> matrix<T>::~matrix()
{
}

//assignment
template<class T> matrix<T>& matrix<T>::operator=(const matrix<T>& rhs)
{
    if (this == &rhs) { return *this; }
    if (rows != rhs.rows || cols != rhs.cols)
    {
        cout << " forbidden: different sizes\n"; exit(1);
    }

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            mat[i][j] = rhs.mat[i][j];
        }
    }
    return *this;
}

//indexing subscript
template<class T> T& matrix<T>::operator() (int row, int col)
{
    if (row >= rows || col >= cols || row < 0 || col < 0) { cout << " out of range\n"; exit(1); }
    return mat[row][col];
}
template<class T> T matrix<T>::operator() (int row, int col) const
{
    if (row >= rows || col >= cols || row < 0 || col < 0) { cout << " out of range\n"; exit(1); }
    return mat[row][col];
}

//different functions for matrices
template<class T> matrix<T>& matrix<T>::making_diagonal_matrix(const T& val)
{
    if (rows != cols) { cout << " impossible to make diagonal matrix, because of rectangular form\n"; exit(1); }
    for (int i = 0; i < rows; i++)
    {
        mat[i][i] = val;
    }
    return *this;
}
template<class T> matrix<T>& matrix<T>::transpose()
{
    matrix<T> result(cols, rows);

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            result.mat[j][i] = mat[i][j];
        }
    }
    rows = result.rows;
    cols = result.cols;
    alloc_space();
    *this = result;
    return *this;
}

```

```

template<> matrix<Fixed_Point_Number>& matrix<Fixed_Point_Number>::transpose()
{
    matrix<Fixed_Point_Number> result(cols, rows);
    resize_matrix_elements(result, this->mat[0][0].get_size_of_integer_part(),
    this->mat[0][0].get_size_of_fractional_part());

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            result.mat[j][i] = mat[i][j];
        }
    }

    rows = result.rows;
    cols = result.cols;
    alloc_space();

    resize_matrix_elements(*this, result.mat[0][0].get_size_of_integer_part(),
    result.mat[0][0].get_size_of_fractional_part());

    *this = result;
    return *this;
}
template<class T> matrix<T> matrix<T>::get_transpose_matrix()
{
    matrix<T> rhs_copy(*this);
    rhs_copy.transpose();
    return rhs_copy;
}
template<class T> T matrix<T>::get_num() const
{
    if (rows > 1 || cols > 1) { cout << " error: matrix has many elements\n"; exit(1); }
    return mat[0][0];
}

//arithmetic operations
template<class T> matrix<T>& matrix<T>::operator+=(const matrix<T>& rhs)
{
    *this = *this + rhs;

    return *this;
}
template<class T> matrix<T>& matrix<T>::operator-=(const matrix<T>& rhs)
{
    *this = *this - rhs;
    return *this;
}
template<class T> matrix<T>& matrix<T>::operator*=(const matrix<T>& rhs)
{
    *this = *this * rhs;
    return *this;
}
template<class T> matrix<T>& matrix<T>::operator*=(const T& val)
{
    *this = *this * val;
    return *this;
}
template<class T> matrix<T>& matrix<T>::operator/=(const T& val)
{
    *this = *this / val;
    return *this;
}

//Helper Functions
template<class T> void matrix<T>::alloc_space()
{
    mat.resize(rows);
    for (int i = 0; i < rows; i++)
    {
        mat[i].resize(cols);
    }
}

//getters
template<class T> int matrix<T>::get_rows() const
{
    return rows;
}

```

```

template<class T> int matrix<T>::get_cols() const
{
    return cols;
}

////////////////////////////////////////////////////////////////// Implementation of Non-Member Functions //////////////////////////////////////////////////////////////////
//input/output
template<class T> istream& operator>>(istream& is, matrix<T>& rhs)
{
    for (int i = 0; i < rhs.rows; i++)
    {
        for (int j = 0; j < rhs.cols; j++)
        {
            cout << " matrix[" << i << "][" << j << "] = ";
            is >> rhs.mat[i][j];
        }
    }
    return is;
}
template<class T> ostream& operator<<(ostream& os, const matrix<T>& rhs)
{
    os << endl;
    for (int i = 0; i < rhs.rows; i++)
    {
        for (int j = 0; j < rhs.cols; j++)
        {
            os << '\t' << rhs.mat[i][j];
        }
        os << '\n';
    }
    return os;
}
//arithmetic operations
template<class T> matrix<T> operator+(const matrix<T>& lhs, const matrix<T>& rhs)
{
    if (lhs.rows != rhs.rows || lhs.cols != rhs.cols) { cout << " forbidden: different sizes\n"; exit(1); }
    matrix<T> result(lhs.rows, lhs.cols);

    for (int i = 0; i < lhs.rows; i++)
    {
        for (int j = 0; j < lhs.cols; j++)
        {
            result.mat[i][j] = lhs.mat[i][j] + rhs.mat[i][j];
        }
    }
    return result;
}

template<> matrix<Fixed_Point_Number> operator+(const matrix<Fixed_Point_Number>& lhs,
                                                const matrix<Fixed_Point_Number>& rhs)
{
    if (lhs.rows != rhs.rows || lhs.cols != rhs.cols) { cout << " forbidden: different sizes\n"; exit(1); }
    matrix<Fixed_Point_Number> result(lhs.rows, lhs.cols);

    resize_matrix_elements(result, lhs.mat[0][0].get_size_of_integer_part(),
                          lhs.mat[0][0].get_size_of_fractional_part());

    for (int i = 0; i < lhs.rows; i++)
    {
        for (int j = 0; j < lhs.cols; j++)
        {
            result.mat[i][j] = lhs.mat[i][j] + rhs.mat[i][j];
        }
    }
    return result;
}
template<class T> matrix<T> operator-(const matrix<T>& lhs, const matrix<T>& rhs)
{
    if (lhs.rows != rhs.rows || lhs.cols != rhs.cols) { cout << " forbidden: different sizes\n"; exit(1); }
    matrix<T> result(lhs.rows, lhs.cols);

    for (int i = 0; i < lhs.rows; i++)
    {
        for (int j = 0; j < lhs.cols; j++)
        {

```

```

        result.mat[i][j] = lhs.mat[i][j] - rhs.mat[i][j];
    }
    return result;
}
template<> matrix<Fixed_Point_Number> operator-(const matrix<Fixed_Point_Number>& lhs,
                                                 const matrix<Fixed_Point_Number>& rhs)
{
    if (lhs.rows != rhs.rows || lhs.cols != rhs.cols) { cout << " forbidden: different sizes\n"; exit(1); }
    matrix<Fixed_Point_Number> result(lhs.rows, lhs.cols);

    resize_matrix_elements(result, lhs.mat[0][0].get_size_of_integer_part(),
                           lhs.mat[0][0].get_size_of_fractional_part());

    for (int i = 0; i < lhs.rows; i++)
    {
        for (int j = 0; j < lhs.cols; j++)
        {
            result.mat[i][j] = lhs.mat[i][j] - rhs.mat[i][j];
        }
    }
    return result;
}
template<class T> matrix<T> operator*(const matrix<T>& lhs, const matrix<T>& rhs)
{
    if (lhs.cols != rhs.rows) { cout << " forbidden multiplication\n"; exit(1); }
    matrix<T> result(lhs.rows, rhs.cols);
    for (int i = 0; i < result.rows; i++)
    {
        for (int j = 0; j < result.cols; j++)
        {
            for (int k = 0; k < lhs.cols; k++)
            {
                result.mat[i][j] += (lhs.mat[i][k] * rhs.mat[k][j]);
            }
        }
    }
    return result;
}
template<> matrix<Fixed_Point_Number> operator*(const matrix<Fixed_Point_Number>& lhs,
                                                 const matrix<Fixed_Point_Number>& rhs)
{
    if (lhs.cols != rhs.rows) { cout << " forbidden multiplication\n"; exit(1); }
    matrix<Fixed_Point_Number> result(lhs.rows, rhs.cols);

    resize_matrix_elements(result, lhs.mat[0][0].get_size_of_integer_part(),
                           lhs.mat[0][0].get_size_of_fractional_part());

    for (int i = 0; i < result.rows; i++)
    {
        for (int j = 0; j < result.cols; j++)
        {
            for (int k = 0; k < lhs.cols; k++)
            {
                result.mat[i][j] += (lhs.mat[i][k] * rhs.mat[k][j]);
            }
        }
    }
    return result;
}
template<class T> matrix<T> operator*(const matrix<T>& lhs, const T& val)
{
    matrix<T> result(lhs.rows, lhs.cols);
    for (int i = 0; i < result.rows; i++)
    {
        for (int j = 0; j < result.cols; j++)
        {
            result.mat[i][j] = lhs.mat[i][j] * val;
        }
    }
    return result;
}
template<> matrix<Fixed_Point_Number> operator*(const matrix<Fixed_Point_Number>& lhs, const Fixed_Point_Number& val)
{
    matrix<Fixed_Point_Number> result(lhs.rows, lhs.cols);

    resize_matrix_elements(result, lhs.mat[0][0].get_size_of_integer_part(),
                           lhs.mat[0][0].get_size_of_fractional_part());
}

```

```

        for (int i = 0; i < result.rows; i++)
    {
        for (int j = 0; j < result.cols; j++)
        {
            result.mat[i][j] = lhs.mat[i][j] * val;
        }
    }
    return result;
}
template<class T> matrix<T> operator*(const T& val, const matrix<T>& rhs)
{
    return (rhs * val);
}
template<class T> matrix<T> operator/(const matrix<T>& lhs, const T& val)
{
    if (val == 0) { cout << " division by 0\n"; exit(1); }
    matrix<T> result(lhs.rows, lhs.cols);
    for (int i = 0; i < result.rows; i++)
    {
        for (int j = 0; j < result.cols; j++)
        {
            result.mat[i][j] = lhs.mat[i][j] / val;
        }
    }
    return result;
}
template<> matrix<Fixed_Point_Number> operator/(const matrix<Fixed_Point_Number>& lhs, const Fixed_Point_Number& val)
{
    if (val == 0) { cout << " division by 0\n"; exit(1); }
    matrix<Fixed_Point_Number> result(lhs.rows, lhs.cols);

    resize_matrix_elements(result, lhs.mat[0][0].get_size_of_integer_part(),
                           lhs.mat[0][0].get_size_of_fractional_part());

    for (int i = 0; i < result.rows; i++)
    {
        for (int j = 0; j < result.cols; j++)
        {
            result.mat[i][j] = lhs.mat[i][j] / val;
        }
    }
    return result;
}
//Helper Functions
void resize_matrix_elements(matrix<Fixed_Point_Number>& object, int size_of_int_part, int size_of_frac_part)
{
    for (int i = 0; i < object.rows; i++)
    {
        for (int j = 0; j < object.cols; j++)
        {
            object.mat[i][j].resize_vector(size_of_int_part, size_of_frac_part);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
typedef Fixed_Point_Number f_num;
template<class Type> matrix<Type> get_Cholesky_decomposition(matrix<Type>& A_mat)
{
    int size = A_mat.get_rows();
    matrix<Type> L_mat(size, size);
    double sum = 0.0;
    for (int j = 1; j <= size; j++)
    {
        for (int k = 1; k <= (j - 1); k++)
        {
            sum += (L_mat(j - 1, k - 1) * L_mat(j - 1, k - 1));
        }
        L_mat(j - 1, j - 1) = sqrt(A_mat(j - 1, j - 1) - sum);
        sum = 0;
        for (int i = j + 1; i <= size; i++)
        {
            for (int k = 1; k <= (j - 1); k++)
            {
                sum += (L_mat(i - 1, k - 1) * L_mat(j - 1, k - 1));
            }
        }
    }
}

```

```

        L_mat(i - 1, j - 1) = (A_mat(i - 1, j - 1) - sum) / L_mat(j - 1, j - 1);
        sum = 0.0;
    }
    return L_mat;
}
template<> matrix<f_num> get_Cholesky_decomposition(matrix<f_num>& A_mat)
{
    int size = A_mat.get_rows();

    int sz_integer_part = A_mat(0, 0).get_size_of_integer_part();
    int sz_fractional_part = A_mat(0, 0).get_size_of_fractional_part();
    matrix<f_num> L_mat(size, size);
    resize_matrix_elements(L_mat, sz_integer_part, sz_fractional_part);
    f_num sum(sz_integer_part, sz_fractional_part, 0);
    for (int j = 1; j <= size; j++)
    {
        for (int k = 1; k <= (j - 1); k++)
        {
            sum += (L_mat(j - 1, k - 1) * L_mat(j - 1, k - 1));
        }
        L_mat(j - 1, j - 1) = sqroot(A_mat(j - 1, j - 1) - sum);
        sum = 0;
        for (int i = j + 1; i <= size; i++)
        {
            for (int k = 1; k <= (j - 1); k++)
            {
                sum += (L_mat(i - 1, k - 1) * L_mat(j - 1, k - 1));
            }
            L_mat(i - 1, j - 1) = (A_mat(i - 1, j - 1) - sum) / L_mat(j - 1, j - 1);
            sum = 0;
        }
    }
    return L_mat;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////// Ellipsoid Algorithm for bounded, empty or full-dimensional polytopes //////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

class Ellipsoid_Method_with_Cholesky_Dec
{
private:
    matrix<f_num> system_of_inequalities;                                // Ax <= b, coefficients of A and B
    int rows;                                                               // m = rows (number of inequalities)
    int cols;                                                               // number of columns(A) + 1(b column)
    int n;                                                                // n = cols - 1 (number of variables)

    double v_min;                                                          // number of iterations
    double v_max;                                                          // the biggest absolute value in A and b
    int K;                                                                // required number of decimal digits before comma
    int U;                                                                // required number of decimal digits after comma
    int integer_part_precision;                                            // E[0], E[1], ... , E[K] current ellipsoid // n_x_n
    int fractional_part_precision;                                         // c[0], c[1], ... , c[K] vector for center of
    matrix<f_num> current_ell;                                           // current ellipsoid // n_x_1
    matrix<f_num> center_of_current_ell;

    bool perturbation;                                                     //output_file
    fstream output_file;                                                    // 1 + 1/[ 4(n+1)^2 ]
    f_num blow_up_parameter;

public:
    void init_the_biggest_U()
    {
        U = abs(system_of_inequalities(0, 0));

        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                if (abs(system_of_inequalities(i, j)) > U)
                    U = abs(system_of_inequalities(i, j));
            }
        }
    }
    void init_K()
    {
        v_min = 1 / (pow(n, n)*pow((n*U), n*n*n + n * n));
    }
}

```

```

        v_max = pow((2 * n), n)* pow((n*U), n*n);
        //K = 2 * (n + 1)*log(v_max / v_min) + 1;
        double delta = 1.0 + 1.0 / (4 * (n + 1)*(n + 1));
        K = 2 * (n + 1)*log((v_max * delta)/ v_min) + 1;
    }
    void init_blow_up_parameter()
    {
        blow_up_parameter.resize_vector(integer_part_precision, fractional_part_precision);
        blow_up_parameter = 1;

        blow_up_parameter += blow_up_parameter / (4 * (n + 1)*(n + 1));
    }
    void construct_initial_ellipsoid();

    //main_algorithm
    void main_algorithm();
    //separation oracle
    bool init_violated_ineq_entries(matrix<f_num>& system, int number_of_variables,
                                    matrix<f_num>& center_of_current_ell, matrix<f_num>& a_vector);
    //initializing
    Ellipsoid_Method_with_Cholesky_Dec(const matrix<int>& mat, int int_part_prec, int frac_part_prec, bool perturb);
    //output_file
    void make_output_file();

};

//////////////////////////////////////////////////////////////////////// implementation //////////////////
////////////////////////////////////////////////////////////////////////

//output_file
void Ellipsoid_Method_with_Cholesky_Dec::make_output_file()
{
    string Path_for_output = "C:\\\\Users\\\\gpu\\\\Desktop\\\\Ellipsoids\\\\";
    chrono::system_clock::time_point today = chrono::system_clock::now();
    time_t time_date = chrono::system_clock::to_time_t(today);
    string solution_starting_time = ctime(&time_date);

    vector<string> list_of_files_in_directory;
    CString directory = Path_for_output.c_str();
    HANDLE dir;
    WIN32_FIND_DATA file_data;
    CString file_name;
    if ((dir = FindFirstFile((directory + "\\*"), &file_data)) == INVALID_HANDLE_VALUE)
    {
        cout << " Invalid directory\\n";
    }
    while (FindNextFile(dir, &file_data))
    {
        file_name = file_data.cFileName;
        if (strcmp((const char*)file_data.cFileName, ".") != 0 && strcmp((const char*)file_data.cFileName, "..") != 0)
        {
            list_of_files_in_directory.push_back(string(CW2A(file_name.GetString())));
        }
    }
    string name_of_output_file; unsigned int h = 0;
    string solution = "solution"; string dot_txt = ".txt";
    vector<string>::iterator it;
    name_of_output_file = solution + dot_txt;
    while ((it = find(list_of_files_in_directory.begin(), list_of_files_in_directory.end(), name_of_output_file)) != list_of_files_in_directory.end())
    {
        name_of_output_file = solution + to_string(h) + dot_txt;
        h++;
    }

    output_file.open(Path_for_output + name_of_output_file, ios::out);
    if (!output_file) { cout << " Ouput File doesn't open\\n"; exit(1); }
    output_file << "\\n " << solution_starting_time <<
    "\\n Ellipsoid Algorithm for Linear Inequalities with Cholesky Decomposition (with Fixed_Point_Number Type):\\n";
}

//main algorithm
void Ellipsoid_Method_with_Cholesky_Dec::main_algorithm()
{

```

```

//////////////////////////// 1/(n + 1) //////////////////////////////
f_num one_div_n_plus_one(integer_part_precision, fractional_part_precision, 1);
one_div_n_plus_one /= (n + 1);
//////////////////////////// 2/(n + 1) //////////////////////////////
f_num two_div_n_plus_one(integer_part_precision, fractional_part_precision, 2);
two_div_n_plus_one *= one_div_n_plus_one;
//////////////////////////// (n * n) / ( (n * n) - 1 ) //////////////////
f_num sqr_n_div_sqr_n_minus_one(integer_part_precision, fractional_part_precision, n*n);
sqr_n_div_sqr_n_minus_one /= (sqr_n_div_sqr_n_minus_one - 1);
////////////////////////////

matrix<f_num> B_tr(n, n); resize_matrix_elements(B_tr, integer_part_precision, fractional_part_precision);
matrix<f_num> B(n, n); resize_matrix_elements(B, integer_part_precision, fractional_part_precision);
matrix<f_num> d(n, 1); resize_matrix_elements(d, integer_part_precision, fractional_part_precision);
matrix<f_num> b(n, 1); resize_matrix_elements(b, integer_part_precision, fractional_part_precision);
matrix<f_num> a_vector(n, 1);
resize_matrix_elements(a_vector, integer_part_precision, fractional_part_precision);
// vector of entries of violated inequality (without rhs)

for (int i = 1; i <= K; i++)
{
    output_file << " Iteration: " << i << endl;
    if (init_violated_ineq_entries(system_of_inequalities, n, center_of_current_ell, a_vector) == true)
    {
        B_tr = get_Cholesky_decomposition(current_ell);
        B = B_tr.get_transpose_matrix();

        b = ((B_tr * B) * a_vector) / sqroot((a_vector.get_transpose_matrix() * (B_tr * B) * a_vector).get_num());

        //calculating next ellipsoid's center
        center_of_current_ell -= (one_div_n_plus_one * b);
        output_file << " Next ellipsoid's center:" << center_of_current_ell;

        //calculating next ellipsoid
        current_ell = (sqr_n_div_sqr_n_minus_one * (current_ell - (two_div_n_plus_one *
                                                               (b * b.get_transpose_matrix()))));
        output_file << " Next ellipsoid:" << current_ell;

        current_ell *= blow_up_parameter; // blowing u
        output_file << " After blowing up:" << current_ell << endl;
    }
    else
    {
        output_file << " Feasible solution:\n" << center_of_current_ell << endl;
        output_file.close();
        return;
    }
}
output_file << " Infeasible\n";
output_file.close();
return;
}

//initializing
Ellipsoid_Method_with_Cholesky_Dec::Ellipsoid_Method_with_Cholesky_Dec(const matrix<int>& mat, int int_part_prec,
                                                                      int frac_part_prec, bool perturb)
{
    rows = mat.get_rows();
    cols = mat.get_cols();
    n = cols - 1;
    perturbation = perturb;

    integer_part_precision = int_part_prec;
    fractional_part_precision = frac_part_prec;
    system_of_inequalities.resize_matrix(rows, cols); resize_matrix_elements(system_of_inequalities,
                                                                           integer_part_precision, fractional_part_precision);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            system_of_inequalities(i, j) = mat(i, j);
        }
    }

    init_the_biggest_U();
    init_K();
    init_blow_up_parameter();

    construct_initial_ellipsoid();
    center_of_current_ell.resize_matrix(n, 1);
}

```

```

resize_matrix_elements(center_of_current_ell, integer_part_precision, fractional_part_precision);
                                            // initial center (0,0,...,0) n_x_1
f_num epsilon(integer_part_precision, fractional_part_precision, n + 1);
if (perturbation == true)
{
    f_num U_f_num(integer_part_precision, fractional_part_precision, U);
    epsilon ^= (n + 2);
    U_f_num ^= (n + 1);
    epsilon *= (2 * U_f_num);
    epsilon = 1 / epsilon;
    for (int i = 0; i < rows; i++)
    {
        system_of_inequalities(i, cols - 1) += epsilon;
    }
}

make_output_file();
output_file << "\n Number of variables: n = " << n << endl;
output_file << " Number of required iterations: K = " << K << endl;
output_file << " Integer part precision: " << integer_part_precision << endl;
output_file << " Fractional part precision: " << fractional_part_precision << endl;
output_file << " Perturbation: epsilon = ";
if (perturbation == true)
{
    output_file << epsilon;
}
else output_file << 0;
output_file << "\n Blow up parameter: delta = " << blow_up_parameter << endl;
output_file << "\n System of inequalities:" << system_of_inequalities << endl;
output_file << "\n Center of Initial Ellipsoid:" << center_of_current_ell;
output_file << "\n Initial Ellipsoid:" << current_ell << endl;
}

void Ellipsoid_Method_with_Cholesky_Dec::construct_initial_ellipsoid()
{
    f_num diag_elem(integer_part_precision, fractional_part_precision, n*U);
    diag_elem ^= (2 * n);

    diag_elem *= n;
    current_ell.resize_matrix(n, n); resize_matrix_elements(current_ell,
                                                                integer_part_precision, fractional_part_precision);
    current_ell.making_diagonal_matrix(diag_elem);
}

//separation oracle
bool Ellipsoid_Method_with_Cholesky_Dec::init_violated_ineq_entries(matrix<f_num>& system, int number_of_variables,
                                                                    matrix<f_num>& center_of_current_ell, matrix<f_num>& a_vector)
{
    a_vector.transpose(); // 1_x_n
    for (int i = 0; i < system.get_rows(); i++)
    {
        for (int j = 0; j < number_of_variables; j++)
        {
            a_vector(0, j) = system(i, j);
        }
        if ((a_vector * center_of_current_ell).get_num() > system(i, number_of_variables))
        {
            a_vector.transpose();
            return true;
        }
    }
    return false;
}

int main()
{
    matrix<int> mat(6, 3);

    mat(0, 0) = -1;      mat(0, 1) = -1;      mat(0, 2) = -1;
    mat(1, 0) = 1;       mat(1, 1) = 1;       mat(1, 2) = -1;
    mat(2, 0) = -1;      mat(2, 1) = 0;       mat(2, 2) = 0;
    mat(3, 0) = 0;       mat(3, 1) = -1;      mat(3, 2) = 0;
    mat(4, 0) = 1;       mat(4, 1) = 0;       mat(4, 2) = 4;
    mat(5, 0) = 0;       mat(5, 1) = 1;       mat(5, 2) = 4;

    Ellipsoid_Method_with_Cholesky_Dec object(mat, 25, 70, true);
    object.main_algorithm();
    return 0;
}

```