Assignment - System of Equations solving Methods

Devesh Khandelwal (72567)

V.1 - Algorithms for Computational Mathematics: Numerical Methods

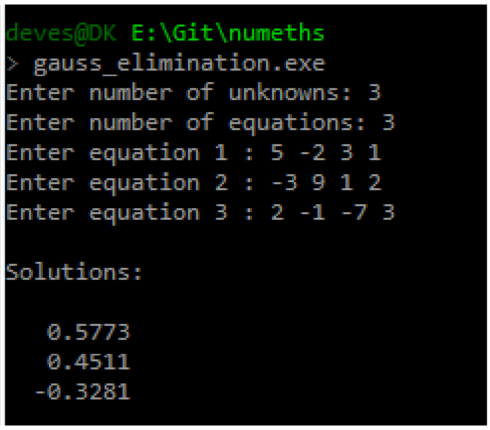
B. Tech. (Information Technology and Mathematical Innovations)

Cluster Innovation Centre, University of Delhi

Gaussian Elimination Method (w/o Pivoting)

```
/**
 * Gaussian Elimination.
 * @author : Devesh Khandelwal
 */
#include <iostream>
#include <armadillo>
#include <cmath>

// Used because the double value may not be exactly zero.
// So, this is to remove almost-singular errors.
#define SINGULARITY_MARGIN 0.00001

using namespace std;
using namespace arma;

/**
 * main function. Inputs a system of equations and outputs the solution to
 * that system, if exists.
 *
 * @param  argc Number of command line arguments.
 * @param  argv Command line arguments.
 * @return      Exit Status.
 */
int main(int argc, char** argv)
{
    int vars, equations;

    // Default system if nothing is provided.
    mat aug_mat = {
        {1, 2, 3, 1},
        {4, 5, 6, 1},
        {1, 0, 1, 1}
    };

    cout << "Enter number of unknowns: ";
    cin >> vars;

    cout << "Enter number of equations: ";
    cin >> equations;

    aug_mat.resize(equations, vars+1);

    // Input equation coefficients. Complete augmented matrix.
    for (unsigned i = 0; i < aug_mat.n_rows ; ++i)
    {
        cout << "Enter equation " << i+1 << " : ";
        for (unsigned j = 0; j < aug_mat.n_cols ; ++ j)
        {
            cin >> aug_mat(i, j);
        }
    }

    // Check singularity.
    if (abs(det(aug_mat.submat(0, 0, aug_mat.n_rows-1, aug_mat.n_cols-2)))
        <SINGULARITY_MARGIN)
    {
        cout << "Error: Coeffecient matrix is singular. No solution exists.";
        return EXIT_FAILURE;
    }

    //  aug_mat.print("\nAugmented matrix before elimination:\n");

    // Eliminate elemnts below the main diagonal i.e. make them zero using
    // elementary row operations.
    for (int i = 0; i < aug_mat.n_rows-1; ++i)
    {

        for (int j = i+1; j < aug_mat.n_rows; ++j)
        {
            double ratio = aug_mat(j, i)/aug_mat(i, i);
            aug_mat.row(j) = aug_mat.row(j) - ratio*aug_mat.row(i);
        }
    }

    //  aug_mat.print("\nAugmented matrix after lower elimination:\n");

    // Eliminate elements above the main diagonal i.e. make them zero using
    // elementary row operations.
    for (int i = aug_mat.n_rows-1; i > 0; --i)
    {
        for (int j = i-1; j >= 0; --j)
        {
            double ratio = aug_mat(j, i)/aug_mat(i, i);
            aug_mat.row(j) = aug_mat.row(j) - ratio*aug_mat.row(i);
        }
    }

    //  aug_mat.print("\nAugmented matrix after upper elimination:\n");

    // Substituting a(i,i)*x(i) = b(i). Calculating x(i).
    // Done by dividing the b vector by the diagonal of the coeffecient matrix.
    vec b = aug_mat.col(aug_mat.n_cols-1)/aug_mat.diag();

    b.print("\nSolutions:\n");

    return EXIT_SUCCESS;
}
```

Output

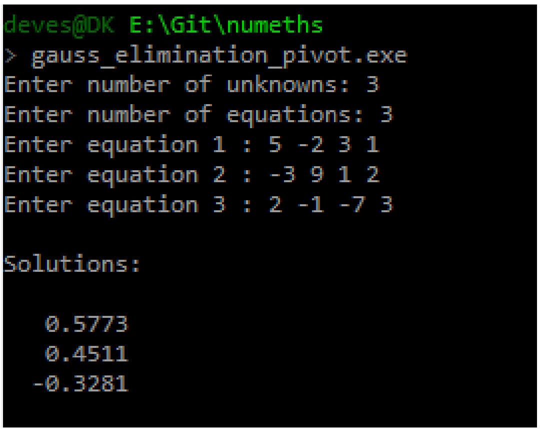


```
deves@DK E:\Git\numeths
> gauss_elimination.exe
Enter number of unknowns: 3
Enter number of equations: 3
Enter equation 1 : 5 -2 3 1
Enter equation 2 : -3 9 1 2
Enter equation 3 : 2 -1 -7 3

Solutions:

   0.5773
   0.4511
  -0.3281
```

Gaussian Elimination Method (w/ Pivoting)

```
/**
 * Gaussian Elimination using pivoting.
 * @author : Devesh Khandelwal
 */

#include <iostream>
#include <armadillo>
#include <cmath>

// Used because the double value may not be exactly zero.
// So, this is to remove almost-singular errors.
#define SINGULARITY_MARGIN 0.00001

using namespace std;
using namespace arma;

/**
 * main function. Inputs a system of equations and outputs the solution to that
 * system, if exists. Uses pivoting.
 * @param  argc Number of command line arguments.
 * @param  argv Command line arguments.
 * @return      Exit Status.
 */
int main(int argc, char** argv)
{
    int vars, equations;

    mat aug_mat = {
        {1, 2, 3, 1},
        {4, 5, 6, 1},
        {1, 0, 1, 1}
    };

    cout << "Enter number of unknowns: ";
    cin >> vars;

    cout << "Enter number of equations: ";
    cin >> equations;

    aug_mat.resize(equations, vars+1);

    // Input equation coefficients. Complete augmented matrix.
    for (unsigned i = 0; i < aug_mat.n_rows ; ++i)
    {
        cout << "Enter equation " << i+1 << " : ";
        for (unsigned j = 0; j < aug_mat.n_cols ; ++ j)
        {
            cin >> aug_mat(i, j);
        }
    }

    // Check singularity.
    if (abs(det(aug_mat.submat(0, 0, aug_mat.n_rows-1, aug_mat.n_cols-2)))<
        SINGULARITY_MARGIN)
    {
        cout << "Error: Coeffecient matrix is singular. No solution exists.";
        return EXIT_FAILURE;
    }

    // aug_mat.print("\nAugmented matrix before pivoting:\n");

    for (int i = 0; i < aug_mat.n_cols-1; ++i)
    {
        uword r;
        aug_mat.col(i).max(r);
        aug_mat.swap_rows(r, i);
    }

    // aug_mat.print("\nAugmented matrix after pivoting:\n");

    // aug_mat.print("\nAugmented matrix before elimination:\n");
    for (int i = 0; i < aug_mat.n_rows-1; ++i)
    {

        for (int j = i+1; j < aug_mat.n_rows; ++j)
        {
            double ratio = aug_mat(j, i)/aug_mat(i, i);
            aug_mat.row(j) = aug_mat.row(j) - ratio*aug_mat.row(i);
        }
    }

    // aug_mat.print("\nAugmented matrix after lower elimination:\n");

    for (int i = aug_mat.n_rows-1; i > 0; --i)
    {
        for (int j = i-1; j >= 0; --j)
        {
            double ratio = aug_mat(j, i)/aug_mat(i, i);
            aug_mat.row(j) = aug_mat.row(j) - ratio*aug_mat.row(i);
        }
    }

    // aug_mat.print("\nAugmented matrix after upper elimination:\n");

    vec b = aug_mat.col(aug_mat.n_cols-1)/aug_mat.diag();

    b.print("\nSolutions:\n");

    return 0;
}
```

Output

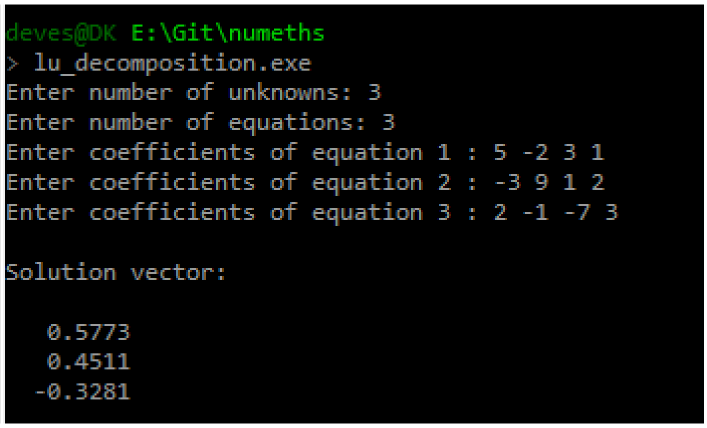


```
deves@DK E:\Git\numeths
> gauss_elimination_pivot.exe
Enter number of unknowns: 3
Enter number of equations: 3
Enter equation 1 : 5 -2 3 1
Enter equation 2 : -3 9 1 2
Enter equation 3 : 2 -1 -7 3

Solutions:

   0.5773
   0.4511
  -0.3281
```

LU Decomposition Method

```
/**
 * LU Decomposition Method
 * @author : Devesh Khandelwal
 */

#include <iostream>
#include <armadillo>
#include <cmath>

// Used because the double value may not be exactly zero.
// So, this is to remove almost-singular errors.
#define SINGULARITY_MARGIN 0.00001

using namespace std;
using namespace arma;

/**
 * main function. Inputs a system of equations and outputs the solution
 * to that system, if exists. Uses LU decomposition method.
 *
 * @param  argc Number of command line arguments.
 * @param  argv Command line arguments.
 * @return      Exit Status.
 *
 */
int main(int argc, char** argv)
{
    int vars, equations;

    // Default system if nothing is provided.
    mat coef_mat = {
        {1, 2, 3},
        {4, 5, 6},
        {1, 0, 1}
    };

    mat l(size(coef_mat)), u(size(coef_mat));
    colvec b, x, y;


    cout << "Enter number of unknowns: ";
    cin >> vars;

    cout << "Enter number of equations: ";
    cin >> equations;

    coef_mat.resize(equations, vars+1);

    // Input equation coefficients. Complete augmented matrix.
    for (unsigned i = 0; i < coef_mat.n_rows ; ++i)
    {
        cout << "Enter coefficients of equation " << i+1 << " : ";
        for (unsigned j = 0; j < coef_mat.n_cols ; ++ j)
        {
            cin >> coef_mat(i, j);
        }
    }

    // coef_mat.print("\nEquation matrix:\n");

    // Check singularity.
    if (abs(det(coef_mat.submat(0, 0, coef_mat.n_rows-1, coef_mat.n_cols-2)))<
        SINGULARITY_MARGIN)
    {
        cout << "Error: Coeffecient matrix is singular. No solution exists.";
        return EXIT_FAILURE;
    }

    // Initialize the b vector with values of last column of coefficients matrix.
    b = coef_mat.col(coef_mat.n_cols-1);

    // Reducing the size of the coefficients matrix by shedding the last column.
    coef_mat.shed_col(coef_mat.n_cols-1);

    //  set elements along main diagonal to one and off-diagonal elements to zero.
    //  Identity matrix
    l.eye();
    u = coef_mat;

    // coef_mat.print("\nCoefficient matrix:\n");

    // b.print("\nAnswer vector:\n");
    // l.print("\nLower triangular matrix before elimination:\n");
    // u.print("\nUpper triangular matrix before elimination:\n");

    for (int i = 0; i < coef_mat.n_rows-1; ++i)
    {

        for (int j = i+1; j < coef_mat.n_rows; ++j)
        {
            double ratio = u(j, i)/u(i, i);

            // Upper triangular matrix(U) by applyring elementary row operations.
            u.row(j) = u.row(j) - ratio*u.row(i);

            // Getting Lower triangular matrix(L) from Identity matrix
            l(j, i) = ratio;
        }
    }

    // l.print("\nLower triangular matrix after elimination:\n");
    // u.print("\nUpper triangular matrix after elimination:\n");

    // 'b' is B vector for 'y' // AX=B
    y = (l.i())*b;
    // y.print("\n y vector:\n");

    // 'y' is B vector for 'x'
    x = (u.i())*y;

    x.print("\nSolution vector:\n");
    return 0;
}
```

Output

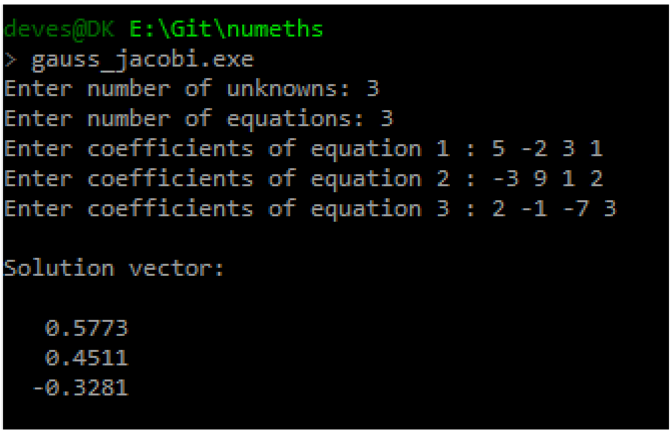


```
deves@DK E:\Git\numeths
> lu_decomposition.exe
Enter number of unknowns: 3
Enter number of equations: 3
Enter coefficients of equation 1 : 5 -2 3 1
Enter coefficients of equation 2 : -3 9 1 2
Enter coefficients of equation 3 : 2 -1 -7 3

Solution vector:

   0.5773
   0.4511
  -0.3281
```

Gauss-Jacobi Method

```
/**
 * Gauss Jacobi Method.
 * @author : Devesh Khandelwal
 */

#include <iostream>
#include <armadillo>
#include <cmath>

// Used because the double value may not be exactly zero.
// So, this is to remove almost-singular errors.
#define SINGULARITY_MARGIN 0.00001
#define ITERATION_COUNT 50

using namespace std;
using namespace arma;

/**
 * main function. Inputs a system of equations and outputs the solution
 * to that system, if exists. Uses Guass-Jacobi method.
 *
 * @param  argc Number of command line arguments.
 * @param  argv Command line arguments.
 * @return      Exit Status.
 *
 */
int main(int argc, char** argv)
{
    int vars, equations;

    // Default system if nothing is provided.
    mat coef_mat = {
        {1, 2, 3, 1},
        {4, 5, 6, 1},
        {1, 0, 1, 1}
    };

    colvec x, xi, b;

    cout << "Enter number of unknowns: ";
    cin >> vars;

    cout << "Enter number of equations: ";
    cin >> equations;

    // Resize the coef matrix
    coef_mat.resize(equations, vars+1);
    x.resize(vars);
    // Set all elements to zero
    x.zeros();
    xi.resize(vars);
    xi.zeros();

    // Input equation coefficients. Complete augmented matrix.
    for (unsigned i = 0; i < coef_mat.n_rows ; ++i)
    {
        cout << "Enter coefficients of equation " << i+1 << " : ";
        for (unsigned j = 0; j < coef_mat.n_cols ; ++ j)
        {
            cin >> coef_mat(i, j);
        }
    }

    // Check singularity.
    if (abs(det(coef_mat.submat(0, 0, coef_mat.n_rows-1, coef_mat.n_cols-2)))<
        SINGULARITY_MARGIN)
    {
        cout << "Error: Coeffecient matrix is singular. No solution exists.";
        return EXIT_FAILURE;
    }

    // coef_mat.print("\nEquation matrix:\n");

    // Initialize the b vector with values of last column of coefficients matrix.
    b = coef_mat.col(coef_mat.n_cols-1);
    // Reducing the size of the coefficients matrix by shedding the last column.
    coef_mat.shed_col(coef_mat.n_cols-1);

    // b.print("\nAnswer vector:\n");
    // x.print("\nSolution vector:\n");

    int count=0;
    while(++count < ITERATION_COUNT)
    {
        for (int i = 0 ; i < x.n_elem ; ++i)
        {
            double tmp=0;
            for (int j = 0; j < coef_mat.n_cols ; ++j)
            {
                if (i!=j)
                {
                    tmp = tmp + -1*coef_mat(i, j)*x(j);
                }
            }
            tmp += b(i);
            xi(i) = tmp/coef_mat(i, i);
        }
        x=xi;
        //x.print("\nSolution vector at iteration " + to_string(count) + " :\n");
    }

    x.print("\nSolution vector:\n");

    return 0;
}
```

Output



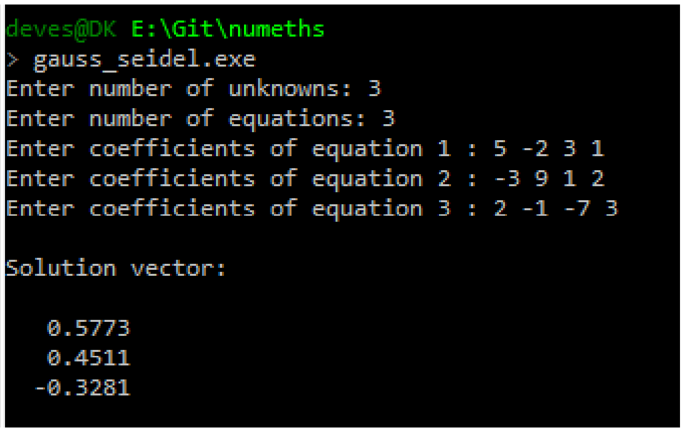
```
deves@DK E:\Git\numeths
> gauss_jacobi.exe
Enter number of unknowns: 3
Enter number of equations: 3
Enter coefficients of equation 1 : 5 -2 3 1
Enter coefficients of equation 2 : -3 9 1 2
Enter coefficients of equation 3 : 2 -1 -7 3

Solution vector:

   0.5773
   0.4511
  -0.3281
```

Gauss-Seidel Method

```
/**
 * Gauss Seidal Method
 * @author : Devesh Khandelwal
 */

#include <iostream>
#include <armadillo>
#include <cmath>

// Used because the double value may not be exactly zero.
// So, this is to remove almost-singular errors.
#define SINGULARITY_MARGIN 0.00001
#define ITERATION_COUNT 50

using namespace std;
using namespace arma;

/**
 * main function. Inputs a system of equations and outputs the solution
 * to that system, if exists. Uses Gauss-Seidel method.
 *
 * @param  argc Number of command line arguments.
 * @param  argv Command line arguments.
 * @return      Exit Status.
 *
 */
int main(int argc, char** argv)
{
    int vars, equations;

    // Default system if nothing is provided.
    mat coef_mat = {
        {1, 2, 3, 1},
        {4, 5, 6, 1},
        {1, 0, 1, 1}
    };

    colvec x, xi, b;

    cout << "Enter number of unknowns: ";
    cin >> vars;

    cout << "Enter number of equations: ";
    cin >> equations;

    // Resize the coef matrix
    coef_mat.resize(equations, vars+1);
    x.resize(vars);
    // Set all elements to zero
    x.zeros();
    xi.resize(vars);
    xi.zeros();

    // Input equation coefficients. Complete augmented matrix.
    for (unsigned i = 0; i < coef_mat.n_rows ; ++i)
    {
        cout << "Enter coefficients of equation " << i+1 << " : ";
        for (unsigned j = 0; j < coef_mat.n_cols ; ++ j)
        {
            cin >> coef_mat(i, j);
        }
    }

    // Check singularity.
    if (abs(det(coef_mat.submat(0, 0, coef_mat.n_rows-1, coef_mat.n_cols-2)))<
        SINGULARITY_MARGIN)
    {
        cout << "Error: Coeffecient matrix is singular. No solution exists.";
        return EXIT_FAILURE;
    }

    // coef_mat.print("\nEquation matrix:\n");

    // Initialize the b vector with values of last column of coefficients matrix.
    b = coef_mat.col(coef_mat.n_cols-1);

    // Reducing the size of the coefficients matrix by shedding the last column.
    coef_mat.shed_col(coef_mat.n_cols-1);

    // b.print("\nAnswer vector:\n");
    // x.print("\nSolution vector:\n");

    int count=0;
    while(++count < ITERATION_COUNT)
    {
        for (int i = 0 ; i < x.n_elem ; ++i)
        {
            double tmp=0, tmpi=0;
            for (int j = 0; j < i ; ++j)
            {
                if (i!=j)
                {
                    tmp = tmp + coef_mat(i, j)*xi(j);
                }
            }
            tmp *= -1;
            for (int j = i; j < coef_mat.n_cols ; ++j)
            {
                if (i!=j)
                {
                    tmpi = tmpi + coef_mat(i, j)*x(j);
                }
            }
            tmpi *= -1;
            xi(i) = (tmp + tmpi + b(i))/coef_mat(i, i);
        }
```

```
            x=xi;
            //x.print("\nSolution vector at iteration " + to_string(count) + " :\n");
        }

        x.print("\nSolution vector:\n");

        return 0;
    }
```

Output



```
deves@DK E:\Git\numeths
> gauss_seidel.exe
Enter number of unknowns: 3
Enter number of equations: 3
Enter coefficients of equation 1 : 5 -2 3 1
Enter coefficients of equation 2 : -3 9 1 2
Enter coefficients of equation 3 : 2 -1 -7 3

Solution vector:

   0.5773
   0.4511
  -0.3281
```

Namaste.

THE END.