**Титульный лист**

**МАЗМҰНЫ**

[ҚОЛДАНЫЛҒАН НОРМАТИВТЕР 3](#_Toc103323018)

[ҚЫСҚАРТУЛАР МЕН БЕЛГІЛЕУЛЕР 4](#_Toc103323019)

[КІРІСПЕ 5](#_Toc103323020)

[1 ГАУСС ӘДІСІ 8](#_Toc103323021)

[1.1 Сипаттамасы және модификациялары 8](#_Toc103323022)

[1.2 Жетекші элементті таңдау алгоритмін бағдарламалау 18](#_Toc103323023)

[1.3 Жетекші элементті таңдау алгоритмін параллельдеу 20](#_Toc103323024)

[2 БІРІКТІРІЛГЕН ГРАДИЕНТТЕР ӘДІСІ 25](#_Toc103323025)

[2.1 Cипаттамасы және модификациялары 25](#_Toc103323026)

[2.2 Шешу алгоритмін бағдарламалау 28](#_Toc103323027)

[2.3 Шешу алгоритмін параллелдеу 28](#_Toc103323028)

[3 ЕСЕПТЕУ ЭКСПЕРИМЕНТТЕРІ ЖӘНЕ ТАЛДАУ 29](#_Toc103323029)

[3.1 Эксперименттің жүргізілу шарттары 29](#_Toc103323030)

[3.2 Эксперименттер қоюға қажетті қосымша класстар жайлы 30](#_Toc103323031)

[3.3 Эксперимент нәтижелерін алу және талдау 32](#_Toc103323032)

[ҚОРЫТЫНДЫ 33](#_Toc103323033)

[ҚОЛДАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ 34](#_Toc103323034)

# **ҚОЛДАНЫЛҒАН НОРМАТИВТЕР**

# **ҚЫСҚАРТУЛАР МЕН БЕЛГІЛЕУЛЕР**

|  |  |
| --- | --- |
| САТЖ | сызықтық алгебралық теңдеулер жүйесі |
|  | өлшемді коэффициенттер матрицасы |
|  | бос мүшелер векторы |
|  | Айнымалылар векторы |
|  | теңдеулер жүйесінің матрицалық түрде жазылуы |
| OpenMP | Open Multi-Processing – C, C++ және Fortran тілдеріндегі бағдарламаларды параллелдеуге арналған ашық стандарт |
| CG | conjugate-gradient – біріктірілген градиент әдісі |
|  | және  векторларын скалярлы көбейту, басқаша |

# **КІРІСПЕ**

Көптеген қолданбалы, оның ішінде экономикалық , табиғат құбылыстары, өндірістік инженерия, кескінді тану және т.б. сызықтық теңдеулер жүйесіне әкелінеді.

 айнымалысы бар және  сызықтық теңдеулерден тұратын жүйе келесідей жазылады:

 (1)

мұндағы  - сәйкесінше айнымалылардың коэффициенттері және теңдеулердің бос мүшелері деп аталатын ерікті сандар.[1]

Қысқаша белгілеуде, жинақтау белгілерін пайдаланып, жүйені келесідей жазуға болады:

 (2)

1. *жүйенің шешімі* деп жүйенің әр теңдеуіндегі белгісіз айнымалы орнына қойғанда  шындыққа айналатын  сандар жиынын айтамыз.[2]

Сызықтық алгебралық теңдеулер жүйесінің дым аз дегенде бір шешімі бар болса, онда оны *үйлесетін* (совместной), ал шешімдері жоқ болса *үйлеспейтін* (несовместной) деп аталады.

Үйлесетін теңдеулер жүйесінің жалғыз шешімі болса, *анықталған*, ал бірнеше шешімі болса, *анықталмаған* деп аталады. Мысалы,  - теңдеулер жүйесі үйлесімді және анықталған, себебі жүйенің бір-ақ қана шешімі бар, және ол ;  - теңдеулер жүйесі үйлеспейтін болып табылады, себебі жүйені қанағаттандыратын ешқандай сандар жиынын таба алмаймыз;  - теңдеулер жүйесі үйлесімді және анықталмаған, себебі жүйенің бірден көп, басқаша айтқанда шексіз көп шешімі бар, және ол  мұндағы  - кез – келген сан.[3-4]

Егер екі жүйенің шешімдер жиыны бірдей болса, онда оларды *эквивалентті* деп атайды. (1) – жүйені элементар түрлендірулерінің көмегімен (мысалы, теңдеулердің екі бөлігін де нөлге тең емес сандарға көбейту; жүйенің теңдеулерін бір – біріне қосу) берілген (1) жүйеге эквивалентті теңдеулер жүйелерін аламыз.

1. Жүйені матрицалық формада жазайық.



мұндағы  - айнымалы алдындағы коэффициенттер матрицасы, немесе жүйе матрицасы;  - айнымалылар баған – матрицасы;  - бос мүшелер баған – матрицасы.

 матрицасының бағандар саны мен  матрицасының жолдар саны тең болғандықтан, оларды көбейтуге болады. Нәтижесінде:



 баған-матрицаны аламыз. Және ол (1) жүйенің сол жағы болып табылады. Матрицалардың теңдік ережесі бойынша (1) жүйені келесі түрде жазсақ болады:

 (3)

Теңдеулер жүйесін шешудің барлық әдістерін шартты түрде дәл (нақты) және жуық деп бөлуге болады. Дәл алгоритмдерге Крамер, Гаусс, Джордан-Гаусс, прогонка және т.б. әдістер жатады. Жуықтап есептеу әдістеріне итерациялық әдістер (Якоби, Зейдель, релаксация, біріктірілген градиенттер және т.б.), квадрат түбірлер әдісін және т.б. жатады.[5]

**Дипломдық жұмыстың өзектілігі.** Тәжірибеде өте үлкен өлшемдегі сызықтық теңдеулер жүйесін шешуге тура келеді. Мысалыға, экономикада кіріс-шығыс балансын құрастыру кезінде белгісіздер саны жүзден асатын жүйелер кездеседі. Бұл тақырыптың өзектілігі осындай сияқты жүйелерді шешуде қолданылатын әдістерді зерттеуге және талдауға арналған

**Дипломдық жұмыстың мақсаты**:

* САТЖ шешудің тура және итерациялық әдістерін зерттеу
* САТЖ шешудің алгоритмін программалау тілінде құру
* Құрылған алгоритм коды бойынша есептеу эксперименттерін жүргізіп, алынған нәтижелер бойынша анализ жасау

**Дипломдық жұмыстың тапсырмалары:**

* САТЖ шешудің тікелей және итерациялық әдістеріне шолу
* Гаусс және CG әдісінің сызықты алгоритмін С++ тілінде жазу
* Гаусс және CG әдісінің параллельді алгоритмін С++ тілінде жазу
* Эксперименттер өткізіп, нәтижесі бойынша талдау жүргізу

**Бірінші бөлімде** САТЖ шешудің ең танымал және қолданбалы әдісі, Гаусс әдісінің қысқаша тарихы, басқа дәл әдістермен салыстырғанда артықшылығы жазылған. Әдістің математикалық алгоритмінің егжей – тегжейлі талдауы, және осы алгоритмнің орындалуының көрнекті бір мысалын көрсетілген. Осы математикалық алгоритмнің  матрицасы үшін коэффициенттерді, содан кейін айнымалыларды есептеудің жалпы формулалары шықты. Осы есептеулердің жалпы формулаларын қолданып, бағдарламалау алгоритмінің блок-схемасы құрылды. Блок – схема құру барысында Гаусстың классикалық әдісінің әлсіз жерлері, яғни диагональ элементі өте кіші сан болған кезде, бөлу операциясы есептеу қателігін өсіріп жіберетіні байқалды. Осылайша есептеу қателігінің өсуі азайту үшін бағдарламалауға Гаусстың модификациялық әдісінің алгоритмі алынды. Сондай-ақ осы жетекші элементті таңдаумен жазылған алгоритм OpenMP көмегімен параллелдеудің нюанстары көрсетілді.

**Екінші бөлімде** біріктірілген градиенттер әдісі жазылған.

**Үшінші бөлімде** алдыңғы бөлімдерде құрылған блок-схема бойынша C++ тілінде бағдарламалардың қалай жазылғаны жайлы айтылған. Алгоритм дұрыс шешім беріп жатқанын анықтау үшін тексеруші бөлікке жазылды. Алгоритмдерде есептеу қайталаулары көп болғандықтан, OpenMP қолданбалы бағдарлама интерфейсі көмегімен есептеу бөліктерін бірнеше ағындарға (потоктарға) бөлу жоспарланды. Енді осы жазылған кодтың тиімділігі анықтау мақсатында үлкен өлшемді матрицалар есептеуге жіберілді. Ондай үлкен өлшемді матрицаларды қолмен толтырып отырмас үшін, кездейсоқ мәндермен толтыру көмекші бағдарлама жазылды. Осы қойылған эксперименттер нәтижесі талдауға алынып, қорытынды жасалынды.

# **ГАУСС ӘДІСІ**

## **Сипаттамасы және модификациялары**

Сызықтық алгебралық теңдеулер жүйесінің(САТЖ) бір шешімі де шексіз көп шешімі де шешімдері жоқ болуы да мүмкін. САТЖ шешудің барлық әдістері екінші жағдайды, яғни жүйенің шексіз көп шешімдері болған жағдайда шешімнің біреуін де таба алмайды. Мысалы, Крамер әдісі мен матрицалық әдіс қолданылмайды, алайда Гаусс әдісімен шешуге болады.

Бұл әдістің тарихына шолу жасайтын болсақ, бұл әдіс Карл Фридрих Гауссқа дейін де белгілі болғанын байқаймыз. Әдістің алғашқы белгілі сипаттамасы біздің дәуірімізге дейінгі I ғасыр және II ғасыр арасында құрастырылған қытайлық «Тоғыз кітаптағы математика» трактатында көрсетілген[6]. САТЖ шешудің Гаусс әдісін кей оқулықтарды Гаусстық жою әдісі деп те атайды.

Гаусс әдісі – сызықтық алгебралық теңдеулер жүйесін шешудің классикалық әдісі. Бұл элементар түрлендірулерді қолдана отырып, теңдеулер жүйесі сатылы (немесе үшбұрышты) түрдегі эквивалентті жүйеге келтірілгенде, айнымалы мәндерді дәйекті жою әдісі, оның ішінен барлық басқа айнымалылар соңғысынан бастап дәйекті түрде табылды. (Кремер, 2010)

Жүйедегі (1)  айнымалысының алдында бірінші теңдеуде нөлге тең емес  деп есептейік. Егер нөлге тең болса, онда теңдеулердің орнын ауыстырып нөлге тең емес жағдайына келтіреміз.

*1 – қадам.* САТЖ – нің бірінші теңдеуін сәйкес сандарға көбейтіп (атап айтқанда ) және алынған теңдеулерді (1) жүйенің сәйкесінше екінші, үшінші, …, - ші теңдеулеріне қоссақ, бірінші теңдеуден басқа, яғни екінші теңдеуден бастап  айнымалысынан құтыламыз.

 (4)

мұндағы үстіндегі  белгісі бірінші қадамнан кейінгі пайда болған жаңа коэффициентті білдіреді.

*2 – қадам.* Жүйеде (4)  деп есептейік. Егер нөлге тең болса, онда теңдеулердің орнын ауыстырып нөлге тең емес жағдайына келтіреміз. (4) жүйенің екінші теңдеуін сәйкес сандарға көбейтіп (атап айтқанда ) және алынған теңдеулерді (4) жүйенің сәйкесінше үшінші, төртінші, …, - ші теңдеулеріне қоссақ, үшінші теңдеуден бастап  айнымалысынан құтыламыз.



мұндағы үстіндегі  белгісі екінші қадамнан кейінгі пайда болған жаңа коэффициентті білдіреді.

Осылайша қадам жалғастыра берсек, әр қадамда  айнымалыларынан құтыламыз. Және соңғы  қадамда келесі жүйені аламыз

 (5)

Соңғы  теңдеуіндегі нөл саны олардың сол жақ бөліктері  түрінде болатынын білдіреді. Егер де (5) жүйенің  бос мүшелерінің ең болмағанда біреуі нөлге тең болмаса, онда осы теңдік қарама-қайшы болады да, (1) жүйе үйлеспейтін болып саналады, яғни (1) жүйенің шешімі жоқ болады.

Осылайша, кез-келген үйлесімді САТЖ-үшін (5) жүйеде  сандары нөлге тең. Бұл жағдайда, (5) жүйенің соңғы  теңдеуі қарама-қайшылық көрсетпейді, және де (1) жүйені шешу кезінде елемеуге болады. Әлбетте, «артық» теңдеулерді алып тастағаннан кейін екі жағдай болуы мүмкін: а) (5) жүйесіндегі теңдеулер саны айнымалылар санына тең, яғни  (бұл жағдайда жүйе үшбұрыш пішінге ие); б) r < n (бұл жағдайда (5) жүйе сатылы пішінге ие).

(1) жүйенің оның эквивалентті жүйесіне (5) өтуі Гаусс әдісінің *тура жүріс*, ал (5) жүйесінен айнымалыларды табу *кері жүріс* деп аталады.

Гаусс түрлендірулерін теңдеулердің өздерімен емес, олардың коэффициенттерінің матрицасымен түрлендіруді орындау арқылы жүргізу ыңғайлы. Келесідей матрицаны қарастырайық:

 (6)

(1) жүйенің *кеңейтілген матрицасы* деп аталады, өйткені ол жүйенің А матрицасына қосымша бос мүшелер бағанын қосылған.

*Мысалы,* Берілген САТЖ Гаусс әдісімен шешімін тап.



*Шешімі.* САТЖ – ның кеңейтілген матрицасы келесі түрде болады:



*1 – қадам.*  болғандықтан, матрицаның бірінші жолын (-2), (-3), (-2) сандарына көбейтіп, және сәйкес екінші, үшінші және төртінші жолдарға қосамыз. Осының арқасында екінші жолдан бастап, бірінші баған элементтері нөлге тең болады, басқаша айтқанда екінші теңдеуден бастап  айнымалысын жоямыз.



Келесі қадам бастамас бұрын нөлге тең көріп тұрмыз, демек матрица жолдарын өзара орнын ауыстырып нөлге тең емес жағдайына келтіреміз.



*2 – қадам.*  матрицаның екінші жолын (-7/4) санына көбейтіп, төртінші жолға қосамыз. Осының арқасында үшінші жолдан бастап, екінші баған элементтері нөлге тең болады, басқаша айтқанда үшінші теңдеуден бастап  айнымалысын жоямыз.



*3 – қадам.*  болғандықтан, матрицаның үшінші жолын (13,5 / 8 = 27 / 16) санына көбейтіп, және сәйкес төртінші жолға қосамыз. Сонда келесі матрица шығады:



Матрица түрін жүйе түріне аударып жазсақ. Келесі жүйені аламыз:



Соңғы жүйені шешу үшін астынан үстіге қарай айнымалы мәндерін есептеп отырамыз. Осы кезеңін Гаусс әдісінің кері жүрісі деп атайды. Төртінші теңдеуден ; осы шешімді алдындағы теңдеуге (атап айтқанда үшінші теңдеуге) қойып есептейміз ; ал екінші теңдеуге қойып ; ал бірінші теңдеуге қойып . Яғни, теңдеу шешімі .

Есептеу сынақтарын жасау кезінде, және талдау кезінде де (1) жүйенің үйлесімді болу мүмкіндігін көбейту үшін біз айнымалылар саны  мен теңдеулер санын  тең қыламыз, басқаша айтқанда  матрицасы квадратты түрде болады.

Гаусс әдісін бағдарламалауды бастамас бұрын, әдістің келесі қадамын есептеудің жалпы математикалық теңдеуін есептеп алайық.

Тура жүріс кезеңінде, келесі қадам үшін таңдалған коэффициент астындағы коэффициенттердің мәнін есептеу формуласы:

 (7)

Ал кері жүріс кезеңінде, коэффициенттерді және де есептелінген айнымалыларды ескере отырып, келесі айнымалыны есептеу формуласы

 (8)

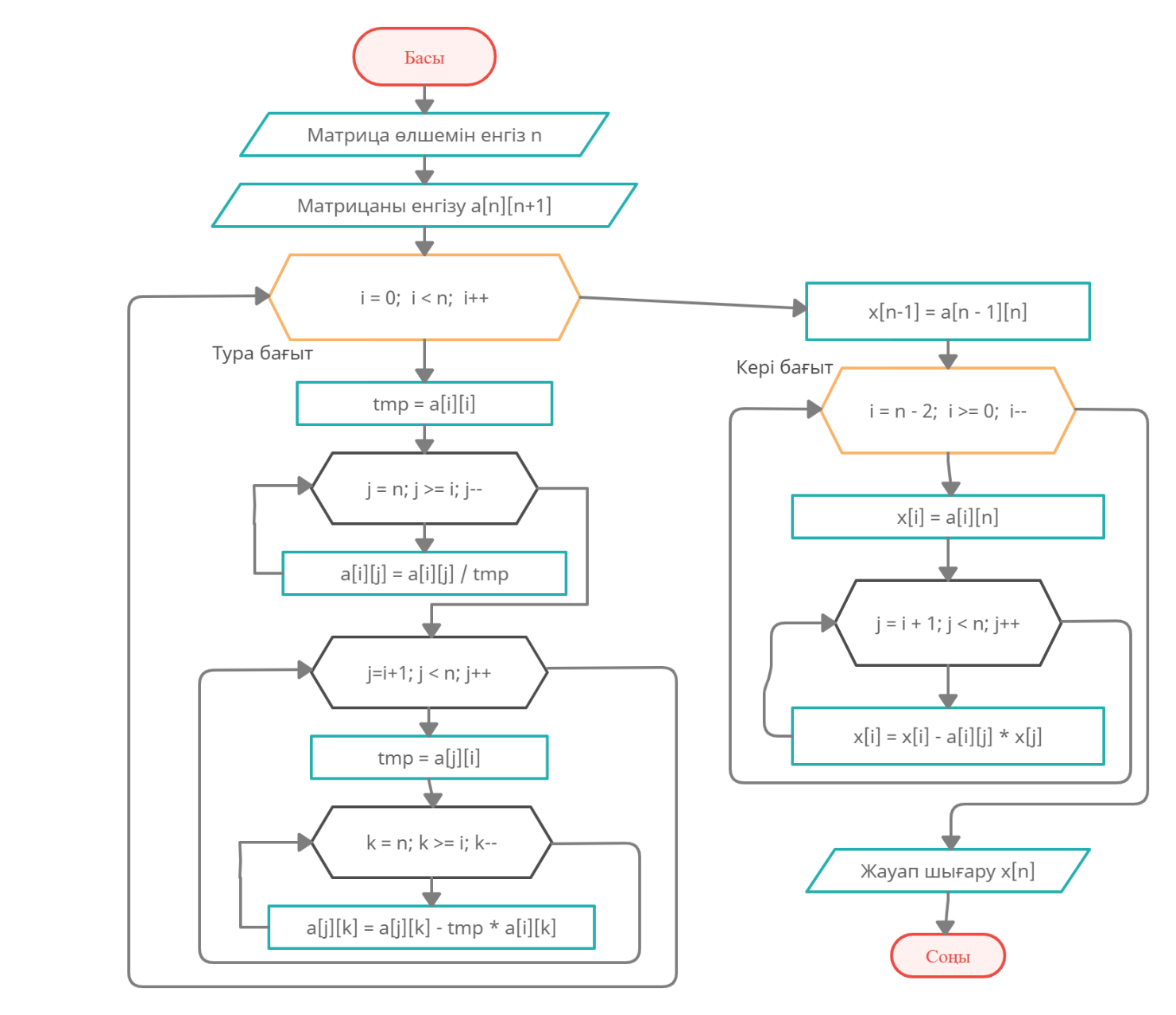
Жүйені (1) және (7) формуланы біріктіріп тура жүріс кезеңін есептеудің жалпы формуласын аламыз:



Ал кері жүріс кезеңіндегі есептеудің жалпы формула



Енді екі кезеңді қосып, Гаусс әдісінің алгоритмін блок-схемасын құрастырайық. (1 – сурет)



1 – сурет. Гаусс әдісінің алгоритмі блок – схема түрінде

Алайда, классикалық алгоритмі бағдарламалап, есептеу кезінде диагональ элементінің кіші болған кезде, қателіктердің жиналуы байқалды, ол есептеуіш бағдарламаның тұрақсыздығына әкеледі.

Бұл мәселені болдырмаудың ықтимал жолы – бағдарламалауға Гаусс әдісінің модификацияланған алгоритмін қолдану. Модификацияланған әдістің классикалық әдістен айырмашылығы тура жүріс кезінде баған элементтері ішіндегі абсолютті мәні бойынша ең үлкен элемент тұрған жолды жетекші қылып, осы бағандағы басқа элементтерді жоямыз. Сондай – ақ есептеу жылдамдығын тағы да өсіру үшін, жолдардың орнын ауыстыру операциясын орындамаймыз. Жою барысында егер де қандай да бір жол осыған дейінгі қадамдарда жетекші болған болса, сол жолдың элементтерін секіріп өтіп кетеміз.

*Мысалы*, Берілген САТЖ – ны Гаусс әдісінің баған бойынша жетекші элементті таңдауымен шешімін тап.



*Шешімі.* САТЖ – ның кеңейтілген матрицасы келесі түрде болады:



*1 – қадам.*  1 – ші бағанда, абсолютті мәні бойынша ең үлкен элемент 4 – ші жолда орналасқан болғандықтан, матрицаның төртінші жолын  сандарына көбейтіп, сәйкес бірінші, екінші және үшінші жолдарға қосамыз. Осының арқасында бірінші, екінші және үшінші жолдардың, бірінші бағанында элементтері нөлге тең болады, басқаша айтқанда төртінші жолдан басқа теңдеулерде  айнымалысын жоямыз.



*2 – қадам.* 2 – ші бағанда, абсолютті мәні бойынша ең үлкен элемент 4 – ші жолда орналасқан. Алайда, 4 – ші жол бізде алдыңғы қадамдарда жетекші жол болғандықтан, оны ала алмаймыз, осы себептен 2 – ші бағанда ең үлкен элемент ретінде 2 – ші жолда тұрған элементті айтамыз. Екінші жолды  сандарына көбейтіп, сәйкесінше бірінші және үшінші жолға қосамыз. Осы арифметикалық түрлендірулердің арқасында бірінші және үшінші жолдың екінші баған элементтері нөлге тең болады, басқаша айтқанда бірінші және үшінші теңдеуде  айнымалысын жоямыз.



*3 – қадам.* Бірінші және үшінші жолдар ішінен 3 – ші бағанында, абсолютті мәні бойынша ең үлкен элемент 1 – ші жолда орналасқан. Бірінші жолды  сандарына көбейтіп, үшінші жолға қосамыз. Осы арифметикалық түрлендірулердің арқасында үшінші жолдың үшінші баған элементі нөлге тең болады, басқаша айтқанда үшінші теңдеуде  айнымалысын жоямыз.

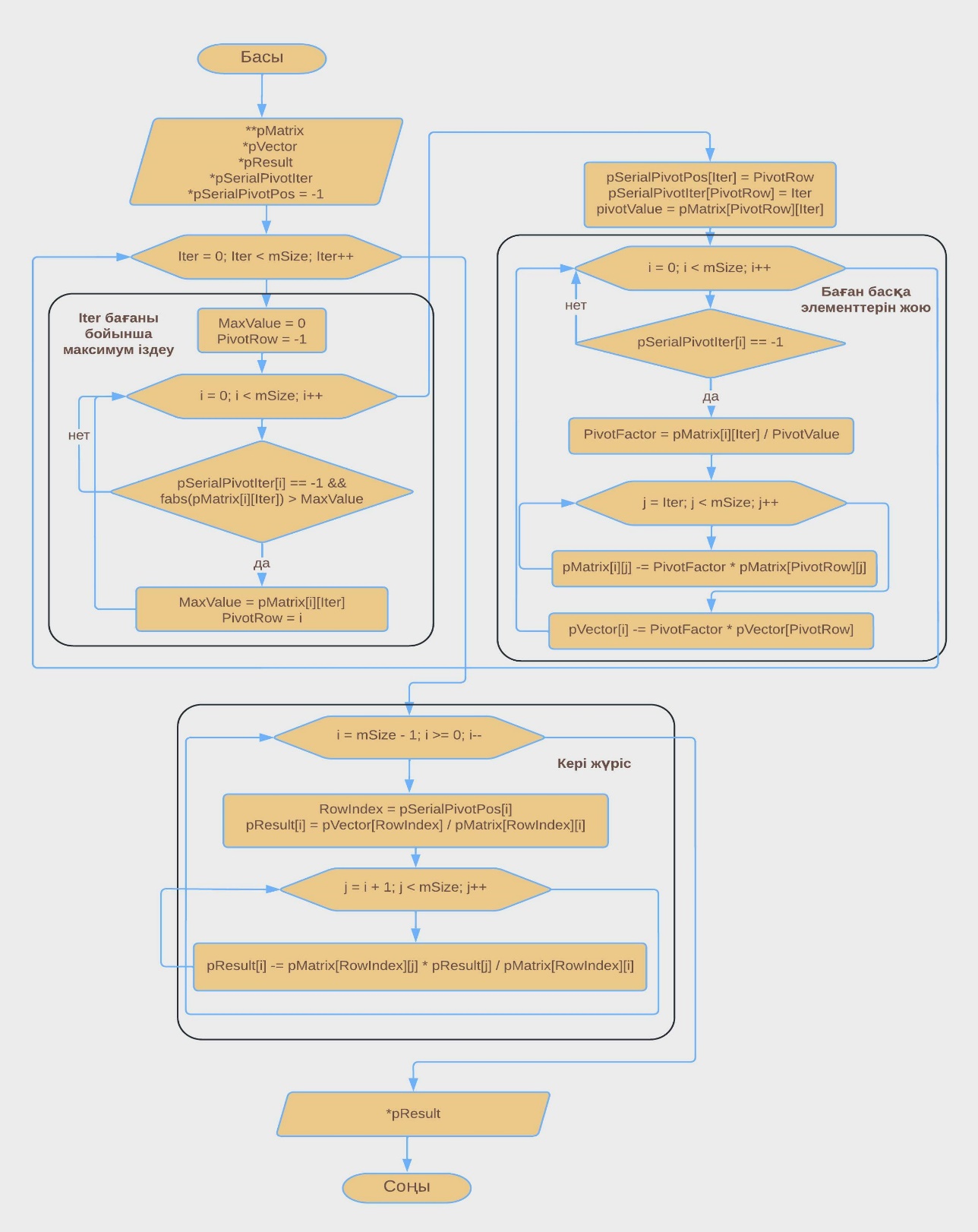


Осымен тура жүріс кезеңі аяқталады. Соңғы шыққан матрица түрін жүйе түріне аударып жазсақ. Келесі жүйені аламыз:



Соңғы жүйені шешу үшін тура жүріске қарсы жүріс жасаймыз. Яғни бірінші үшінші теңдеуді шешеміз, сол кезде біз  шешімін аламыз. Осы шешімді бірінші теңдеуге қойып,  есептеп аламыз; ал екінші теңдеуге қойып ; ал төртінші теңдеуге қойып . Яғни, теңдеу шешімі .

Енді Гаусс әдісінің баған бойынша жетекші элементті таңдау алгоритмін блок-схемасын құрастырайық. (2 – сурет)



2 – сурет. Гаусс әдісінің жетекші элементті таңдауы алгоритмі блок – схемасы

## **Жетекші элементті таңдау алгоритмін бағдарламалау**

Енді 2 – суретте көрсетілген блок – схема бойынша С++ тілінде бағдарлама жазамыз. Бағдарламамызды GaussSerial класс құрудан бастаймыз. Бұл класс бірнеше әдістен тұрады. Әдістер:

* Класстың конструкторы;
* Максимумды іздеу әдісі;
* Баған басқа элементтерін жою әдісі;
* Кері жүріс – айнымалыларды есептеу әдісі;
* Көрсетілген әдістерді ретімен орындау әдісі.

Осы әдістерге тоқталсақ.

*Класс конструкторы*. Есептеу орындалу барысында екі ақпаратты, әр жолдың жетекші болғаны және қай итерацияда қай жол жетекші болғанын сақтайтын екі массив қажет. pSerialPivotIter массиві тура жүріс кезінде алдыңғы итерацияларда жетекші болған жолдарды арналған. Басында бұл массивтің барлық элементі -1 тең болады. pSerialPivotPos массиві кері жүріс кезінде теңдеулерді шешу ретін анықтауға қажет. Бұл әдістің коды келесідей болады:

pSerialPivotIter = new int [mSize];

pSerialPivotPos = new int [mSize];

for (int j = 0; j < mSize; j++) {

pSerialPivotIter[j] = -1;

}

*Максимумды іздеу әдісі.* Бұл әдісте біз баған бойынша барлық жолды циклмен өтіп модулі бойынша үлкен және pSerialPivotIter массивіне жол индексі -1 – ге тең жолдарды іздейді. Әдіс ішінде екі айнымалы қолданылады. MaxVal – цикл бағандағы модулі бойынша үлкен элементтің мәні, PivotRow – цикл бағандағы модулі бойынша үлкен элемент орналасқан жолдың индексі. Бұл әдістің коды келесідей болады:

double MaxVal = 0;

int PivotRow = -1;

for (int i = 0; i < mSize; i++) {

if ((pSerialPivotIter[i] == -1) && (fabs(pMatrix[i][Iteration]) > MaxVal)) {

PivotRow = i;

MaxVal = fabs(pMatrix[i][Iteration]);

}

}

return PivotRow;

*Баған басқа элементтерін жою әдісі*. Бұл әдісте біз алдыңғы итерацияларда жетекші болмаған жолдардың баған элементін жоямыз, нөлге айныдырамыз. Сондай – ақ, жойылатын бағаннан кейінгі тұрған элементтерге де және бос мүшелер векторына да арифметикалық түрлендіруді орындаймыз. PivotValue – баған бойынша үлкен мәні, PivotFactor алгебралық түрлендіру үшін көбейту мәні. Жоюды орындау кезінде, алдыңғы жоюлар орындалған кездегі жетекші болған жолдарға алгебралық түрлендірулер орындалмауы керек. Оны қадағалау үшін pSerialPivotIter массивіндегі сәйкес жолдың мәні -1 – ге тең болмауы керек. Бұл әдістің коды келесідей болады:

Int i, j;

Double; PivotValue = pMatrix[PivotRow][Iteration];

for (i = 0; i < mSize; i++) {

if (pSerialPivotIter[i] == -1) {

double PivotFactor = pMatrix[i][Iteration] / PivotValue;

for (j = Iteration; j < mSize; j++) {

pMatrix[i][j] -= PivotFactor \* pMatrix[PivotRow][j];

}

pVector[i] -= PivotFactor \* pVector[PivotRow];

}

}

*Кері жүріс – айнымалыларды есептеу әдісі.* Бұл әдісте жетекші болу ретімен жазылған pSerialPivotPos массивіне соңынан басына қарай жүре отырып, айнымалыларды есептеп pResult массивіне жазып отырамыз. Бұл әдістің коды келесідей болады:

Int j, k, Row;

for (j = mSize - 1; j >= 0; j--) {

Row = pSerialPivotPos[j];

pResult[j] = pVector[Row] / pMatrix[Row][j];

for (k = j + 1; k < mSize; k++) {

pResult[j] - = pMatrix[Row][k] \* pResult[k] / pMatrix[Row][j];

}

}

*Көрсетілген әдістерді ретімен орындау әдісі.* Бұл әдіс – есептеуді бастаушы әдіс. Мұнда әдістерді орындау реті көрсетіледі. Бағандарды кезегімен жою үшін итерациямен орындаймыз. Бұл әдістің коды келесідей болады:

for (int Iteration = 0; Iteration < mSize; Iteration ++) {

int PivotRow = findPivotRow(pMatrix, Iteration);

pSerialPivotIter[PivotRow] = Iteration;

pSerialPivotPos[Iteration] = PivotRow;

serialColumnElimination(pMatrix, pVector, PivotRow, Iteration);

}

Осы кодтардың бәрін орындау үшін екі өлшемді \*\*pMatrix және бос мүшелер векторы \*pVector енгізіледі. Шешімдер векторы \*pResult - ке жазылады.

## **Жетекші элементті таңдау алгоритмін параллелдеу**

OpenMP кітапханасы математикалық есептеулерде жиі пайдаланылады, өйткені бағдарламаның жеке процедуралар мен алгоритмдердің параллелизациясы өте жылдам және көп қиындықсыз параллелдеуге мүмкіндік береді. Оларға параллель сұрыптау, матрицаны көбейту және сызықтық теңдеулер жүйесін шешу алгоритмдері жатады. Абстракцияның бұл деңгейінде OpenMP сияқты параллельді бағдарламалау технологиясын пайдалану ыңғайлы.

OpenMP (Open Multi-Processing) – көп процессорлы жүйелерде көп ағынды қосымшаларды бағдарламалауға арналған компилятор директивалары, кітапхана процедуралары және орта айнымалы мәндерінің жиынтығы. OpenMP тармақтарды біріктіру параллельді орындау үлгісін пайдаланады. OpenMP бағдарламасы бастапқы ағын деп аталатын орындаудың жалғыз ағыны ретінде басталады. Жіп параллельді құрылымды кездестіргенде, ол өзінен және бірқатар қосымша ағындардан тұратын жаңа ағындар тобын жасайды және жаңа топтың көшбасшысы болады. Жаңа топтың барлық мүшелері (негізгі ағынды қоса) параллельді құрылымда кодты орындайды. Параллельді конструкцияның соңында жасырын тосқауыл бар. Параллельді құрастырудан кейін пайдаланушы кодының орындалуы тек негізгі ағында жалғасады. Басқа параллель аймақтарды параллель аймаққа салуға болады.

«Үлкен параллелизация» идеясының арқасында OpenMP үлкен параллельді циклдері бар (матрицаны матрицаға көбейту, матрицаны векторға көбейту және т.с.с.) есептеу бағдарламаларын қиындықсыз параллельдеудегісі келетін бағдарламашылар үшін өте ыңғайлы. Бағдарламашы жаңа параллельді бағдарламаны құрастырмайды, тек қана құрастырылған бағдарлама алгоритміне OpenMP директиваларын дәйекті түрде қосылады.

Параллельді алгоритмдерді енгізу міндеті өте күрделі, сондықтан деректерді параллель өңдеуді жүзеге асыру құрылғысына кірмей текшелерден бағдарламаларды құруға мүмкіндік беретін параллель алгоритмдердің жеткілікті үлкен саны бар.

Көрсетілген артықшылықтарды ескере отырып, алгоритмді параллелдеуге OpenMP кітапханасы қолданылады. Сондай-ақ, сынақтарды бірнеше ағындар санымен қойылатынын ескеріп, параллелдеу барысында ағын санын оңай басқаратын кілтсөздер қосуымыз керек.

Бағдарламамызды GaussParallel класс құрудан бастаймыз. Бұл класс бірнеше әдістен тұрады. Әдістер:

* Класстың конструкторы;
* Максимумды іздеу әдісі;
* Баған басқа элементтерін жою әдісі;
* Кері жүріс – айнымалыларды есептеу әдісі;
* Көрсетілген әдістерді ретімен орындау әдісі.

Көрсетілген әдістердің бәрін алдыңғы бөлімде жазылған GaussSerial классынан аламыз. Осыдан кейін параллелдеуге келетін бөліктерге параллелдеу кілттерін қойып параллелдеу алгоритмін бастаймыз. Енді қай әдісте, қай бөлікті параллелдейтінімізді нақтырақ көрсетейік.

*Класс конструкторы*. Бұл әдісте pSerialPivotIter -1 – ге меншіктеу қайталау операциясы бар. Осы циклды параллелдеу үшін OpenMP – дің #pragma omp parallel for кілтсөзін циклдің алдына қосып жазамыз. Сынақтарды бірнеше ағындар санымен қойылатынын ескеріп, параллелдеу барысында ағын санын оңай басқаратын кілтсөзді осы классқа қосу керек. OpenMP ағындар санын басқарудың оңай әдісін ұсынады. omp\_set\_num\_threads(threads\_count) функциясы жақша ішіне натурал сан енгізіледі. Біздің жағдайда бұл айнымалылар {2, 4, 6, 8, 10, 12}. Бұл әдістің коды келесідей болады:

omp\_set\_num\_threads(threads\_count);

pSerialPivotIter = new int [mSize];

pSerialPivotPos = new int [mSize];

#pragma omp parallel for

for (int j = 0; j < mSize; j++) {

pSerialPivotIter[j] = -1;

}

*Максимумды іздеу әдісі.* Бұл әдісте максимумды іздеу процессін бірнеше ағынға бөлсек болады. Әр ағын өзі алған бөліктен максимумды тауып, басқа ағындар да тауып болғаннан кейін бір-бірімен салыстырады. Енді осы ойымызды іске асыру үшін әр ағын ішінде MaxVal және PivotRow айнымалыларын құрамыз. Осы екі айнымалыны TThreadPivotRow деген тип құрастырып, осының ішіне салып қояйық. Содан кейін әр ағын басталғанда өздері үшін осы типтегі ThreadPivotRow айнымалысын жасап алсын, OpenMP – дің бөліктерді параллелдеу #pragma omp parallel кілтсөздерін қолданамыз. Осы кілтсөзден кейін біз жүйелік жақша ашып ішінде ThreadPivotRow айнымалысын жасаймыз. Ары қарай максимумды іздеу циклін параллелдейік, сол кезде бір жол бір ағында ғана қаралады. Циклді параллелдеу үшін #pragma omp for кілтсөзін қолданамыз. Осы цикл аяқталған соң, біз әр ағыннан шыққан максимумды басқа ағындардан табылған максимумдер мен салыстырып, барлық ағын арасынан максимумды табамыз. Оны орындау үшін #pragma omp critical кілтсөзін қолданамыз. Бұл әдістің коды келесідей болады:

double MaxVal = 0;

int PivotRow = -1;

int i;

#pragma omp parallel

{

TThreadPivotRow ThreadPivotRow;

ThreadPivotRow. MaxVal = 0;

ThreadPivotRow. PivotRow = -1;

#pragma omp for

for (i = 0; i < mSize; i++) {

if ((pSerialPivotIter[i] == -1) && (fabs(pMatrix[i][Iteration]) > ThreadPivotRow. MaxVal)) {

ThreadPivotRow. PivotRow = i;

ThreadPivotRow. MaxVal = fabs(pMatrix[i][Iteration]);

}

}

#pragma omp critical

{

if (ThreadPivotRow. MaxVal > MaxVal) {

MaxVal = ThreadPivotRow. MaxVal;

PivotRow = ThreadPivotRow. PivotRow;

}

}

}

return PivotRow;

*Баған басқа элементтерін жою әдісі*. Бұл әдісте жолдарға алгебралық түрлендіруді орындау процессін параллелдеуге болады. Егер біз әр жолды әр ағында орындайтын болсақ, есептеуге кететін уақыт азаяды. Демек әр жол үшін өзінің көбейтілу мәні бар болғандықтан, әр ағында да тек өзіне ғана тиісті мәні болу керек. Мұндай мүмкіндікті іске асыруға бізге OpenMP – дің private (PivotFactor) кілтсөзі көмектеседі. Кілтсөзді әр ағында PivotFactor айнымалысы әртүрлі мәнге ие болатындықтан, сол айнымалыны жақшаға ішіне жаздық. Келесі ескеретін жайт, алдыңғы итерацияларда жетекші болған жолдарға алгебралық түрлендірулер орындауға болмайды. Осы жайтты параллелдеу кезінде тиімді пайдалану үшін OpenMP – дің schedule (dynamic, 1) кілтсөзін қолданамыз. Кілтсөз циклдің келесі орындалуын бос ағынға салу үшін, сондай – ақ тек қана бір ғана циклді салу үшін жазылды. Негізгі мақсат – егер ағынға берілген жол алдыңғы итерациялардың бірінде жетекші болған болса, онда ағын осы циклда есептеуін жалғастыра алмайды, басқаша айтқанда босайды сол кезде келесі циклді есептеуді береміз. Бұл әдістің коды келесідей болады:

Int i, j;

Double PivotFactor, PivotValue = pMatrix[PivotRow][Iteration];

#pragma omp parallel for private (PivotFactor) schedule(dynamic, 1)

for (i = 0; i < mSize; i++) {

if (pSerialPivotIter[i] == -1) {

PivotFactor = pMatrix[i][Iteration] / PivotValue;

for (j = Iteration; j < mSize; j++) {

pMatrix[i][j] -= PivotFactor \* pMatrix[PivotRow][j];

}

pVector[i] -= PivotFactor \* pVector[PivotRow];

}

}

*Кері жүріс – айнымалыларды есептеу әдісі.* Бұл әдісте екі цикл бар, алайда сыртқы цикл параллелдеу тиімсіз. Себебі сыртқы цикл алдыңғы цикл есептеулері бітпей басталса онда қате шешім аламыз. Сондықтан біз тек ішкі циклді ғана параллелдейміз. Параллелдеу кезінде бізде tmp айнымалысы ағындар есептеуін аяқтағаннан кейін бір айнымалыға біріктірілуі керек, басқаша айтқанда жойылуы керек. Мұндай мүмкіндікті іске асыруға OpenMP – дің reduction (-:tmp) кілтсөзі көмектеседі. Reduction кілтсөзінің жақша ішінде бірінші параметрі қалай жойылу керектігін көрсетеді, біздің жағдайда азайтылу керек. Ал екінші параметрі ретінде қай айнымалы бойынша жойылу орындалу керектігі көрсетіледі. Бұл әдістің коды келесідей болады:

Int j, k, Row;

for (j = mSize - 1; j >= 0; j--) {

Row = pSerialPivotPos[j];

Double tmp = pVector [Row] / pMatrix [Row][j];

#pragma omp parallel for reduction (-:tmp)

for (k = j + 1; k < mSize; k++) {

tmp - = pMatrix[Row][k] \* pResult[k] / pMatrix[Row][j];

}

pResult[j] = tmp;

}

*Көрсетілген әдістерді ретімен орындау әдісі.* Бұл әдіс – есептеуді бастаушы әдіс. Мұнда әдістерді орындау реті көрсетіледі, сондықтан мұндай ағындарға бөлу тиімсіз болу табылады. Демек, бұл әдіс коды өзгертілмейді.

Енді бұл класста жаңа айнымалы типін құрастыруымыз керек. Типті TThreadPivotRow атаумен болсын. Ал бұл типтің ішінде екі айнымалы болады, біріншісі double типті MaxVal айнымалысы, ал екіншісі int типті PivotRow айнымалысы. Типті құрастыру коды келесідей болады:

typedef struct {

int PivotRow;

double MaxVal;

} TThreadPivotRow;

Осы кодтардың бәрін орындау үшін екі өлшемді \*\*pMatrix және бос мүшелер векторы \*pVector енгізіледі. Шешімдер векторы \*pResult – ке жазылады.

# **БІРІКТІРІЛГЕН ГРАДИЕНТТЕР ӘДІСІ**

## **Сипаттамасы және модификациялары**

Біріктірілген градиенттер (CG) әдісі САТЖ шешімін алуға арналған итерациялық әдіс болып табылады. Әдістің негізгі артықшылығы - ол қадамдардың шектеулі санымен квадраттық оңтайландыру есебін шешеді. Сондықтан алдымен квадраттық функцияны оңтайландырудың біріктірілген градиент әдісі сипатталады, итерациялық формулалар шығарылады және жинақтылық жылдамдығының бағалаулары беріледі. Осыдан кейін ерікті функционалдылықты оңтайландыру үшін біріктірілген градиенттер әдісі қалай жалпыланғаны көрсетіледі, әдістің әртүрлі нұсқалары қарастырылады және конвергенция талқыланады.

«Біріктірілген градиент әдісі» термині мағынасыз тіркестердің үйреншікті болып қабылдануының және ешқандай таң қалдырмайтынының бір мысалы болып табылады. Мәселе мынада, практикалық қызығушылық тудырмайтын нақты жағдайды қоспағанда, градиенттер біріктірілмейді және конъюгаттық бағыттар градиенттермен ешқандай байланысы жоқ. Әдістің атауы шартсыз экстремумды табудың бұл әдісі мақсат функциясының градиенті және конъюгаттық бағыттар ұғымдарын біріктіретінін көрсетеді.

Төменде қолданылатын белгілер туралы бірнеше сөз.

Екі вектордың скаляр көбейтіндісі , деп жазылады және скалярлардың қосындысын көрсетеді: . Және  екенін ескеруіміз керек. Егер  пен  ортогональды болса, онда . Яғни,  және  сияқты 1x1 матрицасына түрлендіретін өрнектер скалярлар ретінде қарастырылады.

Бастапқыда біріктірілген градиент әдісі (3) түрдегі сызықтық алгебралық теңдеулер жүйесін шешу үшін жасалды.  матрицасы берілген, квадраттық, симметриялы, оң анықталған матрица болып табылсын. Бұл жүйені шешу сәйкес квадраттық форманың минимумын табуға тең.

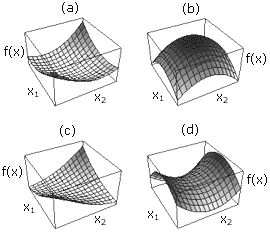
Квадраттық пішін жай ғана скаляр, келесі түрдегі кейбір  векторының квадраттық функциясы:

 (9)

 сызықты түрлендіру матрицасы мен  скаляр функциясы арасында мұндай байланыстың болуы сызықтық алгебраның кейбір формулаларын интуитивті сызбалармен суреттеуге мүмкіндік береді. Мысалы, кез келген нөлге тең емес  векторы үшін мыналар дұрыс болса,  матрицасы оң-анықталған деп аталады:

 (10)

3-суретте матрицалар үшін квадраттық пішіндердің сәйкесінше қалай көрінетіні көрсетілген.оң анықталған матрица (а), теріс анықталған матрица (b), оң анықталмаған матрица (c), анықталмаған матрица (d)

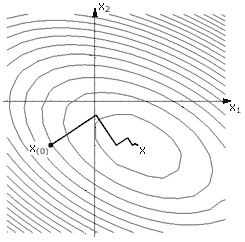


3 – сурет. Әртүрлі анықталған матрицалар үшін квадраттық пішіндері

Яғни, егер  матрицасы оң-анықталған болса, онда (3) теңдеулер жүйесін шешудің орнына оның квадраттық функциясының минимумын табуға болады. Сонымен қатар, біріктірілген градиент әдісі мұны  немесе одан аз қадамдармен орындайды, мұндағы  - белгісіз  векторының өлшемі. Оның минимум нүктесіне жақын орналасқан кез келген тегіс функция квадратпен жақсы жуықталатындықтан, квадраттық емес функцияларды да минимизациялау үшін дәл осындай әдісті қолдануға болады. Бұл жағдайда әдіс ақырлы болуды тоқтатады да итерациялы болады.

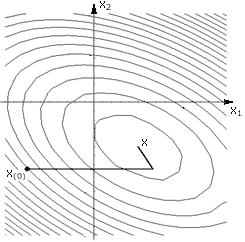
Біріктірілген градиент әдісін қарастыра отырып, функцияның экстремумын табудың қарапайым әдісінен – ең тік түсу әдісінен бастаған жөн. 4-суретте ең тік түсу әдісін қолдану арқылы минималды нүктеге дейінгі қозғалыс траекториясы көрсетілген. Бұл әдістің мәні:

* бастапқы  нүктесінде градиент есептеледі, ал қозғалыс мақсат функциясы төмендегенше антиградиент бағытында жүргізіледі;
* функцияның кемуі тоқтаған жерде градиент қайтадан есептеледі және төмендеу жаңа бағытта жалғасады;
* процесс ең төменгі нүктеге жеткенше қайталанады.



4 – сурет. Ең тік түсу әдісімен минималды нүктеге дейінгі қозғалыс траекториясы.

Бұл жағдайда қозғалыстың әрбір жаңа бағыты алдыңғысына ортогональды болады. Қозғалыстың жаңа бағытын таңдаудың ақылды жолы бар емес пе? Бар және ол біріктірілген бағыттар әдістері деп аталады. Ал біріктірілген градиенттер әдісі тек біріктірілген бағыттар әдістерінің тобына жатады. 5-суретте біріктірілген градиент әдісін қолдану кезінде минималды нүктеге дейінгі қозғалыс траекториясы көрсетілген.



5 – сурет. Біріктірілген градиенттер әдісін қолдану кезінде минималды нүктеге дейінгі қозғалыс траекториясы

Біріктірілген анықтамасы келесідей тұжырымдалған: екі  және  векторлары  - біріктірілген (немесе  матрицасына қатысты біріктірілген) немесе  - ортогональды деп аталады, егер  пен  скаляр көбейтіндісі нөлге тең болса, яғни:

## **Шешу алгоритмін бағдарламалау**

## **Шешу алгоритмін параллелдеу**

# **ЕСЕПТЕУ ЭКСПЕРИМЕНТТЕРІ ЖӘНЕ ТАЛДАУ**

## **Эксперименттің жүргізілу шарттары**

Экспериментте матрицаларды және бос мүше векторларын кездейсоқ сандармен толтырамыз. Есептеудің дұрыстығын автоматты түрде тексеруіміз керек. Шешімнің дұрыстығы кезінде, шешімнің қателігі  кіші болуы қадағаланады.

Кіріс ақпараттары үшін бірнеше параметрлерді анықтап аламыз. Есептеу эксперименттерінің параметрлері келесідей:

* Матрица өлшемі;
* Матрица типтері;
* Ағындар саны.

*Матрица өлшемі*. Есептеу эксперименттерді үлкен өлшемді матрицалар үшін жүргізіледі. Себебі кіші өлшемдегі матрица үшін параллелдеу технологиясы онша көп өсім бермейді, керісінше жадыдан ақпаратты оқып жазу уақытының кесірінен параллелденген алгоритм шешім алуға көп уақыт жұмсауы мүмкін. Эксперимент кезінде матрица өлшемдері 100х100, 500х500, 1000х1000, 1500х1500, 2000х2000, 2500х2500, 3000х3000 болады.

*Матрица типтері*. Практикада, өндірісте есептеулерде қолданылатын матрица түрлері көп. Оның ішінде танымал үш типті матрицаны қолданамыз.

Біріншісі диагоналды матрица деп аталады. Берілген матрицаның диагоналынан басқа элементтері нөлге тең болады. Мысалы,



Екіншісі үш диагоналды симметриялы матрица деп аталады. Берілген матрицаның үш диагоналынан басқа элементтері нөлге тең болады. Мысалы,



Үшіншісі симметриялы толық матрица деп аталады. Берілген матрицаның ешқандай элементі нөлге тең емес болады. Мысалы,



Эксперименттердің бәрі жалғыз ноутбукта орындалады. Ноутбуктың сипаттамасы келесідей:

* Microsoft Windows 10 Professional;
* Компилятор Microsoft Visual C++;
* HexaCore Intel Core i7-9750H, 3000 MHz, 12 Multi CPU (ағын);
* 15,9 GB RAM.

Біздің ноутбук 12 ағынды қолдайтын болғандықтан, эксперимент кезінде максимум ағын саны 12 болады. Сондай-ақ, аралық ағындар санын да экспериментте қолданамыз. Нақтырақ айтсақ, экспериментте берілген жүйені алдымен сызықты алгоритммен, содан кейін 2 ағынмен параллелді шешеміз, содан кейін 4, 6, 8, 10, ал ең соңында 12 ағынмен шешеміз. Есептеуге кеткен уақыттарды талдау оңай болу үшін барлық кеткен уақыттарды \*.csv типте сақтаймыз.

## **Эксперименттер қоюға қажетті қосымша класстар жайлы**

Эксперимент ыңғайлы, әрі қатесіз өткізу үшін бағдарламамызға бірнеше класстар жасау керек. Олар:

* Кіріс ақпараттарын кездейсоқ генерациялау классы;
* Шешімді тексеру, берілген матрицаны тексеру және т.б., көмекші матрица классы;
* \*.csv файлмен жұмыс істеу классы;
* Және соңғысы, басқарушы класс.

*Ақпараттарды генерациялау классы.* Класс атын dataGen деп атайық. Класс ішінде үш типтегі матрицаны, бос мүше векторын кездейсоқ сандармен толтыратын үш әдіс болсын. Енді осы үш әдіске ретімен тоқталсақ.

Бірінші экспериментте, диагональ типтегі матрица болады. Оның қандай болатын алдыңғы бөлімде айтып өттік. Алдымен кездейсоқ сандарды қайтаратын srand(unsigned(clock())) функциясын шақырамыз. Кездейсоқ сандарды алу функциясын инициализация жасағаннан кейін, екі цикл көмегімен \*\*pMatrix, \*pVector кездейсоқ мәндерді rand() көмегімен толтырамыз. Матрица толтыру кезінде тек қана диагональ элементтеріне (i == j) ғана мән береміз, басқа кезде нөлге меншіктейміз. Бұл әдіс коды келесідей:

srand(unsigned(clock()));

for (int i = 0; i < Size; i++) {

pVector[i] = (double)rand() / double(Size);

for (int j = 0; j < Size; j++) {

if (i == j) {

pMatrix[i][j] = (double) rand() / double(Size);

}

else {

pMatrix[i][j] = 0;

}

}

}

Екінші экспериментте, үш диагональді типтегі матрица болады. Оның қандай болатын алдыңғы бөлімде айтып өттік. Алдымен кездейсоқ сандарды қайтаратын srand(unsigned(clock())) функциясын шақырамыз. Кездейсоқ сандарды алу функциясын инициализация жасағаннан кейін, екі цикл көмегімен \*\*pMatrix, \*pVector кездейсоқ мәндерді rand() көмегімен толтырамыз. Матрица толтыру кезінде тек қана диагональден 1 қадам ары жатқан элементтерге (fabs(j - i) < 2) ғана мән береміз, басқа кезде нөлге меншіктейміз. Бұл әдіс коды келесідей:

srand(unsigned(clock()));

for (int i = 0; i < Size; i++) {

pVector[i] = (double)rand() / double(Size);

for (int j = 0; j < Size; j++) {

if (fabs(j - i) < 2){

pMatrix[i][j] = pMatrix[j][i] = (double) rand() / double(Size);

}

else {

pMatrix[i][j] = 0;

}

}

}

Үшінші экспериментте, симметриялы матрица болады. Оның қандай болатын алдыңғы бөлімде айтып өттік. Алдымен кездейсоқ сандарды қайтаратын srand(unsigned(clock())) функциясын шақырамыз. Кездейсоқ сандарды алу функциясын инициализация жасағаннан кейін, екі цикл көмегімен \*\*pMatrix, \*pVector кездейсоқ мәндерді rand() көмегімен толтырамыз. Мұнда алдыңғы эксперементтердегідей шарт болмайды. Бұл әдіс коды келесідей:

srand(unsigned(clock()));

for (int i = 0; i < Size; i++) {

pVector[i] = (double)rand() / double(Size);

for (int j = 0; j < Size; j++) {

pMatrix[i][j] = pMatrix[j][i] = (double) rand() / double(Size);

}

}

*Көмекші матрица классы.* Класс атын matrixHelpers деп атайық. Класс ішінде ең қолданбалы шешімді тексеру, матрица мен векторларды бастапқы қалыпқа әкелу, және матрицаны не векторды экранға шығару әдістері бар. Енді осы әдістерге ретімен тоқталсақ.

Шешімді тексеру әдісінде \*\*OriginalA матрицасын \*pResult векторын көбейтеміз. Көбейтуді жылдамдату үшін OpenMP көмегімен параллелдейміз. Параллелдеу кезінде reduction(+:tmp) кілтсөзін қолданамыз, себебі бізде tmp мәні барлық ағын бойынша қосылып шығу керек. Көбейтіндіні \*pRightPartVector векторына меншіктейміз. Көбейтіп біткеннен кейін \*pRightPartVector векторынан \*pVector бос мүше векторынан азайтамыз, ол айырма абсолютті мәні бойынша 0,01 дәлдігінен кіші болмау керек. Басқаша айтқанда fabs(pRightPartVector[i] - pVector[i]) > Accuracy осы шарт орындалмағаны жөн. Егер шарт орындалса equal мәнін өсіреміз. Осы азайту операциясын да OpenMP кітапханасының reduction(+:equal) кілтсөзінің көмегімен параллелдеп есептейміз. Осы азайту операциясы аяқталғаннан кейін equal мәні 0 – ден үлкен болса, шешім біз қалаған дәлдікпен шешілмеді дегенді білдіреді. Енді осы әдістің кодын келтірейік:

double\* pRightPartVector;

int equal = 0, i, j;

double Accuracy = 0.01f;

pRightPartVector = new double[Size];

for (i = 0; i < Size; i++) {

double tmp = 0;

#pragma omp parallel for reduction(+:tmp)

for (j = 0; j < Size; j++) {

tmp += pMatrix[i][j] \* pResult[j];

}

pRightPartVector[i] = tmp;

}

#pragma omp parallel for reduction(+:equal)

for (i = 0; i < Size; i++) {

if (fabs(pRightPartVector[i] - pVector[i]) > Accuracy) {

equal++;

}

}

if (equal > 0) {

printf(" Wrong.");

}

else {

//printf(" Сorrect.");

}

Матрицаны және векторларды бастапқы қалыпқа алып келу әдісі. Біздің бір экспериментте бір матрицаны және бос мүше векторын бірнеше рет шешетін болғандықтан, матрица мен векторларды әр есептеу біте бастапқы мәндерге алып келіп отыру керек. Бұл операцияны екі циклда орындаймыз. Параллелдеу операцияны жылдамдатуға мүмкіндік береді. Әдіс коды келесідей:

for (int i = 0; i < Size; i++) {

B[i] = OriginalB[i];

X[i] = 0;

#pragma omp parallel for

for (int j = 0; j < Size; j++)

A[i][j] = OriginalA[i][j];

}

Матрицаны немесе векторды экранға шығару екі немесе бір цикл арқылы, printf("%.4f ", matrix[i]) көмекші функциясы көмегімен іске асырамыз. Бұл әдістерде параллелдеуді қолданбаймыз. Егер қолданар болсақ, ағындар бір – бірімен экранға шығару бойынша жарысуы мүмкін. Ал бұл бізге қате тұжырым жасатуы мүмкін.

*Файлмен жұмыс классы.* Баған аттарын және алгоритмдердің есептеуге кеткен уақыттарын жазатын екі әдіс құрастырамыз. Файлдармен fstream кітапханасының көмегімен жұмыс істейміз.

Баған тақырыбын жазу әдісінде көрсетілген бағанда есепті қандай әдіспен шешілгені жазылады. Тақырыптар динамикалық түрде болу үшін экспериментте жүргізілетін ағындар саны массивін осы әдіске жібереміз. Содан кейін осы массивті цикл арқылы өтіп, файлға жазып шығамыз. Ал файл аты result қосылған матрица типінің нөмірі болады. Бұл әдіс коды келесідей болады:

ofstream myfile;

string filename = "result";

filename += to\_string(experiment\_number);

filename +=".csv";

myfile.open(filename);

myfile << "Размер;Гаусс линейно;";

for (int i = 0; i < size; i++) {

myfile << "Гаусс параллель(" << threads\_array[i] << ");";

}

myfile << "CG линейно;";

for (int i = 0; i < size; i++) {

myfile << "CG параллель(" << threads\_array[i] << ");";

}

myfile.close();

Есептеуге кету уақытын жазу әдісінде тақырыптар бойынша керек бағанға жазады. Файлдағы ақпаратты өшірмей астына жаза беру үшін файлды ашқан кезде ios::app атрибутын енгіземіз. Бұл әдіс коды келесідей болады:

ofstream myfile;

string filename = "result";

filename += to\_string(experiment\_number);

filename += ".csv";

myfile.open(filename,ios::app);

myfile << "\n" <<Size << "x" << Size << ";" << times;

myfile.close();

Және осы csv файлды Microsoft Excel бағдарламасында ашқан кезде уақытты айнымалысы double типте болу үшін, уақыттағы нүкте символын үтір символына ауыстыру керек. Оның коды келесідей:

size\_t start\_pos = 0;

while ((start\_pos = str.find(from, start\_pos)) != std::string::npos) {

str.replace(start\_pos, from.length(), to);

start\_pos += to.length();

}

*Басқарушы класс.* Экпериментті бастар алдында матрица типтер саны, ағындар санының массиві көрсетіледі. Файлдағы кестенің тақырыбын жазып шығамыз. Содан кейін матрица көлемінің өлшемін цикл арқылы меншіктеп отырамыз.

## **Эксперимент нәтижелерін алу және талдау**

Есептеу эксперименті алдыңғы бөлімде көрсетілген алгоритмдер мен блок схемалар бойынша C++ тілінде код жазудан басталады. Кодты біз Visual Studio бағдарламалау ортасында жазамыз.

# **ҚОРЫТЫНДЫ**

# **ҚОЛДАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ**

1. Ильин В. А., Позняк Э. Г. Линейная алгебра: Учебник для вузов. — T. 6. – М.: Физматлит, 2004. – 280 бет.
2. Кострикин А.И., Манин Ю.И. Линейная алгебра и геометрия. – М.: Наука, 1986. – 304 бет.
3. Стренг Г. Линейная алгебра и ее применения. – М.: Мир. – 1980. – 454 бет.
4. Ильин В.А., Ким Г.Д. Линейная алгебра и аналитическая геометрия. – М.: Проспект, 2007. – 400 бет.
5. Вержбицкий В. М. Основы численных методов. – М.: Высшая школа, 2009. – 840 бет.
6. Grcar Joseph F. How ordinary elimination became Gaussian elimination // Historia Mathematica. – 2011. – 38: Т. 2. – Б. 163-218.
7. Кремер Наум Шевелевич Высшая математика для экономистов. – М.: ЮНИТИ-ДАНА, 2010. – 3: Т. 1. – 479 бет.