Tian Putra Perdana

21120122130072

Penjelasan Source Code Sistem Persamaan Linear

1. Fungsi luDecomposition()

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        if (j < i)
            L[j][i] = 0;
        else
        {
            L[j][i] = A[j][i];
            for (int k = 0; k < i; k++)
                L[j][i] -= L[j][k] * U[k][i];
        }
    }
    for (int j = 0; j < n; j++)
    {
        if (j < i)
            U[i][j] = 0;
        else if (j == i)
            U[i][j] = 1;
        else
        {
            U[i][j] = A[i][j] / L[i][i];
            for (int k = 0; k < i; k++)
                U[i][j] -= ((L[i][k] * U[k][j]) / L[i][i]);
        }
    }
```

Kode di atas adalah implementasi algoritma dekomposisi LU (Lower-Upper) untuk matriks persegi n x n. Proses dekomposisi ini membagi matriks input A menjadi dua matriks: matriks segitiga bawah L dan matriks segitiga atas U, sehingga A = LU. Langkah-langkahnya dimulai dengan loop pada setiap baris matriks. Di setiap iterasi, elemen-elemen matriks L dan U diinisialisasi sesuai aturan dekomposisi. Pada matriks L , elemen di atas diagonal utama diatur menjadi 0, dan elemen di bawah diagonal utama dihitung menggunakan rumus yang memperhitungkan nilai-nilai dari matriks L dan U . Sedangkan pada matriks U, elemen di bawah diagonal utama diatur menjadi 0, elemen pada diagonal utama diatur menjadi 1, dan elemen di atas diagonal utama dihitung menggunakan rumus yang juga memperhitungkan nilai-nilai dari matriks L

dan U. Setelah proses dekomposisi selesai, matriks A terbagi menjadi L dan U, yang berguna untuk penyelesaian sistem persamaan linear dan invers matriks.

2. Fungsi solveEquationsLU()

```
int n = A.size();

    vector<vector<double>> L(n, vector<double>(n, 0));
    vector<vector<double>> U(n, vector<double>(n, 0));

    luDecomposition(A, L, U, n);

    vector<double> Y(n, 0);
    vector<double> X(n, 0);

    // Solve LY = B
    for (int i = 0; i < n; i++)
    {
        Y[i] = B[i];
        for (int j = 0; j < i; j++)
        {
            Y[i] -= L[i][j] * Y[j];
        }
        Y[i] /= L[i][i];
    }

    // Solve UX = Y
    for (int i = n - 1; i >= 0; i--)
    {
        X[i] = Y[i];
        for (int j = i + 1; j < n; j++)
        {
            X[i] -= U[i][j] * X[j];
        }
        X[i] /= U[i][i];
    }

    return X;
```

Fungsi `solveEquationsLU` bertujuan untuk menyelesaikan sistem persamaan linear menggunakan metode dekomposisi LU. Proses dimulai dengan inisialisasi matriks segitiga bawah L dan segitiga atas U menggunakan fungsi `luDecomposition`, yang menghasilkan dua matriks dari matriks koefisien input A. Setelah mendapatkan matriks L dan U, fungsi tersebut menyelesaikan sistem persamaan linear $LY = B$ dan $UX = Y$, di mana L, U, B, Y, dan X adalah matriks dan vektor yang sesuai. Untuk menyelesaikan $LY = B$, fungsi melakukan substitusi mundur, mengurangkan hasil perkalian elemen-elemen L dengan elemen-elemen Y yang telah diketahui sebelumnya, lalu membagi hasilnya dengan elemen diagonal utama dari L. Selanjutnya, untuk menyelesaikan $UX = Y$, fungsi melakukan substitusi

maju, mengurangkan hasil perkalian elemen-elemen U dengan elemen-elemen X yang telah diketahui sebelumnya, lalu membagi hasilnya dengan elemen diagonal utama dari U. Setelah kedua tahap selesai, vektor X yang berisi solusi dari sistem persamaan linear dikembalikan.

3. Fungsi croutDecomposition()

```
// Decomposition
for (int i = 0; i < n; i++)
{
    U[i][i] = 1;

    // Uupper triangular matrix
    for (int k = i; k < n; k++)
    {
        double sum = 0;
        for (int j = 0; j < i; j++)
            sum += (L[i][j] * U[j][k]);
        U[i][k] = A[i][k] - sum;
    }

    // Lower triangular matrix
    for (int k = i + 1; k < n; k++)
    {
        double sum = 0;
        for (int j = 0; j < i; j++)
            sum += (L[k][j] * U[j][i]);
        L[k][i] = (A[k][i] - sum) / U[i][i];
    }
}
```

Fungsi `croutDecomposition` mengimplementasikan algoritma dekomposisi Crout untuk matriks persegi $n \times n$. Tujuan dekomposisi ini adalah untuk memisahkan matriks input A menjadi dua matriks: matriks segitiga bawah L dan matriks segitiga atas U, sehingga $A = LU$. Proses dimulai dengan menginisialisasi diagonal utama U sebagai matriks identitas. Selanjutnya, elemen-elemen U dihitung untuk setiap kolom k dengan mengurangkan hasil perkalian elemen-elemen L dan U yang telah diketahui sebelumnya dari elemen-elemen matriks A. Kemudian, elemen-elemen L dihitung untuk setiap kolom k, dengan memperhatikan elemen-elemen L dan U yang telah diketahui sebelumnya, dan hasilnya dibagi dengan elemen diagonal utama dari matriks U. Setelah selesai, matriks A terbagi menjadi L dan U, yang berguna untuk penyelesaian sistem persamaan linear dan invers matriks.

4. Fungsi solveEquationsCrout()

```
int n = A.size();
```

```
    vector<vector<double>> L(n, vector<double>(n, 0));
    vector<vector<double>> U(n, vector<double>(n, 0));

    croutDecomposition(A, L, U, n);

    vector<double> Y(n, 0);
    vector<double> X(n, 0);

    // Solve LY = B
    for (int i = 0; i < n; i++)
    {
        Y[i] = B[i];
        for (int j = 0; j < i; j++)
        {
            Y[i] -= L[i][j] * Y[j];
        }
    }

    // Solve UX = Y
    for (int i = n - 1; i >= 0; i--)
    {
        X[i] = Y[i];
        for (int j = i + 1; j < n; j++)
        {
            X[i] -= U[i][j] * X[j];
        }
        X[i] /= U[i][i];
    }

return X;
```

Fungsi `solveEquationsCrout` bertujuan untuk menyelesaikan sistem persamaan linear menggunakan metode Crout decomposition. Prosesnya dimulai dengan inisialisasi matriks segitiga bawah L dan segitiga atas U menggunakan fungsi `croutDecomposition`, yang menghasilkan dua matriks dari matriks koefisien input A. Setelah mendapatkan matriks L dan U, fungsi tersebut menyelesaikan sistem persamaan linear $LY = B$ dan $UX = Y$, di mana L, U, B, Y, dan X adalah matriks dan vektor yang sesuai. Untuk menyelesaikan $LY = B$, fungsi melakukan substitusi maju, yaitu menghitung Y dengan mengurangkan hasil perkalian elemen-elemen L dengan elemen-elemen Y yang telah diketahui sebelumnya dari vektor B. Proses ini dilakukan secara bertahap dari atas ke bawah matriks L, sehingga setiap $Y[i]$ dapat dihitung dengan menggunakan nilai-nilai Y yang sudah diketahui sebelumnya. Selanjutnya, untuk menyelesaikan $UX = Y$, fungsi melakukan substitusi mundur, yaitu menghitung X dengan mengurangkan hasil perkalian elemen-elemen U dengan elemen-elemen X yang telah diketahui sebelumnya dari vektor Y, dan membagi hasilnya dengan elemen diagonal utama dari matriks U. Setelah kedua tahap tersebut selesai, vektor X yang berisi solusi dari sistem persamaan linear dikembalikan.

5. Fungsi inverseMatrix()

```
    int n = matrix.size();

    // Identity matrix
    vector<vector<double>> identity(n, vector<double>(n, 0));
    for (int i = 0; i < n; ++i)
        identity[i][i] = 1;

    vector<vector<double>> A = matrix;

    // Forward elimination
    for (int i = 0; i < n; ++i)
    {
        double pivot = A[i][i];
        for (int j = 0; j < n; ++j)
        {
            A[i][j] /= pivot;
            identity[i][j] /= pivot;
        }
        for (int k = i + 1; k < n; ++k)
        {
            double factor = A[k][i];
            for (int j = 0; j < n; ++j)
            {
                A[k][j] -= factor * A[i][j];
                identity[k][j] -= factor * identity[i][j];
            }
        }
    }

    // Backward elimination
    for (int i = n - 1; i > 0; --i)
    {
        for (int k = i - 1; k >= 0; --k)
        {
            double factor = A[k][i];
            for (int j = 0; j < n; ++j)
            {
                A[k][j] -= factor * A[i][j];
                identity[k][j] -= factor * identity[i][j];
            }
        }
    }

    return identity;
```

Fungsi `inverseMatrix` dirancang untuk menemukan invers dari sebuah matriks persegi $n \times n$. Metode yang digunakan adalah eliminasi Gauss-Jordan, yang mengubah matriks identitas $n \times n$ menjadi invers dari matriks input. Langkah-langkahnya dimulai dengan pembuatan matriks identitas, di mana setiap elemen diagonal utamanya diatur menjadi 1 dan elemen-elemen lainnya menjadi 0. Proses eliminasi dimulai dengan normalisasi setiap baris, yaitu pembagian setiap elemen pada baris matriks input dan

matriks identitas dengan elemen diagonal utama yang disebut pivot. Selanjutnya, untuk setiap baris di bawah pivot, nilai-nilai di bawah diagonal utama pada matriks input dan matriks identitas dieliminasi dengan menggunakan elemen-elemen dari baris yang sedang diproses. Setelah eliminasi maju selesai, langkah mundur dilakukan untuk memastikan bahwa hanya elemen diagonal utama yang memiliki nilai 1, sementara nilai-nilai di atasnya diubah menjadi 0 dengan menggunakan elemen-elemen yang ada di atas diagonal utama. Setelah kedua tahap eliminasi selesai, matriks identitas telah berubah menjadi invers dari matriks input.

6. Fungsi matrixMultiply()

```
        int n = A.size();
        vector<double> result(n, 0);
        for (int i = 0; i < n; ++i)
        {
            for (int j = 0; j < n; ++j)
            {
                result[i] += A[i][j] * B[j];
            }
        }
  return result;
```

Fungsi `matrixMultiply` bertujuan untuk melakukan perkalian antara sebuah matriks A dengan sebuah vektor B . Prosesnya dimulai dengan inisialisasi panjang sisi matriks n , di mana matriks A diasumsikan sebagai matriks persegi n x n . Sebuah vektor hasil result[i] dengan panjang n juga diinisialisasi dengan nilai-nilai awal 0. Selanjutnya, dilakukan iterasi melalui setiap baris i dari matriks A dan setiap kolom j dari matriks A dan vektor B . Pada setiap langkah iterasi, nilai pada indeks i dari vektor hasil result[i] diupdate dengan menjumlahkan perkalian setiap elemen pada baris i dari matriks A dengan nilai yang sesuai pada vektor B . Akhirnya, vektor hasil result yang berisi hasil perkalian matriks A dengan vektor B dikembalikan.