Lab 3 - 实现稀疏矩阵以及高斯赛达尔迭代法

一、实验要求

1.稀疏矩阵

本实验要求自行实现一种稀疏矩阵。推荐的稀疏矩阵存储方式包括：

1. Compressed Row Storage
2. Compressed Sparse Column
3. 其他形式的稀疏矩阵存储方式

同学们请在实验报告中明确指出自己实现的稀疏矩阵方式，并保证实现的稀疏矩阵具备以下的几种最基本的功能：

1. `at(row, col)`：根据`row`和`column`的系数来查询矩阵里面的元素的数值
2. `insert(val, row, col)`： 将`val`替换/插入到`(row, col)`这个位置去
3. `initializeFromVector(rows, cols, vals)`：根据向量来初始化一个稀疏矩阵。其中`rows`，`cols`，`vals`皆为等长度的向量。`rows`里面存的是行系数，`cols`里面存的是列系数，`vals`里面存的是数值。
4. 其余的基本功能可以参考Matlab里面的sparse函数，或者Eigen Library里面的Sparse Matrix的介绍。

2.稀疏矩阵的高斯赛达尔迭代法

本实验要求在自己实现的稀疏矩阵的表达的基础上，实现高斯赛达尔迭代法（Gauss-Seidel Method），用于求解大规模的稀疏线性方程组。

关于高斯赛达尔方法的介绍已经在课件中进行了详细的介绍。请同学们参照课件进行代码的编写。

作为验证实现的正确性，这里给出一个小的Test Case:

$$A = \begin{pmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{pmatrix}$$

求解 得到的结果应该为：

$x = [1, 2, -1, 1]$;

3.共轭梯度法

参考相关资料实现共轭梯度法求解线性方程组。

二、运行环境

编译环境：MinGW-GCC

运行环境：Windows10

三、具体实现

1.稀疏矩阵

稀疏矩阵，其定义为非零元素个数很少的矩阵。在日常的实际操作中，如果我们用传统的方式去存储稀疏矩阵，则村粗大量的0是非常浪费空间的，尤其是矩阵的尺寸量级非常大时。

因此为了方便操作和节省空间，我们只存储非0元素，这样在处理稀疏矩阵时便能很好地解决这些问题。课堂上已经介绍了压缩行存储、压缩列存储等存储方式，这些都是非常有效的数据结构。

在本次实验中我采用十字链表法，基于此方法存储稀疏矩阵并实现高斯赛达尔迭代法及共轭梯度法求解线性方程组。

```
class Node{
    public:
        int row;//行
        int col;//列
        double value;//值
        Node *col_next;//指向下一列
        Node *row_next;//指向下一行
        Node(int row,int col,double value){
            this->row=row;
            this->col=col;
            this->value=value;
            this->col_next=nullptr;
            this->row_next=nullptr;
        }
        Node(){
            this->row=0;
            this->col=0;
            this->value=0;
            this->col_next=nullptr;
            this->row_next=nullptr;
        }
};
```

1.1矩阵类

矩阵类的完整定义如下：

```
class Sparsematrix{
    public:
        int rows;//矩阵行数
        int cols;//矩阵列数
        int number;//矩阵非零元素数量
        Node *element;
        Sparsematrix(int row_number,int col_number);
        double at(int row,int col);
        Node *insert(double val,int row,int col);
        void initializeFromVector(vector<double> rows, vector<double> cols,
vector<double> vals);
        void printmatrix();
        void Gauss_Seidel(double B[],double x[]);
        vector<double> Ax(vector<double> x);
        void Conjugate_gradient(double B[],double x[]);
};
```

1.2 `at(row, col)` 函数

本函数要求我们根据 `row` 和 `column` 的系数来查询矩阵里面的元素的数值。本函数的实现思路较为直接，只需要根据行列系数进行遍历找到相应位置的数值并返回即可。

```
double at(int row,int col){
    Node *row_now=&element[row];
    Node *ptr=nullptr;
    while(row_now){
        if(row_now->col==col){
            ptr=row_now;
        }
        row_now=row_now->col_next;
    }
    if(ptr==nullptr){
        return 0;
    }
    else{
        return ptr->value;
    }
}
```

1.3 `insert(val, row, col)` 函数

本函数要求我们将 `val` 替换/插入到 `(row, col)` 这个位置去。

本函数的思路也非常直接，对当前位置的各种情形进行判断，并对每个情形采取相应的措施将值存入指定位置。当插入值不为零食，总共有三种插入的情形，其中第一种和第三种情形较为简单，判断条件较为直接，而第二种情形的判别条件比较复杂，需要同时满足 `row_now->col<col` 和 `row_now->col_next->col>col`，这说明我们需要插入的位置位于已有的两个结点中间，且同时需要有 `row_now->col_next!=nullptr`。

```
Node *insert(double val,int row,int col){
    if(val==0) return nullptr;//若值为0则不需要进行插入操作
    Node *row_now=&element[row];//找到指定行数
    while(row_now){
        if(row_now->col==col){//找到指定位置
            if(row_now->value==0){
                number++;//若原先位置值为0则矩阵非零元素数加一
            }
            row_now->value=val;
            return row_now;
        }
        if(row_now->col<col&&row_now->col_next&&row_now->col_next->col>col){//插
入在两个结点中间
            Node *nele=new Node(row,col,val);
            number++;
            nele->col_next=row_now->col_next;
            row_now->col_next=nele;
            return nele;
        }
        if(row_now->col_next==nullptr){//插入在末尾
            Node *nele=new Node(row,col,val);
            number++;
            row_now->col_next=nele;
            return nele;
```

```
        }
        row_now=row_now->col_next;
    }
}
```

1.4 `initializeFromVector(rows, cols, vals)` 函数

本函数要求我们根据向量来初始化一个稀疏矩阵。其中 `rows`，`cols`，`vals` 皆为等长度的向量。`rows` 里面存的是行系数，`cols` 里面存的是列系数，`vals` 里面存的是数值。

我采用了一种较为简便的方法，调用此函数时每次更新矩阵的行列数等各参数，创建一个新的矩阵，再根据行列系数将相应的数值利用 `insert()` 函数存储进入新的矩阵中。

```
void initializeFromVector(vector<double> rows, vector<double> cols,
vector<double> vals){
    delete this->element;
    this->number=0;
    this->rows=*max_element(rows.begin(), rows.end()) + 1;
    this->cols=*max_element(cols.begin(), cols.end()) + 1;
    element = new Node[this->rows];;
    for(int i=0;i<rows.size();i++){
        insert(vals[i],rows[i],cols[i]);
    }
}
```

1.5其它函数

高斯赛达尔迭代法的函数和共轭梯度法的函数在下一部分作详细阐述，在本题的具体实现过程中，还利用了矩阵的打印函数：

```
void printmatrix() {
    cout << "matrix " << this->rows << "x" << this->cols << ":" << endl;
    for (int i = 0; i < this->rows; i++) {
        for (int j = 0; j < this->cols; j++) {
            cout << this->at(i, j) << "  ";
        }
        cout << endl;
    }
}
```

2.高斯赛达尔迭代法

Gauss-Seidel方法是一种依次求解线性方程组$Ax = b$的技术，我们可以直接利用已有的结论进行求解：

$$x_i^{(k)} = \frac{b_i - \sum_{j<i} a_{ij}x_j^{(k)} - \sum_{j>i} a_{ij}x_j^{(k-1)}}{a_{ii}}$$

从矩阵的角度看，Gauss-Seidel方法也可以理解为$x^{(k)} = (D - L)^{-1}(Ux^{(k-1)} + b)$，其中几个矩阵分别表示$A$的对角线、严格下三角、严格上三角部分。

```
vector<double> Gauss_Seidel(double B[]){
    vector <double> result;
    vector <double> pre_result;
    result.assign(this->rows,0);//初始化解向量
```

```
    int num=0;
    double error=0.000001;
    do{
        num++;
        pre_result=result;
        if(num==1) pre_result.assign(this->rows,1.0);
        for(int i=0;i<this->rows;i++){
            if(at(i,i)==0) continue;
            double sum1=0;
            double sum2=0;
            for(int j=0;j<i;j++){
                sum1+=at(i,j)*result[j];-------------------------------------(1)
            }
            for(int j=i+1;j<this->rows;j++){
                sum2+=at(i,j)*pre_result[j];---------------------------------(2)
            }
            result[i]=(B[i]-sum1-sum2)/at(i,i);--------------------------(3)
        }
    }while(!limit(result,pre_result,error));//循环终止条件
    return result;-----------------------------------------------------------(4)
}
```

对于`(1)`、`(2)`、`(3)`部分，根据公式求解$x_i^{(k)}$，`(4)`返回解向量。

其中`limit()`函数定义如下：

```
static bool limit(vector<double>& v1, vector<double>& v2,double error) {
    double res = 0.0;
    for (int i = 0; i < v1.size(); i++) {
        res += abs(v1[i] - v2[i]);
    }
    if (res <=error) return true;
    else return false;
}
```

根据此函数判断高斯赛达尔迭代法是否终止迭代，当终止时说明已经求得误差范围内的解。

3.共轭梯度法

根据维基百科中的共轭梯度法，我们需要用到以下几个数学公式：

$r_0 = b - Ax_0 = p_0$，$\alpha_0 = \frac{r_0^T r_0}{p_0^T A p_0}$，$x_1 = x_0 + \alpha_0 p_0$，

$r_1 = r_0 - \alpha_0 A p_0$，$\beta_0 = \frac{r_1^T r_1}{r_0^T r_0}$，$p_1 = r_1 + \beta_0 p_0$，$\alpha_1 = \frac{r_1^T r_1}{p_1^T A p_1}$，$x_2 = x_1 + \alpha_1 p_1$

其中α_k和β_k还可以写为$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$，$\beta_k = -\frac{r_{k+1}^T A p_k}{p_k^T A p_k}$的形式。

接下来我们直接根据公式进行求解，首先我们需要写一个求矩阵和向量乘积的函数，常用来求解`Ax`，其返回值为一个向量。

```
vector<double> Ax(vector<double> x){
    vector<double> Ax_k;
    for (int i=0;i<this->rows;i++) {
        double sum=0;
        for (int j=0;j<this->cols;j++) {
            sum+=this->at(i, j)*x[j];
        }
        Ax_k.push_back(sum);
    }
    return Ax_k;
}
```

下面是共轭梯度法的具体实现过程：

```
vector<double> Conjugate_gradient(double B[]){
    vector<double> xk;
    xk.assign(this->rows,0);
    vector<double> b;
    for(int i=0;i<this->rows;i++){
        b.push_back(B[i]);
    }
    vector<double> rk;
    for(int i=0;i<b.size();i++){
        vector<double> Ax_k;
        Ax_k=this->Ax(xk);
        rk.push_back(b[i]-Ax_k[i]);-------------------------------------(1)
    }
    vector<double> pk=rk;
    int k=0;
    while(1){
        double rrk=0;
        for(int i=0;i<rk.size();i++){
            rrk+=rk[i]*rk[i];
        }
        vector<double> Ap_k=this->Ax(pk);
        double pAp=0;
        for(int i=0;i<pk.size();i++){
            pAp+=pk[i]*Ap_k[i];
        }
        double alpha_k=rrk/pAp;-----------------------------------------(2)
        vector<double> xk1;
        for(int i=0;i<xk.size();i++){
            xk1.push_back(xk[i]+alpha_k*pk[i]);-------------------------(3)
        }
        xk=xk1;
        vector<double> rk1;
        double err=0;
        for(int i=0;i<rk.size();i++){
            rk1.push_back(rk[i]-alpha_k*(this->Ax(pk)[i]));
            err+=abs(rk1[i]);
        }
        if(err<0.00000001) break;
        double rrk1=0;
        for(int i=0;i<rk1.size();i++){
            rrk1+=rk1[i]*rk1[i];
        }
```

```
        double beta_k=rrk1/rrk;----------------------------------------(4)
        vector<double> pk1;
        for(int i=0;i<rk1.size();i++){
            pk1.push_back(rk1[i]+beta_k*pk[i]);-------------------------(5)
        }
        pk=pk1;
        rk=rk1;
        k++;
    }
    return xk;-----------------------------------------------------------(6)
}
```

非常容易发现 `(1)`、`(2)`、`(3)`、`(4)`、`(5)` 部分便是上述数学公式的具体代码实现，最后 `(6)` 返回解向量。

四、实验结果及分析

4.1测试结果

测试 `insert()` 函数和 `at()` 函数：

```
Sparsematrix s(3 ,3);
s.insert(6, 0, 0);
s.insert(7, 0, 1);
s.insert(5, 1, 2);
s.insert(4, 2, 1);
cout << s.at(0, 0) << endl;
cout << s.at(0, 1) << endl;
cout << s.at(1, 0) << endl;
cout << s.at(2, 1) << endl;
s.printmatrix();
```

结果为：

```
6
7
0
4
matrix 3×3:
6  7  0
0  0  5
0  4  0
```

函数实现效果符合预期，且经过多次验证后均得到正确结果。

测试 `initializeFromVector()` 函数：

```
vector<double> r1 = { 0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3 };
vector<double> c1 = { 0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3 };
vector<double> v1 = { 10,-1,2,0,-1,11,-1,3,2,-1,10,-1,0,3,-1,8 };
s.initializeFromVector(r1, c1, v1);
s.printmatrix();
```

结果为：

```
matrix 4×4:
10  -1  2  0
-1  11  -1  3
2  -1  10  -1
0  3  -1  8
```

上面的结果符合预期，接下来利用此矩阵样例测试高斯赛达尔函数：

```
double B[] = { 6,25,-11,15 };
vector<double> x=s.Gauss_Seidel(B);
cout << "Solution:" << endl;
for (int i = 0; i < s.cols; i++)
    cout << x[i] << " ";
cout << endl;
```

结果为：

```
Solution:
1 2 -1 1
```

与测试样例的预期结果相符合，经过多次验证后我们均得到了正确结果，在此不一一展示。

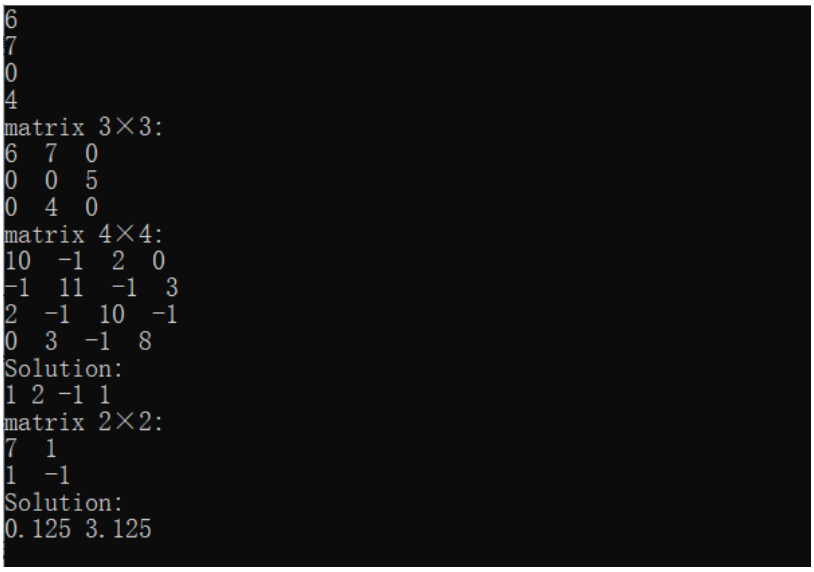
下面测试共轭梯度法函数的正确性：

```
vector<double> r2 = { 0,0,1,1 };
vector<double> c2 = { 0,1,0,1 };
vector<double> v2 = { 7,1,1,-1 };
s.initializeFromVector(r2, c2, v2);
s.printmatrix();
double B2[] = { 4,-3 };
vector <double>x2=s.Conjugate_gradient(B2);
cout << "Solution:" << endl;
for(int i=0;i<s.rows;i++){
    cout << x2[i] << " ";
}
cout << endl;
```

结果为：

```
matrix 2×2:
7  1
1  -1
Solution:
0.125 3.125
```

与预期结果相符合，验证了共轭梯度法函数的正确性。



4.2问题分析与改进

我原本希望实现系数矩阵的加减乘除，但在尝试的过程中发现，如果遇到运算后某原先值不为零的结点其值变为了零或是某原先值为零的结点其值不为零的情况，该如何保持节省空间和便于存储的"初心"成了较大的问题，需要对结点值进行判断并在需要的情况下释放特定的空间，但如何花费较小的时间和空间解决这个问题因为时间和精力有限我并未解决。

同时由于时间和能力有限，在本次实验中我并未选取大规模的数据进行样例的测试，对于函数的性能测试尚且不足，这也是程序进一步可以改进的地方。

五、心得体会

本次作业耗时依旧较久，花费了一定时间学习了`C++`类的相关知识，才能够较好地理解和完成本次作业，其中和数理相关部分的程序我可以较好的理解和实现，但其余部分的完成花费了较多的时间和精力，也参考了网上较多的代码和讲解。

总体而言这次作业我实现的功能比较简单，相对而言并没有什么出彩之处，但是目前能力也仅限于此了，希望在之后的过程中能够在完成基本要求的基础之上有一定的突破。

六、参考文献

(1) http://mathworld.wolfram.com/Gauss-SeidelMethod.html

(2) http://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method

(3) https://en.wikipedia.org/wiki/Conjugate_gradient_method

(4) https://www.runoob.com/cplusplus/cpp-classes-objects.html

(5) http://data.biancheng.net/view/138.html