Nama : Rachel Savitri

NIM : 21120122140111

Kelas : C

Penjabaran Kode pada Metode

Matriks Balikan

```
#include <iostream>
#include <vector>

using namespace std;

// fungsi untuk mencari invers dari sebuah matriks
vector<vector<double>> inverse(vector<vector<double>> A) {
    int n = A.size();
    vector<vector<double>> I(n, vector<double>(n, 0));
    for (int i = 0; i < n; i++) {
        I[i][i] = 1;
    }

    for (int i = 0; i < n; i++) {
        double div = A[i][i];
        for (int j = 0; j < n; j++) {
            A[i][j] /= div;
            I[i][j] /= div;
        }

        for (int j = 0; j < n; j++) {
            if (i != j) {
                double factor = A[j][i];
                for (int k = 0; k < n; k++) {
                    A[j][k] -= factor * A[i][k];
                    I[j][k] -= factor * I[i][k];
                }
            }
        }
    }

    return I;
}
```

1. Pertama, fungsi menghitung ukuran matriks *input* dan menyimpannya dalam variabel n. Kemudian, sebuah matriks identitas I dibuat dengan ukuran yang sama seperti matriks input A. Setiap elemen diagonal dari matriks identitas diinisialisasi dengan nilai 1, sementara elemen-elemen lainnya diinisialisasi dengan nilai 0.
2. Selanjutnya, fungsi melakukan iterasi melalui setiap baris matriks A. Pada setiap iterasi, dilakukan normalisasi pada baris tersebut dengan membagi setiap elemen baris tersebut dengan elemen diagonalnya, sehingga elemen diagonal menjadi 1.

3. Selanjutnya, dilakukan eliminasi Gauss-Jordan pada kolom `'i'` dari matriks `'A'` dan `'I'` untuk membuat semua elemen di luar diagonal menjadi 0. Ini dilakukan dengan mengurangkan baris lainnya dengan faktor yang sesuai dengan elemen di posisi `'[j][i]'`, di mana `'j'` adalah indeks baris selain `'i'`.
4. Iterasi berlanjut hingga semua elemen di luar diagonal menjadi nol. Setelah selesai, matriks `'A'` akan menjadi matriks identitas, dan matriks `'I'` akan menjadi *invers* dari matriks `'A'`.
5. Mengembalikan matriks `'I'`, yang merupakan *invers* dari matriks `'A'`.

```
        // fungsi untuk mencari solusi SPL menggunakan metode invers
        vector<double>solveSPL(vector<vector<double>>A,
        vector<double> b) {
            int n = A.size();

            // mencari invers dari matriks A
            vector<vector<double>> A_inv = inverse(A);

            // menghitung solusi SPL dengan menggunakan invers dari A
            vector<double> x(n, 0);
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    x[i] += A_inv[i][j] * b[j];
                }
            }

            return x;
}
```

1. Fungsi menerima dua argumen (matriks koefisien `'A'` dan vektor konstanta `'b'`). Kemudian, fungsi menghitung ukuran matriks `'A'` dan menyimpannya dalam variabel `'n'`.
2. Selanjutnya, fungsi memanggil fungsi *inverse* untuk mencari *invers* dari matriks `'A'`. Yang kemudian disimpan dalam variabel `'A_inv'`.
3. Setelah itu, fungsi menginisialisasi vektor solusi `'x'` dengan ukuran `'n'` dan nilai awal 0.

4. Fungsi melakukan iterasi melalui setiap baris matriks `'A_inv'` dan mengalikan setiap elemen baris tersebut dengan elemen yang sesuai dari vektor konstanta `'b'`. Hasil perkalian kemudian ditambahkan ke elemen vektor solusi `'x'` yang sesuai.
5. Setelah iterasi selesai, vektor `'x'` akan berisi solusi dari SPL.

```
        // Kode Testing
        int main() {
            // contoh SPL
            // 6x + 8y = 12
            // 10x + 2y = 40
            vector<vector<double>> A = {{6, 8}, {10, 2}};
            vector<double> b = {12, 40};

            // mencari solusi SPL menggunakan metode invers
            vector<double> x = solveSPL(A, b);

            // pencetakan solusi
            cout << "Solusi SPL:" << endl;
            cout << "x = " << x[0] << endl;
            cout << "y = " << x[1] << endl;

            return 0;
}
```

1. Di bagian `'main()'`, SPL contoh diberikan sebagai input, kemudian fungsi `'solveSPL'` dipanggil untuk menyelesaikan SPL menggunakan metode *invers*. Solusi yang ditemukan dicetak untuk memverifikasi keakuratannya. Proses ini membantu memastikan bahwa implementasi fungsi `'solveSPL'` berhasil menemukan solusi yang benar untuk SPL contoh yang diberikan.

Hasil Output