Gauss Elimination Using Scaled Partial Pivoting

Algorithm

1. Enter the no. of equations, iDim
2. Create an iDim * (iDim+1) matrix, which will be the augmented matrix.
3. Create an array ‘x’ of size ‘iDim’ which will store the solutions.
4. Enter the elements of augmented matrix.
5. selectMax:
    select max absolute value from each row and create scale vector
6. solve
    repeat the scaled partial pivoting and forward elimination for iDim-1 times
    Scaled Partial Pivoting:
        scale the first element in each row and store in the scaledElements vector
        determine the index of the max scaled element and swap its row with the first row
    Forward Elimination:
        zero out all the values under the diagonal and print the matrix after elimination
7. Back-substitution:
    calculate values of x and check the solution validity
8. Print the solution i.e. the elements of x.

Code

main.cpp

```cpp
#include <iostream>
#include <vector>
#include "gauss_elimination.h"

using namespace std;

int main()
{
    int iDim;
    std::vector<float> _scale;
    std::cout.precision(4);        //set precision
    std::cout.setf(ios::fixed);    //display using the fixed precision
    std::cout << "\nEnter the no. of equations\n";
    std::cin >> iDim;                //input the no. of equations

    float *p[iDim];
    float arrA[iDim][iDim+1];  //Array declaration to store the augmented-matrix elements
    std::cout << "\nEnter the elements of the augmented-matrix row-wise:\n";
    for (int i = 0; i < iDim; i++)
        for (int j = 0; j <= iDim; j++)
        {
            std::cout << "A[" << i << "][" << j << "]=";
            std::cin >> arrA[i][j];
        };
    for (int i = 0; i < iDim; ++i)
        p[i] = arrA[i];

    GaussElimination matrix = GaussElimination();
    std::cout << "\nThis is the matrix to be solved\n";
    matrix.printMatrix(p, iDim);
    matrix.selectMax(p, iDim, _scale);
    for (int step = 0; step < iDim-1; step++)
        matrix.solve(p, iDim,_scale, step);
    matrix.backwardSubstitution(p, iDim);
    return 0;
}
```

gauss_elimination.h

```
//Gauss Elimination by Asmaa Ali - ID: 1910069

using namespace std;

class GaussElimination
{
    public:
        /**
        @param float **a  the augmented matrix
        @param  n  array size
        @param  scale the scale vector
        @param  step the forward elimination step
        */
        void printMatrix(float **a, int n);

        void selectMax(float **a, int n, vector<float> &scale);

        void backwardSubstitution(float **a, int n);

        void solve(float **a, int n, vector<float> &scale, int step);
        /**
        Checks solution validity
        */
        bool valid_solution = false;

    private:
        /**
        @param float **a  the augmented matrix
        @param  n  array size
        @param  scale the scale vector
        @param  step the forward elimination step
        */
        void scaledPartialPivoting(float **a, int n, vector<float> &scale, int step);

        void forwardElimination(float **a, int n, int step);
};

```

gauss_elimination.... ×

```cpp
//Gauss Elimination
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <cstdint>
#include <cmath>
#include "gauss_elimination.h"

using namespace std;

void GaussElimination::printMatrix(float **a, int n)
{
    int i,j;
    for (i=0;i<n;i++)            //print the new matrix
    {
        for (j=0;j<=n;j++)
            std::cout << a[i][j] << std::setw(16);
        std::cout << "\n";
    };
    std::cout << "\n";
}

static bool abs_compare(int a, int b)
{
    return (std::abs(a) < std::abs(b));
}

void GaussElimination::selectMax(float **a, int n, vector<float> &scale)
{
    for (int i = 0; i < n; i++)
    {
        float max = *std::max_element(a[i], a[i] + n, abs_compare);
        scale.push_back(max);
    }
}

void GaussElimination::scaledPartialPivoting(float **a, int n, vector<float> &scale, int step)
{
    std::vector<float> scaledElements;
    for (int i = step; i < n; i++)
        scaledElements.push_back(std::abs(a[i][step]/scale[i]));
        //std::cout << a[i][step] << "\t/ scale\t" << scale[i] << "\t= scaledElement  (" << scaledElements[i-step] << ")\n" ;
    int maxScaledElementidx = std::max_element(scaledElements.begin(),scaledElements.end()) - scaledElements.begin();
    //std::cout << "\n" << maxScaledElementidx+step << "\n\n";
    if (maxScaledElementidx+step > step)
    {
        std::swap(a[step], a[maxScaledElementidx+step]);
        std::swap(scale[step], scale[maxScaledElementidx+step]);
    }
    //std::printf("This is the matrix after selecting num %d pivotEQ equation\n", step+1);
    //printMatrix(a, n);
}

void GaussElimination::forwardElimination(float **a, int n, int step)
{
    for (int i = step+1; i < n; i++)
    {
        float x = a[i][step];
        for (int j = step; j <= n; j++)
            a[i][j] = a[i][j] - ((x/a[step][step]) * a[step][j]);
    }

    std::printf("\nThe matrix after step %d of elimination\n", step+1);
    printMatrix(a, n);
}

void GaussElimination::solve(float **a, int n, vector<float> &scale, int step)
{
    scaledPartialPivoting(a, n, scale, step);
    forwardElimination(a, n, step);
}
```

```
74  void GaussElimination::backwardSubstitution(float **a, int n)
75  {
76      float x[n], R[n];
77      for (int i = n-1; i >= 0; i--)
78      {
79          x[i] = a[i][n];
80          for (int j = i+1; j < n; j++)
81              if (j != i)
82                  x[i] = x[i] - a[i][j] * x[j];
83          x[i] = x[i] / a[i][i];
84          std::printf("X%d =\t %.4f\n", i+1, x[i]);
85      }
86      std::cout << "\n";
87      //R = AX-B
88      for (int i = 0; i < n; i++)
89          for (int j = 0; j < n; j++)
90              R[i]+=( a[i][j]*x[j]);
91
92      for (int i = 0; i < n; i++)
93      {
94          R[i] = R[i] - a[i][n];
95          std::printf("R%d =\t %.4f\n", i+1, R[i]);
96      }
97      //check solution validity
98      if (*max_element(R , R + n) <= 0.04)
99          std::cout << "\nvalid_solution = true\n\n";
100     else
101         std::cout << "\nvalid_solution = false\n\n";
102 }
103
```

Output

```
asmaa@asmaa-ali:~/Gauss Elimination SPP$ ./gauss_elimination

Enter the no. of equations
3

Enter the elements of the augmented-matrix row-wise:
A[0][0]=2
A[0][1]=1
A[0][2]=-1
A[0][3]=0
A[1][0]=1
A[1][1]=4
A[1][2]=3
A[1][3]=14
A[2][0]=-1
A[2][1]=2
A[2][2]=7
A[2][3]=30

This is the matrix to be solved
2.0000          1.0000          -1.0000          0.0000
1.0000          4.0000           3.0000         14.0000
-1.0000          2.0000           7.0000         30.0000


The matrix after step 1 of elimination
2.0000          1.0000          -1.0000          0.0000
0.0000          3.5000           3.5000         14.0000
0.0000          2.5000           6.5000         30.0000


The matrix after step 2 of elimination
2.0000          1.0000          -1.0000          0.0000
0.0000          3.5000           3.5000         14.0000
0.0000          0.0000           4.0000         20.0000

X3 =     5.0000
X2 =     -1.0000
X1 =     3.0000

R1 =     0.0019
R2 =     0.0000
R3 =     0.0019

valid_solution = true
```

```
asmaa@asmaa-ali:~/Gauss Elimination SPP$ g++ -o gauss_elimination gauss_elimination.cpp main.cpp
asmaa@asmaa-ali:~/Gauss Elimination SPP$ ./gauss_elimination

Enter the no. of equations
4

Enter the elements of the augmented-matrix row-wise:
A[0][0]=1
A[0][1]=-1
A[0][2]=2
A[0][3]=1
A[0][4]=1
A[1][0]=3
A[1][1]=2
A[1][2]=1
A[1][3]=4
A[1][4]=1
A[2][0]=5
A[2][1]=8
A[2][2]=6
A[2][3]=3
A[2][4]=1
A[3][0]=4
A[3][1]=2
A[3][2]=5
A[3][3]=3
A[3][4]=-1

This is the matrix to be solved
1.0000         -1.0000          2.0000          1.0000          1.0000
3.0000          2.0000          1.0000          4.0000          1.0000
5.0000          8.0000          6.0000          3.0000          1.0000
4.0000          2.0000          5.0000          3.0000         -1.0000


The matrix after step 1 of elimination
4.0000          2.0000          5.0000          3.0000         -1.0000
0.0000          0.5000         -2.7500          1.7500          1.7500
0.0000          5.5000         -0.2500         -0.7500          2.2500
0.0000         -1.5000          0.7500          0.2500          1.2500


The matrix after step 2 of elimination
4.0000          2.0000          5.0000          3.0000         -1.0000
0.0000         -1.5000          0.7500          0.2500          1.2500
0.0000          0.0000          2.5000          0.1667          6.8333
0.0000          0.0000         -2.5000          1.8333          2.1667


The matrix after step 3 of elimination
4.0000          2.0000          5.0000          3.0000         -1.0000
0.0000         -1.5000          0.7500          0.2500          1.2500
0.0000          0.0000         -2.5000          1.8333          2.1667
0.0000          0.0000          0.0000          2.0000          9.0000

X4 =     4.5000
X3 =     2.4333
X2 =     1.1333
X1 =     -7.2333

R1 =     0.0000
R2 =     5.0000
R3 =     0.0000
R4 =     0.0000

valid_solution = false
```