Л.Н. Гумилев атындағы Еуразия ұлттық университеті

ӘӨЖ 519.61 Қолжазба құқығында

**ЖУСУПОВА ДИНАРА СЕРИКЖАНОВНА**

# Алгебралық теңдеулер жүйелерін шешуді параллельдеудің кейбір есептері

6D060100 – Математика

Философия докторы (PhD)
дәрежесін алу үшін дайындалған диссертация

Ғылыми жетекшісі
физика-математика ғылымдарының докторы,
Өтелбаев М.

Шетел ғылыми кеңесшісі
физика-математика ғылымдарының докторы
профессор,
Шкаликов А.А.
(М.В. Ломоносов ат. ММУ)

Республика Казахстан
Астана 2025

## МАЗМҰНЫ

## БЕЛГІЛЕУЛЕР МЕН ҚЫСҚАРТУЛАР

| | |
|---|---|
| $\acute{f}, \tilde{f}$ | – вектордың сәйкес нақты және жуықтап тәжірибе кезінде алынған мәндері |
| $\bar{A}, \tilde{A}$ | – матрицаның сәйкес нақты және жуықтап тәжірибе кезінде алынған мәндері |
| $J_\varepsilon(x)$ | – Тихонов функционалы |
| $R^n$ | – $n$ өлшемді евклидтік кеңістік |
| $\|\cdot\|$ | – вектордың $R^n$ - дегі нормасы |
| $\|\|\cdot\|\|$ | – матрицаның нормасы |
| $A^*$ | $-A$ операторына түйіндес оператор |
| $s_1(A), s_2(A), \dots, s_n(A)$ | – $A$ операторының сингулярлы сандары |
| $e_1, e_2, \dots, e_n$ | – $A$ операторының сингулярлы сандарына сәйкес меншікті векторлары |
| $\langle\cdot,\cdot\rangle$ | – екі вектордың скалярлық көбейтіндісі |

# КІРІСПЕ

**Тақырыптың өзектілігі.** XX ғасырдың 50 -ші жылдардан бастап, есептеу әдістері бірталай қарқынды дамуда. Бұл әсіресе сызықты алгебралық жүйелерді шешуге арналған жоғары деңгейдегі бағдарламалардың пайда болуымен сипатталады. Мысал ретінде EISPACK пен LINPACK бағдарламалар жинақтарының көптеген қолданбалы есептерде кең қолданылуы жоғары деңгейлі бағдарламалардың жазылуына үлес қосты. Аталған бағдарламалар жинақтарындағы бөлек модульдерді пайдалана отырып, ғалымдар мен инженерлер айналысқан мәселелеріне сәйкес күрделі бағдарламалық құралдарды құрастырады.

Үлкен сызықты алгебралық теңдеулер жүйелерін шешу мәселесі соңғы үлгідегі қуатты есептеу техниканың пайдалануы максималды тиімділігін әкелетін мәселелерге жатады. Кейбір жағдайда есептің параметрлері өте үлкен болып, бір процессорлы компьютер жадының сыйымдылығына немесе жұмсалатын уақытқа байланысты шешу мүмкін емес болғанда, есептеулерді жүргізуде параллель алгоритм құру қажеттілігі туындайды. Бұндай мәселелер ғылымның әр түрлі салаларында пайда болады: кванттық физика (элементар бөлшектердің физикасы, ядерлік физика, өрістің кванттық теориясы), молекулалар физикасы, кванттық химия, биология, экология, Жер туралы ғылымдарда (атмосфера физикасы, метеорология, климатология, мұхит физикасы), экономика және эконометрия (есептеу экономикасы, макроэкономика, тиімді басқару теориясы), математикалық лингвистика (дауысты тану, мәтінді талдау және автоматты аудару), информатика (мәліметтер базасын енгізу, бейнелерді тану), әлеуметтік ғылымдар, гидродинамика, газодинамика, медицина, фармацевтика және т. с. с. Параллель жүйелерді пайдалану арқылы бұрын өте көп уақытты жұмсайтын есептеулер бүгінгі күні белгілі бір шектеулі уақытты алатын болды. Тізбектелген алгоритмді көп процессорлы жүйеге қою көп жағдайда есептің шешілуін жеделдетуге әкелмейді. Бұл жағдайда жүйенің архитектуралық құрылысын ескеріп, арнайы параллель алгоритм құру жөн. Әсіресе, дискретті математиканың сандық қатарлар, комбинаторлық есептер, сызықты алгебра, тордың түйіндеріндегі мәндерді есептеу, графтарды өңдеу сияқты есептерді шешуде параллель алгоритмдер жиі қолданысқа ие. Қазіргі кезде ең қуатты суперкомпьютер Қытайдың Чанша қаласында орналасқан. Оның аталуы Tianhe-2 (National University of Defense Technology), оның қуаты - 33,862.7 TFlop/s немесе секундына 33,862.7 квадриллион үтірі жылжымалы болатын операцияларды орындай алады. Одан жылдамдығы жағынан төмен Titan (USA, DOE/SC/Oak Ridge National Laboratory), ол 17,590.0 TFlop/s немесе секундына 17,590.0 квадриллион операция орындайды. Суперкомпьютерлердің құрылысы мен жұмысы туралы негізгі мағлұматтар келесі мақалаларда кең қарастырылған [1-4]. Көппроцессорлы жүйелердің жұмыс істеу қағидалары [5-9].

Параллель сызықты алгебралық есептеулер әдебиеттерде кең қарастырылған. Параллель есептеулер әдістерінің алгоритмдері арнайы

(үшбұрыштық, үш диагоналды) матрицалар үшін [10-12] монографияларда зерттелген. Сызықты алгебралық теңдеулер жүйелерін шешуде параллель алгоритмдердің кейбір түрлері жалпы жағдай үшін [13] еңбегінде келтірілген. Сонымен қоса аталған әдебиетте осы түрдегі есептердің тиімділігінің талдауы және уақытты қысқарту көрсеткіштері зерттелген. Параллель алгоритмдерін бағдарлама жазу мәселелерін зерттеуде [14-17] пайдалы болады. Сызықты алгебралық теңдеулер жүйесін шешудің тура және итерациялық әдістері [18, 19] еңбектерде келтірілген.

**Жұмыстың мақсаты**. Сызықты алгебралық және сызықты емес алгебралық теңдеулер жүйелерін жуықтап шешу үрдісінің параллель есептеу әдістерін құру.

**Зерттеу оъектісі**. Сызықты алгебралық теңдеулер жүйелері, сызықты емес алгебралық теңдеулер жүйелері.

**Зерттеу әдістері**. Сызықты алгебралық теңдеулер жүйесін жуықтап шешу әдісін құрастыруда

$$Ax = f$$

теңдеуді жуықтап шешу мәселесі оған эквивалентті

$$J(x) = |Ax - f|^2$$

функционалды минимизациялау есебіне көшу әдісі пайдаланылған. Көп жағдайда

$$Ax = f$$

теңдеуінің нақты шешімін табудан гөрі

$$|Ax - f|$$

мәнін кішірейту тиімді болады. Осы ұстанымға сай $A$ матрицасы шенелген және қайтымды жағдай үшін жуықтап шешу вариациялық әдісі диссертациялық жұмыста келтірілген. Ал $A$ нашар шартталған (яғни қайтымды емес немесе меншікті мәндері өте аз шама болатын жағдайда) Тихоновтың $\varepsilon \geq 0$ саны үшін келесі функционалы

$$J_\varepsilon(x) = |Ax - f|^2 + \varepsilon|x|^2.$$

қолданылған. Бұл жағдайда егер $A$ матрицасының кері операторы болмаса, онда бірнеше шешімдердің ішінде нормасы ең аз болатын шешімді іздеу әдісі келтірілген.

**Ғылыми жаңалығы**. Диссертациялық жұмыста алынған ғылыми нәтижелердің жаңалығы алгебралық теңдеулер жүйелерін жуықтап шешу үрдісін параллель есептеу алгоритмдеріне сәйкес құрастыруы болып табылады.

**Алынған нәтижелердің теориялық және практикалық маңыздылығы**. Диссертациялық жұмыс нәтижелерінің теориялық және практикалық маңызы бар. Алынған нәтижелерді пайдаланып, сызықты алгебралық жүйелерді матрицасы қайтымды немесе нашар шартталған жағдайда параллель есептеуге болады, сонымен қатар сызықты емес алгебралық жүйені параллельдеудің де алгоритмі келтірілген.

**Алынған нәтижелердің талқылануы**. Диссертациялық жұмыстың негізгі нәтижелері келесі ғылыми конференциялар мен ғылыми семинарларда талқыланды:

1. Халықаралық "Ломоносов - 2011" атты ғылыми конференция (Астана, Қазақстан).
2. Халықаралық "Операторлардың спектрлік теориясы және оның қолданыстары" атты ғылыми конференция (13 - 16 маусым 2011, Уфа, Ресей).
3. Congress of the Turkic World Mathematical Society (TWMS) атты ғылыми симпозиум (1 - 3 шілде 2011, Баку, Әзірбайжан).
4. "Функционалдық талдау және оның қолданыстары" атты ғылыми жазғы мектеп (Нұр-Сұлтан, Қазақстан).
5. М. Өтелбаев, Р. Ойнаров, Е. Нұрсұлтанов, Қ. Оспанов, Н. Боқаев жетекшілігімен өтетін "Функционалдық талдау және оның қолданыстары" атты ғылыми семинар (Астана, Қазақстан).
6. Бүкіл әлемдік инженерлер конгрессі IAENG WCE - 2013 (3 - 5 шілде, 2013, Лондон, Ұлыбритания).
7. Помпажды аймағынан қорғау үшін жанармай шығынын азайту үшін газ компрессорлық станциясының жұмысын модельдеу. Конференция Совр. проблемы дифференциальных уравнений и их приложения, 23-25 қараша - 2023, Ташкент.
8. Minimizing compressor fuel cost on large natural gas pipeline transmission networks. AIP Conf. Proc. 9 October 2023.
9. Modeling the operation of a gas compressor station to minimize fuel costs for protection against surge zone // Procced. “APMAS-2024” conf. – 2024.

**Жарияланымдар**.

1. Mathematical modelling of the process of natural gas transportation via pipe networks using crossing branch method. Journal of Mathematics, Mechanics and Computer Science, 2024, 121(1), 12–26 (CiteScore2023=0.4, процентиль=19)
2. Modeling Gas Compressor Station Operation to Minimize Fuel Costs for Surge Zone Protection, International Journal of Rotating Machinery, vol. 2024, Article ID 5560308, 18 pages, 2024 (CiteScore2023=2.4, процентиль=50)
3. Criterion for the Boundedness and Compactness of a Class of Sets in L[0;1) // Differential Equations, 2019, Vol. 55, No. 9, pp. 1301-1304 (CiteScore2023=1.0, процентиль=48).

4. On a Method of Parallel Computation for Solving Linear Algebraic System with Ill-Conditioned Matrix // TWMS Journal of Pure and Applied Mathematics. - 2013. -V. 4. - No. 2. - p. 115 - 124.

5. On a Method of Finding Approximate Solutions of Ill-conditioned Algebraic Systems and Parallel Computation //Eurasian Mathematical Journal. -2011.- Vol.2.- No. 1.- pp.149-151.

6. Сызықты емес теңдеуді шешудің параллельдеу әдісі // Вестник ЕНУ, серия естественно-технических наук. (95), 2013, С. 113-116.

**Диссертацияның құрылымы**. Диссертациялық жұмыс 5 бөлімнен (әр бөлім бөлімшелерге бөлінген), кіріспеден, қорытынды, әдебиеттер тізімі мен қосымшалардан құрылады. Барлығы беттер саны - 52.

**Диссертацияның негізгі мазмұны**. Диссертациялық жұмыста

$$Ax = f \tag{1}$$

операторлық теңдеуі $R^n$ кеңістігінде қарастырылады. 1-ші бөлімде (1)-ші теңдеуді жуықтап шешу әдісі келтірілген. $A$ матрицасы қайтымды және шенелген болған жағдайда (1)-ші теңдеуді жуықтап шешу мәселесі эквивалентті $J = |Ax - f|^2$ функционалдың мәнін кішірейту есебімен алмастырылып, келесі түрдегі жуық шешімдер тізбегі үшін

$$x_{k+1} = x_k + \varepsilon_k \omega_k, \tag{2}$$

мұнда

$$\varepsilon_k = \frac{-\langle Ax_k - f, A\omega_k \rangle}{|A\omega_k|^2} \text{ және } \omega_k = A^* Ax_k - A^* f. \tag{3}$$

келесі тұжырымдар дәлелденді:

$$|Ax_k - f|^2 = J(x_k) \leq \rho^k |f|^2$$

және

$$|x_k - \mathring{x}| \leq \| A^{-1} \| \rho^{\frac{k}{2}} |f|, \rho = 1 - \left( \frac{1}{\| A^{*-1} \| \| A \|} \right)^2.$$

2-ші бөлімде $A$ матрицасы кері матрицасы жоқ немесе нашар шартталған жағдай үшін (1)-теңдеуді жуықтап шешу әдісі құрастырылады. Бұл жағдайда диссертациялық жұмыста $\varepsilon \geq 0$ саны үшін келесі Тихоновтың функционалы

$$J_\varepsilon(x) = |Ax - f|^2 + \varepsilon |x|^2.$$

пайдаланылды. Онда (1)-теңдеуді шешу мәселесі эквивалентті

$$inf J_\varepsilon(x) = J_\varepsilon(\mathring{x})$$

есебімен алмастырылды. Шешімді құрастыру кезінде

$$x_j = \delta \sum_{k=0}^{j-1} [E - \delta(A^*A + \varepsilon E)]^k A^* f,$$

түрінде анықталған векторлар тізбегі мен

$$0 < \delta < \frac{2}{\|A^*A\| + \varepsilon}$$

сандары үшін

$$x_j - \mathring{x} = -[E - \delta(A^*A + \varepsilon E)]^j \mathring{x}$$

теңдігі алынып, $j \to +\infty$ ұмтылған жағдайда $x_j$ тізбегі $\mathring{x}$ векторына геометриялық прогрессияның жылдамдығымен жинақталатыны дәлелденді.
$\varepsilon \geq 0$ және $x_j$ $(j = 1,2, \dots)$ векторлар тізбегі үшін:

а) $\gamma > 0$ мен $j$ нөмірі

$$(1 - \delta(\gamma + \varepsilon))^{2j} \leq \gamma$$

шартынан алынса, онда

$$\left|Ax_j(\varepsilon) - f\right| \leq 2|f|\sqrt{\gamma} + \gamma|\mathring{x}(\varepsilon)| + |A\mathring{x}(\varepsilon) - f|$$

ә) егер $j$ нөмірі

$$(1 - \delta(\gamma + \varepsilon))^{2j} \leq \gamma^2,$$

шартты қанағаттандырса, онда

$$\left|A^*A\left(x_j(\varepsilon) - \mathring{x}(\varepsilon)\right)\right|^2 \leq \gamma^2[|A^*A\mathring{x}|^2 + |\mathring{x}|^2];$$

б) егер $j$ нөмірі

$$(1 - \delta(\gamma + \varepsilon))^{2j} \leq \frac{2}{5\|A^*\|}\gamma,$$

шартты қанағаттандырса, онда

$$\left|A^*A\left(x_j(\varepsilon)-\mathring{x}(\varepsilon)\right)\right|^2 \le 8\gamma|f|^2;$$

в) егер $\varepsilon = 0$ үшін $j$ нөмірі

$$(1-\delta\gamma)^{2j} \le \frac{2}{5\|A\ \|}\gamma,$$

формуладан алынса, онда

$$\left|A^*A\left(x_j(0)-f\right)\right|^2 \le 8\gamma|f|^2.$$

3-ші бөлімде сызықты емес алгебралық теңдеулер жүйесі қарастырылды және келесідей нәтижелер алынды. Жүйе

$$A(u) = f \tag{4}$$

$A$ - $H$ Гильберт кеңістігін өзіне бейнелейтін оператор болсын. $A$ үшін

$$A(0) = 0, |A(u)| \le C < \infty, |u| < C_1 \tag{5}$$

шарттары орындалсын. $C_1$ - $C$ - дан ғана тәуелді. Сонымен қатар келесі шарт орындалсын:

$$|A(u+v) - A(u) - B(u)v| \le C_2(|u|,|v|)|v|^2, \tag{6}$$

мұнда $B(u)$ - әр $u \in H$ үшін сызықты, үзіліссіз ($A$ операторының $u$ нүктесінде Гато туындысы), $C_2(|u|,|v|)$ - шенелген, монотонды кемитін үзіліссіз функция. $B(u)$ операторы қайтымды және

$$|B^{-1}(u)| \le C_3(|u|) \tag{7}$$

орындалсын. $u_0$ - (4) теңдеудің бастапқы жуықтауы, онда келесі жуықтауды $u_{n+1} = u_n + \varepsilon_n v_n, n = 0,1,\ldots$ формула бойынша анықталады. Келесі тұжырым ақиқат:

**Теорема 1**. *(5) - (7) шарттары орындалсын. Онда (4) теңдеудің шешімі бар болады. Егер $u_0$ - $H$ -тың кез келген элементі болса, онда $C_4 > 0$ саны мен $\{u_n\}_{n=0}^{\infty}$ үшін*

$$u_{n+1} = u_n + \varepsilon_n B^{-1}(u_n)S_n, S_n = A(u_n) - f \text{ және}$$
$$\frac{1}{2a_n^2} < 1; \text{үшін } \varepsilon_n = \frac{-1}{2a_n^2}, \frac{1}{2a_n^2} \ge 1 \text{үшін } \varepsilon_n = -1, \text{мұндағы } a_n^2 = |S_n|C_4.$$

*а)* $u_n$ *тізбегі* $A(u) - f = 0$ *теңдеудің шешіміне* $n \to \infty$ *ұмтылғанда,* $\lim\limits_{n\to\infty}|S_n| = \lim\limits_{n\to\infty}|A(u_n) - f| = 0$;

*ә)* $|S_n| = |A(u_n) - f|$ *өрнегі нөлге монотонды ұмтылады,* $n > n_0$ *үшін*

$$\left|u_{n_0+k} - u_{n_0+k_1}\right| \le C_5(|u_0|)2^{-2^{k-k_1}}$$

*теңсіздігі орындалады.*

Автор ғылыми жетекшісі ҚР ҰҒА академигі, ф. - м. ғ.д. М. Өтелбаевқа оқу барысында қойған есебі, ол есепті түсіндіруі, ғылыми нәтижелерді алуға көмек көрсеткені үшін, жарияланымдарды дайындауда кеңестер беріп, ғылыми жұмыспен айналысуға қолдау жасап, көп көңіл бөлгеніне шын жүректен алғысын білдіреді.

# 1 СЫЗЫҚТЫ АЛГЕБРАЛЫҚ ТЕҢДЕУЛЕР ЖҮЙЕСІН МАТРИЦАСЫ ШЕНЕЛГЕН ЖАҒДАЙДАҒЫ ЖУЫҚТАП ШЕШУ ҮРДІСІН ПАРАЛЛЕЛЬ ЕСЕПТЕУ ӘДІСІ

Бұл бөлімде сызықты алгебралық теңдеулер жүйесін матрицасы шенелген жағдайдағы жуықтап шешу үрдісін параллель есептеу әдісі келтіріледі.

Алдымен жақсы шартталған матрицасы бар сызықты алгебралық теңдеулер жүйесін жуықтап шешудің кейбір итерациялық әдістері келтіріледі. Бұл бөлімде келтірілген нәтижелер [20, 21] мақалаларда жарияланды.

## 1.1 Сызықты алгебралық теңдеулер жүйесін матрицасы шенелген жағдайдағы жуықтап шешу әдісі

Сызықты алгебралық теңдеулер жүйесін итерациялар әдісімен шешу [22, 23]. Сызықты алгебралық теңдеулер жүйесі берілсін:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}. \qquad (1.1.1)$$

Ұсынылып отырған итерациялық әдіс градиенттік әдістің қарапайым түрлерінің бірі болып табылады. Бұл әдіс (1.1.1) жүйені жуықтап шешуді тиімді әдіспен параллель есептеуге мүмкіндік береді. Бұл есеп [24] мақалада қарастырылған.

$N$- процессорлы жүйе үшін $(N \geq 1)$ $P_1, P_2, \ldots, P_N, P_{N+1}$ - операциялар саны $s(3n^2 + 2n) + n^2$ көбейту, $s$ - бөлу және көбейту мен бөлу амаладарын қосқанда барлығының саны $4sn^2$ -қа тең, $N$ - нен тәуелсіз. Мұндағы $s$ - итерациялар саны. Егер бір компьютер (бір процессорлы жүйе) жұмыс істесе, онда көбейту амалдар саны $3sn^2$ аз емес. $n$ үлкен болғанда, итерациялар саны $s$ $n$ - ге қарағанда өте үлкен болады, сондықтанда құрастырылған әдіс жеткілікті деңгейде тиімді болады.

Итерациялық әдісті параллель есептеу үрдісі берілген жүйедегі матрицаны есептеуіш элементтерге таспалы матрицаларды жіктеу арқылы іске асырылады. $A$ және оған түйіндес $A^*$ матрицалары $N$ таспалы матрицаларға жіктеліп, есептеулердің алдында таспалы матрицалар сәйкес процессорлардың локальді жадына таратылады. Таспалы матрицаларға берілген матрицаны бөлу кезінде процессорлардың жылдамдығы ескерілсе, бұл жайт есептеулерді шамамен бір уақытта аяқтауға мүмкіндік береді. Барлық есептеулер процессорлардың есептеу нәтижелерін түбірлік процессорда $P_{N+1}$ жинақталуы арқылы орындылады. Процессорлардың арасындағы қатынасу саны шамамен $8Ns$ санына тең болады.

Сонымен қатар шартты - корректті есептерді шешу кезінде (1.1.1) түріндегі жүйелерде көп жағдайда $A$ матрицасы нашар шартталған болады.

Келесі бөлімде осы жағдай үшін жуықтап шешу үрдісінің параллель есептеу әдісі зерттеледі.

Ең алдымен (1.1.1)-ші теңдеуді жуықтап шешу әдісін келтіреміз. $A$ матрицасы қайтымды және

$$\| A \| \leq c_0, \| A^{-1} \| \leq c_1$$

теңсіздігі орындалсын. Мұндағы $\|.\|$ белгісі $l_2^{(n)}$ кеңістігіндегі оператордың нормасын, яғни

$$\| A \| = sup_{|g|_{l_2^{(n)}}=1} |A|_{l_2^{(n)}} |g|^{-1}_{l_2^{(n)}}$$

ал $|\cdot|_{l_2^{(n)}}$ белгісі вектордың гильберттік норманы білдіреді, яғни $|g|^2_{l_2^{(n)}} = \sum_{j=1}^{n} {g_j}^2$, егер $g = (g_1, g_2, \dots, g_n)$. Кейде $|\cdot|_{l_2^{(n)}}$ белгісінің орнына қысқаша $|\cdot|$ деп жазамыз.

(1.1.1) - ші теңдеуді жуықтап шешу үшін $J$ функционалын пайдаланамыз.

Келесі тұжырым ақиқат:

*Лемма 1.1.1. а) Егер $J(\mathring{x}) = 0$ болса, онда $\mathring{x}$ векторы (1.1.1) - ші теңдеудің шешімі болады; ә) керісінше, егер $\mathring{x}$ векторы (1.1.1) - ші теңдеудің шешімі болса, онда $J(\mathring{x}) = 0$.*

Бұл леммадан (1.1.1)-ші теңдеуді жуықтап шешу мәселесі $J(\cdot)$ функционалдың мәнін кішірейту есебімен эквивалентті екенін байқауға болады. Сондықтанда төменде біз $J(\cdot)$ функционалдың мәнін кішірейту мақсатында $x_0, x_1, \dots$ векторлар тізбегін құрастырамыз. $x_0 = (x_{01}, x_{02}, \dots, x_{0n})$ векторды кез келген немесе нақты $x_0 = 0$ деп алуға болады. $x_0, x_1, \dots, x_k$ таңдап алынған болсын. $x_{k+1} = (x_{k1}, x_{k2}, \dots, x_{kn})$ векторды есептеу үрдісін келесі түрде орындаймыз:

$$x_{k+1} = x_k + \varepsilon_k \omega_k, \tag{1.1.2}$$

мұнда $\varepsilon_k$ саны мен $\omega_k$ векторы төменде анықталу ережесі көрсетіледі. Онда

$$J(x_{k+1}) = |Ax_{k+1} - f|^2 = |Ax_k - f + \varepsilon_k A\omega_k|^2 = |Ax_k - f|^2 + 2\varepsilon_k \langle Ax_k - f, A\omega_k \rangle + \varepsilon_k^2 |A\omega_k|^2$$

орындалады. $\varepsilon_k$ санын таңдап алайық. Егер $\langle Ax_k - f, A\omega \rangle = 0$ болса, онда $\varepsilon_k = 0$ орындалады. Егер $\langle Ax_k - f, A\omega \rangle \neq 0$ болса, онда $\varepsilon_k$ саны үшін ең тиімді мәні келесі формула бойынша анықталады:

$$\varepsilon_k = \frac{-\langle Ax_k - f, A\omega_k \rangle}{|A\omega_k|^2}. \tag{1.1.3}$$

$\varepsilon_k$ үшін $J(x_{k+1})$ функционалдың мәні

$$J(x_{k+1}) = J(x_k) - \frac{\langle Ax_k - f, A\omega_k\rangle^2}{|A\omega_k|^2} = J(x_k) - \frac{\langle A^*(Ax_k - f), \omega_k\rangle^2}{|A\omega_k|^2} \quad (1.1.4)$$

$\omega_k$ векторын

$$\omega_k = A^*Ax_k - A^*f \quad (1.1.5)$$

түрінде аламыз. Онда (1.1.4) формуласы бойынша

$$J(x_{k+1}) = J(x_k) - \frac{|\omega_k|^4}{|A\omega_k|^2}$$

Бұдан

$$J(x_k) - J(x_{k+1}) = \frac{|\omega_k|^4}{|A\omega_k|^2} \geq \frac{|\omega_k|^4}{\|A\||\omega_k|^2} = \| A \|^{-1} |\omega_k|^2 = \frac{1}{\|A\|}|A^*(Ax_k - f)|^2 = \frac{1}{\|A\|^2} \| A^{*-1} \|^{-2} |Ax_k - f|^2 \quad (1.1.6)$$

теңсіздігіне келеміз. Жоғарыда жазылған теңсіздікті шығару үшін

$$|A\omega_k| \leq \| A \| |\omega_k|, |(Ax_k - f)| \leq \left|A^{*-1}A^*(Ax_k - f)\right| \leq \| A^{*-1} \| |A^*(Ax_k - f)|$$

теңсіздіктерін пайдаландық. Енді (1.1.5) бойынша

$$J(x_{k+1}) \leq J(x_k)\left(1 - \frac{1}{\|A^{-1}\|^2\|A\|^2}\right), \quad (1.1.7)$$

мұнда $k = 0,1,2, \ldots$.

Келесі тұжырым орындалады:

*Теорема 1.1.1.* Егер A матрицасы қайтымды (яғни $A^{-1}$ бар болады), $\{x_k\}_{k\geq 0}$, $\{\omega_k\}_{k\geq 0}$ векторлар тізбегі және $\{\varepsilon_k\}_{k\geq 0}$ сандар тізбегі (1.1.2), (1.1.3) мен (1.1.5) формулалар бойынша анықталса, онда келесі бағалаулар орындалады:

$$|Ax_k - f|^2 = J(x_k) \leq \rho^k |f|^2$$

және

$$|x_k - \mathring{x}| \leq \| A^{-1} \| \rho^{\frac{k}{2}} |f|,$$

мұнда

$$\rho = 1 - \left(\frac{1}{\|A^{*-1}\|\|A\|}\right)^2.$$

*Дәлелдеуі*. $x_0 = 0$ деп алғанда, (1.1.7) теңсіздікті тізбектеп пайдалану арқылы теореманың 1-ші теңсіздігіне келеміз. Егер $\mathring{x}$ векторы $A\mathring{x} = f$ теңдеудің шешімі болса, онда

$$|Ax_k - A\mathring{x}| = |Ax_k - f - A\mathring{x} + f| \leq |Ax_k - f|$$

теңдігі орындалатыны айқын. Енді дәлелденген 1 - ші теңсізідікті қолдансақ, онда алатынымыз

$$|Ax_k - A\mathring{x}| = |Ax_k - f| \leq \rho^{\frac{k}{2}}|f|.$$

Бұдан

$$|x_k - \mathring{x}| = |A^{-1}(Ax_k - A\mathring{x})| \leq \| A^{-1} \| \rho^{\frac{k}{2}}|f|$$

формуласы шығады. Ал келесі теңдік орындалатындықтан $\| A^{-1} \| = \| A^{*-1} \|$ теореманың 2-ші теңсізідігі дәлелденді.

$\left|Ax_{k_\varepsilon} - f\right|^2 \leq \varepsilon$ дәлдікпен теңдеуді шешу үшін теореманың 1-ші теңсізідігін логарифмдеу арқылы итерациялар саны үшін келесі бағалауды аламыз:

$k_\varepsilon \geq \frac{ln\varepsilon^{-1}|f|^2}{ln\rho}$, мұндағы $\varepsilon$ өте аз шама. Ал $\left|x_{k_\varepsilon} - \mathring{x}\right|^2 \leq \varepsilon$ дәлдікпен есепті шешу үшін 2 - ші теңсізідікті пайдалану арқылы қажетті $k_\varepsilon$ итерация саны $k_\varepsilon \geq \frac{ln\varepsilon^{-1}\|A^{*-1}\|^2|f|^2}{ln\rho}$. Теорема дәлелденді.

*Ескерту 1.1.1*. (1.1.5)-*формулада* $\omega_k$ *векторын кездейсоқ деп алуға болады, ал* $\varepsilon_k$ *санын келесі жолмен таңдап алайық:*

$$\varepsilon_k = -\langle Ax_k - f, A\omega_k\rangle |A\omega_k|^{-2}.$$

*Онда*

$$J(x_{k+1}) = J(x_k) - \frac{\langle Ax_k - f, A\omega_k\rangle^2}{|A\omega_k|^2} = J(x_k) - \frac{|A^*(Ax_k - f)|^2}{|A\omega_k|^2}\langle r_k, S_k\rangle^2|\omega_k|^2$$
$$\left(S_k = \frac{\omega_k}{|\omega_k|}, r_k = \frac{A^*(Ax_k - f)}{|A^*(Ax_k - f)|}\right)$$

формуласы орындалады. Бұл әдіс кейбір жағдайда есептің шешімін тез табуға әкелуі мүмкін.

**1.2 Сызықты алгебралық теңдеулер жүйесін матрицасы шенелген жағдайдағы жуықтап шешу үрдісін параллельдеу**

Сызықты алгебраның негізгі алгоритмдеріне матрицалық - векторлық көбейту және матрицалардың көбейтіндісін есептеу операциялары жатады.

*Матрицалық - векторлық көбейту*. $N$ - процессорлы жүйе берілсін. $A$ - $m \times n$ өлшемді тікбұрышты матрица, ал $x$ пен $b$ $n \times 1$ өлшемді векторлар болсын. $Ax = b$ көбейтіндіні есептейтін екі әдіс бар: вектордың матрицаға көбейтіндісі болатын вектордың келесі түрдегі жіктелуіне байланысты құрылады:

$$b = \begin{pmatrix} \langle a_1, x \rangle \\ \langle a_2, x \rangle \\ \vdots \\ \langle a_m, x \rangle \end{pmatrix}$$

мұнда $a_i$ - $A$ матрицасының-ші жолы, ал

$\langle a_i, x \rangle$ - индекске сәйкес жолдың $x$ векторына скалярлық көбейтіндісі; $A$ матрицасының бағандардың сызықты комбинациясы арқылы құрылатын әдіс:

$$b = \sum_{i=1}^{n} x_j \, a_j,$$

мұнда $a_j$ - $A$ матрицасының -ші бағаны, ал

$x_j$ - $x$ векторының $j$ - ші координатасы.

Жұмыста аталған алгоритмдердің 1-ші әдісі қолданылғандықтан, сол әдістің негізгі қадамдарына тоқталып өтейік. Жазуларды ықшамдау үшін жалпылықты бұзбай, жүйенің өлшемі $m$ саны жүйедегі процессорлар санына $N$-ге еселі деп аламыз. Онда алгоритмнің сызбанұсқасын келесі түрде келтіруге болады:

1. Процессорларға $A$ матрицасының жолдары мен $x$ вектордың координаттары таратылады.
2. Параллель уақытта барлық процессорларда мынадай есептеулер орындалады:

– $P_1$ процессорында $\langle a_1, x \rangle = b_1, \ldots, \langle a_m, x \rangle = b_m$ скалярлық көбейтінділері есептелінеді;

– $P_2$ процессорында - $\langle a_{p+1}, x \rangle = b_{p+1}, \ldots, \langle a_{2p}, x \rangle = b_{2p}$ есептелінеді;

– ...;

– $P_N$ процессорында - $\langle a_{(N-1)p+1}, x \rangle = b_{(N-1)p+1}, \ldots, \langle a_{Np=m}, x \rangle = b_{Np=m}$.

3. Түбірлік процессорға көбейтудің нәтижелерін жіберіп, ол қосу арқылы $b$ векторын есептейді.

Матрицалық - векторлық көбейтіндінің скалярлық көбейтінділер арқылы жүзеге асатын алгоритмінің тиімділік бағалауы төменде келтіріледі. Алдымен қажетті анықтамаларды келтіреміз.

*Анықтама 1.3.1*. Коммуникациялық желінің диаметрі деп кез келген екі түйіннің арасындағы максималды ара қашықтығын атайды.

*Анықтама 1.3.2*. Коммуникациялық желінің латенттілігі деп түйіндер желісі арқылы ақпаратты жіберуге дайындық уақытын атайды.

*Анықтама 1.3.3*. Коммуникациялық желінің өткізгіш қабілеті деп түйін желісі арқылы ақпараттардың жеткізілуінің уақыты немесе бір уақыт бірлігінде түйіндер арасында жіберілетін ақпараттардың көлемін атайды.

Жазуларды қысқарту мақсатында $P_1, P_2 \dots P_N$ процессорларында екі санның үтірі жылжымалы амалдар көбейту мен қосу үшін бірдей $t$ уақыт жұмсалады деп аламыз. Сонымен қоса $m = n$ жағдайды қарастырамыз. Онда $\langle a_i, x \rangle = b_i$ бір скалярлық көбейтіндіні есептеу үшін $(n + (n-1))t$ уақыт жұмсалады, ал әр процессордың есептеулер шығындары $T_{calc}$ саны мынаған тең:

$$T_{calc} = (2n - 1) \approx 2\frac{n^2}{N}t.$$

Әр процессордың коммуникациялық шығындары $T_{com}$ сәйкес $a_i$ вектордың $np$ компоненттерін мен $x$ вектордың $n$ координаттарын қабылдау, есептеу нәтижесі болатын $b$ вектордың $p$ компоненттерін жіберуден тұрады. Онда коммуникациялық желінің диаметрі 1 - ге тең деп алғанда,

$$T_{com} = 2S + \frac{(np + n + p)l}{R} \approx 2S + \left(\frac{n^2}{N} + n\right) \cdot \frac{l}{R},$$

мұнда $S$ - коммуникациялық желінің латенттілігі,

$l$ - нақты санның байтпен алынған ұзындығы,

$R$ - коммуникациялық желінің өткізгіш қабілеті (өлшем бірлігі - байт/сек).

$N$ - процессорлы жүйеде жоғарыда жазылған алгоритмге жұмсалатын уақыты

$$T_N \approx 2\frac{n^2}{N}t + 2S + \left(\frac{n^2}{N} + n\right) \cdot \frac{l}{R},$$

ал 1 процессорда бұл сан мынаған тең:

$$T_1 = (2n - 1) \approx 2n^2 t.$$

Егер $t = 10 * 10^{-9}[c], l = 32S = 50 * 10^{-6}[c]$ және $R = 80 * 10^6$[байт/с] болса, онда алгоритмінің жеделдетуі үшін келесі формуланы аламыз:

$$S_N = \frac{T_1}{T_N} \approx \frac{20n^2N}{20n^2 + 400(n^2 + nN) + 10^5N}.$$

Енді 1.1-ші тармақта құрастырылған екі әдістің параллель есептеу үрдісінің алгоритмдерін келтіреміз. $n$ үлкен сан берілсін. $N+1$ - процессорлы ([wcm_eq1]) - жүйені шешу үшін пайдаланамыз: $P_1, P_2, \dots, P_N, P_{N+1}$.

$k_0, k_1, \dots, k_N$ бүтін сандар $k_j + 1 < k_{j+1}, j = 0,1, \dots, N, k_0 = 0, k_N = n$ шартты қанағаттандыратын болсын. $A_j, j = 1,2, \dots, N$ матрицаларды енгіземіз. Оның нөмірлері $k_{j-1} + 1, \dots, k_j$ болатын жолдардың элементтері $A$ матрицасының сәйкес жолдар элементтерімен бірдей, қалғандары нөлдер:

$$A_1 = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_1 n \\ \dots & \dots & \dots & \dots & \dots \\ a_{k_{11}} & a_{k_{12}} & a_{k_{13}} & \dots & a_{k_1 1 n} \\ 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

$$A_j = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 \\ a_{k_{j-1}+1,1} & a_{k_{j-1}+1,2} & a_{k_{j-1}+1,3} & \dots & a_{k_{j-1}+1,n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{k_j,1} & a_{k_j,2} & a_{k_j,3} & \dots & a_{k_j,n} \\ 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0. \end{pmatrix} (j = 1,2, \dots, N)$$

Дәл солай $A$ матрицасы оған түйіндес $A^*$ матрицасына алмастырылып, $(A^*)_j (j = 1,2, \dots, N)$ матрицалары анықталады.

$P_j$ процессорына $A_j, (A^*)_j$ матрицасы және $(A^*)_j f (j = 1,2, \dots, N)$ векторы таратылады. Теорема 1.1.1 негізінде жуықтап шешудің параллель есептеу алгоритмін төменде келтіреміз.

1. $P_{N+1}$ түбірлік процессоры $x_k (k = 0,1, \dots)$ векторын есептеп, $P_j$ $(j = 1,2, \dots, N)$ процессорларына жібереді. Әр процессор $P_j$ $A_j x_k$ вектордың мәнін есептейді. Ол кезде $(k_j - k_{j-1})n$ көбейту мен $(k_j - k_{j-1})(n-1)$ қосу амалдары орындалады. Кейін процессорлар $P_{N+1}$ түбірлік процессорға $A_j x_k$ есептеу нәтижелерін жібереді.

2. $P_{N+1}$ барлық $A_j x_k (j = 1,2, \dots, N)$ векторларын алып, қосу арқылы $Ax_k = \sum_{j=1}^{N} A_j x_k$ векторын есептейді. Бұнда $(n-1)N$ қосу орындалады. Қайтадан $Ax_k$ векторын $P_j (j = 1,2, \dots, N)$ процессорларына таратады. Әр $P_j$ процессоры:

$$(A^*)_j(Ax_k) + (-(A^*)_j f) = (A^*)_j(Ax_k - f)$$

есептейді. Бұнда орындайтын операциялар саны: $(k_j - k_{j-1})n$ көбейту, $(k_j + 1 - k_{j-1})(n-1)$ қосу. $(A^*)_j(Ax_k - f)$ векторы $P_{N+1}$ - ге қайтып барады.

3. $P_{N+1}$ процессоры

$$\omega_k = \sum_{j=1}^{N} (A^*)_j(Ax_k - f) = A^*(Ax_k - f)$$

векторын есептеп, $P_j(j = 1,2,\dots,N)$ - ларға жібереді. $(n-1)N$ қосу амалы орындалады. $P_j$ процессорлары $A_j\omega_k, |A_j\omega_k|^2$ және $|\omega_k|_j{}^2 = \sum_{r=k_{j-1}}^{k_j} \omega_{k,r}{}^2$ есептеп, есептеу нәтижелерін $P_{N+1}$ түбірлік процессорға жібереді. Бұнда $(k_j - k_{j-1})(n+2)$ көбейту мен $(k_j - k_{j-1})(n-1) + 2(k_j - k_{j-1} - 1)$ қосу амалдары орындалады.

4. $P_{N+1}$ процессоры:

$$|\omega_k|^2 = \sum_{j=1}^{N} |\omega_k|_j{}^2, |A\omega_k|^2 = \sum_{j=1}^{N} |A_j\omega_k|_j{}^2, \varepsilon_k = \frac{-|\omega_k|^2}{|A\omega_k|^2}$$

есептеп, $x_{k+1}$ жуық шешімін:

$$x_{k+1} = x_k + \varepsilon_k\omega_k$$

формуласы бойынша құрастырады. Ол кезде 1 бөлу, $n$ көбейту мен $n - 1 + 2(N-1)$ қосу амалдары орындалады. Есептің дәлдігі тексеріледі, егер алдын ала бекітілген дәлдікті шешім қанағаттандыратын болса, онда үрдіс тоқтатылады. Кері жағдайда, цикл қайта жалғасады.

$k = 0$ үшін $P_{N+1}$ процессоры алғашқы жауық шешім ретінде кездейсоқ векторды (мысалы, $x_k = 0$) алады. $P_{N+1}$ процессоры бір циклде $n$ көбейту мен $n - 1 + 2(N-1) + 2N(n-1)$ қосу амалдарды орындайды. Ал $P_j$ циклде $3(k_j - k_{j-1})n + 2(k_j - k_{j-1})$ көбейту, $3(k_j - k_{j-1})(n-1) + n - 1 + 2(k_j - k_{j-1} - 1)$ қосу амалын орындайды. Барлық процессорлар $P_j(j = 1,2,\dots,N)$ бір цикл үшін $3n^2 + 2n$ көбейту, $3n(n-1) + N(n-1) + 2n - 2N$ қосу амалын орындайды. Ал $P_{N+1}$ процессоры $2(n-1)N + n - 1 + 2(N-1) = 2nN + n - 3$ қосу, $n$ көбейту және бір бөлу операцияларын орындайды. $s$ циклде $(A^*)_j f$ векторын есептеумен бірге барлық операциялар саны $s[3(n^2+n)+1] + n^2$ санына тең және $N$ - нен тәуелсіз.

$k_j(j = 0,1,\dots,N)$ бүтін сандарды таңдау ережесін келтіреміз:

$$k_0 = 0, k_{j+1} > k_j + 1(j = 0,1,\dots,N-1)$$

$$k_{j+1} = k_j + \left[\frac{C}{v_{j+1}}\right],$$

мұнда $v_{j+1}$ белгілеуі $P_{j+1}$ процессордың жылдамдығын, ал $[.]$ санның бүтін бөлігін білдіреді. $C$ тұрақты саны:

$$n = C\left[\frac{1}{v_1} + \frac{1}{v_2} + \cdots + \frac{1}{v_N}\right]$$

теңдігінен анықталады. Аталған шарттар $P_j(j = 1,2,\ldots,N)$ процессорлардың жұмысын шамамен бір уақытта аяқтауға септігін тигізеді. $P_{N+1}$ процессоры аз операцияны орындағандықтан, оның жұмысын $n$ үлкен болған жағдайда аз деп қабылдауға болады.

*Ескерту 1.2.1.* $\mathrm{P_{N+1}}$ жұмысын $\mathrm{P_j(j = 1,2,\ldots,N)}$ процессорладың ішіндегі ең қуатты процессорға жүктеуге болады. Ол кезде алғашқы кезде $\mathrm{N} + 1$ емес N - процессорлы жүйені пайдалануға болады.

*Ескерту 1.2.2.* Қолданбалы теорияда нөлдік емес элементтері әр түрлі болатын матрицалар кездеседі. $\mathrm{k_j(j = 1,2,\ldots,k_N)}$ сандарды таңдағанда, осы жайтты қолдануға болады.

1.2.1-ші ескертуге сүйеніп, келесі алгоритмді келтіреміз:

1. $P_j(j = 1,2,\ldots,N)$ процессорларына $A_j(j = 1,2,\ldots,N)$ матрицалар мен $F_j(j = 1,2,\ldots,N)$ векторлар таратылады. $F_j(j = 1,2,\ldots,N)$ вектордың координаттардың нөмірлері $k_{j-1} + 1, k_{j-1} + 2, \ldots, k_j$ болатын сәйкес $f$ вектордың элементтерімен бірдей, қалғандары нөлдер, яғни

$$F_1 = (f_1, \ldots, f_{k_1}, 0, \ldots, 0)$$
$$\ldots$$
$$F_j = (0, \ldots, 0, f_{k_{j-1}+1}, f_{k_{j-1}+2}, \ldots, f_{k_j}, 0, \ldots, 0)$$
$$\ldots$$

2. $P_{N+1}$ процессоры $x_k$ векторын есептеп, кездейсоқ $\omega_k$ векторын таңдап алады ($k = 0$ үшін $x_k$ нөлдік вектор деп алынады). $P_{N+1}$ процессоры $x_k$ және $\omega_k$ векторларын $P_j(j = 1,2,\ldots,N)$ процессорларға таратады. $P_j$ процессорлары:

$$\langle A_j x_k - f_j, A_j \omega_k \rangle, \left|A_j \omega_k\right|^2$$

есептеп, $P_{N+1}$-ға жібереді.

3. $P_{N+1}$ түбірлік процессор:

$$\frac{\sum_{j=1}^{N} \langle A_j x_k - f_j, A_j \omega_k \rangle}{\sum_{j=1}^{N} \left|A_j \omega_k\right|^2} \equiv \varepsilon_k$$

векторды есептеп, $x_{k+1}$ жуық шешімін:

$$x_{k+1} = x_k + \varepsilon_k \omega_k$$

формуласы бойынша құрастырады. Есептің дәлдігі тексеріледі, егер алдын ала бекітілген дәлдікті шешім қанағаттандыратын болса, онда үрдіс тоқтатылады. Кері жағдайда, цикл қайта жалғасады.

Бұл алгоритмде барлық операциялар саны $s(3n^2 + n + 1)$ санына тең, мұндағы $s$ - итерациялар саны.

## 2 СЫЗЫҚТЫ АЛГЕБРАЛЫҚ ТЕҢДЕУЛЕР ЖҮЙЕСІН МАТРИЦАСЫ НАШАР ШАРТТАЛҒАН ЖАҒДАЙДАҒЫ ЖУЫҚТАП ШЕШУ ҮРДІСІН ПАРАЛЛЕЛЬ ЕСЕПТЕУ ӘДІСІ

Бұл бөлімде сызықты алгебралық теңдеулер жүйесін матрицасы нашар шартталған жағдайдағы жуықтап шешу үрдісін параллель есептеу әдісі келтіріледі.

### 2.1 Сызықты алгебралық теңдеулер жүйесін матрицасы нашар шартталған жағдайдағы жуықтап жуықтап шешу әдісі

$A$ мен $f$ (2.1.4) теңдеуінде берілсін. $\varepsilon \geq 0$ саны үшін келесі функционалды енгізейік [25-27]:

$$J_\varepsilon(x) = |Ax - f|^2 + \varepsilon|x|^2.$$

Біз келесі есептің шешімі болатын $\mathring{x}$ векторын іздейміз:

$$inf J_\varepsilon(x) = J_\varepsilon(\mathring{x}). \qquad (2.1.1)$$

Теңдіктің оң жағында $infimum\ R^n$ кеңістігіндегі барлық $x$ векторлар бойынша алынады. $R^n$ - дегі бірлік шар компакт болғандықтан, (2.1.1) - есептің шешімі әрқашан бар болады.

*Ескерту 2.1.1.* А матрицасы қайтымды болса, онда $\varepsilon = 0$ үшін (2.1.1)-ші есептің жалғыз шешімі бар және ол мынаған тең: $\mathring{x} = A^{-1}f$; ал егер А матрицасының кері матрицасы жоқ болса, онда $A\mathring{x}$ векторы f векторының Ах түріндегі векторлардың жуықтау мәні болып табылады. Егер $\varepsilon \neq 0$ және А қайтымды болса, онда $\mathring{x}$ векторы $Ax = f$ теңдеудің жуық шешімі болады.

Егер $\varepsilon = 0$ және $A$ қайтымды болмаса, онда (2.1.1) - есептің бірнеше шешімі болады. Бұндай жағдайда біз нормасы ең аз болатын шешімді іздейміз.

*Лемма 2.1.1.* Егер $\varepsilon \geq 0$ және $\mathring{x}$ - (2.1.1)-ші есептің шешімі болса, онда келесі теңдік орындалады:

$$A^*(A\mathring{x} - f) + \varepsilon\mathring{x} = 0.$$

*Дәлелдеуі*. $\mathring{x}$ (2.1.1)-ші есептің шешімі болсын. Кері жорып,

$$\omega = A^*(A\mathring{x} - f) + \varepsilon\mathring{x} \neq 0$$

деп аламыз. $J_\varepsilon(\mathring{x} + \delta\omega)$ мәнін есептейік:

$$J_\varepsilon(\mathring{x} + \delta\omega) = J_\varepsilon(\mathring{x}) + 2\delta\langle A^*(A\mathring{x} - f) + \varepsilon\mathring{x}, \omega\rangle + \delta^2(|A\omega|^2 + \varepsilon|\omega|^2) = J_\varepsilon(\mathring{x}) + 2\delta|\omega|^2 + \delta^2(|A\omega|^2 + \varepsilon|\omega|^2)$$

$\delta$ саны келесі шарттарды қанағаттандырсын:

$$\delta < 0, -2\delta > \delta^2 \frac{|A\omega|^2 + \varepsilon|\omega|^2}{|\omega|^2}.$$

Бұндай сан $\omega \equiv A^*(A\mathring{x} - f) + \varepsilon\mathring{x} \neq 0$ жоруымыз бойынша бар болады. Онда $J_\varepsilon(\mathring{x} + \delta\omega) < J_\varepsilon(\mathring{x})$ орындалады. Қайшылыққа келдік. Лемма дәлелденді.

*Лемма 2.1.2.* $\varepsilon \geq 0$ және $\mathring{x}$ (2.1.1)-ші есептің шешімі болсын. Онда кез келген $x \in H$ үшін келесі теңдік ақиқат:

$$\varepsilon x + A^*(Ax - f) = (\varepsilon + A^*A)(x - \mathring{x}).$$

*Дәлелдеуі*. 2.1.1 - лемма бойынша алатынымыз:

$$\varepsilon x + A^*(Ax - f) = \varepsilon x + A^*(Ax - f) - \varepsilon\mathring{x} - A^*(A\mathring{x} - f) =$$
$$\varepsilon(x - \mathring{x}) + A^*A(x - \mathring{x}) = (\varepsilon + A^*A)(x - \mathring{x}).$$

Лемма дәлелденді.

$x_j (j = 1,2,\dots)$ векторлар тізбегін келесі түрде анықтаймыз:

$$x_j = \delta \sum_{k=0}^{j-1} [E - \delta(A^*A + \varepsilon E)]^k A^* f, \tag{2.1.2}$$

мұнда $\delta$ саны

$$0 < \delta < \frac{2}{\|A^*A\| + \varepsilon} \tag{2.1.3}$$

шартты қанағаттандыратын кез келген сан.

*Теорема 2.1.1.* $\varepsilon \geq 0$, $\delta$ сандары (2.1.3)-ші формуладан алынсын. $\mathring{x}$ - (2.1.1)-ші есептің шешімі, ал $x_j$ (2.1.2)-ші формула бойынша құрастырылсын. Онда:

$$x_j - \mathring{x} = -[E - \delta(A^*A + \varepsilon E)]^j \mathring{x} \tag{2.1.4}$$

*және $j \to +\infty$ ұмтылған жағдайда $x_j$ тізбегі $\mathring{x}$ векторына геометриялық прогрессияның жылдамдығымен жинақталады, яғни белгілі бір $\rho > 0$ саны үшін келесі теңсіздік орындалады:*

$$\left|x_j - \mathring{x}\right| \leq C \cdot \rho^j,$$

мұнда C - $\delta$ мен $\varepsilon$ сандарынан тәуелді тұрақты константа.

*Дәлелдеуі*. 2.1.1 - лемма бойынша:

$$A^* f = \varepsilon\mathring{x} + A^*A\mathring{x}.$$

$A^*f$ өрнекті (2.1.2)-ші формулаға қоямыз. Онда

$$x_j = \delta \sum_{k=0}^{j-1} [E - \delta(A^*A + \varepsilon E)]^k [\varepsilon \mathring{x} + A^*A\mathring{x}] =$$

$$= \sum_{k=0}^{j-1} [E - \delta(A^*A + \varepsilon E)]^k [E - E + \delta(\varepsilon + A^*A)]\mathring{x} =$$

$$= -\sum_{k=1}^{j} [E - \delta(A^*A + \varepsilon E)]^k \mathring{x} + \sum_{k=0}^{j-1} [E - \delta(A^*A + \varepsilon E)]^k \mathring{x} =$$

$$= -[E - \delta(A^*A + \varepsilon E)]^j \mathring{x} + \mathring{x}$$

теңдігіне келеміз. Бұдан (2.1.4) - ші формула шығады.
$E - \delta(A^*A + \varepsilon E)$ матрицасы өзара түйіндес болғандықтан, оның нормасы меншікті мәндердің модульдерінің максимумына тең болады. Оның меншікті мәндері мынаған тең: $1 - \delta({s_j}^2 + \varepsilon)(j = 1,2,\dots,n)$. Сондықтанда:

$$\|E - \delta(A^*A + \varepsilon E)\| < 1$$

теңдігі $j = 1,2,\dots,n$ үшін:

$$-1 < 1 - \delta({s_j}^2 + \varepsilon) < 1.$$

теңсіздігімен бір уақытта орындалады. Бұл теңсіздіктер $\delta(\max\limits_{j=1,2,\dots,n} {s_j}^2 + \varepsilon) < 2$ және $\delta > 0$ шарттар орындалғандықтан ақиқат болады. Бірақ $\max\limits_{j=1,2,\dots,n} {s_j}^2 = \|A^*A\|$. Сондықтанда (2.1.3) - ші формуладан (2.1.4) шығады [28-33]. [25, p. 3-422] кітапта алынған нәтижелерге жақын формулалар бар екенін атап өтейік [25, p. 238].

$R_0^{(n)}$ арқылы $A^*A$ матрицасының нөлдік меншікті мәндерге сәйкес болатын меншікті векторлармен туындалған кеңістікті белгілейміз, яғни $x \in R_0^{(n)}$, $x = \sum_{k=j_0}^{n} x_k e_k$ болса, онда $k = j_0, \dots, n$ үшін $A^*Ae_k = 0$ орындалады. $R_0^{(n)}$ - $A^*A$ матрицасының ядросы.

Егер $A$ матрицасы қайтымды болса, онда $R_0^{(n)}$ кеңістігі бос жиын болады.

*Лемма 2.1.3.* Егер $x \in R_0^{(n)}$ болса, онда $\langle A^*f, x\rangle = 0$, яғни $A^*f$ векторы $R_0^{(n)}$ кеңістігінің ортогоналды толықтауышы болатын $R^{(n)} \ominus R_0^{(n)}$ кеңістігінде жатыр.

*Дәлелдеуі.* $\varepsilon = 0$ саны үшін 2.1.1-ші мен 2.1.2 - леммалардан алатынымыз:

$$A^*f = A^*A\mathring{x}.$$

Егер $x \in R_0^{(n)}$ болса, онда

$$\langle A^* f, x\rangle = \langle A^* A\mathring{x}, x\rangle = \langle \mathring{x}, A^* Ax\rangle = 0.$$

Лемма дәлелденді.

*Лемма 2.1.4*. $\varepsilon > 0$ саны және $\mathring{\mathrm{x}}$ - (2.1.1)-ші есептің шешімі болсын. Егер $\mathrm{x} \in \mathrm{R}_0^{(\mathrm{n})}$ болса, онда $\langle \mathring{\mathrm{x}}, \mathrm{x}\rangle = 0$ орындалады, яғни $\mathring{\mathrm{x}}$ векторы $\mathrm{R}^{(\mathrm{n})} \ominus \mathrm{R}_0^{(\mathrm{n})}$ кеңістігінде жатады.

*Дәлелдеуі*. $\varepsilon > 0$ саны үшін 2.1.1-ші мен 2.1.2-леммалардан алатынымыз:

$$\varepsilon\langle \mathring{x}, x\rangle = \langle A^* f, x\rangle = \langle A^* A\mathring{x}, x\rangle = -\langle \mathring{x}, A^* Ax\rangle = 0.$$

Бұдан $\varepsilon > 0$ үшін лемманың орындалатыны шығады.

$\varepsilon = 0$ үшін (2.1.1)-ші есептің шешімі $Ax = 0$ теңдеудің шешімінің қосылғышына дейінгі дәлдікпен анықталады. Бірақ $x_j (j = 1,2,\dots)$ тізбегі 2.1.2 - лемма бойынша $R^{(n)} \ominus R_0^{(n)}$ кеңістігінде жататын (2.1.1)-ші есептің шешіміне жинақталады. $\varepsilon = 0$ саны үшін (2.1.2)-ші формуладағы $x_j$ тізбегінің шегі үшін $\mathring{x}(0)$ белгісін пайдаланамыз.

(2.1.1)-ші есептің шешімі -нан тәуелді болғандықтан, бұл тәуелділікті біз $\mathring{x} = \mathring{x}(\varepsilon)$ белгілеу арқылы көрсетеміз.

*Лемма 2.1.5*. Егер $\mathring{\mathrm{x}}(0)$ (2.1.1)-і есептің шешімі болса, онда $\varepsilon > 0$ және $\delta \geq 0$ сандары үшін:

$$\mathring{x}(\varepsilon) = (A^*A + \varepsilon E)^{-1} A^* A\mathring{x}(0) = (A^*A + \varepsilon E)^{-1} A^* f,$$
$$\mathring{x}(0) = (E + \varepsilon (A^*A)^{-1})\mathring{x}(\varepsilon),\ \mathring{x}(\varepsilon) = (A^*A + \varepsilon E)^{-1}(A^*A + \delta E)\mathring{x}(\delta),$$
$$\mathring{x}(\varepsilon) - \mathring{x}(\delta) = (\delta - \varepsilon)(A^*A + \varepsilon E)^{-1}\mathring{x}(\delta).$$

*Дәлелдеуі*. 2.1.1 - лемма бойынша кез келген $\varepsilon, \delta \geq 0$ үшін%

$$(A^*A + \varepsilon E)\mathring{x}(\varepsilon) = (A^*A + \delta E)\mathring{x}(\delta)$$

теңдігі орындалады. Бұдан лемма шығады.

Дәлелденген леммалар мен 2.1.1-теорема бойынша келесі тұжырым шығады:

*Теорема 2.1.2*. а) (2.1.1)-ші есептің шешімі болатын $\mathring{\mathrm{x}}(\varepsilon)$ векторы $\varepsilon > 0$ санынан үзіліссіз тәуелді және

$$\mathring{x}(\varepsilon) = (A^*A + \varepsilon E)^{-1} A^* f$$

теңдігі орындалады.

ә) $\mathrm{j} \to +\infty$ ұмтылғанда (2.1.2)-ші формула бойынша құрастырылған $\mathrm{x_j}(\varepsilon)$ тізбегінің шегі $\varepsilon \geq 0$ санынан үзіліссіз тәуелді.

*б) егер* $s_1 \geq s_2 \geq \cdots \geq s_{j_0} > 0, s_{j_0+1} = s_{j_0+2} = \ldots = 0$ *сандары* $A^*A$ *матрицасының меншікті мәндері және* $e_1, e_2, \ldots, e_n$ *сәйкес ортонормаланған меншікті векторлары, ал* $\mathring{x}(\varepsilon)(\varepsilon > 0)$ *(2.1.1) - ші есептің шешімі және* $x_j(\varepsilon)$ *тізбегі (2.1.2)-ші формуладан алынса, онда* $\mathring{x}(\varepsilon), x_j(\varepsilon) \in R^{(n)} \ominus R_0^{(n)}, (j = 1,2, \ldots)$

$$1 \leq k \leq j_0 \text{ үшін } x_{jk}(\varepsilon) - \mathring{x}_k(\varepsilon) = (1 - \delta(s_k^2 + \varepsilon))^j \mathring{x}_k(0),$$
$$j_0 + 1 \leq k \text{ үшін } x_{jk}(\varepsilon) = \mathring{x}_k(\varepsilon).$$

мұнда

$$x_{jk}(\varepsilon) = \langle x_j(\varepsilon), e_k \rangle, \mathring{x}_k(\varepsilon) = \langle \mathring{x}(\varepsilon), e_k \rangle.$$

*в) 2.1.1 - теоремадағы* $\rho > 0$ *саны мынаған тең:*

$$\rho = max\{(1 - \delta(s_{j_0}^2 + \varepsilon)), (1 - \delta(\|A^*A\| + \varepsilon))\} < 1.$$

Егер $A^*A$ матрицасының нөлдік меншікті мәндері болмаса, онда $j_0$ саны - ге тең деп алынады.

2.1.2 - теореманың б) тармағындағы тұжырымнан $\mathring{x}(\varepsilon)$ векторы - $\varepsilon = 0$ саны үшін (2.1.4)-ші есептің шешімдерінің ішіндегі норма мәні ең аз болатын вектор. Сонымен қатар $\varepsilon \geq 0 \mathring{x}(\varepsilon)$ және $x_j(\varepsilon)(j = 1,2, \ldots)$ векторлары $R^{(n)} \ominus R_0^{(n)}$ кеңістігінде жатады, мұндағы $R_0^{(n)}$ - $A^*A$ матрицасының ядросы.

## 2.2 Сызықты алгебралық теңдеулер жүйесін матрицасы нашар шартталған жағдайдағы жуықтап жуықтап шешу әдісі

2.1.1-ші мен 2.1.2-теоремаларға сүйеніп, (2.1.1)-ші есептің жуық шешімін табу үрдісінің параллель есептеу әдісін келтіреміз.

$n$ үлкен саны және $N + 1$ - процессорлы жүйе берілсін. $k_{m-1} + 1 < k_m, m = 0,1, \ldots, N, k_0 = 0, k_N = n$ шартты қанағаттандыратын $k_0, k_1, \ldots, k_N$ бүтін сандар берілсін. $A_m$ мен $(A^*)_m, m = 1,2, \ldots, N$ матрицаларды келесі түрде енгіземіз:

$$A_m = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 0 \\ a_{k_{m-1}+1,1} & a_{k_{m-1}+1,2} & a_{k_{m-1}+1,3} & \ldots & a_{k_{m-1}+1,n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{k_m,1} & a_{k_m,2} & a_{k_m,3} & \ldots & a_{k_m,n} \\ 0 & 0 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 0. \end{pmatrix}$$

$$(A^*)_m = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 \\ \alpha_{k_{m-1}+1,1} & \alpha_{k_{m-1}+1,2} & \alpha_{k_{m-1}+1,3} & \dots & \alpha_{k_{m-1}+1,n} \\ \dots & \dots & \dots & \dots & \dots \\ \alpha_{k_m,1} & \alpha_{k_m,2} & \alpha_{k_m,3} & \dots & \alpha_{k_m,n} \\ 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0. \end{pmatrix} (m = 1,2, \dots, N)$$

Бұл матрицалардың нөмірлері $k_{m-1} + 1$-ден бастап, $k_m$-ге дейінгі жолдардың элементтері сәйкес $A$ мен $A^*$ матрицаларының жолдарындағы элементтерімен бірдей, қалғаны нөлдер. Мұнда $\alpha_{kj}$ және $a_{kj}$ - $A$ мен $A^*$ матрицаларының элементтері, сәйкес $\alpha_{kj} = a_{jk}$ теңдіктері орындалады. Сонымен қоса есептеулерде:

$$\omega^j = [E - \delta(A^*A + \varepsilon E)]\omega^{j-1}, \omega^0 = \delta A^* f, j = 1,2, \dots$$

түріндегі векторлар қолданылады. Онда (2.1.2)-ші формуланы:

$$x_{j+1} = x_j + \omega^j, x_1 = \omega^0, j = 1,2, \dots$$

түрінде жазуға болады.

Есептеу үрдісінің алдында $P_m(m = 1,2, \dots, N)$ процессорларына келесі мәліметтер жіберіледі: $A_m$ мен $(A^*)_m$ матрицалары, ал түбірлік процессорға $\omega^0 = \delta A^* f$ векторы енгізіледі. Алгоритм келесі түрде іске асырылады:

1. $P_{N+1}$ процессоры $j$ - ші жуық $x_j$ шешімін есептеп, $\omega^{j-1}$ векторын барлық $P_m(m = 1,2, \dots, N)$ процессорларына жібереді. Әр $P_m$ процессоры $A_m\omega^{j-1}$ мәнін есептеу үшін $(k_m - k_{m-1})n$ көбейту мен $(k_m - k_{m-1})(n-1)$ қосу амалын орындайды. $P_{N+1}$ түбірлік процессорға вектордың мәнін қайтарады.

2. $P_{N+1}$ процессоры жинаған векторлады қосу арқылы:

$$A\omega^{j-1} = \sum_{m=1}^{N} A_m\omega^{j-1}$$

векторын алады. Ол кезде $(n-1)N$ қосу амалын орындайды. $P_{N+1}$

$$A\omega^{j-1}$$

векторын процессорларға таратады.

3. $P_m$ процессоры $(A^*)_m A\omega^{j-1}$ векторды есептеп, $P_{N+1}$ процессорына жібереді. Бұған $(k_m - k_{m-1})n$ көбейту мен $(k_m - k_{m-1})(n-1)$ қосу амалы жұмсалады. $P_m$ процессорлары есептеу нәтижелерін $(A^*)_m A\omega^{j-1}$ $P_{N+1}$ - на жібереді.

4. $P_{N+1}$ жинаған векторларды қосу арқылы:

$$A^*(A\omega^{j-1}) = \sum_{m=1}^{N} (A^*)_m A\omega^{j-1}$$

векторын құрастырады. Сонымен қатар $\omega^j = (1-\delta\varepsilon)\omega^{j-1} - \delta(A^*)_m A\omega^{j-1}$ векторды есептеп, $x_{j+1} = x_j + \omega^j, j = 1,2,\ldots$ жуық шешімді анықтайды. Бұнда $2n$ көбейту мен $(n-1)N + 2n$ қосу амалы орындалады.

$P_{N+1}$ түбірлік процессордың цикл бойынша орындаған операциялар саны: $2n$ көбейту мен $2(n-1)N + 2n$ қосу. Әр процессор $P_m(m = 1,2,\ldots,N)$ циклде орындайтын операциялар саны: $2(k_m - k_{m-1})n$ көбейту мен $2(k_m - k_{m-1})(n-1)$ қосу. Барлық процессорлардың бір циклде орындайтын амалдар саны мынаған тең: $2n^2$ көбейту және $2n(n-1)$ қосу. Ал $s$ циклде барлығы $2sn^2 + n^2 + n$ көбейту мен $2sn(n-1) + n(n-1)$ қосу амалдарын орындайды.

2.1.2-теорема бойынша (2.1.2)-ші формуланың тиімділігі $A^*A$ матрицасының нөлдік емес бірақ нөлге жақын меншікті мәндерге байланысты екенін байқауға болады. Ал, керісінше $A$ матрицасының нөлдік меншікті мәндердің бар болуы формулаға әсер етпейді. Осы кезде нөлге жақын меншікті мәндердің есептеулерге әсерін бәсеңдету мәселесі туындайды. Оны қалай шешуге болатынын келесі мысалда көрсетеміз.

$\varepsilon = 0{,}01$ саны мен

$$A = \begin{pmatrix} 1 & 1 \\ 3 & 3{,}001 \end{pmatrix}, f = \begin{pmatrix} 2 \\ 6{,}006 \end{pmatrix}$$

жүйесі берілсін.

$$A^*A = \begin{pmatrix} 1 & 3 \\ 1 & 3{,}001 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 3 & 3{,}001 \end{pmatrix}$$

матрицасының нөлге тең емес өте аз шамаға тең меншікті мәні бар. Сондықтанда (2.1.4)-ші формула бойынша шешімді табу үрдісі ұзаққа созылуы мүмкін. Бірақ $A$ матрицасын:

$$\tilde{A} = \begin{pmatrix} 1 & 1 \\ 3 & 3 \end{pmatrix}, \tilde{f} = \begin{pmatrix} 2 \\ 6 \end{pmatrix} \qquad (2.2.1)$$

матрицасына алмастыратын болсақ, онда

$$A^*A = \begin{pmatrix} 1 & 3 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 3 & 3 \end{pmatrix} = \begin{pmatrix} 10 & 10 \\ 10 & 10 \end{pmatrix}$$

Бұл матрицаның меншікті мәндері $\lambda_1^2 = 20, \lambda_2^2 = 0$ және нормасы 20-ға тең. Сондықтанда $\delta$ санын $(0, \frac{1}{10})$ интервалынан алуға болады. $\delta = \frac{1}{20} < \frac{1}{10}$ деп аламыз. Онда 2.1.2-теорема бойынша $\rho = 0$. Яғни (2.2.1)-шы формуладан $A$ және $f$ мәндерімен алынған (2.1.4)-ші есепті аталған итерация бір қадаммен шешеді.

(2.1.4)-ші есептің шешімі $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ векторы болады.

$$A\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 6 \end{pmatrix}$$

теңдеуінің шешімі $x_1 + x_2 = 2$ шартты қанағаттандыратын кез келген $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ векторы болғандықтан, олардың ішінде $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ векторы да бар. Табылған $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ векторы (2.2.1) - формуладағы $A$ мен $f$ - пен алынған (2.1.1)-ші теңдеудің шешімі болады. Шынында да

$$\begin{pmatrix} 1 & 1 \\ 3 & 3{,}001 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 2 \\ 6{,}006 \end{pmatrix} = \begin{pmatrix} 0 \\ 0{,}006 \end{pmatrix}.$$

Сондықтанда $|A\mathring{x} - f| = 0{,}006 \approx 0$. (Біз $|x|$ норманы үлкейтпей, $|Ax - f|$ нормасын кішірейтуіміз қажет екенін тағы да ескертіп айтамыз.) Берілген

$$\begin{pmatrix} 1 & 1 \\ 3 & 3{,}001 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 6{,}006 \end{pmatrix}$$

жүйенің шын шешімі $\tilde{x} = \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} = \begin{pmatrix} -4 \\ 6 \end{pmatrix}$ векторы болып табылады.

$\mathring{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ үшін алатынымыз:

$$|A\mathring{x} - f|^2 + \varepsilon|\mathring{x}|^2 = \left|\begin{matrix} 0 \\ 0{,}006 \end{matrix}\right|^2 + 0{,}01\left|\begin{matrix} 1 \\ 1 \end{matrix}\right|^2 = (0{,}006)^2 + 0{,}01 \approx 0{,}01.$$

Ал $\begin{pmatrix} -4 \\ 6 \end{pmatrix}$ үшін:

$$|A\tilde{x} - f|^2 + \varepsilon|\tilde{x}|^2 = 0 + 0{,}01(16 + 24) = 0{,}4$$

орындалады. $\varepsilon = 0{,}01\mathring{x} = \begin{pmatrix}1\\1\end{pmatrix}$ векторы $\bar{x} = \begin{pmatrix}-4\\6\end{pmatrix}$ векторына қарағанда, (2.1.4) -ші есептің шешіміне жақын орналасады.

Келесі тұжырым ақиқат:

*Теорема 2.2.1.* $\varepsilon \geq 0$ және $x_j$ $(j = 1,2,\dots)$ векторлар тізбегі (2.1.2)-ші формула бойынша анықталсын. Онда

а) $\gamma > 0$ мен j нөмірі $(1 - \delta(\gamma + \varepsilon))^{2j} \leq \gamma$, шартынан алынса, онда

$$\left|Ax_j(\varepsilon) - f\right| \leq 2|f|\sqrt{\gamma} + \gamma|\mathring{x}(\varepsilon)| + |A\mathring{x}(\varepsilon) - f|$$

*ә) егер j нөмірі*

$$(1 - \delta(\gamma + \varepsilon))^{2j} \leq \gamma^2,$$

*шартты қанағаттандырса, онда*

$$\left|A^*A\left(x_j(\varepsilon) - \mathring{x}(\varepsilon)\right)\right|^2 \leq \gamma^2[|A^*A\mathring{x}|^2 + |\mathring{x}|^2];$$

*б) егер j нөмірі*

$$(1 - \delta(\gamma + \varepsilon))^{2j} \leq \frac{2}{5\|A^*\|}\gamma,$$

*шартты қанағаттандырса, онда*

$$\left|A^*A\left(x_j(\varepsilon) - \mathring{x}(\varepsilon)\right)\right|^2 \leq 8\gamma|f|^2;$$

*в) егер $\varepsilon = 0$ үшін j нөмірі*

$$(1 - \delta\gamma)^{2j} \leq \frac{2}{5\|A^*\|}\gamma,$$

*формуладан алынса, онда*

$$\left|A^*A\left(x_j(0) - f\right)\right|^2 \leq 8\gamma|f|^2$$

*Дәлелдеуі*. $\varepsilon \geq 0$ саны беріліп,

$$\inf_{\{x\}}(|Ax - f|^2 + \varepsilon|x|^2) = |A\mathring{x}(\varepsilon) - f|^2 + \varepsilon|\mathring{x}(\varepsilon)|^2$$

орындалсын. Кез келеген $u$ векторы үшін

$$|Au|^2 = \langle Au, Au\rangle = \langle A^*Au, u\rangle = \left|(A^*A)^{\frac{1}{2}}u\right|^2.$$

орындалады. $(A^*A)^{\frac{1}{2}}e_k = s_k e_k$ болғандықтан, (2.1.4)-ші теңдіктерді пайдалана отырып, 2.1.2 - теорема бойынша:

$$\left|A\left(x_j(\varepsilon) - \mathring{x}(\varepsilon)\right)\right|^2 = \left|(A^*A)^{\frac{1}{2}}\left(x_j(\varepsilon) - \mathring{x}(\varepsilon)\right)\right|^2 = \delta\sum_{k=1}^{n} s_k^2\left(x_{jk}(\varepsilon) - \mathring{x}_k(\varepsilon)\right)^2 =$$

$$= \delta\sum_{k=1}^{n} s_k^2\left(1 - \delta(s_k^2+\varepsilon)\right)^{2j}|\mathring{x}_k(\varepsilon)|^2$$

теңдіктерді аламыз. $\gamma > 0$ үшін алатынымыз:

$$\begin{gathered}(1-\delta(s_k^2+\varepsilon))^{2j}|\mathring{x}_k(\varepsilon)|^2 \le (1-\delta(s_k^2+\varepsilon))^{2j}|\mathring{x}_k(\varepsilon)|^2 \\ \le \left(1-\delta(\gamma+\varepsilon)\right)^{2j}\delta\sum_{s_k^2>\gamma} s_k^2\,|\mathring{x}_k(\varepsilon)|^2 + \\ +\gamma\delta\textstyle\sum_{s_k^2\le\gamma}|\mathring{x}_k(\varepsilon)|^2 \le \left(1-\delta(\gamma+\varepsilon)\right)^{2j}\delta\sum_{k=1}^{n} s_k^2\,|\mathring{x}_k(\varepsilon)|^2 + \gamma\delta\sum_{k=1}^{n}|\mathring{x}_k(\varepsilon)|^2 = \\ = \left(1-\delta(\gamma+\varepsilon)\right)^{j}\left|(A^*A)^{\frac{1}{2}}\mathring{x}(\varepsilon)\right|^2 + \gamma|\mathring{x}(\varepsilon)|^2 = \\ = (1-\delta(\gamma+\varepsilon))^{j}|A\mathring{x}(\varepsilon)|^2 + \gamma|\mathring{x}(\varepsilon)|^2\end{gathered} \tag{2.2.2}$$

Бірақ

$$\begin{gathered}|A\mathring{x}(\varepsilon)|^2 = |A\mathring{x}(\varepsilon) - f + f|^2 \le 2(|A\mathring{x}(\varepsilon) - f|^2 + |f|^2) \le \\ \le 2\left[\left(\inf_{\{x\}}(|Ax-f|^2 + \varepsilon|x|^2)\right) + |f|^2\right] \le 2(|f|^2 + |f|^2) = 4|f|^2.\end{gathered} \tag{2.2.3}$$

Жазылған теңдік пен (2.2.2)-формуладан:

$$\left|A\left(x_j(\varepsilon) - \mathring{x}(\varepsilon)\right)\right|^2 \le 4[1-\delta(\gamma+\varepsilon)]^{2j}|f|^2 + \gamma|\mathring{x}(\varepsilon)|^2.$$

шығады. Онда

$$\begin{gathered}\left|Ax_j(\varepsilon) - f\right| = \left|A\left(x_j(\varepsilon) - \mathring{x}(\varepsilon)\right) + A\mathring{x}(\varepsilon) - f\right| \le \\ \le \left[\left|A\left(x_j(\varepsilon) - \mathring{x}(\varepsilon)\right)\right| + |A\mathring{x}(\varepsilon) - f|\right] \le \\ \le 2|f|(1-\delta(\gamma+\varepsilon))^{j} + |\mathring{x}(\varepsilon)|\sqrt{\gamma} + |A\mathring{x}(\varepsilon) - f|\end{gathered}$$

Келесі шарттар

$$(1-\delta(\gamma+\varepsilon))^{2j} \le \gamma$$

орындалса, онда соңғы теңсіздіктен:

$$\left|Ax_j(\varepsilon)-f\right| \le 2|f|\sqrt{\gamma}+\sqrt{\gamma}|\mathring{x}(\varepsilon)|+|A\mathring{x}(\varepsilon)-f|$$

теңдігі шығады.

(2.1.4)-формуладан шығатыны:

$$\left|A^*A\left(x_j(\varepsilon)-\mathring{x}(\varepsilon)\right)\right|^2 = \delta\sum_{k=1}^{n} s_k^4\,(1-\delta(s_k^2+\varepsilon))^{2j}|\mathring{x}_k(\varepsilon)|^2 = \quad (2.2.4)$$

$$= \delta\sum_{s_k^2>\gamma} s_k^4\,(1-\delta(s_k^2+\varepsilon))^{2j}|\mathring{x}_k(\varepsilon)|^2 + \delta\sum_{s_k^2\le\gamma} s_k^4\,(1-\delta(s_k^2+\varepsilon))^{2j}|\mathring{x}_k(\varepsilon)|^2 \le$$

$$\le \delta\sum_{s_k^2>\gamma} s_k^4\,(1-\delta(s_k^2+\varepsilon))^{2j}|\mathring{x}_k(\varepsilon)|^2 + \delta\sum_{s_k^2\le\gamma} s_k^4\,(1-\delta(s_k^2+\varepsilon))^{2j}|\mathring{x}_k(\varepsilon)|^2.$$

Бұл теңсіздіктен алатынымыз:

$$\left|A^*A\left(x_j(\varepsilon)-\mathring{x}(\varepsilon)\right)\right|^2 \le (1-\delta(\gamma+\varepsilon))^{2j}|A^*A\mathring{x}|^2+\gamma|\mathring{x}|^2$$

$j$ нөмірін

$$(1-\delta(\gamma+\varepsilon))^{2j} \le \gamma^2,$$

теңсіздіктен таңдап алғанда,

$$\left|A^*A\left(x_j(\varepsilon)-\mathring{x}(\varepsilon)\right)\right|^2 \le \gamma^2[|A^*A\mathring{x}|^2+|\mathring{x}|^2]$$

теңсіздігі шығады. Бұдан теореманың ә) тармағындағы тұжырым шығады.

(2.2.4)-формуладан тағы келесідей теңсіздіктерді шығаруға болады:

$$\left|A^*A\left(x_j(\varepsilon)-\mathring{x}(\varepsilon)\right)\right|^2 \le \delta\sum_{k=1}^{n}(1-\delta(\gamma+\varepsilon))^{2j}\left|s_k^2\mathring{x}_k(\varepsilon)\right|^2+\delta\sum_{k=1}^{n}s_k^2\,|\mathring{x}_k(\varepsilon)|^2 =$$

$$= (1-\delta(\gamma+\varepsilon))^{2j}|A^*A\mathring{x}_k(\varepsilon)|^2+\gamma|A^*A\mathring{x}_k(\varepsilon)|^2 =$$

$$= (1-\delta(\gamma+\varepsilon))^{2j}|A^*(A\mathring{x}_k(\varepsilon)-f)+A^*f|^2+\gamma|A\mathring{x}_k(\varepsilon)|^2 \leq$$
$$\leq 2(1-\delta(\gamma+\varepsilon))^{2j}(|A^*(A\mathring{x}_k(\varepsilon)-f)|^2+|A^*f|^2)+\gamma|A\mathring{x}_k(\varepsilon)|^2.$$

(2.2.3) бойынша:

$$|A^*(A\mathring{x}_k-f)|^2 \leq \|A^*\|^2(|A\mathring{x}_k|^2+|f|^2) \leq 5\|A^*\|^2|f|^2,$$
$$|A\mathring{x}_k(\varepsilon)|^2 \leq 4|f|^2.$$

Сондықтанда

$$\left|A^*A\left(x_j(\varepsilon)-\mathring{x}(\varepsilon)\right)\right|^2 \leq 2(1-\delta(\gamma+\varepsilon))^{2j}[5\|A^*\|^2|f|^2+|A^*f|^2]++4\gamma|f|^2 \leq$$
$$\leq 10(1-\delta(\gamma+\varepsilon))^{2j}\|A^*\|^2|f|^2+4\gamma|f|^2.$$

$j$ нөмірін

$$(1-\delta(\gamma+\varepsilon))^{2j}10\|A^*\|^2 \leq 4\gamma,$$

шартынан таңдап алғанда,

$$\left|A^*A\left(x_j(\varepsilon)-\mathring{x}(\varepsilon)\right)\right|^2 \leq 8\gamma|f|^2.$$

теңсіздікке келеміз. Теореманың б) тармағы дәлелденді.
$\varepsilon = 0$ үшін 2.1.2 - лемма бойынша

$$A^*A\mathring{x} = Af$$

орындалады. Сондықтанда теореманың в) тармағы б) - дан шығады. Теорияның қолданбалы бөлімдерінде көп жағдайда (2.1.4)-ші теңдеудің нақты шешімін табудан гөрі $Ax-f$ айырманы кішірейтуге ұмтылады. Сондықтанда дәлелденген теорема (2.1.1)-ші есепті ұтымды шешуге мүмкіндік береді. Сонымен қатар $x_j$ үшін формуланы пайдаланған кезде $\varepsilon > 0$ емес, $\varepsilon = 0$ алынуы жөн. 2.1.3-теоремаға сүйеніп, параллель есептеу алгоритмін келтіруге болады.

Ол жоғарыда жазылған алгоритмінен келесідей айырмашылықтары бар болады: есептің дәлдігіне жауап беретін:

10. $\gamma > 0$ саны алынады;
11. $\varepsilon = 0$ деп алынады;
12. әр циклден кейін 2.2.1-теореманың в) тармағындағы шартының орындалатыны тексеріледі. Егер (2.2.2) орындалса, онда цикл тоқтатылады.

# 3 СЫЗЫҚТЫ ЕМЕС АЛГЕБРАЛЫҚ ТЕҢДЕУЛЕР ЖҮЙЕСІН ЖУЫҚТАП ШЕШУ ҮРДІСІН ПАРАЛЛЕЛЬ ЕСЕПТЕУ ӘДІСІ

Бұл бөлімде сызықты емес алгебралық теңдеулер жүйесін жуықтап шешу үрдісін параллель есептеу әдісі келтіріледі.

## 3.1 Сызықты емес алгебралық теңдеулер жүйесін жуықтап шешудің Ньютон әдісі

Бүгінгі күні есептеу жүйелер, сонымен қоса көп процессорлы жүйелер немесе суперкомпьютерлер қарқынды дамуда. Бұл параметрлері көп болатын есепті шешуде есептеулерді параллель тармақтарға жіктеу арқылы уақытты үнемдеуге мүмкіндік береді. Сызықты емес алгебралық жүйенің жуықтап шешу үрдісін құрастырып, оның параллель есептеу әдісін келтіреміз. Алдымен сызықты емес алгебралық теңдеулер жүйесін жуықтап шешудің әдістері туралы қысқаша тоқталып өтейік. Бұндай жүйелерді шешуде И. Ньютон әдісі жиі пайдаланылады [34].

$$Px = 0, \tag{3.1.1}$$

мұнда $P$ операторы - $\Omega \subset H$ ашық жиынды $H$ кеңістігіне бейнелейтін оператор. Кез келген $x_0 \in \Omega$ элементті алайық. $P$ операторының $\Omega$- да туындысы бар деп жорығанда, $P(x_0) = P(x_0) - P(x^*)$ элементті (мұндағы $x^*$ - (3.1.1) теңдеудің шешімі) оған ұқсас $P'(x_0)(x_0 - x^*)$ элементімен алмастырып, осы алмастыруға сүйеніп,

$$P'(x_0)(x_0 - x^*) = P(x_0) \tag{3.1.2}$$

теңдеуінің шешімі $x^*$ векторына жақын деп алуға болады. (3.1.2) теңдеуі сызықты болғандықтан, оның шешімі келесі формуламен табылады:

$$x_1 = x_0 - [P'(x_0)]^{-1}(P(x_0)).$$

Осы үрдісті жалғастырғанда, біз келесі түрдегі $\{x_n\}$ тізбекке келеміз:

$$x_{n+1} = x_n - [P'(x_n)]^{-1}(P(x_n))(n = 0,1,2,\dots). \tag{3.1.3}$$

$x_n$ тізбегін құру әдісі Ньютон әдісі болып табылады.

Ньютон әдісі кейбір жағдайда іске аспауы мүмкін: $x_n$ тізбегінің элементтері кейбір $n$ үшін $\Omega$ жиынынан шығып кетуі мүмкін немесе $[P'(x_n)]^{-1}$ анықталмаған болуы мүмкін.

Егер $\{x_n\}$ тізбегі $x^*$ векторына жинақталып, $x_0$ векторы $x^*$ - на жеткілікті жақын алынса, онда $P'$ опертордың үзіліссіздігі бойынша $P'(x_n)$ мен $P'(x_0)$

айырмашылығы аз болады. Бұған сүйеніп, (3.1.3) формуласының орнына ықшамдалған түрін жазуға болады:

$$x'_{n+1} = x'_n - [P'(x_0)]^{-1}(P(x'_n))(n = 0,1,2,\dots; x_0 = x'_0).$$

$\{x'_n\}$ тізбегін Ньютон әдісінің модификацияланған түрі деп аталады.

*Теорема 3.1.1*. [34, с. 89-184; 35] P операторы Ω жиынында анықталып, $\Omega_0(|x - x_0| \le r)$ тұйық шарда екінші ретті туындысы бар болсын. Сонымен қатар келесі шарттар орындалсын:

*1)* $\Gamma_0 = [P'(x_0)]^{-1}$ *операторы анықталған;*

*2)* $|\Gamma_0 P(x_0)| \le \eta;$

*3)* $|\Gamma_0 P'(x)| \le K(x \in \Omega_0).$

*Егер*

$$h = K\eta \le \frac{1}{2}$$

*және*

$$r \ge r_0 = \frac{1-\sqrt{1-2h}}{h}\eta,$$

шарттары орындалса, онда $x_n$ мен $x'_n$ тізбектері (3.1.1) теңдеудің шешімі болатын $x^*$ векторына жинақталады. Сонымен қоса

$$|x - x_0| \le r_0$$

теңсіздігі орындалады. Егер $h < \frac{1}{2}$ үшін

$$r < r_1 = \frac{1+\sqrt{1-2h}}{h}\eta,$$

*ал* $h = \frac{1}{2}$ *үшін*

$$r \le r_1,$$

орындалса, онда $\Omega_0$ шарында $x^*$ шешімі жалғыз болады.

Үрдістің жылдамдығы келесі теңсіздік арқылы сипатталады:

$$|x^* - x_n| \le \frac{1}{2^n}(2h)^{2^n}\frac{\eta}{h}(n = 0,1,2,\dots),$$

ал модификацияланған әдісте $h < \frac{1}{2}$ үшін

$$|x^* - x'_n| \le \frac{\eta}{h}\left(1 - \sqrt{1-2h}\right)^{n+1}(n = 0,1,2,\dots).$$

[36] мақалада құрастырылған сызықты емес алгебралық теңдеулер жүйені шешудің итерациялық әдісін келтіреміз. Жүйе

$$A(u) = f \tag{3.1.4}$$

$A$ - $H$ гильберт кеңістігін өзіне бейнелейтін оператор болсын. $A$ үшін [36]

$$A(0) = 0, |A(u)| \leq C < \infty, |u| < C_1 \tag{3.1.5}$$

шарттары орындалсын. $C_1$ - $C$ - дан ғана тәуелді. Сонымен қатар келесі шарт орындалсын:

$$|A(u+v) - A(u) - B(u)v| \leq C_2(|u|, |v|)|v|^2, \tag{3.1.6}$$

мұнда $B(u)$ - әр $u \in H$ үшін сызықты, үзіліссіз ($A$ операторының $u$ нүктесінде Гато туындысы), $C_2(|u|, |v|)$ - шенелген, монотонды кемитін үзіліссіз функция. $B(u)$ операторы қайтымды және

$$|B^{-1}(u)| \leq C_3(|u|) \tag{3.1.7}$$

орындалсын. $u_0$ - (3.1.4) теңдеудің бастапқы жуықтауы, онда келесі жуықтауды $u_{n+1} = u_n + \varepsilon_n v_n, n = 0,1, \ldots$ формула бойынша анықталады. Келесі тұжырым ақиқат:

*Теорема 3.1.2.* (3.1.5)-(3.1.7) шарттары орындалсын. Онда (3.1.4) теңдеудің шешімі бар болады. Егер $\mathrm{u}_0$ - Н-тың кез келген элементі болса, онда $\mathrm{C}_4 > 0$ саны мен $\{\mathrm{u_n}\}_{\mathrm{n=0}}^{\infty}$ үшін

$$u_{n+1} = u_n + \varepsilon_n B^{-1}(u_n) S_n, S_n = A(u_n) - f$$

*және*

$$\frac{1}{2a_n^2} < 1; \text{үшін } \varepsilon_n = -\frac{1}{2a_n^2},$$
$$\frac{1}{2a_n^2} \geq 1 \text{үшін } \varepsilon_n = -1,$$

мұнда $a_n^2 = |S_n|C_4$.

*а)* $u_n$ *тізбегі* $A(u) - f = 0$ *теңдеудің шешіміне* $n \to \infty$ *ұмтылғанда,*

$$\lim_{n\to\infty} |S_n| = \lim_{n\to\infty} |A(u_n) - f| = 0;$$

*ә)* $|S_n| = |A(u_n) - f|$ *өрнегі нөлге монотонды ұмтылады,* $n > n_0$ *үшін*

$$\left|u_{n_0+k} - u_{n_0+k_1}\right| \leq C_5(|u_0|)2^{-2^{k-k_1}}$$

теңсіздігі орындалады.

**3.2 Сызықты емес алгебралық теңдеулер жүйесін жуықтап шешу әдісі**

$H$ - $N$ - өлшемді ($1 \leq N \leq \infty$) Гильберт кеңістігі болсын.

Келесі алгебралық теңдеулер жүйесін қарастырайық:

$$A(u) - f = 0, u \in H. \tag{3.2.1}$$

Егер $-1 \leq \varepsilon \leq 1$ және $u, \omega \in H$ болса, онда

$$A(u + \varepsilon\omega) = A(u) + \varepsilon B_u \omega + \varepsilon^2 D(u, \omega, \varepsilon). \tag{3.2.2}$$

мұнда $B_u$ - $\omega \in H$ кеңістігінде әсер ететін әр $u \in H$ үшін сызықты үзіліссіз оператор және $D(\cdot,\cdot,\cdot)$ -әр $(u, \omega)$ жұбына әсер ететін сызықты емес түрлендіру. Егер $A(\cdot)$ - сызықты оператор болса, онда $B_u$ $A$ операторымен беттеседі. $A, B_u$ және $D(\cdot,\cdot,\cdot)$ операторлары қосымша келесі шарттарды қанағаттандыратын болсын:

$$A(0) = 0, \| B_u(\omega) \vee \leq C_1(|A(u)|)|\omega|^* \varepsilon \in (-1; 1), |\omega| \leq 1 |D(u, \omega, \varepsilon)| \leq C_2(|A(u)|),$$

мұнда $C_1(\cdot)$ және $C_2(\cdot)$ үзіліссіз монотонды кемімейтін $C_1(0) > 0$, $C_2(0) > 0$ шарттарды қанағаттандыратын функциялар. (3.2.3) - формулада және төменде $\vee$ белгісі элементтердің нормасын, сонымен қоса скаляр шаманың модулін білдіреді.

[35, с. 113-115] мақалада (3.2.1) есепті жуықтап шешу мен оны параллель есептеу әдісі құрастырылды. Ұсынылып отырған әдіс $n$ - компьютер (процессор) пайдаланған кезде есептеуге жұмсалатын уақыт бір компьютерге қарағанда шамамен $n$ есе аз болады. Бұндай нәтиже бұрыннан белгілі сиретілген немесе таспалы матрицасы бар сызықты алгебралық теңдеулер жүйелері үшін көрсетілмеген [11, p. 3-675].

(3.2.1)-ші есепті сызықты емес жағдайда шешуді параллельдеуде біз қосымша шартты қолданамыз:

$$\| B_u^{-1} \| = \| B_u^{-1} \|_{H \to H} \leq C_3, \tag{3.2.4}$$

мұнда $C_3$ - $u \in H$ тәуелсіз тұрақты сан. (3.2.2), (3.2.3) және (3.2.4) шарттардың орындалуын пайдаланып, біз аталған есептің шешімі бар туралы 3.2.1-теореманы дәлелдейміз. 3.2.1-ші теореманы қатаң шарттарды алып тастап, дәлелдеуге болады [34, с. 89-184; 36, с. 20-24], бірақ 3.2.1-теореманың дәлелдеуін ықшамдау үшін $A$ операторы қосымша шартқа тәуелді деп аламыз.

Келесі тұжырым орындалады:

*Теорема 3.2.1.* Егер (3.2.2), (3.2.3) және (3.2.4) шарттар орындалса, онда жүйенің шешімі бар болады. $\mathrm{u}_0, \mathrm{u}_1, \ldots, \mathrm{u}_n$ векторлар тізбегі H-тан алынып, келесі рекуррентті формулалармен анықталсын:

$$u_0 = 0, u_{n+1} = u_n - \varepsilon_n \omega_n,$$
$$\omega_n = \frac{B^*_{u_n}(A(u_n)-f)}{|B^*_{u_n}(A(u_n)-f)|}; \varepsilon_n = \sqrt{J_u}\frac{1}{4c^3(1+f\vee)} < 1,$$

Онда n ≥ 0 үшін

$$J_n \le J_0\rho^n = |f|^2\rho^n (n = 0,1,2,\dots)$$

мұнда

$$\rho = 1 - \frac{1}{C^4(1 + f\vee)} < 1$$

бағалаулар орындалады. (3.2.1) есептің шешімі болатын $\mathring{u}$ үшін келесі теңсіздіктер орындалады:

$$\mathring{u} - u_n \vee \le \sqrt{J_0}\frac{1}{4C^3(1+f\vee)} \cdot \frac{\rho^n}{1-\sqrt{\rho}}$$

мұнда $J_u = |A(u) - f|^2, C = 1 + max\{C_1(2|f|), C_2(2|f|), C_3(2|f|)\}$.

Әдебиеттерде бұл теореманың нұсқалары алуан түрлі. Бірақ осы теореманың дәлелдеуін келтіру қажет.

*Дәлелдеуі.* $n = 0$ үшін

$$u_1 = \varepsilon_1\omega_1, \omega_1 = -B_0^* f(B_0^*|f|^{-1}),$$
$$\varepsilon_0 = |f|[4C^3(1 + |f|)]^{-1}, J_0 \le J_0\rho^0 = J_0.$$

Егер $\omega_n$ және $\varepsilon_n$ (3.2.5)-ші формуладан алынса, онда $n = 0$ үшін (3.2.6) орындалады. $n = 0,1,\dots,k$ үшін (3.2.5)-ші формула бойынша анықталса, онда $n \le k$ үшін (3.2.6) дәлелденді.

$n = k + 1$ үшін дәлелдейік. (3.2.2) бойынша

$$J_{k+1} = |A(u_k + \varepsilon_k\omega_k) - f|^2 = \left|A(u_k - f + \varepsilon_k B_{u_k}\omega_k) + \varepsilon_k^2 D(u_k, \omega_k\varepsilon_k)\right|^2.$$

Бұл теңдіктен және $0 < \varepsilon_k < 1$ мен (3.2.3) шарттарынан

$$J_{k+1} \le J_k + 2\varepsilon_k < (A(u_k) - f), B_{u_k}\omega_k > +$$
$$+2\varepsilon_k^2\left[A(u_k) - f|D(u_k, \omega_k\varepsilon_k)| + B_{u_k}|\omega_k|^2 + |D(u_k, \omega_k\varepsilon_k)|^2\right]$$

шығады. Енді $\omega_k$ мен келесі формулалардан алатынымыз:

$$(A(u_k) - f), B_{u_k}\omega_k > B_k^*(A(u_k) - f), \omega_k >,$$
$$|A(u_k) - f| = |(B_k^*)^{-1}B_k^*(A(u_k) - f)| \le \| (B_k^*)^{-1} \| |B_k^*(A(u_k) - f)|,$$
$$|A(u_k)| = |A(u_k) - f + f| \le |A(u_k) - f| + |f| \le 2.$$

Онда (3.2.3) және (3.2.4) шарттарынан келесі теңсіздікке келеміз:

$$J_{k+1} \le J_k - 2\varepsilon_k |B_u^*(A(u_k) - f)| + 4\varepsilon_k^2 \left(\sqrt{J_k} + C\right) C^3 \le$$
$$\le J_k - 2\varepsilon_k \sqrt{J_k} C^{-1} + 4\varepsilon_k^2 C^3 (|f| + 1).$$

(3.2.5) формуласы бойынша анықталған $\varepsilon_k$ мәнін орнына қойғанда, $J_{k+1} \le J_k \rho$ теңсіздігін аламыз. Бұдан (3.2.5)-ші формуланың орындалатыны шығады. Енді (3.2.7) дәлелдейік. $n > 1, k > 1$ үшін:

$$u_{n+1} = u_n - \varepsilon_n \omega_n, \dots, u_{n+k+1} = u_{n+k} - \varepsilon_{n+k} \omega_{n+k}.$$

Жоғарыда жазылған теңсіздіктер мен (3.2.6)-шы формуладан алатынымыз Теорема дәлелденді.

*Ескерту 3.2.1*. Теореманың шарттары қатаң болып табылады. Бұл теореманы күшейтуге болады. Бірақ біздің негізгі мақсатымыз құрастырылған жуықтау әдісін параллельдеудің мүмкіндігін көрсету болғандықтан, теореманың дәлелдеуін ықшамды түрде келтіруге тырыстық.

### 3.3 Сызықты емес алгебралық теңдеулер жүйесін жуықтап шешу үрдісін параллельдеу

3.2.1-теоремаға сүйеніп, (3.2.1) есепті шешуді параллель есептеу әдісін келтіреміз [35, с. 113-115; 36, с. 20-24].

Процессорлар саны $k$ және $N$ - кеңістіктің өлшемі болсын. $A$ операторын

$$A(u) = (a_1(u), a_2(u), \dots, a_N(u))$$

вектор түрінде, ал $B_u^*$ операторын

$$B_u^* = \{b_{ji}(u)\}$$

матрица түрінде өрнектейік.

Алдын ала $|f|$ және $C$ шамаларын есептейік (қажет болса, осы үрдісті де параллель есептеуге болады).

$m_l + 1 < m_{l+1}, l = 0,1, \dots, k$ және $m_0 = 0, m_k = N$ шарттарды қанағаттандыратын $m_0, m_1, \dots m_N$ бүтін сандар болсын. $l = 1,2, \dots, N$ үшін:

$$B_u^l = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 \\ b_{m_{l-1}+1,1}(u) & b_{m_{l-1}+1,2}(u) & b_{m_{l-1}+1,3}(u) & \dots & b_{m_{l-1}+1,N}(u) \\ \dots & \dots & \dots & \dots & \dots \\ b_{m_l,1}(u) & b_{m_l,2}(u) & b_{m_l,3}(u) & \dots & b_{m_l,N}(u) \\ 0 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

түрінде матрицаларды енгізейік. $B_u^l$ матрицасының нөмірлері:

$$m_{l-1}+1, m_{l-1}+2, \dots, m_l, l = 1,2, \dots, N$$

болатын жолдардың элементтері $B_u^*$ матрицасының элементтерімен бірдей, қалғандары нөл болады. Сонымен қатар:

$$A_l(u) = \left(0, \dots, a_{m_{l-1}+1}(u), a_{m_{l-1}+1}(u), \dots, a_{m_l}(u), 0, \dots, 0\right)$$

және

$$f_l = \left(0, \dots, f_{m_{l-1}+1}, f_{m_{l-1}+1}, \dots, f_{m_l}, 0, \dots, 0\right)$$

векторларын енгіземіз. Есептеулредің алдында $P_l, l = 1,2, \dots, N$ процессорларына келесідей ақпарат таратылады: $\{m_l\}$ сандарына сәйкес таспалы матрицаларға тармақталған $B_u^* = \left\{b_{ji}(u)\right\}$ матрицасы және $A_l(u), f_l$ векторлары.

$n$ - ші итерацияда біз $u_n$ жуық шешімін алдық. Келесі $u_{n+1}$ жуық шешімін (3.2.5) формулалар бойынша есептейік. Ол үшін $P_l, l = 1,2, \dots, N$ процессорларына $u_n$ векторын таратамыз. Бұл вектор арқылы $B_n^l = B_{u_n}^l$ матрицасы және $A_n^l = A_l(u_n)$ векторлары есептелінеді.

1. Алдымен $A(u_n) - f$ векторын және оның нормасын есептейміз. $P_l$ процессоры $A_n^l - f_l$ айырманы есептеп, нәтижені түбірлік процессорға жібереді. Түбірлік процессор $P_{root}$ барлық процессорладан нәтижелерді қабылдап, олардың қосындысын $A(u_n) - f$ векторын есептейді. Кейін бұл вектордың нормасы есептелінеді.

2. $A(u_n) - f$ векторды әр $P_l$ процессоры алғаннан кейін, келесі көбейтіндіні $B_n^l(A(u_n) - f)$ есептеп, алған нәтижелерді түбірлік процессорға жібереді. $P_{root}$ процессоры:

$$B_{u_n}^*(A(u_n) - f) = \sum_{l=1}^{N} B_n^l(A(u_n) - f)$$

векторды және оның нормасын есептейді.

3. $P_{root}$ келесі векторды:

$$\omega_n = \frac{B^*_{u_n}(A(u_n) - f)}{\left|B^*_{u_n}(A(u_n) - f)\right|}$$

және

$$\varepsilon_n = \sqrt{|A(u_n) - f|^2}\,\frac{1}{4C^3(1 + f\vee)}$$

санды есептейді.

4. Келесі қадамда:

$$u_{n+1} = u_n - \varepsilon_n \omega_n$$

жуық шешімі есептелінеді.

5. Есептің дәлдігі тексеріледі. Егер дәлдік есептің шартына сәйкес болса, онда шешім ретінде $u_{n+1}$ жуық шешімі алынады. Кері жағдайда итерациялық үрдіс жалғасады. Сонымен қоса есептеу қателердің жинақталуы да есепке алынады.

## 4 ЕСЕПТЕУЛЕР НӘТИЖЕЛЕРІ

Бұл бөлімде сызықты алгебралық теңдеулер жүйесін жуықтап шешу үрдісін тізбектей және параллель есептеудің нәтижелері келтіріледі. Тесттік матрицаларды құрастыру әдістері баяндалған және сәйкес псевдокодтар (Қосымша А) келтірілген.

### 4.1 Есептеулер орындалған кластер мен арнайы қолданылған кітапхана туралы мәліметтер

Жұмыста ұсынылатын алгоритмдер арнайы кластерде тесттік жүйелер үшін жүзеге асырылды. Есептеулер орындалған кластердің сипаттамаларын келтірейік. Параллельдеу CPU мен GPU процессорларда орындалды. CPU типті процессорда ядроларға thread технологиясы қолданылды.

CPU сипаттамасы:

– core model: Intel(R) Xeon(R) CPU 2.00GHz;
– cpu MHz: 2000.186;
– cache size: 39424 KB;
– number of cores: 4.

GPU сипаттамасы:

– model: Tesla P100;
– RAM: 16 GB;
– Driver Version: 560.35.03.

CUDA Version: 12.6.

### 4.2 Есептеулер нәтижелері (жақсы шартталған матрицасы бар жүйелер үшін)

Есептеулер бөлімше 1.2 көрсетілген әдіс бойынша тізбектей орындалды. Бағдарламалық код (Қосымша Ә) келтірілген.

Құрастырылған әдіске сынақтарды жүргізу үшін тесттік түрлі өлшемді жақсы шартталған матрицалар пайдаланылды. Жақсы шартталған матрицалар ретінде бірқалыпты меншікті мәндері бар симметриялы матрица алынды. Оны құрастырудың әдісін келітрейік.

$Q$ ортогоналды матрица - кездейсоқ матрицаның $QR$ декомпозициясынан алынған болсын. Содан кейін бірқалыпты меншікті мәндері бар симметриялы $A$ матрицасы келесі түрде құрастырылады:

$$A = Q^T \Lambda Q, \tag{4.2.1}$$

мұнда $\Lambda$ - бірқалыпты үлестірілген мәндері бар диагоналды матрица:

$$\Lambda = diag(\lambda_1, \lambda_2, \dots, \lambda_n), \lambda_i \in [a, b], a > 0.$$

Бұндай матрицалар симметриялы және оң анықталған болып табылады. Симметриялы болады, себебі (4.2.1) формулада $\Lambda$ матрицасы диагоналды. Ал

оң анықталған - себебі оның барлық меншікті мәндері оң. Жүргізілген тесттік есептеулердің нәтижелері 4.1-кестеде бейнеленген.

Кесте 4.1 – 1-ші алгоритм (тізбектей)

| Матрица өлшемі $(n \times n)$ | $\lvert Ax - f \rvert$ мәні | Мақсатқа жетудің итерация нөмірі | Ағындар (threads) саны |
|---|---|---|---|
| 100 | 0.000118045 | 115 | 1 |
| 500 | 0.000110087 | 113 | 1 |
| 1000 | 0.000118319 | 115 | 1 |
| 10000 | – | – | – |

Есептеулер Бөлімше 1.3 көрсетілген параллельдеу әдісі бойынша орындалды. Бағдарламалық код (Қосымша Б) келтірілген.

Жүргізілген тесттік есептеулердің нәтижелері 4.2-кестеде бейнеленген.

Кесте 4.2 – 1-ші алгоритм (параллель OMP)

| Матрица өлшемі $(n \times n)$ | $\lvert Ax - f \rvert$ мәні | Мақсатқа жетудің итерация нөмірі | Ағындар (threads) саны |
|---|---|---|---|
| 100 | 0.000117819 | 115 | 4 |
| 500 | 0.000110759 | 113 | 4 |
| 1000 | 0.000119207 | 115 | 4 |
| 10000 | – | – | – |

Графиктік процессорда орындалған есептеулердің нәтижелері 4.3-кестеде келтірілген. Бағдарламалық код (Қосымша В) ұсынылған.

Кесте 4.3 – 1-ші алгоритм (параллель CUDA)

| Матрица өлшемі $(n \times n)$ | $\lvert Ax - f \rvert$ мәні | Мақсатқа жетудің итерация нөмірі | Ағындар (threads) саны |
|---|---|---|---|
| 100 | $5.25941e-05$ | 65 | 16 blocks × ((n + 16 - 1) / 16) = 7 |
| 500 | $4.37354e-05$ | 66 | 16 blocks × ((n + 16 - 1) / 16) = 32 |
| 1000 | $4.97848e-05$ | 68 | 16 blocks × ((n + 16 - 1) / 16) = 63 |
| 10000 | $4.41976e-05$ | 83 | 64 blocks × ((n + 64 - 1) / 64)=157 |

### 4.3 Есептеулер нәтижелері (нашар шартталған және сингулярлы матрицасы бар жүйелер үшін)

Әдісте қолданылатын параметрлер 4.4-кестеде ұсынылған.

Кесте 4.4 – Параметрлер мәндері

| р/с | Матрица өлшемі $(n \times n)$ | Матрица түрі | $\varepsilon$ | $\delta$ |
|---|---|---|---|---|
| 1 | 100 | Сингулярлы | 0.001 | 0.1 |
| 2 | 100 | нашар шартталған | 0.00001 | 0.1 |
| 3 | 500 | Сингулярлы | 0.001 | 0.1 |
| 4 | 500 | нашар шартталған | 0.00001 | 0.1 |
| 5 | 1000 | Сингулярлы | 0.000001 | 0.1 |
| 6 | 1000 | нашар шартталған | 0.000001 | 0.1 |
| 7 | 10000 | нашар шартталған | 0.0000001 | 0.1 |

Тесттік мысалдарда пайдаланылатын сингулярлы матрицаларды құрастыру әдісі төменде келтірілген.

Сингулярлы матрицаны құрастыру алгоритмі SVD жіктелуге негізделіп, нөлдік меншікті мәні бар матрицаны алады.

Сингулярлы мәндік декомпозициясы бойынша:

$$A = U\Sigma V^T \tag{4.3.1}$$

мұнда $\Sigma$ – сингулярлы мәндердің диагоналды матрицасы. Егер олардың кем дегенде біреуі нөлге тең болса, матрица сингулярлы болады:

$$\Sigma = diag(\sigma_1, \sigma_2, \dots, \sigma_{n-1}, 0).$$

Сонда кездейсоқ квадраттық матрицаның SVD жіктелуінен $\Sigma$-дағы соңғы элементін 0-ге алмастырып, үш матрицаның көбейтіндісі сингулярлы матрицаны береді.

Ал нашар шартталған матрицаларды құрастыруда келесі әдіс қолданылған. Сонда кездейсоқ квадраттық матрицаның SVD жіктелуінен $\Sigma$-дағы соңғы элементін 0-ге алмастырып, үш матрицаның көбейтіндісі сингулярлы матрицаны береді.

Нашар шартталған матрицаны құрастыру алгоритмі сингулярлы мәндерінің үлкен ауытқуларын пайдалануға негізделген. Ол үшін $\sigma_{min}$ мен $\sigma_{max}$ сингулярлы мәндері үлкен қашықтықта орналастыру арқылы таңдап алуға болады.

Сингулярлы мәндері бар $\Sigma$ матрицаның элементтерін реті бойынша алыс етіп алуға болады:

$$\Sigma = diag(\sigma_1, \sigma_2, \dots, \sigma_n),$$

мұнда $\sigma_i$ геометриялық прогрессияны құрайды, мысалы:

$$\sigma_i = \sigma_1 \cdot r^{i-1}, r \ll 1.$$

Содан кейін нәтижелі нашар шартталған матрица SVD (4.3.1) көмегімен құрастырылады.

Есептеулер Бөлімше 2.4 көрсетілген әдіс бойынша тізбектей орындалды. Бағдарламалық код (Қосымша Г) келтірілген.

Жүргізілген тесттік есептеулердің нәтижелері 4.5-кестеде бейнеленген.

Кесте 4.5 – 2-ші алгоритм (тізбектей)

| Матрица өлшемі $(n \times n)$ | Матрица түрі | $\|Ax - f\|$ мәні | Мақсатқа жетудің итерация нөмірі | Ағындар (threads) саны | Жұмсалған уақыты (msec) |
|---|---|---|---|---|---|
| 100 | сингулярлы | 1.03247 | > 1000 | 1 | 1 071 |
| 100 | нашар шартталған | 9.51426 | > 1000 | 1 | 355 |
| 500 | сингулярлы | 4.65388 | > 1000 | 1 | 129 110 |
| 500 | нашар шартталған | 20.3827 | > 1000 | 1 | 21 406 |
| 1000 | сингулярлы | 3.46996 | > 1000 | 1 | 293 478 |
| 1000 | нашар шартталған | 28.6048 | > 1000 | 1 | 133 460 |

Есептеулер Бөлімше 2.4 көрсетілген параллельдеу әдісі бойынша орындалды. Бағдарламалық код (Қосымша Ғ) келтірілген.

Жүргізілген тесттік есептеулердің нәтижелері 4.6-кестеде бейнеленген.

Кесте 4.6 – 2-ші алгоритм (параллель OMP)

| Матрица өлшемі $(n \times n)$ | Матрица түрі | $\|Ax - f\|$ мәні | Мақсатқа жетудің итерация нөмірі | Ағындар (threads) саны | Жұмсалған уақыты (msec) |
|---|---|---|---|---|---|
| 100 | сингулярлы | 1.03247 | > 1000 | 4 | 698 |
| 100 | нашар шартталған | 9.50772 | > 1000 | 4 | 276 |
| 500 | сингулярлы | 2.74712 | > 1000 | 4 | 19 558 |
| 500 | нашар шартталған | 20.2943 | > 1000 | 4 | 17 453 |
| 1000 | сингулярлы | 3.46996 | > 1000 | 4 | 203 291 |
| 1000 | нашар шартталған | 28.3898 | > 1000 | 4 | 159 544 |

Есептеулер Бөлімше 2.4 көрсетілген параллельдеу әдісі бойынша орындалды. Бағдарламалық код (Қосымша Д) келтірілген.

Жүргізілген тесттік есептеулердің нәтижелері 4.7-кестеде бейнеленген.

Кесте 4.7 – 2-ші алгоритм (параллель CUDA)

| Матрица өлшемі ($n \times n$) | Матрица түрі | $\|Ax - f\|$ мәні | Мақсатқа жетудің итерация нөмірі | Ағындар (threads) саны | Жұмсалған уақыты (msec) |
|---|---|---|---|---|---|
| 100 | сингулярлы | 1.03345 | > 1000 | 16 blocks × ((n + 16 - 1) / 16)=7 | 1 177 |
| 100 | нашар шартталған | 9.5066 | > 1000 | 16 blocks × ((n + 16 - 1) / 16)=7 | 465 |
| 500 | сингулярлы | 2.74493 | > 1000 | 16 blocks × ((n + 16 - 1) / 16)=32 | 4 166 |
| 500 | нашар шартталған | 20.2608 | > 1000 | 16 blocks × ((n + 16 - 1) / 16)=32 | 3 701 |
| 1000 | сингулярлы | 3.45769 | > 1000 | 16 blocks × ((n + 16 - 1) / 16)=63 | 13 413 |
| 1000 | нашар шартталған | 28.4224 | > 1000 | 16 blocks × ((n + 16 - 1) / 16)=63 | 9 414 |
| 10000 | сингулярлы | 9.91053 | > 1000 | 64 blocks × ((n + 64 - 1) / 64)=157 | 544 865 |
| 10000 | нашар шартталған | 89.2931 | > 1000 | 64 blocks × ((n + 64 - 1) / 64)=157 | 303 422 |

### 4.4 Әдістердің қолданыстары

Құрастырылған әдістер келесідей есептерді шешуде өз қолданыстарын тапқан:

1. Жанармай газының шығындарын азайту үшін компрессорлық станция жұмысын модельдеу [37-40].

2. Көлденең тармақталу әдісін қолдана отырып, құбыр желілері арқылы табиғи газды тасымалдау процесін математикалық модельдеу [41].

# 5 ЖУЫҚТАП ШЕШУ ӘДІСТЕРІМЕН САЛЫСТЫРУ

Бұл бөлімде жұмыста ұсынлатын сызықты алгебралық теңдеулер жүйесін жуықтап шешу үрдістері басқа әдістермен салыстырулар келтіріледі.

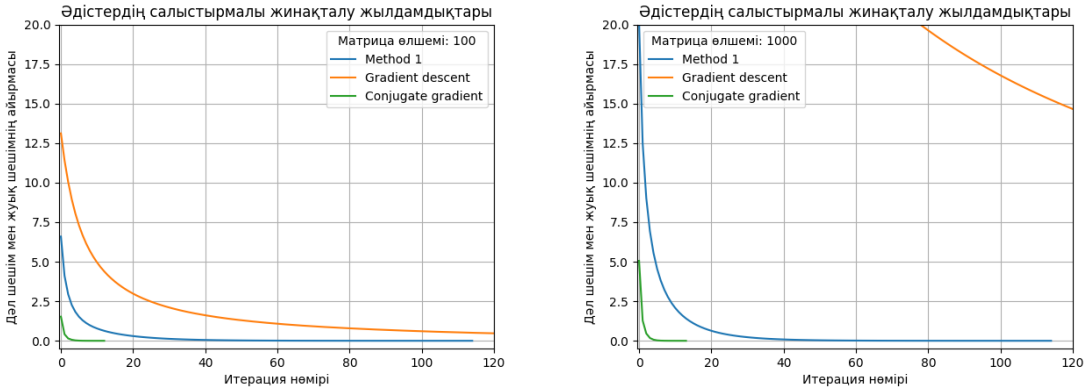
## 5.1 Жақсы шартталған матрицасы бар жүйені шешудің 1-алгоритмін градиенттік түсу мен түйіндес градиенттер әдістерімен салыстыру

Градиенттік түсу әдісі [26, с. 3-284] сызықты алгебралық теңдеулер жүйесін шешуде қолданылады. Соңғы уақытта бұл әдістің модификацияларының санының көбеюі Машиналық оқытуда қолдануына байланысты болып тұр. Терең оқыту модельдерін үйретуде Адам оптимайзер модификациясы [27, с. 3-227] жиі қолданылады.

Құрастырылған әдістерді басқа әдістермен салыстыру үшін екінші - алгоритм түйіндес градиенттер әдісі қарастырылады. Түйіндес градиенттер әдісі - өзара түйіндес және оң анықталған матрицасы бар жүйені жуықтап шешу әдісі. Тек нашар шартталған мен сингулярлы матрицалар үшін бұл әдіс тұрақсыз болып табылады.

Жинақталу жылдамдықтарын тесттік мысалдар арқылы тексерілді. Суретте үш әдістің $|Ax - f|$ қалдығының азаю графигі бейнеленген.

Үш әдіспен жуықтап шешудің нәтижелері 5.1-суретте келтірілген.




а ә

Сурет 5.1 – Үш түрлі әдістерімен жуықтап шешудің салыстырулары

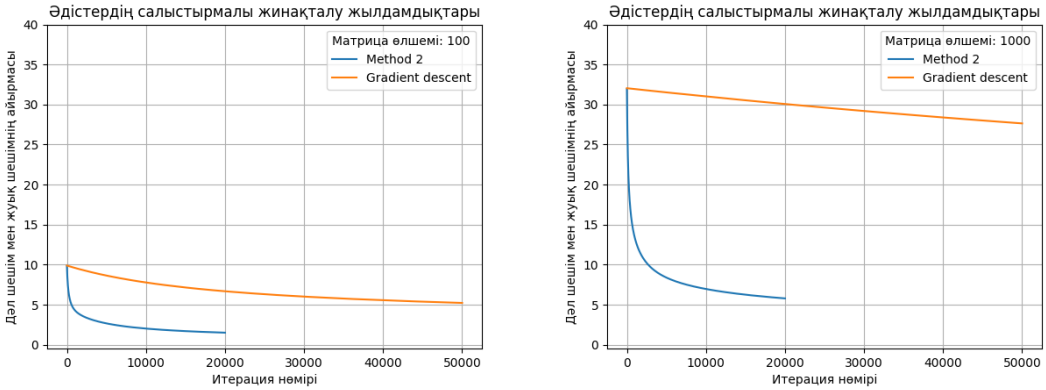
## 5.2 Нашар шартталған немесе сингулярлы матрицасы бар жүйені шешудің 2-алгоритмін градиенттік түсу әдісімен салыстыру

Екі түрлі матрицалар үшін есептеулер жүргізілді. Алдымен сингулярлы, яғни анықтаушы нөлге тең матрицасы бар сызықты жүйелер үшін жуықтап шешу әдістері қолданылды.

Сингулярлы мен нашар шартталған матрицалар үшін түйіндес градиенттер әдісі қолданылған жоқ, себебі құрастыру әдісі бойынша негізгі шарт симметриялылық пен оң анықталғандық қасиеттері жалпы жағдайда орындалмайды. Құрастырылған сингулярлы матрицалар симметриялы болмауы мүмкін, себебі (4.3.1) формулада $U$ мен $V$ матрицалар бірдей емес жалпы

жағдайда. Жәнеде оң анықталғандық шарты да бұзылады, бір меншікті мәні 0-ге тең. Ал нашар шартталған матрицалар оң анықталған болғанымен, (4.3.1) формуласындағы $U \neq V$ болуы мүмкін.

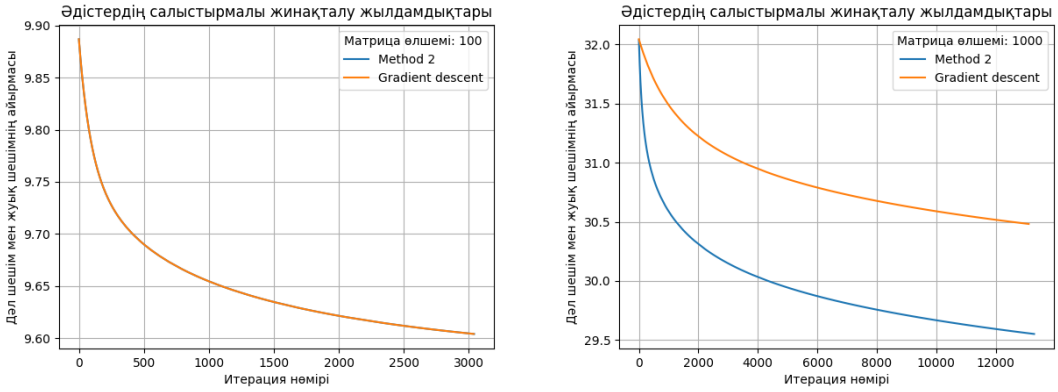
Екі әдіспен жуықтап шешудің нәтижелері 5.2-суретте келтірілген.




а ә

Сурет 5.2 – Екі түрлі әдістерімен жуықтап шешудің салыстырулары

Нашар шартталған матрицасы бар сызықты жүйелер үшін жуықтап шешу әдістері қолданылды. Екі әдіспен жуықтап шешудің нәтижелері 5.3-суретте келтірілген.




а ә

Сурет 5.3 – Екі түрлі әдістерімен жуықтап шешудің салыстырулары

# ҚОРЫТЫНДЫ

Диссертациялық жұмыста алынған негізгі нәтижелер келесідей:

– шенелген, жақсы шартталған матрицасы үшін сызықты алгебралық теңдеулер жүйесін жуықтап шешу әдісі құрастырылып, оның паралелльдеу алгоритмі жасалды;

– нашар шартталған немесе сингулярлы матрицасы бар сызықты алгебралық теңдеулер жүйесін жуықтап шешу әдісі және оның параллельдеу алгоритмі құрастырылды;

– сызықты емес теңдеуді шешудің жуықтап шешу әдісі құрастырылып, оның паралелль алгоритмі жасалды.

Диссертациялық жұмыста алынған ғылыми нәтижелердің жаңалығы алгебралық теңдеулер жүйелерін жуықтап шешу үрдісін параллель есептеу алгоритмдеріне сәйкес құрастыруы болып табылады.

Диссертациялық жұмыс нәтижелерінің теориялық және практикалық маңызы бар. Алынған нәтижелерді пайдаланып, сызықты алгебралық жүйелерді матрицасы қайтымды немесе нашар шартталған жағдайда параллель есептеуге болады, сонымен қатар сызықты емес алгебралық жүйені параллельдеудің де алгоритмі келтірілген.

## ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ


1 Grigori L., Demmel J., Xiang H. Communication-Avoiding Gaussian Elimination // Procced. conf. “Supercomputing”. – Berkeley, 2008. – P. 1-12.

2 Burnim J., Sen K. Asserting and checking determinism for multithreaded programs // ESEC/FSE ’09: proceed. of the the 7th joint meeting of the European Software Engineering conf. and the ACM SIG- SOFT sympos. on the Foundations of Software Engineering. – NY., 2009. – P. 3-12.

3 Higham N.J. Accuracy and stability of numerical algorithms // Journal of the American Statistical Association. – 2002. – Vol. 94, Issue 445. – P. 349-350.

4 Asanovic K., Bodik R., Demmel J. et al. A view of the parallel computing landscape // Communications of ACM. – 2009. – Vol. 52, Issue 10. – P. 56-67.

5 Demmel J., Grigori L., Xiang H. CALU: A Communication Optimal LU Factorization Algorithm // Siam J. Matrix Anal. appl. – 2011. – Vol. 32, Issue 4. – P. 1317-1350.

6 Demmel J., Hoemmen M., Mohiyuddin M. et al. Avoiding communication in sparse matrix computations // Procced. IEEE internat. Parallel and Distributed Processing sympos. (IPDPS 08). – Miami, 2008. – P. 1-13.

7 Kamil S., Oliker L., Pinar A. et al. Communication Requirements and Interconnect Optimization for High-End Scientific Applications // IEEE Transactions on Parallel and Distributed Systems. – 2010. – Vol. 21, Issue 2. – P. 188-202.

8 Mattson T., Sanders B., Massingill B. Patterns for parallel programming. – Ed. 1st. – Bristol: Addison-Wesley Professional, 2004. – 355 p.

9 Patterson D., Gannon D., Wrinn M. The Berkeley Par Lab: Progress in the Parallel Computing Landscape. – NY., 2013. – 1719 p.

10 Ortega J.M. Introduction to parallel and vector solution of linear systems. – NY., 1982. – 305 p.

11 Golub G.H., van Loan Ch.F. Matrix Computations. – Ed. 3rd. – Baltimore: Johns Hopkins, 1996. – 694 p.

12 Bertsekas D.P., Tsitsiklis J.N. Parallel and Distributed Computation: Numerical Methods. – Belmont, 1997. – 730 p.

13 Brucel P.I. Parallel programming an introduction. – NY., 1993. – 270 p.

14 Heroux M., Raghavan P., Simon H. Parallel Processing for Scientific Computing. – Philadelphia, 2006. – 676 p.

15 Stewart G.W. Introduction to Matrix Computations. – NY., 1973. – 422 p.

16 Самарский А.А., Гулин А.В. Численные методы: учеб. пос. – М., 1989. – 432 с.

17 Воеводин В.В., Воеводин В.В. Параллельные вычисления. – СПб.: БХВ Петербург, 2002. – 608 с.

18 Strang G. Linear Algebra and Its Applications. – Belmont, 2006. – 487 p.

19 Traub J.F. Iterative Methods for the Solution of Equations. – New Jersey: Prentice-Hall, 1964. – 310 p.

20 Otelbaev M., Tuleuov B., Zhussupova D. On a Method of Finding Approximate Solutions of Ill-conditioned Algebraic Systems and Parallel Computation // Eurasian Mathematical J. – 2011. – Vol. 2, Issue 1. – P. 149-151.

21 Отелбаев М., Жусупова Д., Тулеуов Б. Распараллеливание линейной алгебраической системы с обратимой матрицей // Вестник Башкирского университета. – 2011. – Т. 16. – С. 1129-1133.

22 Trefethen L. N., Bau D. Numerical Linear Algebra. – Philadelphia, 1997. – 361 p.

23 Watkins D. S. Fundamentals of Matrix Computation. – NY., 1991. – 618 p.

24 Балдыбек Ж., Отелбаев М. К задачам распараллеливания // Математический журнал. – 2011. – Т. 11, №39. – С. 53-58.

25 Gohberg I., Goldberg S., Kashoek Marinus A. Basic Classes of Linear Operators. – Boston: Birkhauser Verlag, 2003. – 423 p.

26 Тихонов А.Н., Арсенин В.Я. Методы решения некорректных задач. – М.: Наука, 1979. – 285 с.

27 Тихонов А.Н., Гончарский А.В., Степанов В.В. и др. Численные методы решения некорректных задач. – М., 2012. – 228 с.

28 Отелбаев М., Сейткулов Е., Жусупова Д. және т.б. Корреляциялық геокосмостық тәуелділікті математикалық модельдеу және болжаудың әдістері // ЕҰУ хабаршысы. – 2012. – №4(89). – Б. 6-14.

29 Отелбаев М., Тулеуов Б., Жусупова Д. Один метод распараллеливания процесса решения нелинейных алгебраических систем // Вестник ВКГТУ им. Д. Серикбаева. – 2013. – Ч. 1. – С. 233-236.

30 Otelbaev M., Tuleuov B., Zhussupova D. On an Orthogonal Method of Finding Approximate Solutions of Ill-Conditioned Algebraic Systems and Parallel Computation // Procced. IAENG WCE-2013 conf. – London, 2013. – P. 54-58.

31 Otelbaev M., Tuleuov B., Zhussupova D. On a Method of Parallel Computation for Solving Linear Algebraic System with Ill-Conditioned Matrix // Journal of Pure and Applied Mathematics. – 2013. – Vol. 4, Issue 2. – P. 115-124.

32 Жусупова Д., Отелбаев М. Один метод приближенного решения системы линейных алгебраических уравнений с плохо обусловленной матрицей // Современные методы математической физики и их приложения: тез. докл. республ. науч. конф. – Ташкент, 2015. – С. 60.

33 Otelbaev M., Sultanaev Ya.T., Zhusupova D. Criterion for the Boundedness and Compactness of a Class of Sets in L[0;1) // Differential Equations. – 2019. – Vol. 55. – P. 1301-1304.

34 Канторович Л.В. Функциональный анализ и прикладная математика // УМН. – 1948. – Т. 3, №6(28). – С. 89-185.

35 Жусупова Д. Сызықты емес теңдеуді шешудің параллельдеу әдісі // Вестник ЕНУ. – 2013. – №(95). – С. 113-116.

36 Zhussupova D., Abdikalykova ZБ. Modeling the operation of a gas compressor station to minimize fuel costs for protection against surge zone // Procced. “APMAS-2024” conf. – Muğla, 2024. – P. 124-125.

37 Zhussupova D., Otelbaev M., Burgumbayeva S. Modeling Gas Compressor Station Operation to Minimize Fuel Costs for Surge Zone Protection // International Journal of Rotating Machinery. – 2024. – Vol. 2024. – P. 5560308-1-5560308-18.

38 Отелбаев М., Рысбайулы Б. Приближенный метод решения нелинейных уравнений: итерационный процесс, оценка скорости сходимости // Докл. академии наук. – 1999. – Т. 5. – С. 20-25.

39 Жусупова Д., Абдикаликова З., Ыдырыс А. и др. Моделирование работы газокомпрессорной станции для минимизации затрат топлива на защиту от помпажной зон // Сб. тр. конф. «Современные проблемы дифференциальных уравнений и их приложения». – Ташкент, 2023. – С. 159-161.

40 Burgumbayeva S., Zhussupova D. Minimizing compressor fuel cost on large natural gas pipeline transmission networks // AIP Conf. Proc. – 2023. – Vol. 2879, Issue 1. – P. 050001.

41 Burgumbayeva S., Zhussupova D., Koshkarova B. Mathematical modelling of the process of natural gas transportation via pipe networks using crossing branch method // Journal of Mathematics, Mechanics and Computer Science. – 2024. – Vol. 121, Issue 1. – P. 12-26.

# ҚОСЫМША А

Тесттік матрицаларды құрастырудың программалық коды

```
def symmetric_uniform_eigen(n, l_min=1, l_max=2):
Q, _ = qr(np.random.randn(n, n))
L = np.diag(np.linspace(l_min, l_max, n))
return Q.T @ L @ Q

def singular_svd(n):
U, _, Vt = svd(np.random.randn(n, n))
Sigma = np.diag(np.linspace(1, 0, n))  # Last singular value = 0
return U @ Sigma @ Vt.
def ill_conditioned_svd(n, ratio=1e10):
U, _, Vt = svd(np.random.randn(n, n))
Sigma = np.diag(np.geomspace(1, 1/ratio, n))
# Геометрическая прогрессия
return U @ Sigma @ Vt
```

## ҚОСЫМША Ә

Бірінші алгоритмнің тізбектей программалық коды

```
//compile command g++ bolim1.cpp -o bolim1.bin
//execute command ./bolim1.bin 100

#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>
#include <chrono>
#include <omp.h>

#define MAX_ITERATIONS 50000

int main(int argc, char *argv[]){
   int NN = atoi(argv[1]);
   float matA[NN][NN];
   float vecf[NN], xVec[NN];
   float omega[NN] = {};
   float vec[NN] = {};
   float normValNumerator = 0, normValDenominator = 0;
   float epsilon, val = 0, valPrev = 0, tol = 1e-05;
   unsigned it = 0;
   std::string readFile = std::string("/kaggle/working/data_") + std::string(argv[1]) +
std::string("x") + std::string(argv[1]) + std::string(".txt");
   std::ifstream readDataFile(readFile);

   std::ofstream writeLogsFile("/kaggle/working/logs_bolim1_sequential",
std::ios_base::app);

   std::string writeResidualFile = "/kaggle/working/residual_output_sequential_" +
std::string(argv[1]);
   std::ofstream writeDataFile(writeResidualFile, std::ios_base::trunc);

   if (readDataFile.is_open()) {
      for (int k = 0; k < NN; k++)
         for (int l = 0; l < NN; l++)
            readDataFile >> matA[k][l];
      for (int l = 0; l < NN; l++) {
         readDataFile >> vecf[l];
      }
      //fill initial approximation
```

```
        for (int l = 0; l < NN; l++) {
            readDataFile >> xVec[l];
            xVec[l] += 0.05;
        }
        readDataFile.close();
    }
    else std::cout << "Unable to open read file";
    if (writeLogsFile.is_open()) {
        writeLogsFile << "-------S E Q U E N T I A L-------\n";
        writeLogsFile << "dimension=" << NN << "x" << NN << std::endl;
    }
    else std::cout << "Unable to open output file";
    std::chrono::steady_clock::time_point start =
std::chrono::steady_clock::now();//zasekayem vremya
    int i,j;
    for (i = 0; i < NN; i++) {
        omega[i] = 0;
        for (j = 0; j < NN; j++) {
            omega[i] += matA[i][j] * xVec[j];
        }
    }

    for (i = 0; i < NN; i++)
        vec[i] = omega[i] - vecf[i];

    while(true){
        it ++;
        normValNumerator = 0;
        normValDenominator = 0;

        for (i = 0; i < NN; i++) {
            omega[i] = 0;
            for (j = 0; j < NN; j++) {
                omega[i] += matA[j][i] * vec[j];
            }
        }
        for (i = 0; i < NN; i++)
            normValNumerator += omega[i] * omega[i];

        for (i = 0; i < NN; i++) {
            vec[i] = 0;
            for (j = 0; j < NN; j++) {
                vec[i] += matA[i][j] * omega[j];
            }
```

```
            normValDenominator += vec[i] * vec[i];
        }
        epsilon = normValNumerator / normValDenominator;
        for (i = 0; i < NN; i++) {
            xVec[i] -=  epsilon * omega[i];
        }
        for (i = 0; i < NN; i++) {
            omega[i] = 0;
            for (j = 0; j < NN; j++) {
                omega[i] += matA[i][j] * xVec[j];
            }
        }
        val = 0;
        for (i = 0; i < NN; i++) {
            vec[i] = omega[i] - vecf[i];
            val += vec[i] * vec[i];
        }
        val = sqrt(val);
        if (it > MAX_ITERATIONS){
            if (writeLogsFile.is_open()) {
                writeLogsFile << "No convergence! Iterations number greater than max
value " << MAX_ITERATIONS << std::endl;
                writeLogsFile << "|A x_" << it << " - f| = " << val << std::endl;
            }
            else std::cout << "Unable to open output file";
            std::cout << "No convergence! Iterations number greater than max value " <<
MAX_ITERATIONS << std::endl;
            std::cout << "|A x_" << it << " - f| = " << val << std::endl;
            break;
        }
        if (writeDataFile.is_open()) {
                writeDataFile << val << " ";
        }
        if (fabs(val - valPrev) < tol || val < tol){

            if (writeLogsFile.is_open()) {
                writeLogsFile << "Process converged! |A x_" << it << " - f| = " << val <<
std::endl;
            }
            else std::cout << "Unable to open output file";
            std::cout << "Process converged! |A x_" << it << " - f| = " << val << std::endl;
            break;
        }
        valPrev = val;
```

```
    }
    std::chrono::steady_clock::time_point end =
std::chrono::steady_clock::now();//ostanavlivayem schetchik
    std::cout << "time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
milliseconds" << std::endl;
    if (writeLogsFile.is_open()) {
        writeLogsFile << "time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
milliseconds\n\n";
        writeLogsFile.close();
    }
    else std::cout << "Unable to open output file";
    return 0;
}
```

## ҚОСЫМША Б

OMP технологияны қоладанып бірінші алгоритмнің параллель программалық коды

```
//compile command g++ -fopenmp bolim1_OMP.cpp -o bolim1_OMP.bin
//execute command ./bolim1_OMP.bin 100

#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>
#include <chrono>
#include <omp.h>

#define MAX_ITERATIONS 50000

int main(int argc, char *argv[]) {
   int NN = atoi(argv[1]);
   float matA[NN][NN];
   float vecf[NN], xVec[NN];
   float omega[NN] = {};
   float vec[NN] = {};
   float normValNumerator = 0, normValDenominator = 0;
   float epsilon, val = 0, valPrev = 0, tol = 1e-05;
   unsigned it = 0;
   std::string readFile = std::string("/kaggle/working/data_") + std::string(argv[1]) +
std::string("x") + std::string(argv[1]) + std::string(".txt");
   std::ifstream readDataFile(readFile);

   std::ofstream writeLogsFile("/kaggle/working/logs_bolim1_parallel",
std::ios_base::app);

   std::string writeResidualFile = "/kaggle/working/residual_output_parallel_" +
std::string(argv[1]);
   std::ofstream writeDataFile(writeResidualFile, std::ios_base::trunc);

   if (readDataFile.is_open()) {
      for (int k = 0; k < NN; k++)
         for (int l = 0; l < NN; l++)
            readDataFile >> matA[k][l];
      for (int l = 0; l < NN; l++) {
         readDataFile >> vecf[l];
      }
```

```
        for (int l = 0; l < NN; l++) {
            readDataFile >> xVec[l];
            xVec[l] += 0.05;
        }
        readDataFile.close();
    }
    else std::cout << "Unable to open read file";

    if (writeLogsFile.is_open()) {
        writeLogsFile << "-------P A R A L L E L-------\n";
        writeLogsFile << "dimension=" << NN << "x" << NN << std::endl;
    }
    else std::cout << "Unable to open output file";

    std::chrono::steady_clock::time_point start = std::chrono::steady_clock::now();

    int i, j;

    // Initial omega calculation
    #pragma omp parallel for private(j) schedule(static)
    for (i = 0; i < NN; i++) {
        omega[i] = 0;
        for (j = 0; j < NN; j++) {
            omega[i] += matA[i][j] * xVec[j];
        }
    }

    #pragma omp parallel for schedule(static)
    for (i = 0; i < NN; i++)
        vec[i] = omega[i] - vecf[i];

    while (true) {
        it++;
        normValNumerator = 0;
        normValDenominator = 0;

        #pragma omp parallel for private(j) reduction(+:normValNumerator)
schedule(static)
        for (i = 0; i < NN; i++) {
            omega[i] = 0;
            for (j = 0; j < NN; j++) {
                omega[i] += matA[j][i] * vec[j];
            }
            normValNumerator += omega[i] * omega[i];
```

```
        }

        #pragma omp parallel for private(j) reduction(+:normValDenominator)
schedule(static)
        for (i = 0; i < NN; i++) {
            vec[i] = 0;
            for (j = 0; j < NN; j++) {
                vec[i] += matA[i][j] * omega[j];
            }
            normValDenominator += vec[i] * vec[i];
        }

        epsilon = normValNumerator / normValDenominator;

        #pragma omp parallel for schedule(static)
        for (i = 0; i < NN; i++) {
            xVec[i] -= epsilon * omega[i];
        }

        #pragma omp parallel for private(j) schedule(static)
        for (i = 0; i < NN; i++) {
            omega[i] = 0;
            for (j = 0; j < NN; j++) {
                omega[i] += matA[i][j] * xVec[j];
            }
        }

        val = 0;

        #pragma omp parallel for reduction(+:val) schedule(static)
        for (i = 0; i < NN; i++) {
            vec[i] = omega[i] - vecf[i];
            val += vec[i] * vec[i];
        }

        val = sqrt(val);

        if (it > MAX_ITERATIONS) {
            if (writeLogsFile.is_open()) {
                writeLogsFile << "No convergence! Iterations number greater than max
value " << MAX_ITERATIONS << std::endl;
                writeLogsFile << "|A x_" << it << " - f| = " << val << std::endl;
            }
            std::cout << "No convergence! Iterations number greater than max value " <<
```

```
MAX_ITERATIONS << std::endl;
            std::cout << "|A x_" << it << " - f| = " << val << std::endl;
            break;
        }

        if (writeDataFile.is_open()) {
            writeDataFile << val << " ";
        }

        if (fabs(val - valPrev) < tol || val < tol) {
            if (writeLogsFile.is_open()) {
                writeLogsFile << "Process converged! |A x_" << it << " - f| = " << val <<
std::endl;
            }
            std::cout << "Process converged! |A x_" << it << " - f| = " << val << std::endl;
            break;
        }

        valPrev = val;
    }

    std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();

    std::cout << "time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
milliseconds" << std::endl;

    if (writeLogsFile.is_open()) {
        writeLogsFile << "time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
milliseconds\n\n";
        writeLogsFile.close();
    }

    return 0;
}
```

## ҚОСЫМША В

CUDA технологияны қоладанып бірінші алгоритмнің параллель программалық коды

```
// compile command: nvcc -o bolim1_cuda.bin bolim1_cuda.cu
// execute command: ./bolim1_cuda.bin 100 256

#include <cuda_runtime.h>
#include <iostream>
#include <cmath>
#include <fstream>
#include <string>
#include <chrono>
#include <fstream>
#include <vector>

#define MAX_ITERATIONS 50000
#define TOL 1e-05

__global__ void matrixVectorMultiply(float *matA, float *vec, float *result, int NN)
{
    int row = blockIdx.x * blockDim.x + threadIdx.x;
    if (row < NN) {
        float sum = 0.0f;
        for (int col = 0; col < NN; col++) {
            sum += matA[row * NN + col] * vec[col];
        }
        result[row] = sum;
    }
}

__global__ void vectorSubtract(float *vec1, float *vec2, float *result, int NN) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < NN) {
        result[idx] = vec1[idx] - vec2[idx];
    }
}

__global__ void computeNorm(float *vec, float *result, int NN) {
    __shared__ float cache[256];
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int threadIdxLocal = threadIdx.x;

    float temp = (idx < NN) ? vec[idx] * vec[idx] : 0.0f;
    cache[threadIdxLocal] = temp;
```

```
    __syncthreads();

    // Reduction within the block
    for (int stride = blockDim.x / 2; stride > 0; stride /= 2) {
        if (threadIdxLocal < stride) {
            cache[threadIdxLocal] += cache[threadIdxLocal + stride];
        }
        __syncthreads();
    }

    if (threadIdxLocal == 0) {
        atomicAdd(result, cache[0]);
    }
}

__global__ void updateSolution(float *xVec, float *omega, float epsilon, int NN) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < NN) {
        xVec[idx] -= epsilon * omega[idx];
    }
}

int main(int argc, char *argv[]) {
    int NN = atoi(argv[1]);
    size_t sizeMatrix = NN * NN * sizeof(float);
    size_t sizeVector = NN * sizeof(float);

    // Host memory allocation
    float *matA = new float[NN * NN];
    float *matA_con = new float[NN * NN];
    float *vecf = new float[NN];
    float *xVec = new float[NN];
    float *omega = new float[NN]();
    float *vec = new float[NN]();

    // Device memory allocation
    float *d_matA, *d_matA_con, *d_vecf, *d_xVec, *d_omega, *d_vec, *d_norm;
    cudaMalloc(&d_matA, sizeMatrix);
    cudaMalloc(&d_matA_con, sizeMatrix);
    cudaMalloc(&d_vecf, sizeVector);
    cudaMalloc(&d_xVec, sizeVector);
    cudaMalloc(&d_omega, sizeVector);
    cudaMalloc(&d_vec, sizeVector);
    cudaMalloc(&d_norm, sizeof(float));

    // Load data from file
```

```
   std::string readFile = std::string("/kaggle/working/data_") + std::string(argv[1]) +
std::string("x") + std::string(argv[1]) + std::string(".txt");
   std::ifstream readDataFile(readFile);

   std::ofstream writeLogsFile("/kaggle/working/logs_bolim1_cuda",
std::ios_base::app);

   std::string writeResidualFile = "/kaggle/working/residual_output_cuda_" +
std::string(argv[1]);
   std::ofstream writeDataFile(writeResidualFile, std::ios_base::trunc);
   if (readDataFile.is_open()) {
      for (int i = 0; i < NN; i++)
         for (int j = 0; j < NN; j++)
            readDataFile >> matA[i * NN + j];
      for (int i = 0; i < NN; i++)
         readDataFile >> vecf[i];
      for (int i = 0; i < NN; i++) {
         readDataFile >> xVec[i];
         xVec[i] += 0.05;
      }
      readDataFile.close();
   } else {
      std::cerr << "Unable to open file!" << std::endl;
      return -1;
   }
   for (int i = 0; i < NN; i++)
         for (int j = 0; j < NN; j++)
            matA_con[i * NN + j] = matA[j * NN + i];

   // Copy data to device
   cudaMemcpy(d_matA, matA, sizeMatrix, cudaMemcpyHostToDevice);
   cudaMemcpy(d_matA_con, matA_con, sizeMatrix, cudaMemcpyHostToDevice);
   cudaMemcpy(d_vecf, vecf, sizeVector, cudaMemcpyHostToDevice);
   cudaMemcpy(d_xVec, xVec, sizeVector, cudaMemcpyHostToDevice);

   unsigned int it = 0;
   float val = 0, valPrev = 0, epsilon = 0;

   dim3 blockDim(std::atoi(argv[2]));
   dim3 gridDim((NN + blockDim.x - 1) / blockDim.x);

   // Matrix-vector multiplication
   matrixVectorMultiply<<<gridDim, blockDim>>>(d_matA, d_xVec, d_omega,
NN);
   cudaDeviceSynchronize();

   // Compute residual vector
```

```
    vectorSubtract<<<gridDim, blockDim>>>(d_omega, d_vecf, d_vec, NN);
    cudaDeviceSynchronize();

    if (writeLogsFile.is_open()) {
        writeLogsFile << "-------C U D A-------\n";
        writeLogsFile << "dimension=" << NN << "x" << NN << std::endl;
    }
    else std::cout << "Unable to open output file";
    std::chrono::steady_clock::time_point start =
std::chrono::steady_clock::now();//zasekayem vremya

    while (true) {
        it++;

        // Compute omega vector omega=A^T*(Ax-f)
        matrixVectorMultiply<<<gridDim, blockDim>>>(d_matA_con, d_vec,
d_omega, NN);
        cudaDeviceSynchronize();

        // Compute numerator for epsilon
        float normNumerator = 0;
        cudaMemcpy(d_norm, &normNumerator, sizeof(float),
cudaMemcpyHostToDevice);
        computeNorm<<<gridDim, blockDim>>>(d_omega, d_norm, NN);
        cudaMemcpy(&normNumerator, d_norm, sizeof(float),
cudaMemcpyDeviceToHost);
        //

        // Compute A * omega vector
        matrixVectorMultiply<<<gridDim, blockDim>>>(d_matA, d_omega, d_vec,
NN);
        cudaDeviceSynchronize();

        // Compute denominator for epsilon
        float normDenominator = 0;
        cudaMemcpy(d_norm, &normDenominator, sizeof(float),
cudaMemcpyHostToDevice);
        computeNorm<<<gridDim, blockDim>>>(d_vec, d_norm, NN);
        cudaMemcpy(&normDenominator, d_norm, sizeof(float),
cudaMemcpyDeviceToHost);

        epsilon = normNumerator / normDenominator;

        // Update solution
        updateSolution<<<gridDim, blockDim>>>(d_xVec, d_omega, epsilon, NN);
        cudaDeviceSynchronize();
```

```
        // Matrix-vector multiplication
        matrixVectorMultiply<<<gridDim, blockDim>>>(d_matA, d_xVec, d_omega,
NN);
        cudaDeviceSynchronize();

        // Compute residual vector
        vectorSubtract<<<gridDim, blockDim>>>(d_omega, d_vecf, d_vec, NN);
        cudaDeviceSynchronize();

        val = 0;
        cudaMemcpy(d_norm, &val, sizeof(float), cudaMemcpyHostToDevice);
        computeNorm<<<gridDim, blockDim>>>(d_vec, d_norm, NN);
        cudaMemcpy(&val, d_norm, sizeof(float), cudaMemcpyDeviceToHost);

        if (it > MAX_ITERATIONS){
           if (writeLogsFile.is_open()) {
              writeLogsFile << "No convergence! Iterations number greater than max
value " << MAX_ITERATIONS << std::endl;
              writeLogsFile << "|A x_" << it << " - f| = " << val << std::endl;
           }
           else std::cout << "Unable to open output file";
           std::cout << "No convergence! Iterations number greater than max value " <<
MAX_ITERATIONS << std::endl;
           std::cout << "|A x_" << it << " - f| = " << val << std::endl;
           break;
        }
        if (writeDataFile.is_open()) {
              writeDataFile << val << " ";
        }
        if (fabs(val - valPrev) < TOL || val < TOL){
           if (writeLogsFile.is_open()) {
              writeLogsFile << "Process converged! |A x_" << it << " - f| = " << val <<
std::endl;
           }
           else std::cout << "Unable to open output file";
           std::cout << "Process converged! |A x_" << it << " - f| = " << val << std::endl;
           break;
        }

        // Compute the new residual norm
        valPrev = val;
     }

     std::chrono::steady_clock::time_point end =
std::chrono::steady_clock::now();//ostanavlivayem schetchik
     std::cout << "time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
```

```
milliseconds" << std::endl;
    if (writeLogsFile.is_open()) {
        writeLogsFile << "time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
milliseconds\n\n";
        writeLogsFile.close();
    }
    else std::cout << "Unable to open output file";

    // Copy results back to host
    cudaMemcpy(xVec, d_xVec, sizeVector, cudaMemcpyDeviceToHost);

    // Cleanup
    cudaFree(d_matA);
    cudaFree(d_matA_con);
    cudaFree(d_vecf);
    cudaFree(d_xVec);
    cudaFree(d_omega);
    cudaFree(d_vec);
    cudaFree(d_norm);

    delete[] matA;
    delete[] matA_con;
    delete[] vecf;
    delete[] xVec;
    delete[] omega;
    delete[] vec;

    std::cout << "Converged in " << it << " iterations, residual norm: " << val <<
std::endl;

    return 0;
}
```

## ҚОСЫМША Г

Екінші алгоритмнің тізбектей программалық коды

```
//compile command g++ bolim2.cpp -o bolim2.bin
//execute command ./bolim2.bin 100 0.01 0.00001 singular

#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>
#include <chrono>
#define MAX_ITERATIONS 50000


int main(int argc, char *argv[]){
   int NN = std::atoi(argv[1]);
   float matA[NN][NN];
   float vecf[NN];
   float xVec[NN] = {}, vec[NN] = {}, omega[NN] = {}, vecCalc[NN] = {};
   float delta = std::stof(argv[2]), epsilon = std::stof(argv[3]), val = 0, val_prev=0, tol
= 1e-05;
   unsigned it = 0;
   std::string readFile;
   if (argv[4] == "general")
      readFile = std::string("/kaggle/working/data_") + std::string(argv[1]) +
std::string("x") + argv[1] + std::string(".txt");
   else
      readFile = std::string("/kaggle/working/data_") + std::string(argv[1]) +
std::string("x") + argv[1] + std::string("_")
               + std::string(argv[4]) + std::string(".txt");

   std::ifstream readDataFile(readFile);

   std::ofstream writeLogsFile("/kaggle/working/logs_bolim2", std::ios_base::app);

   std::string writeResidualFile;
   writeResidualFile = "/kaggle/working/residual_output_" + std::string(argv[4]) +
std::string("_") + std::string(argv[2])
            + std::string("_") + std::string(argv[1]);

   std::ofstream writeDataFile(writeResidualFile, std::ios_base::trunc);

   if (readDataFile.is_open()) {
```

```
        for (int k = 0; k < NN; k++)
            for (int l = 0; l < NN; l++)
                readDataFile >> matA[k][l];
        for (int l = 0; l < NN; l++) {
            readDataFile >> vecf[l];
        }

        readDataFile.close();
    }
    else std::cout << "Unable to open read file!!!";
    if (writeLogsFile.is_open()) {
        writeLogsFile << "-------S E Q U E N T I A L-------\n";
        writeLogsFile << "matrix type = " << argv[4] << std::endl;
        writeLogsFile << "dimension = " << NN << "x" << NN << std::endl;
        writeLogsFile << "delta = " << delta << std::endl;
        writeLogsFile << "epsilon = " << epsilon << std::endl;
    }
    else std::cout << "Unable to open output file";
    std::chrono::steady_clock::time_point start =
std::chrono::steady_clock::now();//zasekayem vremya
    int i,j;
    for (i = 0; i < NN; i++) {
        omega[i] = 0;
        for (j = 0; j < NN; j++) {
            omega[i] += matA[j][i] * vecf[j];
        }
    }
    while(true){
        it ++;
        val = 0;
        for (i = 0; i < NN; i++)
            xVec[i] += delta * omega[i];
        for (i = 0; i < NN; i++) {
            vec[i] = 0;
            for (j = 0; j < NN; j++) {
                vec[i] += matA[i][j] * omega[j];
            }
        }

        for (i = 0; i < NN; i++) {
            vecCalc[i] = 0;
            for (j = 0; j < NN; j++) {
                vecCalc[i] += matA[j][i] * vec[j];
            }
```

```
        }
        for (i = 0; i < NN; i++) {
            omega[i] = (1 - delta * epsilon) * omega[i] - delta * vecCalc[i];
        }
        for (i = 0; i < NN; i++) {
            vec[i] = 0;
            for (j = 0; j < NN; j++) {
                vec[i] += matA[i][j] * xVec[j];
            }
            val += (vec[i] - vecf[i]) * (vec[i] - vecf[i]);
        }
        val = sqrt(val);
        if (it > MAX_ITERATIONS){
            if (writeLogsFile.is_open()) {
                writeLogsFile << "No convergence! Iterations number greater than max
value " << MAX_ITERATIONS << std::endl;
                writeLogsFile << "|A x_" << it << " - f| = " << val << std::endl;
            }
            else std::cout << "Unable to open output file";
            std::cout << "No convergence! Iterations number greater than max value " <<
MAX_ITERATIONS << std::endl;
            std::cout << "|A x_" << it << " - f| = " << val << std::endl;
            break;
        }
        if (writeDataFile.is_open()) {
                writeDataFile << val << " ";
        }
        if (fabs(val - val_prev) < tol){
            if (writeLogsFile.is_open()) {
                writeLogsFile << "Process converged! |A x_" << it << " - f| = " << val <<
std::endl;
            }
            else std::cout << "Unable to open output file";
            std::cout << "Process converged! |A x_" << it << " - f| = " << val << std::endl;
            break;
        }
        val_prev = val;
    }
    std::chrono::steady_clock::time_point end =
std::chrono::steady_clock::now();//ostanavlivayem schetchik
    std::cout << "time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
milliseconds" << std::endl;
    if (writeLogsFile.is_open()) {
```

```
        writeLogsFile << "time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
milliseconds\n\n";
        writeLogsFile.close();
    }
    else std::cout << "Unable to open output file";
    return 0;
}
```

## ҚОСЫМША Ғ

OMP технологияны қоладанып екінші алгоритмнің параллель программалық коды

```
//g++ -fopenmp bolim2_OMP.cpp -o bolim2_OMP.bin
//./bolim2_OMP.bin 100 0.01 0.00001 singular

#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>
#include <chrono>
#include <omp.h>
#define MAX_ITERATIONS 50000

int main(int argc, char *argv[]) {
   int NN = atoi(argv[1]);
   float matA[NN][NN];
   float vecf[NN];
   float xVec[NN] = {}, vec[NN] = {}, omega[NN] = {}, vecCalc[NN] = {};
   float delta = std::stof(argv[2]), epsilon = std::stof(argv[3]), val = 0, val_prev = 0,
tol = 1e-05;
   unsigned it = 0;
   std::string readFile;
   if (argv[4] == "general")
      readFile = std::string("/kaggle/working/data_") + std::string(argv[1]) +
std::string("x") + argv[1] + std::string(".txt");
   else
      readFile = std::string("/kaggle/working/data_") + std::string(argv[1]) +
std::string("x") + argv[1]
               + std::string("_") + std::string(argv[4]) + std::string(".txt");

   std::ifstream readDataFile(readFile);
   std::ofstream writeLogsFile("/kaggle/working/logs_bolim2", std::ios_base::app);
   std::string writeResidualFile;
   writeResidualFile = "/kaggle/working/residual_output_" + std::string(argv[4]) +
std::string("_")
            + std::string(argv[2]) + std::string("_") + std::string(argv[1]) +
std::string(".txt");
   std::ofstream writeDataFile(writeResidualFile, std::ios_base::trunc);

   if (readDataFile.is_open()) {
      for (int k = 0; k < NN; k++)
```

```
            for (int l = 0; l < NN; l++)
                readDataFile >> matA[k][l];
        for (int l = 0; l < NN; l++) {
            readDataFile >> vecf[l];
        }
        readDataFile.close();
    } else {
        std::cout << "Unable to open read file";
        return 1;
    }
    if (writeLogsFile.is_open()) {
        writeLogsFile << "-------P A R A L L E L-------\n";
        writeLogsFile << "matrix type = " << argv[4] << std::endl;
        writeLogsFile << "dimension = " << NN << "x" << NN << std::endl;
        writeLogsFile << "delta = " << delta << std::endl;
        writeLogsFile << "epsilon = " << epsilon << std::endl;
    } else {
        std::cout << "Unable to open output file";
        return 1;
    }

    auto start = std::chrono::steady_clock::now();

    // Compute initial omega in parallel
    #pragma omp parallel for
    for (int i = 0; i < NN; i++) {
        omega[i] = 0;
        for (int j = 0; j < NN; j++) {
            omega[i] += matA[j][i] * vecf[j];
        }
    }

    while (true) {
        it++;
        val = 0;

        // Update xVec in parallel
        #pragma omp parallel for
        for (int i = 0; i < NN; i++) {
            xVec[i] += delta * omega[i];
        }

        // Compute vec in parallel
        #pragma omp parallel for
```

```
        for (int i = 0; i < NN; i++) {
            vec[i] = 0;
            for (int j = 0; j < NN; j++) {
                vec[i] += matA[i][j] * omega[j];
            }
        }

        // Compute vecCalc in parallel
        #pragma omp parallel for
        for (int i = 0; i < NN; i++) {
            vecCalc[i] = 0;
            for (int j = 0; j < NN; j++) {
                vecCalc[i] += matA[j][i] * vec[j];
            }
        }

        // Update omega in parallel
        #pragma omp parallel for
        for (int i = 0; i < NN; i++) {
            omega[i] = (1 - delta * epsilon) * omega[i] - delta * vecCalc[i];
        }

        // Calculate val in parallel with reduction
        #pragma omp parallel for reduction(+:val)
        for (int i = 0; i < NN; i++) {
            vec[i] = 0;
            for (int j = 0; j < NN; j++) {
                vec[i] += matA[i][j] * xVec[j];
            }
            val += (vec[i] - vecf[i]) * (vec[i] - vecf[i]);
        }
        val = sqrt(val);

        if (it > MAX_ITERATIONS) {
            if (writeLogsFile.is_open()) {
                writeLogsFile << "No convergence! Iterations number greater than max
value " << MAX_ITERATIONS << std::endl;
                writeLogsFile << "|A x_" << it << " - f| = " << val << std::endl;
            }
            std::cout << "No convergence! Iterations number greater than max value " <<
MAX_ITERATIONS << std::endl;
            std::cout << "|A x_" << it << " - f| = " << val << std::endl;
            break;
        }
```

```
        if (writeDataFile.is_open()) {
            writeDataFile << val << " ";
        }

        if (fabs(val - val_prev) < tol) {
            if (writeLogsFile.is_open()) {
                writeLogsFile << "Process converged! |A x_" << it << " - f| = " << val <<
std::endl;
            }
            std::cout << "Process converged! |A x_" << it << " - f| = " << val << std::endl;
            break;
        }
        val_prev = val;
    }

    auto end = std::chrono::steady_clock::now();
    std::cout << "time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
milliseconds" << std::endl;
    if (writeLogsFile.is_open()) {
        writeLogsFile << "time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
milliseconds\n\n";
        writeLogsFile.close();
    }

    return 0;
}
```

## ҚОСЫМША Д

CUDA технологияны қоладанып екінші алгоритмнің параллель программалық коды

```
//nvcc -o bolim2_cuda.bin bolim2_cuda.cu
//./bolim2_cuda.bin 100 0.01 0.00001 singular 256

#include <iostream>
#include <cmath>
#include <fstream>
#include <chrono>
#include <cuda_runtime.h>

#define MAX_ITERATIONS 50000

// CUDA kernel for matrix-vector multiplication
__global__ void matVecMult(float *matA, float *vec, float *result, int NN) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < NN) {
        result[i] = 0;
        for (int j = 0; j < NN; j++) {
            result[i] += matA[i * NN + j] * vec[j];
        }
    }
}

// CUDA kernel for matrix-vector multiplication (for omega update)
__global__ void matVecMultOmega(float *matA, float *vec, float *result, int NN) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < NN) {
        result[i] = 0;
        for (int j = 0; j < NN; j++) {
            result[i] += matA[j * NN + i] * vec[j];
        }
    }
}

__global__ void computeNorm(float *vec, float *result, int NN) {
    __shared__ float cache[256];
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int threadIdxLocal = threadIdx.x;

    float temp = (idx < NN) ? vec[idx] * vec[idx] : 0.0f;
```

```
    cache[threadIdxLocal] = temp;

    __syncthreads();

    // Reduction within the block
    for (int stride = blockDim.x / 2; stride > 0; stride /= 2) {
        if (threadIdxLocal < stride) {
            cache[threadIdxLocal] += cache[threadIdxLocal + stride];
        }
        __syncthreads();
    }

    if (threadIdxLocal == 0) {
        atomicAdd(result, cache[0]);
    }
}

__global__ void updateOmega(float *omega, float *vecCalc, float epsilon, float
delta, int NN) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < NN) {
        omega[idx] = (1 - delta * epsilon) * omega[idx] - delta * vecCalc[idx];
    }
}

__global__ void vectorSubtract(float *vec1, float *vec2, float *result, int NN) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < NN) {
        result[idx] = vec1[idx] - vec2[idx];
    }
}

__global__ void updateSolution(float *xVec, float *omega, float delta, int NN) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < NN) {
        xVec[idx] += delta * omega[idx];
    }
}

int main(int argc, char *argv[]) {
    int NN = atoi(argv[1]);
    float *matA, *vecf, *xVec, *vec, *omega, *vecCalc;
    float *d_matA, *d_vecf, *d_xVec, *d_vec, *d_omega, *d_vecCalc, *d_norm;
    float delta = std::stof(argv[2]), epsilon = std::stof(argv[3]), val = 0, val_prev = 0,
```

```
tol = 1e-05;
   unsigned it = 0;

   // Allocate host memory
   matA = new float[NN * NN];
   vecf = new float[NN];
   xVec = new float[NN]();
   vec = new float[NN]();
   omega = new float[NN]();
   vecCalc = new float[NN]();

   // Allocate device memory
   cudaMalloc((void**)&d_matA, NN * NN * sizeof(float));
   cudaMalloc((void**)&d_vecf, NN * sizeof(float));
   cudaMalloc((void**)&d_xVec, NN * sizeof(float));
   cudaMalloc((void**)&d_vec, NN * sizeof(float));
   cudaMalloc((void**)&d_omega, NN * sizeof(float));
   cudaMalloc((void**)&d_vecCalc, NN * sizeof(float));
   cudaMalloc(&d_norm, sizeof(float));

   std::string readFile;

   if (argv[4] == "general")
      readFile = std::string("/kaggle/working/data_") + std::string(argv[1]) +
std::string("x") + argv[1] + std::string(".txt");
   else
      readFile = std::string("/kaggle/working/data_") + std::string(argv[1]) +
std::string("x") + argv[1] + std::string("_")
               + std::string(argv[4]) + std::string(".txt");

   std::ifstream readDataFile(readFile);

   std::ofstream writeLogsFile("/kaggle/working/logs_bolim2_cuda",
std::ios_base::app);

   std::string writeResidualFile;
   writeResidualFile = "/kaggle/working/residual_output_cuda_" +
std::string(argv[4]) + std::string("_") + std::string(argv[2])
            + std::string("_") + std::string(argv[1]);

   std::ofstream writeDataFile(writeResidualFile, std::ios_base::trunc);

   if (readDataFile.is_open()) {
      for (int k = 0; k < NN; k++)
```

```
            for (int l = 0; l < NN; l++)
                readDataFile >> matA[k * NN + l];
        for (int l = 0; l < NN; l++) {
            readDataFile >> vecf[l];
        }

        readDataFile.close();
    }
    else std::cout << "Unable to open read file!!!";
    if (writeLogsFile.is_open()) {
        writeLogsFile << "-------C U D A-------\n";
        writeLogsFile << "matrix type = " << argv[4] << std::endl;
        writeLogsFile << "dimension = " << NN << "x" << NN << std::endl;
        writeLogsFile << "delta = " << delta << std::endl;
        writeLogsFile << "epsilon = " << epsilon << std::endl;
    }
    else std::cout << "Unable to open output file";

    // Copy data from host to device
    cudaMemcpy(d_matA, matA, NN * NN * sizeof(float),
cudaMemcpyHostToDevice);
    cudaMemcpy(d_vecf, vecf, NN * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_xVec, xVec, NN * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_vec, vec, NN * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_omega, omega, NN * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_vecCalc, vecCalc, NN * sizeof(float),
cudaMemcpyHostToDevice);

    dim3 blockDim(std::atoi(argv[5]));
    dim3 gridDim((NN + blockDim.x - 1) / blockDim.x);

    auto start = std::chrono::steady_clock::now();

    // Initial omega computation
    matVecMultOmega<<<gridDim, blockDim>>>(d_matA, d_vecf, d_omega, NN);

    // Main iterative loop
    while (true) {
        it++;

        // Update solution
        updateSolution<<<gridDim, blockDim>>>(d_xVec, d_omega, delta, NN);
        cudaDeviceSynchronize();
```

```
        // Compute vec in parallel
        matVecMult<<<gridDim, blockDim>>>(d_matA, d_omega, d_vec, NN);
        cudaDeviceSynchronize();

        // Compute vecCalc in parallel
        matVecMultOmega<<<gridDim, blockDim>>>(d_matA, d_vec, d_vecCalc,
NN);
        cudaDeviceSynchronize();

        // Update omega in parallel
        updateOmega<<<gridDim, blockDim>>>(d_omega, d_vecCalc, epsilon, delta,
NN);
        cudaDeviceSynchronize();

        // Matrix-vector multiplication
        matVecMult<<<gridDim, blockDim>>>(d_matA, d_xVec, d_vec, NN);
        cudaDeviceSynchronize();

        // Compute residual vector
        vectorSubtract<<<gridDim, blockDim>>>(d_vec, d_vecf, d_vecCalc, NN);
        cudaDeviceSynchronize();

        val = 0;
        cudaMemcpy(d_norm, &val, sizeof(float), cudaMemcpyHostToDevice);
        computeNorm<<<gridDim, blockDim>>>(d_vecCalc, d_norm, NN);
        cudaMemcpy(&val, d_norm, sizeof(float), cudaMemcpyDeviceToHost);
        val = sqrt(val);  // Take square root to get the norm

        // Convergence check
        if (it > MAX_ITERATIONS) {
            std::cout << "No convergence! Iterations number greater than max value " <<
MAX_ITERATIONS << std::endl;
            std::cout << "|A x_" << it << " - f| = " << val << std::endl;
            break;
        }

        if (fabs(val - val_prev) < tol) {
            std::cout << "Process converged! |A x_" << it << " - f| = " << val << std::endl;
            break;
        }
        val_prev = val;
    }

    auto end = std::chrono::steady_clock::now();
```

```
    std::cout << "Time = " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << "
milliseconds" << std::endl;

    // Free device memory
    cudaFree(d_matA);
    cudaFree(d_vecf);
    cudaFree(d_xVec);
    cudaFree(d_vec);
    cudaFree(d_omega);
    cudaFree(d_vecCalc);
    cudaFree(d_norm);

    // Free host memory
    delete[] matA;
    delete[] vecf;
    delete[] xVec;
    delete[] vec;
    delete[] omega;
    delete[] vecCalc;

    return 0;
}
```