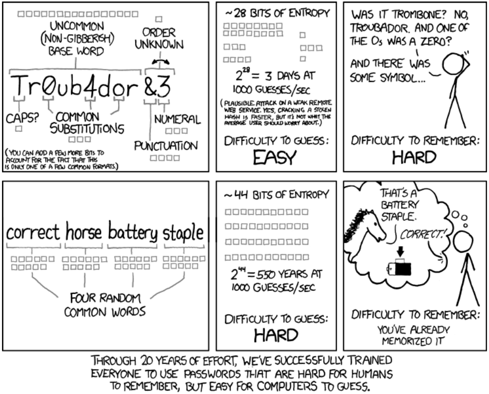
CS319: Scientific Computing (with C++)

Week 7: The Password Problem; Vectors & Matrices

9am, 23 March, and 4pm, 24 March, 2021



http://xkcd.com/936

[Originally, the Week 6 class was titled "The Password Problem"; but I didn't actually get 'round to it!]

Usual reminders...

	Mon	Tue	Wed	Thu	Fri
9 – 10		LECTURE	✗		
10 – 11		LAB			
11 – 12					
12 – 1					
1 – 2		LAB			
2 – 3					
3 – 4					
4 – 5			LECTURE		

1. Two recorded classes this week: Tuesday at 09.00, and Wednesday at 16.00.
2. **Lab times: Tuesday 10.00-10:50, and 13.00-13.50**. You should try to attend at least one of these.

3. Lab due Monday at 5pm!!

Usual reminders...

1. Part 0: Feedback on Feedback
2. Part 1: The Password Problem
   - Algorithm (high-level)
   - Implementation
3. Part 2: Vectors and Matrices
4. Part 3: A vector `class`
   - Vectors
   - C++ "Project"
   - Adding two vectors
   - Exercise
5. Part 4: Solving Linear Systems
   - Introduction
   - Jacobi's Method
   - Implementation
6. Part 5: A matrix `class`
   - MatVec
7. Part 6: Coding Jacobi's method
   - Exercise

Tuesday

Today.

CS319 – Week 7
Week 7: The Password Problem; Vectors & Matrices

Start of ...

PART 0: Feedback on Feedback

Part 0: Feedback on Feedback

- Thank-you to the 8 of you that completed the feedback form circulated by Noelle Cannon.
- On average, it took 1 minutes, 49 seconds to complete.
- Mostly very positive.
- A small number of people are "unsure" or "disagree somewhat" with the statement that "The feedback I have received is helping me to improve my learning". Which is fair! (Will do better!).
- The "live-but-recorded" lectures seem to be popular (which I was unsure of, since the quality is not very high).
- Some good suggestions for improvement, including
  - **"An example of longer code from start to finish, I find it hard to see how the code works as a whole when I only see snippets of code"**. [Response: Fair point. Although the entire code is made available separately, and the snippets have line-numbers, I will do some start-to-finish examples soon.]

CS319 – Week 7
Week 7: The Password Problem; Vectors & Matrices

END OF PART 0

CS319 – Week 7
Week 7: The Password Problem; Vectors & Matrices

Start of ...

PART 1: The Password Problem (finally!)

Part 1: The Password Problem

(Week 5)

Recall from ~~last~~ week that our aim is to take a very long list of passwords and to determine the most common.

The source of the data is the infamous **RockYou** password file, a list of over 30,000,000 unencrypted passwords stolen from RockYou in 2009, and now widely available online.

The file contains one password per line, in no particular order. The first few are

```
password
mekster11
mekster11
progr4sm
khas8950
emilio1
holiday2
caitlin1
```

(although, some lines are blank)

Also: no password has spaces)

These are in the bitbucket repro

See, Eg,

UserAccount-1e6.txt

Given a list of 30,000,000 passwords, how shall we work out which 10 (say) occur most frequently?

Idea:

1. Read the list of passwords, from the file. (into a long array).
2. $\boxed{\text{Sort}}$ the list alphabetically.
3. Calculate the frequency of each word, while removing duplicates.
4. Make a new list of the unique words, and their frequencies.
5. Sort this list by frequency.

(Will return in a few weeks, after more templates)

(By alphabetically, mean lexiographically)
We can write

```
if (string1 < string2)
{ ~ };
```

The first step is to open the file, and count the number of lines, and the length of the longest line.

if = "input file"

00SortPasswords.cpp

```
std::ifstream InFile;
std::string InFileName="UserAccount-1e4.txt";

InFile.open(InFileName.c_str());
if (InFile.fail() )
{
  std::cerr << "Error: Cannot open " << InFileName <<
    " for reading." << std::endl;
  exit(1);
}

// Need to know the number of lines, and the length of the longest one
unsigned int LineCount, LongestLine;
LineCount =  FileLength(InFile, LongestLine);
```

could change to, eg, 1e3, 1e6

~~counts~~ num

(Skipped # includes, variable defs, etc).

00SortPasswords.cpp

```
int FileLength(std::ifstream &InFile, unsigned int &LongestLine)
{
  InFile.clear(); → resets any "flags", Eg End-of-file
  InFile.seekg(std::ios::beg); // Rewind to the start of the file → week 5
  char c;
  InFile.get( c );
  unsigned int LineCount=0, ThisLineLength=0;
  LongestLine=0;
  while( ! InFile.eof() ) {
    if (c != '\n') ← End of line.
      ThisLineLength++;
    else  { ( c is '\n' ).
      LineCount++;
      if (LongestLine<ThisLineLength)
        LongestLine = ThisLineLength;
      ThisLineLength=0;
    }
    InFile.get( c ); ← → Reading one char at a time.
  }
  InFile.clear();
  InFile.seekg(std::ios::beg);    // Rewind
  return(LineCount);
}
```

Now read the file (again) and store the passwords in an array. Again, we write a single stand-alone function to do this.

Store passwords

00SortPasswords.cpp

```
void ReadPasswords(std::ifstream &InFile, std::string *Passwords,
        unsigned int &LineCount, unsigned int LongestLine)
{
  int WordsRead=0;
  char *c_string_word = new char [LongestLine+1];
  for (unsigned int Line=0; Line < LineCount; Line++)
  {
    InFile.getline(c_string_word, LongestLine+1);
    Passwords[Line] = c_string_word;
    if (Passwords[Line].length() == 0) // that was a blank line
      Line--;
    else
      WordsRead++;
  }
  LineCount = WordsRead;
  delete [] c_string_word;
}
```

The next step (main, Line 55) is to call the MergeSort() function. We then have the task of finding which word occurs most frequently. The approach is to create to new arrays:

(a) a new list of strings, called UniqueWords, where each password appears, but only once.

(b) a corresponding int array WordFreq. When we are done, if WordFreq[k]=x, then UniqueWords[k] appeared x times in the original list.

00SortPasswords.cpp

```
  std::string *UniqueWords = new std::string [LineCount+1];
  unsigned int *WordFreq  = new unsigned int [LineCount+1];
  unsigned int UniqueWordsFound;

// The first one can't already be on the list
  UniqueWords[0] = Passwords[0];
  WordFreq[0] = 1;
  UniqueWordsFound=1;
```

continued...

00SortPasswords.cpp

```
74    for (unsigned int i=0; i<LineCount; i++)
      {
76      if (Passwords[i] !=  UniqueWords[UniqueWordsFound-1])
        { // We have found a new word
78        UniqueWords[UniqueWordsFound] = Passwords[i];
          WordFreq[UniqueWordsFound] = 1;
80        UniqueWordsFound++;
        }
82      else
          WordFreq[UniqueWordsFound-1]++;
84    }
```

Explanation:

74: for loop: iterating over every password.

76: IF the current password is not the same as previous one: it is "new"

78: add this to the list of unique pwds.

79: Set freq of this pwd to 1.

83: Otherwise (ie, not a new word) increment the freq value.

Our next step is to create a list to the 10 most frequently used. This information will be stored in two arrays:

```
string Top10[10];
int Top10Freq[10];
```

We will keep this list ordered. Then iterate through the `UniqueWords` list. If we find a word that occurs more often than the (current) 10th most common, we insert it into the list:

00SortPasswords.cpp

```
  // Insert the 1st into list and set rest to 0
  Top10[0]=UniqueWords[0];
  Top10Freq[0]=WordFreq[0];
  for (int i=1; i<10; i++)
  {
    Top10[i]="";
    Top10Freq[i]=0;
  }
// See if this is at least as freq as the 10th most
  for (unsigned int i=1; i<UniqueWordsFound; i++)
    if (WordFreq[i] > Top10Freq[9])
      Insert(Top10, Top10Freq, UniqueWords[i], WordFreq[i]);
```

Check if the current word should be in Top 10 list.

To finish, we'll see how the Insert function works:

00SortPasswords.cpp

```
// Insert NewString into the list Top10, ordered by
// NewCount in Top10Freq, bumping anything if needed
void Insert(std::string *Top10, unsigned int *Top10Freq,
            std::string NewString, unsigned int NewCount)
{
  if (NewCount <= Top10Freq[9])
    std::cerr << "Error: new entry would not make top 10" << std::endl;
  else
  {
    Top10[9]=NewString;
    Top10Freq[9]=NewCount;
    for (int i=8; i>=0; i--)
    {
      if (Top10Freq[i]<NewCount)
      {
        Top10[i+1] = Top10[i];
        Top10Freq[i+1] = Top10Freq[i];
        Top10[i]=NewString;
        Top10Freq[i]=NewCount;
      }
    }
  }
}
```

CS319 – Week 7
Week 7: The Password Problem; Vectors & Matrices

END OF PART 1

CS319 – Week 7
Week 7: The Password Problem; Vectors & Matrices

Start of ...

PART 2: Vectors and Matrices

Motivation

Part 2: Vectors and Matrices

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

- optimisation; (Lab 3?)
- searching and list processing. (Password problem).

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; Eg, how to define + for our own class.
- `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 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 3 equations in 4 unknowns: *find* x_1, x_2, x_3, *such that*

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

(Check : solution is $x_1 = 1$, $x_2 = 2$, $x_3 = 3$)

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

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

$$A \quad x = b.$$

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

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

A is a square $N \times N$ matrix.

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

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

x & b are vectors with N entries.

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

So we will solve

$$Ax = b.$$

But not as $x = A^{-1}b$.

CS319 – Week 7
Week 7: The Password Problem; Vectors & Matrices

END OF PART 2

CS319 – Week 7
Week 7: The Password Problem; Vectors & Matrices

Start of ...

PART 3: A vector class

Recorded Wed @ 4p

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

Recall that a vector is a "list" of n numbers, eg

$$v = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad (2\text{-vector})$$

$$u = \begin{pmatrix} -3 \\ 5 \\ 7 \end{pmatrix} \quad (3\text{-vector})$$

$$z = \begin{pmatrix} -3.14159 \\ 2.735 \\ 2.111 \\ 0.999 \end{pmatrix} \quad (4\text{-vector})$$

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_N \end{pmatrix}$$

N-vector where all the b's are numbers (floats)

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

(a) Size of the vector (= order, dimension).

(b) Entries — the values stored in the vector

Operations:

- Addition: $z = x + y$ (z, x, y all vectors
- Scalar Multiplication: $z = \alpha x$ (α is some number)
- norm of vector

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.

[Make sure only one file has a main function].

See Vector.h for more details

```
// File:  Vector.h (Version W07.1)
// Author:  Niall Madden (NUI Galway) Niall.Madden@NUIGalway.ie
// Date:  Week 7 of 2021-CS319
// What:  Header file for vector class
// See also:  Vector.cpp and 01TestVector.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.
};
```

array where we store elements

number of entries

constructor

included code

return the i^{th} entry.

Set i^{th} entry.

Vector.cpp

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

Constructor

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

destructor

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

no negitive numbers

Vector.cpp continued

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

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

Vector.cpp continued

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

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

Compute

$$\sqrt{x_0^2 + x_1^2 + \dots + x_{N-1}^2}$$

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

See 01TestVector.cpp for more details

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

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

-- check a & b are of the same size

Eg if $a = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, $b = \begin{pmatrix} -1 \\ 4 \\ 0 \end{pmatrix}$, $\alpha = -1$, $\beta = 2$

$$c = (-1)\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + (2)\begin{pmatrix} -1 \\ 4 \\ 0 \end{pmatrix} = \begin{pmatrix} -3 \\ 6 \\ -3 \end{pmatrix}$$

Exercise (7.1)

The method `Vector::norm()` *computes the Euclidian norm of a vector:*

$$\|v\|_2 = \Big(\sum_{i=1}^{n}(v_i)^2\Big)^{1/2}. = \sqrt{v_1^2+v_2^2+\cdots+v_N^2}$$

This is a special case of the so-called p-norm:

$$\|v\|_p = \Big(\sum_{i=1}^{n}|v_i|^p\Big)^{1/p}.$$

where $p \geq 1$. Rewrite the `Vector::norm()` *function so that it takes a double p as an optional second argument, and computes the p-norm of the vector. If p is not provided, it should default to $p = 2$. In addition, if $p = 0$ is given, it should compute the max-norm:*

$$\|v\|_\infty = \max_{i=1}^{n}|v_i|.$$

CS319 – Week 7
Week 7: The Password Problem; Vectors & Matrices

END OF PART Part 3

CS319 – Week 7
Week 7: The Password Problem; Vectors & Matrices

Start of ...

PART 4: Solving Linear Systems

We now move towards learning about **matrices**. When implementing the class, we will learn about

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

important C++.

One of the most ubiquitous problems in scientific computing 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* $\boxed{x_1, x_2, \ldots, x_N}$ *such that*

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

Write as

$$Ax = b.$$

This means, e.g., $x = A^{-1}b$.

There are several classic approaches:

1. Gaussian Elimination;
2. Related: LU- and Cholesky factorisation;

Projects, maybe

3. Stationary Iterative schemes such as **Jacobi's method**, **Gauss-Seidel** and Successive Over Relaxation (SOR);

Project

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 annotated slides.

See video or annotated slides

Idea:

Suppose we want to solve

$$3x_1 + 2x_2 + 4x_3 = 19$$
$$x_1 + 2x_2 + 3x_3 = 14$$
$$5x_1 + x_2 + 6x_3 = 25$$

In Matrix-Vector form, this is:

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

$$3x_1 + 2x_2 + 4x_3 = 19$$
$$x_1 + 2x_2 + 3x_3 = 14$$
$$5x_1 + x_2 + 6x_3 = 25$$

See video or annotated slides

Suppose I know x_2 & x_3, and would like to compute x_1. Then, from the 1st Eqn

$$x_1 = \frac{1}{3}(19 - 2x_2 - 4x_3)$$

If I know x_1 & x_3, I can get that

$$x_2 = \frac{1}{2}(14 - x_1 - 3x_3).$$

Similarly

$$x_3 = \frac{1}{6}(25 - 5x_1 - x_2).$$

But we don't know any pair of x_1, x_2, x_3. However, if we had a guess for 2 of these, we could use it to get a guess for the other 2. **See video or annotated slides**

Lets write our initial guess as $x_1^{(0)}, x_2^{(0)}, x_3^{(0)}$, and the set

$$x_1^{(1)} = \frac{1}{3}\left(19 - 2x_2^{(0)} - 4x_3^{(0)}\right)$$

$$x_2^{(1)} = \frac{1}{2}\left(14 - x_1^{(0)} - 3x_3^{(0)}\right).$$

$$x_3^{(1)} = \frac{1}{6}\left(25 - 5x_1^{(0)} - x_2^{(0)}\right).$$

It turns out $x_1^{(1)}, x_2^{(1)}, x_3^{(1)}$ are better estimates than $x_1^{(0)}, x_2^{(0)}, x_3^{(0)}$.

See video or annotated slides

The we repeat the process

set $x_1^{(2)} = \frac{1}{3}\left(19 - 2x_2^{(1)} - 4x_3^{(1)}\right)$

$x_2^{(2)} = \frac{1}{2}\left(14 - x_1^{(1)} - 3x_3^{(1)}\right).$

$x_3^{(2)} = \frac{1}{6}\left(25 - 5x_1^{(1)} - x_2^{(1)}\right).$

again improving the estimate.

Next (week) we'll see how to write this in Matrix form. | Questions??