Program do zadania Nr.5:

1. Generowanie macierzy, main oraz print

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <numeric>
#include <chrono>

using namespace std;
using vec = vector<double>;
const double NEARZERO = 1.0e-10;

vector<vec> genMatrix(int n) {
    vector <vec> matrix(n, vec(n));
    for (auto & i : matrix){
        for (double & j : i){
            j = 1;
        }
    }
    matrix[0][0] = 5;
    matrix[0][4] = 2;
    matrix[1][0] = 2;
    matrix[4][0] = 2;
    matrix[n-2][n-1] = 2;
    matrix[n-1][n-1] = 5;
    for(int i = 1; i < n-1; i++){
        matrix[i][i] = 5;
        matrix[i][i+4] = 2;
        matrix[i+1][i] = 2;
        matrix[i-1][i] = 2;
    }
    for (int i = 1; i < n-4; i++) {
        matrix[i+4][i] = 2;
    }
    matrix[n-1][0] = 1;
    matrix[0][n-1] = 1;
    return matrix;
}
vec genResults(int n){
    vec res(n, 1.0);
    return res;
}
```

```cpp
void print(const vec &V){
    cout.precision(10);
    int n = V.size();
    cout << "Solution: \n";
    for (int i = 0; i < n; i++){
        double x = V[i];
        if (abs(x) < NEARZERO) x = 0.0;
        cout << "x" << i+1 << "-->" << x << "\n";
    }
    cout << '\n';
}

int main() {
    int i = 64;
        vector<vector<double>> array = genMatrix(i);
        vector<double> res0 = genResults(i);
    auto begin = std::chrono::steady_clock::now();
        print(ShermanMorrison(array, res0));
    auto end = std::chrono::steady_clock::now();
    auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
    std::cout << "The time: " << elapsed_ms.count() << " ms\n";
}
```

2. Metoda gradientu sprzężonego

```cpp
/** METODA GRADIENTOW SPRZEZONYCH **/
// Iloczyn wewnetrzny U i V.
double innerProduct(const vec &U, const vec &V){
    return inner_product(U.begin(), U.end(), V.begin(), init: 0.0);
}
// Norma wektora
double vectorNorm(const vec &V){
    return sqrt(innerProduct(V, V));
}
vec matrixTimesVector(const vector<vec>& array, const vec &V){
    int n = array.size();
    vec C(n);
    for (int i = 0; i < n; i++)
        C[i] = innerProduct(array[i], V);
    return C;
}
// Liniowa kombinacja wektorów
vec vectorCombination(double a, const vec &U, double b, const vec &V){
    int n = U.size();
    vec W(n);
    for (int j = 0; j < n; j++)
        W[j] = a * U[j] + b * V[j];
    return W;
}

```

```cpp
vec conjugateGradientSolver(const vector<vec>& array, const vec& arr ){
    double MARGIN = 1.0e-10;
    int n = array.size();
    vec X(n, value: 0.0);
    vec R = arr;
    vec P = R;
    int k = 0;
    while (k < n){
        vec Rold = R;            // Przechowanie poprzednich pozostałości
        vec AP = matrixTimesVector(array,P);
        double alpha = innerProduct(R,R) / max(innerProduct(P,AP),NEARZERO);
        X = vectorCombination( a: 1.0,X,alpha,P);           // Następne oszacowanie rozwiązania
        R = vectorCombination( a: 1.0,R,-alpha,AP);         // Pozostała
        if (vectorNorm(R) < MARGIN)
            break;              // Test konwergencji
        double beta = innerProduct(R,R) / max(innerProduct(Rold,Rold),NEARZERO);
        P = vectorCombination( a: 1.0,R,beta,P);            // Następny gradient
        k++;
    }
    return X;
}
```

3. Wzór Shermana-Morissona

```
vec ShermanMorrison(vector<vec>& one,vec& two){
    int n = one.size();
    vec u(n);
    vec v(n);
    double gammaU = 1;
    u[0] = gammaU;
    for (int i = 1; i < n; i++) {
        u[i] = 1.0;
    }
    for (int k = 0; k < n; k++) {
        v[k] = 1.0;
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            one[i][j] -= 1.0;
        }
    }
    vec y = conjugateGradientSolver(one,two);
    vec z = conjugateGradientSolver(one,u);
    vec result(n);
    for (int i = 0; i < n; i++) {
        result[i] = y[i] - (innerProduct(v,y))/((1 + innerProduct(v,z))) * z[i] ;
    }
    return result;
}
```

Wyniki do zadania Nr.5:

Wyniki znajdują się w pliku tekstowym: Wyniki.txt

Komentarze do zadania Nr.5:

W tym zadaniu dla obliczania układu równań A*x=e, gdzie e_i = 1 (i = 1,2, …, 64) macierzy A ∈ R^{64x64} było skorzystano z metody gradientu sprzężonego, a następnie ze wzoru Shermana-Morrisona dla modyfikacji rozwiązania.

1. Metoda gradientu sprzężonego:

    Opis metody:

Metoda gradientu sprzężonego jako metoda iteracyjna:
Jeśli właściwie dobierzemy sprzężone wektory pk, możemy nie potrzebować ich wszystkich do dobrej aproksymacji rozwiązania x*. Możemy więc spojrzeć na CG jak na metodę iteracyjną. Co więcej, pozwoli nam to rozwiązać układy równań, gdzie n jest tak duże, że bezpośrednia metoda zabrałaby zbyt dużo czasu. Oznaczmy punkt startowy przez x0. Bez starty ogólności możemy założyć, że x0 = 0 (w przeciwnym przypadku, rozważymy układ Az = b − Ax0). Zauważmy, że rozwiązanie x* minimalizuje formę kwadratową:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \mathbf{x}^{\mathrm{T}}\mathbf{b}, \quad \mathbf{x} \in \mathbf{R}^n.$$

Co sugeruje, by jako pierwszy wektor bazowy p1 wybrać gradient f w x = x0, który wynosi Ax0−b, a ponieważ wybraliśmy x0 = 0, otrzymujemy −b. Pozostałe wektory w bazie będą sprzężone do gradientu (stąd nazwa metoda gradientu sprzężonego).

Niech $\mathbf{r}_k$ oznacza rezyduum w *k*-tym kroku:

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k.$$

Zauważmy, że rk jest przeciwny do gradientu f w x = xk, więc metoda gradientu prostego nakazywałaby ruch w kierunku rk. Tutaj jednak założyliśmy wzajemną sprzężoność kierunków pk, więc wybieramy kierunek najbliższy do rk pod warunkiem sprzężoności. Co wyraża się wzorem:

$$\mathbf{p}_{k+1} = \mathbf{r}_k - \frac{\mathbf{p}_k^\top \mathbf{A}\mathbf{r}_k}{\mathbf{p}_k^\top \mathbf{A}\mathbf{p}_k}\mathbf{p}_k.$$

Upraszczając, otrzymujemy poniższy algorytm rozwiązujący Ax = b, gdzie macierz A jest rzeczywista, symetryczna i dodatnio określona. x_0 jest punktem startowym.

$r_0 := b - Ax_0$
$p_0 := r_0$
$k := 0$
repeat

$\alpha_k := \dfrac{r_k^\top r_k}{p_k^\top A p_k}$

$x_{k+1} := x_k + \alpha_k p_k$

$r_{k+1} := r_k - \alpha_k A p_k$

if r_{k+1} jest "wystarczająco mały" **then** exit loop **end if**

$\beta_k := \dfrac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$

$p_{k+1} := r_{k+1} + \beta_k p_k$

$k := k + 1$

end repeat

Wynikiem jest x_{k+1}

2. Wzór Shermana-Morissona:

Używamy wzoru Shermana-Morrisona. W notacji równania:

$$(\mathbf{A} + \mathbf{u} \otimes \mathbf{v}) \cdot \mathbf{x} = \mathbf{b}$$

zdefiniujmy wektory u i v, które mają być:

$$\mathbf{u} = \begin{bmatrix} \gamma \\ 0 \\ \vdots \\ 0 \\ \alpha \end{bmatrix} \qquad \mathbf{v} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ \beta/\gamma \end{bmatrix}$$

Tutaj y jest na razie dowolne. Wtedy macierz A jest ze zmodyfikowanymi dwoma współczynnikami:

$$b'_1 = b_1 - \gamma, \qquad b'_N = b_N - \alpha\beta/\gamma$$

Teraz rozwiązujemy równania:

A · y = b **A · z = u**

Gdzie y i z są wektorami.

za pomocą **metody gradientu sprzężonego**, a następnie uzyskujemy rozwiązanie z równania:

$$\mathbf{x} = \mathbf{y} - \left[\frac{\mathbf{v} \cdot \mathbf{y}}{1 + (\mathbf{v} \cdot \mathbf{z})}\right] \mathbf{z}$$