CS319: Scientific Computing (with C++)

Computing with vectors and matrices in C++

Week 7: **9am** and **4pm**, 22 Feb 2017

Lab 4...

Lab 4 is due 9am, Monday. before you submit, test against the six test programs, and check the rubric.

Name: **CS319 Lab 4**

Description: **Grading scheme for Lab 4 (Syntax checker)**

Exit

Grid View | List View

	Unsatisfactory	Good	Excellent
Code compiles	**0** (0%) Program does not compile	**1.5** (5%) Program compiles, but with 1 or more warnings	**3** (10%) Program compiles without any warnings
Name, ID, and Comments	**0** (0%) Name and ID number missing	**1.5** (5%) Name and ID number given. Code documents how it works.	**3** (10%) Name and ID number given. Code documents how it works (comments and useful variable names). Correct explanation of output provided
File handling	**0** (0%) File not opened correctly. Data not correctly read from file	**1.5** (5%) File opened, but no check that this is done correctly. Data read from file, but not perfectly (e.g., end-of-file not correctly detected).	**3** (10%) File opened, and operation verified. Data correctly read from file.
Stack usage	**0** (0%) Stack not used correctly	**1.5** (5%) Imperfect use of stack for storing braces	**3** (10%) Stack used fully correctly.
Results for the Test Case 1-3	**0** (0%) Numerous errors misidentified. Useful output not provided.	**2.7** (9%) Passes 2 of the tests. Useful output provided.	**4.5** (15%) Passes all 3 tests. Informative output provided.
Results for Test Cases 4-6	**0** (0%) Numerous errors not identified	**2.7** (9%) Passes 2 of the tests.	**4.5** (15%) Passes all 3 tests.
Programming skill demonstrated	**1.2** (4%) Little programming skill demonstrated.	**3.6** (12%) Programming proficiency is obvious	**6** (20%) Skill level shown goes well beyond the minimum acceptable standard.
Documentation	**0** (0%) No information provided on the capabilities and limitations of the program	**1.8** (6%) Some information provided on the capabilities and limitations of the program	**3** (10%) Resp9onds of program to all reasonable scenarios provided.

Introduction

This is a course in Scientific Computing. "Sci-Comp" problems that we've looked at so far include

- optimisation (Lab 2)
- searching and list processing (Week 6)

Many of the more advanced and more general problems in Scientific Computing are based around **vectors** and **matrices**. So one of our goals is to implement C++ classes for such structures, along with standard operations such as matrix-vector multiplication.

Along the way, we'll learn about

- operator overloading;
- `friend` functions and the `this` pointer;
- `class template`s;
- static variables.
- and much more

Our first step will be to study some problems and applications so that, before we design any classes or algorithms, we'll know what we will use them for. These problems include:

1. Basic analysis of matrices, for example with applications to image processing, graphs and networks.
2. Solution of linear systems of equations, for example with applications to data fitting;
3. Estimation of (certain) eigenvalues, for example with applications to search engine analysis.

Of these problems, probably the most ubiquitous is the solution of (large) systems of simultaneous equations. That is, we want to solve a linear system of N equations in N unknowns: *find* $x_1, x_2, \ldots, x_N$, *such that*

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N &= b_2 \\ &\vdots \\ a_{N1}x_1 + a_{N2}x_2 + \cdots + a_{NN}x_N &= b_N. \end{aligned}$$

Solving linear systems

There are several classic approaches:

1. Gaussian Elimination;
2. Related: LU- and Cholesky factorisation;
3. Stationary Iterative schemes such as **Jacobi's method**, **Gauss-Seidel** and Successive Over Relaxation (SOR);
4. Krylov subspace methods, of which Conjugate Gradients is the best known;
5. Enhancements of the Methods 3 and 4, using preconditioning with, for example, MultiGrid and Incomplete LU-factorisation.

Of the approaches listed above, Jacobi's is by far the simplest to implement, and so is the one we will study first.

See notes.

Jacobi's Method

Implementation

Now that we know the method, let us summarise the steps, so as to work out what standard operations on vectors and matrices we need.

We expressed the problem as a matrix-vector equation: *Find* $\mathbf{x}$ *such that*

$$A\mathbf{x} = \mathbf{b},$$

where A *is a* $N \times N$ *matrix, and* $\mathbf{b}$ *and* $\mathbf{x}$ *are (column) vector with* N *entries.*

We then derived **Jacobi's method**: choose $\mathbf{x}^{(0)}$ and set

$$x^{(k+1)} = D^{-1}(b + Tx^{(k)}).$$

where $D = \mathrm{diag}(A)$ and $T = D - A$.

Looking at this we see that the fundamental operations are: **vector addition** and **matrix-vector multiplication**.

Vectors

Our first focus will be on defining a class of vectors. Intuitively, we know it needs the following components:

C++ "Project"

Due to the level of detail in the matrix and vector classes, the following example is divided into three source files:

1. `Vector.h`, the header file which contains the class definition. Include this header file in another source file with: `#include "Vector.h"`
   Note that this is **not** `<Vector.h>`
2. `Vector.cpp`, which includes the code for the methods in the `vector` class;
3. `01TestVector.cpp`, a test stub.

The test stub can be compiled from the command line with

```
g++ -Wall Vector.cpp 01TestVector.cpp
```

Using `Code::blocks` you need to create a new "project" and include all three source files.

C++ "Project"

See `Vector.h` for more details

```
// File:  Vector.h
// Author:  Niall Madden (NUI Galway) Niall.Madden@NUIGalway.ie
// Date:  22/02/2017 (Week 7 of 1617-CS319)
// What:  Header file for vector class
// See also:  Vector.cpp and 02TestVector.cpp

class vector {
private:
  double *entries;
  unsigned int N;
public:
  vector (unsigned int Size=2);
  ~vector(void);

  unsigned int size(void) {return N;};
  double geti (unsigned int i);
  void seti (unsigned int i, double x);

  void print(void);
  double norm(void); // Compute the 2-norm of a vector
  void zero(void); // Set entries of vector to zero.
};
```

C++ "Project"

Vector.cpp

```
#include "Vector.h"

vector::vector (unsigned int Size) {
  N = Size;
  entries = new double[Size];
}

vector::~vector () {
  delete [] entries;
}

void vector::seti (unsigned int i, double x) {
  if (i<N)
    entries[i]=x;
  else
    cerr << "vector::seti(): Index out of bounds." << std::endl;
}

double vector::geti (unsigned int i) {
  if (i<N)
    return(entries[i]);
  else {
    cerr << "vector::geti(): Index out of bounds." << std::endl;
    return(0);
  }
}
```

Vector.cpp continued

```
38  void vector::print (void)
    {
40    for (unsigned int i=0; i<N; i++)
        std::cout << "[" << entries[i] << "]" << std::endl;
42  }

44  double vector::norm (void)
    {
46    double x=0;
      for (unsigned int i=0; i<N; i++)
48      x+=entries[i]*entries[i];
      return (sqrt(x));
50  }

52  void vector::zero(void)
    {
54    for (unsigned int i=0; i<N; i++)
        entries[i]=0;
56  }
```

Adding two vectors

Here is a simple implementation of a function that computes $\mathbf{c} = \alpha\mathbf{a} + \beta\mathbf{b}$

See `01TestVector.cpp` for more details

```
14 // c = alpha*a + beta*b where a,b are vectors; alpha, beta are scalars
   void VecAdd (vector &c, vector &a, vector &b,
16         double alpha, double beta)
   {
18   unsigned int N;
     N = a.size();

     if ( (N != b.size()) )
22     std::cerr << "dimension mismatch in VecAdd " << std::endl;
     else
24   {
       for (unsigned int i=0; i<N; i++)
26       c.seti(i, alpha*a.geti(i)+beta*b.geti(i) );
     }
28 }
```

The matrix class

Our next step is to write a `class` implementation for a matrix, along with the associated functions.

Then we need to define a function to multiply a matrix by vector.

First though, we consider the matrix representation. The most natural approach might seem to be to construct a two dimensional array. This can be done as follows:

A simpler, faster approach is to store the N^2 entries of the matrix in a single, one-dimensional, array of length N^2, and then take care how the access is done:

The matrix class

See `Matrix.h` for more details

```
// File:  Matrix.h
// Author:  Niall Madden (NUI Galway) Niall.Madden@NUIGalway.ie
// Date:  22/02/2017 (Week 7 of 1617-CS319)
// What:  Implementation of a square matrix
// See also:  Matrix.cpp and 02TestMatrix.cpp

class matrix {
private:
  double *entries;
  unsigned int N;
public:
  matrix (unsigned int Size=2);
  ~matrix(void) { delete [] entries; };

  unsigned int size(void) {return (N);};
  double getij (unsigned int i, unsigned int j);
  void setij (unsigned int i, unsigned int j, double x);

  void print(void);
};
```

The matrix class

Matrix.cpp

```
 8  #include "Matrix.h"

10  matrix::matrix (unsigned int Size)
    {
12    N = Size;
      entries = new double [N*N];
14  }

16  void matrix::setij (unsigned int i, unsigned int j, double x)
    {
18    if (i<N && j<N)
        entries[i*N+j]=x;
20    else
        cerr << "matrix::setij(): Index out of bounds." << std::endl;
22  }

24  double matrix::getij (unsigned int i, unsigned int j)
    {
26    if (i<N && j<N)
        return(entries[i*N+j]);
28    else
      {
30      cerr << "matrix::getij(): Index out of bounds." << std::endl;
        return(0);
32    }
    }
```

MatVec

To test this, we will implement a function for matrix-vector multiplication:

02TestMatrix.cpp

```
// File:      03Testmatrix.h (Set v=A*u)
// Author:    Niall Madden (NUI Galway) Niall.Madden@NUIGalway.ie
// Date:      22/02/2017  (Week 7 of 1617-CS319)
// What:      Test the implementation of a square matrix
```

```
void MatVec(matrix &A, vector &u, vector &v)
{
  unsigned int N;
  N = A.size();

  if ( (N != u.size()) || ( N != v.size() ) )
    std::cerr << "dimension␣mismatch␣in␣MatVec␣" << std::endl;
  else
  {
    for (unsigned int i=0; i<N; i++)
    {
      double x=0;
      for (unsigned int j=0; j<N; j++)
        x += A.getij(i,j)*u.geti(j);
      v.seti(i,x);
    }
  }
}
```

Implementing Jacobi's method

Now we can implement Jacobi's method:

See `03Jacobi.cpp` for more details

```
100 // Use Jacobi's method to solve Ax=b,
    // On entry :  x is the initial guess
102 // On exit :  x is the estimate for the solution
    void Jacobi(matrix &A, vector &b, vector &x,
104             unsigned int &count, double tol)
    {
106   unsigned int N=A.size();
      count=0;
108   if ( (N != b.size()) || (N != x.size() ) )
        std::cout << "Jacobi: error - A must be the same size as b,x\n";
110
      matrix Dinv(N), T(N);    // The diagonal and off-diagonal matrices
112
      for (unsigned int i=0; i<N; i++)
114     for (unsigned int j=0; j<N; j++)
          if (j != i) {
116         T.setij(i,j, -A.getij(i,j));
            Dinv.setij(i,j,0.0);
118       }
          else {
120         T.setij(i,j, 0.0);
            Dinv.setij(i,j, 1.0/A.getij(i,j));
122       }
```

Implementing Jacobi's method

```
124   // Now implement the algorithm:
      vector d(N), r(N);
126   do
      {
128     count++;
        MatVec(T,x,d);        // Set d=T*x
130     VecAdd(d, b, d);      // set d=b+d (so d=b+T*x)
        MatVec(Dinv, d, x); // set x = inverse(D)*(b+T*x)
132
        MatVec(A, x, r);      // set r=A*x
134     VecAdd(r, b, r, 1.0, -1.0); // set r=b-A*r

136   }   while ( r.norm() > tol);
}
```

Of course, the above code would be a lot neater, and much more readable, if we were able to write, for example, `r=A*x` instead of `MatVec(A,x,r)` ....