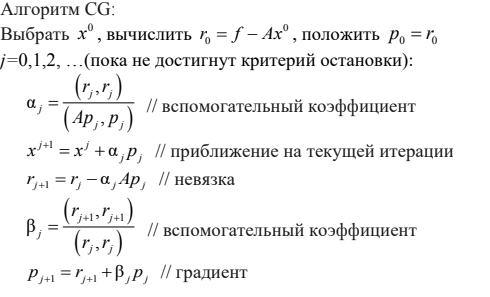
**Лабораторная работа №5**

**Задача.**

Разработать программу численного решения СЛАУ методом сопряженных градиентов. Результаты вычислительных экспериментов сравнить с аналогичными результатами, полученными с помощью разработанной ранее программы численного решения СЛАУ на основе LDLT-разложения. 

**Входные данные.**

Size = 10, m = 14, k = 2, lit = 50, eps = 0.0001

**Листинг программы.**

#include <iostream>  
#include <vector>  
#include <cmath>  
#include <chrono>  
  
int size = 1700**;**int m = 14**;**int k = 2**;**int lit = 50**;**double eps = 0.0001**;**std::vector<double> generateVectorF(size\_t size) {  
 std::vector<double> ans(size**,** 0)**;** for (size\_t i = 0**;** i < size**;** ++i) {  
 ans[i] = m + i**;** }  
 return ans**;**}  
  
double scalMul(std::vector<double> x**,** std::vector<double> y) {  
 double ans = 0**;** for (int i = 0**;** i < x.size()**;** i++) {  
 ans += x[i]\*y[i]**;** }  
 return ans**;**}  
  
std::vector<double> multiply(std::vector<std::vector<double>> A**,** std::vector<double> x) {  
 std::vector<double> ans(x.size()**,** 0)**;** {  
 for (int i = 0**;** i < x.size()**;** i++) {  
 double sum = 0**;** for (int j = 0**;** j < x.size()**;** j++) {  
 sum += A[i][j] \* x[j]**;** }  
 ans[i] = sum**;** }  
 }  
 return ans**;**}  
  
std::vector<std::vector<double>> generateMatrix(int size) {  
 std::vector<std::vector<double>> matrix(size**,** std::vector<double>(size**,** 0))**;** for (int i = 0**;** i < size**;** i++)  
 {  
 double diagElement = 0**;** for (int j = 0**;** j < size**;** j++)  
 {  
 if (i != j && matrix[i][j] == 0){  
 matrix[i][j] = (double)rand() / RAND\_MAX \* -1000**;** matrix[j][i] = matrix[i][j]**;** diagElement -= matrix[i][j]**;** }  
  
 }  
 if (i == 0)  
 {  
 matrix[i][i] = diagElement + std::pow(10**,** 2 - k)**;** }  
 else{  
 matrix[i][i] = diagElement**;** }  
 }  
 return matrix**;**}  
void LdltRtRDecomposition(std::vector<std::vector<double>> &matrix)  
{  
 std::vector<double> t(size)**;** for (int k = 0**;** k < size - 1**;** ++k)  
 {  
 for (int i = k + 1**;** i < size**;** ++i)  
 {  
 t[i] = matrix[i][k]**;** matrix[i][k] /= matrix[k][k]**;** for (int j = k + 1**;** j <= i**;** ++j)  
 {  
 matrix[i][j] -= matrix[i][k] \* t[j]**;** }  
 }  
 }  
}  
  
// функция решения системы Ly=b  
std::vector<double> SolveLyEqB(const std::vector<std::vector<double>> &lMatrix**,** const std::vector<double> &bVector)  
{  
 std::vector<double> y(size)**;** for (int i = 0**;** i < size**;** i++)  
 {  
 double sum = 0.0**;** for (int j = 0**;** j < i**;** j++)  
 {  
 sum += lMatrix[i][j] \* y[j]**;** }  
 y[i] = bVector[i] - sum**;** }  
  
 return y**;**}  
  
// функция решения системы Dz=y  
std::vector<double> SolveDzEqY(const std::vector<std::vector<double>> &DMatrix**,** const std::vector<double> &yVector)  
{  
 std::vector<double> z(size)**;** for (int i = 0**;** i < size**;** i++)  
 {  
 z[i] = yVector[i] / DMatrix[i][i]**;** }  
  
 return z**;**}  
  
// функция решения системы L^Tx=z  
std::vector<double> SolveLTxEqZ(const std::vector<std::vector<double>> &ltMatrix**,** const std::vector<double> &zVector)  
{  
 std::vector<double> x(size)**;** for (int i = size - 1**;** i >= 0**;** i--)  
 {  
 double sum = 0.0**;** for (int j = i + 1**;** j < size**;** j++)  
 {  
 sum += ltMatrix[j][i] \* x[j]**;** }  
 x[i] = zVector[i] - sum**;** }  
  
 return x**;**}  
  
// функция решения СЛАУ на основе LDL^T разложения  
std::vector<double> SolveSystem(std::vector<std::vector<double>> &matrix**,** const std::vector<double> &vector)  
{  
 LdltRtRDecomposition(matrix)**;** const std::vector<double> y = SolveLyEqB(matrix**,** vector)**;** const std::vector<double> z = SolveDzEqY(matrix**,** y)**;** const std::vector<double> x = SolveLTxEqZ(matrix**,** z)**;** return x**;**}  
  
  
  
  
void print5(std::vector<double> vector) {  
 for (int i = 0**;** i < 5**;** i++)  
 std::cout << vector[i] << " "**;** std::cout << "\n"**;**}  
  
  
double relativeError(std::vector<double> x\_**,** std::vector<double> x) {  
 double max\_1 = -1**;** for (int i = 0**;** i < x.size()**;** i++) {  
 max\_1 = std::max(std::abs(x\_[i] - x[i])**,** max\_1)**;** }  
 double max\_2 = -1**;** for (int i = 0**;** i < x.size()**;** i++) {  
 max\_2 = std::max(std::abs(x[i])**,** max\_2)**;** }  
 return max\_1 / max\_2**;**}  
  
bool check(std::vector<double> &ans){  
 for (int i = 0**;** i < size**;** ++i){  
 if (std::abs(ans[i]) <= eps)  
 return true**;** }  
 return false**;**}  
  
std::vector<double> cg(std::vector<std::vector<double>> &A**,** std::vector<double> &f**,** std::vector<double> &xZero){  
 std::vector<double> xl = xZero**;** std::vector<double> fslae(size)**;** std::vector<double> mvxl = multiply(A**,** xl)**;** for (size\_t i = 0**;** i < size**;** i++){  
 fslae[i] = f[i]**;** xl[i] = xZero[i]**;** }  
 std::vector<double> rl(size**,** 0)**;** std::vector<double> pl(size**,** 0)**;** for (size\_t i = 0**;** i < size**;** i++){  
 rl[i] = fslae[i] - mvxl[i]**;** pl[i] = rl[i]**;** }  
  
 double scal\_rl\_rl = scalMul(rl**,** rl)**;** for (size\_t i = 0**;** i < lit**;** i++){  
 double rl\_rl = scal\_rl\_rl**;** std::vector<double> mvpl = multiply(A**,** pl)**;** double scal\_mvpl\_pl = scalMul(mvpl**,** pl)**;** double alpha = rl\_rl/scal\_mvpl\_pl**;** for (size\_t j = 0**;** j < size**;** j++){  
 xl[j] += alpha\*pl[j]**;** rl[j] -= alpha\*mvpl[j]**;** }  
  
 scal\_rl\_rl = scalMul(rl**,** rl)**;** double beta = scal\_rl\_rl/rl\_rl**;** for (size\_t j = 0**;** j < size**;** j++){  
 pl[j] = rl[j] +beta\*pl[j]**;** }  
 if (check(xl))  
 break**;** }  
 return xl**;**}  
  
double vectorNevyazki(std::vector<std::vector<double>> &matrix**,**const std::vector<double> &x\_counted**,** std::vector<double> &f) {  
 std::vector<double> f\_new = multiply(matrix**,** x\_counted)**;** double max = -1**;** for (int i = 0**;** i < size**;** i++) {  
 max = std::max(std::abs(f\_new[i] - f[i])**,** max)**;** }  
 return max**;**}  
  
int main()  
{  
 std::vector<std::vector<double>> matrix = generateMatrix(size)**;** std::vector<double> x = generateVectorF(size)**;** std::vector<double> f = multiply(matrix**,** x)**;** std::vector<double> xZero(size**,** 0)**;** auto start = std::chrono::steady\_clock::now()**;** std::vector<double> x\_ = cg(matrix**,** f**,** xZero)**;** auto end = std::chrono::steady\_clock::now()**;** const int time1 = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count()**;** start = std::chrono::steady\_clock::now()**;** const std::vector<double> y = SolveSystem(matrix**,** f)**;** end = std::chrono::steady\_clock::now()**;** const int time2 = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count()**;** std::cout << "cg" << "\n"**;** std::cout << "5 cordinats of solution "**;** print5(x\_)**;** std::cout << "otnositelnaya pogreshnost " << relativeError(x\_**,** x) << "\n"**;** std::cout << "Norma vectora nevyazki " << vectorNevyazki(matrix**,** x\_**,** f) << "\n"**;** std::cout << "Runtime " << time1 << " ms\n"**;** std::cout << "LDL\n"**;** std::cout << "5 cordinats of solution "**;** print5(y)**;** std::cout << "Otnositelnaya pogreshnost " << relativeError(y**,** x) << "\n"**;** std::cout << "Norma vectora nevyazki " << vectorNevyazki(matrix**,** y**,** f) << "\n"**;** std::cout << "Runtime " << time2 << " ms\n"**;** return 0**;**}

**Выходные данные.**

cg

5 cordinats of solution 14.0031 15.0144 15.9943 16.9897 18.0073

otnositelnaya pogreshnost 0.0069556

Norma vectora nevyazki 1.28179e+09

Runtime 1158 ms

LDL

5 cordinats of solution 14 15 16 17 18

Otnositelnaya pogreshnost 2.03505e-10

Norma vectora nevyazki 1.28179e+09

Runtime 5229 ms

**Выводы.**

Метод сопряженных градиентов на больших данных в несколько раз быстрее LDLT-разложения. Также он менее точен.