**Мазмұны**

[**Кіріспе** 2](#_Toc101269265)

[**Гаусс әдісі** 4](#_Toc101269266)

[**Гаусс әдісінің С++ сызықты есептеу коды.** 9](#_Toc101269267)

[**Гаусс әдісінің С++ параллель есептеу коды.** 12](#_Toc101269268)

[**CG әдісінің С++ сызықты есептеу коды.** 15](#_Toc101269269)

[**CG әдісінің С++ параллель есептеу коды.** 18](#_Toc101269270)

# **Кіріспе**

Көптеген қолданбалы, соның ішінде экономикалық есептер сызықтық теңдеулер жүйесіне әкелінеді.

 айнымалысы бар және  сызықтық теңделерден тұратын жүйе келесідей жазылады:

 (1)

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

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

 (2)

1. *Жүйенің шешімі* деп жүйенің әр теңдеуіндегі белгісіз айнымалы орнына қойғанда  шындыққан айналатын  сандар жиынын айтамыз.

Теңдеулер жүйесінің ең болмағанда бір шешімі болса онда оны *үйлесетін* (совместной), ал шешімдері жоқ болса *үйлеспейтін* (несовместной) деп аталады.

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

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

1. Жүйені матрицалық формада жазайық.



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

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



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

 (3)

# **Гаусс әдісі**

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

Бұл әдістің тарихына шолу жасайтын болсақ, бұл әдіс Карл Фридрих Гауссқа дейін де белгілі болғанын байқаймыз. Әдістің алғашқы белгілі сипаттамасы біздің дәуірімізге дейінгі I ғасыр және және II ғасыр арасында құрастырылған қытайлық «Тоғыз кітаптағы математика» трактатында көрсетілген (Grcar, 2011). САТЖ шешудің Гаусс әдісін кей оқулықтарды Гаусстық жою әдісі деп те атайды.

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

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

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

 (4)

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

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



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

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

 (5)

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

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

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

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

 (6)

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

*Мысалы,* Берілген САТЖ Гаусс әдісімен шешімін тап.



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



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



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



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



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



Матрица түрін жүйе түріне аударып жазсақ. Келесі жүйені аламыз:



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

# **Гаусс әдісінің С++ сызықты есептеу коды.**

gaussSerial::gaussSerial(int size) {

mSize = size;

pSerialPivotIter = new int[size];

pSerialPivotPos = new int[size];

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

pSerialPivotIter[i] = -1;

}

}

gaussSerial::gaussSerial(const gaussSerial& orig) {

}

gaussSerial::~gaussSerial() {

}

int gaussSerial::resultCalculation(double\*\* pMatrix, double\* pVector, double\* pResult) {

serialGaussianElimination(pMatrix, pVector);

serialBackSubstitution(pMatrix, pVector, pResult);

return 0;

}

int gaussSerial::findPivotRow(double\*\* pMatrix, int Iter) {

int PivotRow = -1;

double MaxValue = 0;

int i;

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

if ((pSerialPivotIter[i] == -1) && (fabs(pMatrix[i][Iter]) > MaxValue)) {

PivotRow = i;

MaxValue = fabs(pMatrix[i][Iter]);

}

}

return PivotRow;

}

int gaussSerial::serialColumnElimination(double\*\* pMatrix, double\* pVector, int Pivot, int Iter) {

double PivotValue, PivotFactor;

PivotValue = pMatrix[Pivot][Iter];

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

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

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

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

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

}

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

}

}

return 0;

}

int gaussSerial::serialGaussianElimination(double\*\* pMatrix, double\* pVector) {

int Iter;

int PivotRow;

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

PivotRow = findPivotRow(pMatrix, Iter);

pSerialPivotPos[Iter] = PivotRow;

pSerialPivotIter[PivotRow] = Iter;

serialColumnElimination(pMatrix, pVector, PivotRow, Iter);

}

return 0;

}

/\*

\* Обратный ход метода Гаусса

\*/

int gaussSerial::serialBackSubstitution(double\*\* pMatrix, double\* pVector, double\* pResult) {

int RowIndex, Row;

for (int i = mSize - 1; i >= 0; i--) {

RowIndex = pSerialPivotPos[i];

pResult[i] = pVector[RowIndex] / pMatrix[RowIndex][i];

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

Row = pSerialPivotPos[j];

pVector[j] -= pMatrix[Row][i] \* pResult[i];

pMatrix[Row][i] = 0;

}

}

return 0;

}

# **Гаусс әдісінің С++ параллель есептеу коды.**

gaussParallel::gaussParallel(int size) {

mSize = size;

pSerialPivotIter = new int[size];

pSerialPivotPos = new int[size];

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

pSerialPivotIter[i] = -1;

}

}

gaussParallel::gaussParallel(const gaussParallel& orig) {

}

gaussParallel::~gaussParallel() {

}

int gaussParallel::resultCalculation(double\*\* pMatrix, double\* pVector, double\* pResult) {

gaussianElimination(pMatrix, pVector);

backSubstitution(pMatrix, pVector, pResult);

return 0;

}

int gaussParallel::findPivotRow(double\*\* pMatrix, int Iter) {

int PivotRow = -1;

double MaxValue = 0;

int i;

#pragma omp parallel

{

TThreadPivotRow ThreadPivotRow;

ThreadPivotRow.MaxValue = 0;

ThreadPivotRow.PivotRow = -1;

#pragma omp for

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

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

ThreadPivotRow.PivotRow = i;

ThreadPivotRow.MaxValue = fabs(pMatrix[i][Iter]);

}

}

#pragma omp critical

{

if (ThreadPivotRow.MaxValue > MaxValue) {

MaxValue = ThreadPivotRow.MaxValue;

PivotRow = ThreadPivotRow.PivotRow;

}

}

}

return PivotRow;

}

int gaussParallel::columnElimination(double\*\* pMatrix, double\* pVector, int Pivot, int Iter) {

double PivotValue, PivotFactor;

PivotValue = pMatrix[Pivot][Iter];

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

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

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

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

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

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

}

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

}

}

return 0;

}

int gaussParallel::gaussianElimination(double\*\* pMatrix, double\* pVector) {

int Iter;

int PivotRow;

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

PivotRow = findPivotRow(pMatrix, Iter);

pSerialPivotPos[Iter] = PivotRow;

pSerialPivotIter[PivotRow] = Iter;

columnElimination(pMatrix, pVector, PivotRow, Iter);

}

return 0;

}

/\* Обратный ход метода Гаусса

\*/

int gaussParallel::backSubstitution(double\*\* pMatrix, double\* pVector, double\* pResult) {

int RowIndex, Row;

for (int i = mSize - 1; i >= 0; i--) {

RowIndex = pSerialPivotPos[i];

pResult[i] = pVector[RowIndex] / pMatrix[RowIndex][i];

#pragma omp parallel for private (Row)

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

Row = pSerialPivotPos[j];

pVector[j] -= pMatrix[Row][i] \* pResult[i];

pMatrix[Row][i] = 0;

}

}

return 0;

}

# **CG әдісінің С++ сызықты есептеу коды.**

CGSerial::CGSerial() {

}

CGSerial::CGSerial(const CGSerial& orig) {

}

CGSerial::~CGSerial() {

}

void CGSerial::resultCalculation(double\*\* pMatrix, double\* pVector, double\* pResult, int Size) {

double \*CurrentApproximation, \*PreviousApproximation;

double \*CurrentGradient, \*PreviousGradient;

double \*CurrentDirection, \*PreviousDirection;

double \*Denom, \*tempPointer;

double Step;

int Iter = 1, MaxIter = Size + 1;

float Accuracy = 0.00001f;

CurrentApproximation = new double[Size];

PreviousApproximation = new double[Size];

CurrentGradient = new double[Size];

PreviousGradient = new double[Size];

CurrentDirection = new double[Size];

PreviousDirection = new double[Size];

Denom = new double[Size];

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

PreviousApproximation[i] = 0;

PreviousDirection[i] = 0;

PreviousGradient[i] = -pVector[i];

}

do {

if (Iter > 1) {

tempPointer = PreviousApproximation;

PreviousApproximation = CurrentApproximation;

CurrentApproximation = tempPointer;

tempPointer = PreviousGradient;

PreviousGradient = CurrentGradient;

CurrentGradient = tempPointer;

tempPointer = PreviousDirection;

PreviousDirection = CurrentDirection;

CurrentDirection = tempPointer;

}

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

CurrentGradient[i] = -pVector[i];

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

CurrentGradient[i] += pMatrix[i][j] \* PreviousApproximation[j];

}

double IP1 = 0, IP2 = 0;

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

IP1 += CurrentGradient[i] \* CurrentGradient[i];

IP2 += PreviousGradient[i] \* PreviousGradient[i];

}

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

CurrentDirection[i] = -CurrentGradient[i] +

PreviousDirection[i] \* IP1 / IP2;

}

IP1 = 0;

IP2 = 0;

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

Denom[i] = 0;

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

Denom[i] += pMatrix[i][j] \* CurrentDirection[j];

IP1 += CurrentDirection[i] \* CurrentGradient[i];

IP2 += CurrentDirection[i] \* Denom[i];

}

Step = -IP1 / IP2;

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

CurrentApproximation[i] = PreviousApproximation[i] + Step \* CurrentDirection[i];

}

Iter++;

} while

((diff(PreviousApproximation, CurrentApproximation, Size) > Accuracy)

&& (Iter < MaxIter));

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

pResult[i] = CurrentApproximation[i];

iterationsCount = Iter;

delete CurrentApproximation;

delete PreviousApproximation;

delete CurrentGradient;

delete PreviousGradient;

delete CurrentDirection;

delete PreviousDirection;

delete Denom;

}

double CGSerial::diff(double \*vector1, double\* vector2, int Size) {

double sum = 0;

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

sum += fabs(vector1[i] - vector2[i]);

}

return sum;

}

# **CG әдісінің С++ параллель есептеу коды.**

CGParallel::CGParallel() {

}

CGParallel::CGParallel(const CGParallel& orig) {

}

CGParallel::~CGParallel() {

}

void CGParallel::resultCalculation(double\*\* pMatrix, double\* pVector, double\* pResult, int Size) {

double \*CurrentApproximation, \*PreviousApproximation;

double \*CurrentGradient, \*PreviousGradient;

double \*CurrentDirection, \*PreviousDirection;

double \*Denom, \*tempPointer;

double Step;

int Iter = 1, MaxIter = Size + 1;

float Accuracy = 0.00001f;

CurrentApproximation = new double[Size];

PreviousApproximation = new double[Size];

CurrentGradient = new double[Size];

PreviousGradient = new double[Size];

CurrentDirection = new double[Size];

PreviousDirection = new double[Size];

Denom = new double[Size];

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

PreviousApproximation[i] = 0;

PreviousDirection[i] = 0;

PreviousGradient[i] = -pVector[i];

}

do {

if (Iter > 1) {

tempPointer = PreviousApproximation;

PreviousApproximation = CurrentApproximation;

CurrentApproximation = tempPointer;

tempPointer = PreviousGradient;

PreviousGradient = CurrentGradient;

CurrentGradient = tempPointer;

tempPointer = PreviousDirection;

PreviousDirection = CurrentDirection;

CurrentDirection = tempPointer;

}

#pragma omp parallel for

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

CurrentGradient[i] = -pVector[i];

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

CurrentGradient[i] += pMatrix[i][j] \* PreviousApproximation[j];

}

double IP1 = 0, IP2 = 0;

#pragma omp parallel for reduction(+:IP1,IP2)

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

IP1 += CurrentGradient[i] \* CurrentGradient[i];

IP2 += PreviousGradient[i] \* PreviousGradient[i];

}

#pragma omp parallel for

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

CurrentDirection[i] = -CurrentGradient[i] +

PreviousDirection[i] \* IP1 / IP2;

}

IP1 = 0;

IP2 = 0;

#pragma omp parallel for reduction(+:IP1,IP2)

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

Denom[i] = 0;

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

Denom[i] += pMatrix[i][j] \* CurrentDirection[j];

IP1 += CurrentDirection[i] \* CurrentGradient[i];

IP2 += CurrentDirection[i] \* Denom[i];

}

Step = -IP1 / IP2;

#pragma omp parallel for

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

CurrentApproximation[i] = PreviousApproximation[i] + Step \* CurrentDirection[i];

}

Iter++;

} while

((diff(PreviousApproximation, CurrentApproximation, Size) > Accuracy)

&& (Iter < MaxIter));

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

pResult[i] = CurrentApproximation[i];

iterationsCount = Iter;

delete CurrentApproximation;

delete PreviousApproximation;

delete CurrentGradient;

delete PreviousGradient;

delete CurrentDirection;

delete PreviousDirection;

delete Denom;

}

double CGParallel::diff(double \*vector1, double\* vector2, int Size) {

double sum = 0;

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

sum += fabs(vector1[i] - vector2[i]);

}

return sum;

}