CS319: Scientific Computing

Projects; Vectors and Matrices (DRAFT)

Dr Niall Madden

Week 9: 12 + 14 March, 2025

Slides and examples: https://www.niallmadden.ie/2425-CS319

0. Outline

1. Projects!
2. Review of classes
3. Vectors and Matrices
4. A vector `class` (Basic version)
   - C++ "Project"
   - Adding two vectors
5. Copy Constructors
   - A new constructor
6. Operator Overloading
   - Syntax
   - Eg 1: Adding two vectors
7. The `->`, `this`, and = operators
   - The `this` pointer
   - Overloading =
8. Unary Operators
   - Two - operators
9. `friend` functions
   - Overloading the **insertion** operator
10. Preprocessor Directives
    - `#define`
    - `#include`
    - `#ifndef`
11. Solving Linear Systems
12. A matrix `class`

1. Projects!

Notes for this part are at:

https://www.niallmadden.ie/2425-CS319/2425-CS319-Projects.pdf

2. Review of classes

class

In C++, we define new class with the `class` keyword.
An instance of the class is called an "*object*".
A `class` combines by data and functions (called "methods").

Within a class, code and data may be either

- **Private**: accessible only to another part of that object, or
- **Public**: other parts of the program can access it.

Roughly,

- keep data elements `private`,
- make function elements `public`.

└ methods

2. Review of classes

The basic syntax for defining a class:

```
class class-name {
private:
    ...     // private functions and variables
public:
    ...     // public functions and variables
};
```

class-name becomes a new object type—one can now declare objects to be of type *class-name*.

This is only a declaration. Therefore,

- functions are not defined, though the prototype is given,
- variables are declared but are not initialised,
- the declaration block is delineated by { and }, and terminated with a semicolon.
- *scope resolution operator*, `::` , used in function definition.

2. Review of classes

- A **Constructor** is a public method of a class, that has the same name as the class. It's return type is not specified explicitly. It is executed whenever a new instance of that class is created.
- A **destructor** is a method that is called on an object whenever it goes out of scope. The name of the destructor is the same as the class, but preceded by a tilde.

3. Vectors and Matrices

This is a course in Scientific Computing.

Many advanced and 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;
- 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 Network Science.

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 3 equations in 3 unknowns: *find* x_1, x_2, x_3, *such that*

$$\begin{aligned} 3x_1 + 2x_2 + 4x_3 &= 19 \\ x_1 + 2x_2 + 3x_3 &= 14 \\ 5x_1 + 1x_2 + 6x_3 &= 25 \end{aligned}$$

This can be expressed as a **matrix-vector equation:**

$$\begin{pmatrix} 3 & 2 & 4 \\ 1 & 2 & 3 \\ 5 & 1 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 19 \\ 14 \\ 25 \end{pmatrix}$$

More generally, the linear system of N equations in N unknowns: *find* $x_1, x_2, \ldots, x_N$, *such that*

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

This, as a **matrix-vector equation** is:

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1N} \\ a_{21} & a_{22} & \ldots & a_{2N} \\ \vdots & & \ddots & \vdots \\ a_{N1} & a_{N2} & \ldots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}$$

So, to proceed, we need to be able to represent **vectors** and **matrices** in our codes.

4. A vector `class` (Basic version)

Our first focus will be on defining a class of vectors. This version will be quite **basic** (hence the file names). Will be developed later.

Intuitively, we know it needs the following components:

A class to represent a (mathematical) vector...

Data	Methods
vector entry values. size of vector ie number of elements.	addition Scalar multiplication. Set and get values of entries, (Transpose ... later...)

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

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

In whatever compiler you are using, you'll need to create a **project** that contains all the files. (Ask Niall for help if needed).

See VectorBasic.h for more details

```
// File: VectorBasic.h (simple version)
// Author: Niall Madden ¡Niall.Madden@UniversityOfGalway.ie¿
// Date: Week 9 of 2425-CS319
// What: Header file for vector class
// See also: VectorBasic.cpp and 01TestVector.cpp
class Vector {
private:
  double *entries;
  unsigned int N; // number of entries
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.
};
```

pointer to array for storing values.

default values.

VectorBasic.cpp

```
12 Vector::Vector(unsigned int Size)
   {
14   N = Size;
     entries = new double[Size];
16 }

18 Vector::~Vector()
   {
20   delete [] entries;
   }

   void Vector::seti(unsigned int i, double x)
24 {
     if (i<N)
26     entries[i]=x;
     else
28     std::cerr << "Vector::seti(): Index out of bounds."
                 << std::endl;
30 }
```

Constructor

destructor.

index of Entry

value of Entry.

Console Error.

VectorBasic.cpp continued

```
32 double Vector::geti(unsigned int i)
   {
34   if (i<N)
       return(entries[i]);
36   else {
       std::cerr << "Vector::geti(): Index out of bounds."
38               << std::endl;
       return(0);
40   }
   }

   void Vector::print(void)
44 {
     for (unsigned int i=0; i<N; i++)
46     std::cout << "[" << entries[i] << "]" << std::endl;
   }
```

VectorBasic.cpp continued

```
   double Vector::norm(void)
50 {
     double x=0;
52   for (unsigned int i=0; i<N; i++)
       x+=entries[i]*entries[i];
54   return (sqrt(x));
   }

   void Vector::zero(void)
58 {
     for (unsigned int i=0; i<N; i++)
60     entries[i]=0;
   }
```

If v is a vector, its norm is

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

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

See 01TestVectorBasic.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 }
```

Finished here Wednesday

Notice : we pass by reference....