Errors and Flow Control

Dr. Johnson

School of Mathematics

Semester 1 2012
LAST WEEK...

- **Topics:**
  - Computers and Programs;
  - Syntax of C++;
  - Data and Variables;
  - Input and Output.

- **Aims - week 1:**
  - Understand the idea of programming a computer;
  - Write a simple program to input and output data.
**Part 1:**
- Precision and Errors
- Flow control - loop, if conditions, functions

**Part 2:**
- Program Structure and Functions
- Pointers - dynamic allocation, pass by reference

**Aims - week 2:**
- Understand precision and errors within a program.
- Use functions, loops and if statement to control and structure a program.
- Understand the concept of a pointer.
In order to quantify errors in our solutions we need to define a measure for the error.
In order to quantify errors in our solutions we need to define a measure for the error.

Absolute error -
If \( \phi^* \) is an approximation to a quantity \( \phi \) then the absolute error is defined by

\[ |\phi - \phi^*| \]
In order to quantify errors in our solutions we need to define a measure for the error.

**Absolute error** -
If $\phi^*$ is an approximation to a quantity $\phi$ then the absolute error is defined by

$$|\phi - \phi^*|$$

**Relative error** -
The Relative error is defined by

$$\frac{|\phi - \phi^*|}{|\phi|}, \quad \phi \neq 0$$
An interesting part of solving any numerical problem is that we don’t know what the solution is... This begs the question: How to measure errors when we don’t know \( \phi \)?
An interesting part of solving any numerical problem is that we don’t know what the solution is...

This begs the question:

**How to measure errors when we don’t know \( \phi \)?**

We may show that our approximation \( \phi^* \) converges towards what we believe to be \( \phi \).
An interesting part of solving any numerical problem is that we don’t know what the solution is... This begs the question:

**How to measure errors when we don’t know \( \phi \)?**

We may show that our approximation \( \phi^* \) converges towards what we believe to be \( \phi \)

- The idea of limits and convergence are important concepts in this course
- When presenting numerical work we must try to convince the reader that the solutions are correct!
When using computers to solve a numerical problem, errors appear at every step.

Most numbers cannot be stored exactly to computer precision—floating-point numbers.

Some could never be stored exactly—\( \pi \), \( \sqrt{2} \) etc.

Double precision numbers are stored in 64 bits (0s and 1s), and have around 15-16 decimal digits.
When using computers to solve a numerical problem, errors appear at every step.

Most numbers cannot be stored exactly to computer precision—floating-point numbers.

Some could never be stored exactly—\( \pi \), \( \sqrt{2} \) etc.

Double precision numbers are stored in 64 bits (0s and 1s), and have around 15-16 decimal digits.

Errors may be magnified under certain sequences of calculations.

We can control this by rearranging equations or specifying the order of operations.
**Example - Errors increasing**

**Multiplying by a large number**

Using 3s.f. evaluate \( f(x) = x^3 - 6.1x^2 + 3.2x + 1.5 \) at \( x = 4.71 \)

- **Exact** -
  \[
  f(4.71) = 104.487111 - 135.32301 + 15.072 + 1.5 = -14.263899;
  \]

- **Approx** -
  \[
  f(4.71) = ((105. - 135.) + 15.1) + 1.5 = -13.4
  \]

**NB.** here we have rounded to three s.f. after each multiplication
EXAMPLE - ERRORS INCREASING

MULTIPLYING BY A LARGE NUMBER

Using 3s.f. evaluate \( f(x) = x^3 - 6.1x^2 + 3.2x + 1.5 \) at \( x = 4.71 \)

- Exact -
  \[ f(4.71) = 104.487111 - 135.32301 + 15.072 + 1.5 = -14.263899; \]
- Approx - \( f(4.71) = ((105. - 135.) + 15.1) + 1.5 = -13.4 \)
  NB. here we have rounded to three s.f. after each multiplication
- Nested - \( f(4.71) = ((4.71 - 6.1) \cdot 4.71 + 3.2) \cdot 4.71 + 1.5 = -14.3 \)
  NB. not only is this more accurate but we also perform less operations
Modelling Errors

- Replacing the full Navier-Stokes equations with Euler equations.
- Neglect of viscous terms (assuming they are small).
- We can then only solve an approximation of the full problem.
Modelling Errors

- Replacing the full Navier-Stokes equations with Euler equations.
- Neglect of viscous terms (assuming they are small).
- We can then only solve an approximation of the full problem.
- Small terms are often neglected to make the problem easier.
- In certain regions these small terms may become big! See singular perturbations etc.
These are all too familiar!
The computer is only doing what you ask it to do.
Even NASA has made blunders.
Programming Errors – Bugs

- These are all too familiar!
- The computer is only doing what you ask it to do.
- Even NASA has made blunders.
- Never believe what comes out of a computer.
- You need to be able to prove that solutions are the correct ones.
There are many situations where your code will show all the classic signs (stability, convergence, etc) of being correct, but can be **completely** wrong!!!

Such errors may be:
- changing the value of a parameter by accident;
- entering the scheme incorrectly;
There are many situations where your code will show all the classic signs (stability, convergence, etc) of being correct, but can be **completely** wrong!!!

Such errors may be:
- changing the value of a parameter by accident;
- entering the scheme incorrectly;

Avoid this by checking against known solutions . . .

or other solvers
More Subtle Errors

Suppose in your code you have the two variables

$$\phi = O(10^{-8}), \text{ and } \phi^* = O(10^{-8})$$

then evaluate their difference under some conditions

```c
diff = abs(phi-phistar);
tol= 1.e-6;
if(diff < tol) break;
```
Suppose in your code you have the two variables

\[ \phi = O(10^{-8}) \], and \[ \phi^* = O(10^{-8}) \]

then evaluate their difference under some conditions

\[
\text{diff} = \text{abs} (\phi - \phi^*)
\]

\[
\text{tol} = 1.0e-6;
\]

\[
\text{if (diff < tol) break;}
\]

The condition is always satisfied even though relative error is \( O(1) \).
We can use `if`, `else if`, and `else` to control flow through the program.

```cpp
int i;
cout << " Enter a number " << endl;
cin >> i;
if(i<0)cout << " i is negative" << endl;
else if(i==0)cout << " i is zero" << endl;
else cout << " i is positive" << endl;
```
To execute more than one command on an `if` condition use blocks

```java
if(condition){
    // lots of commands in here
}
else {
    // and in here too.
}
```
To execute more than one command on an if condition use blocks

```java
if(condition){
    // lots of commands in here
}
else {
    // and in here too.
}
```

Try question 2.4 on examples sheet 2.
FOR LOOPS

- The general form for a loop is

```c
for(initialisation; condition; increment)
    statement;
```

We can loop over multiple commands using a block
```c
for(int i=0;i<10;i++)
    temp = i*10;
    cout << " value " << temp << endl;
```

Try changing the condition (e.g. "≤" or ">") and the increment to something else (e.g. "i = i + 2" or "i = i - 5")
What happens?
The general form for a loop is

```plaintext
for(initialisation; condition; increment) statement;
```

We can loop over multiple commands using a block

```plaintext
for(int i=0; i<10; i++){
    temp = i*10;
    cout << " value " << temp << endl;
}
```
FOR LOOPS

The general form for a loop is

```cpp
for(initialisation; condition; increment)
statement;
```

We can loop over multiple commands using a block

```cpp
for(int i=0; i<10; i++){
    temp = i*10;
    cout << " value " << temp << endl;
}
```

Try changing the condition (e.g. “≤” or “>”) and the increment to something else (e.g. “i = i + 2” or “i = i - 5”). What happens?
**Exiting a loop**

- The command `break` can be used to exit a loop.

```c
for(int loop=0;loop<iter_max;loop++)
{
    solve_for_U(u,y,U);
    if(residual(x,y,U)<tolerance)break;
}
```
Evaluate the expression:

\[ \Gamma(n) = \left(1 + \frac{1}{n}\right)^n \]

for values of \(0 < n \leq 10000\) and print them to screen in a table.
Evaluate the expression:

\[ \Gamma(n) = \left(1 + \frac{1}{n}\right)^n \]

for values of 0 < n ≤ 10000 and print them to screen in a table.

Use a \textit{break} command to stop the \textit{for} loop when:

\[ \Gamma(n) - \Gamma(n - 1) \leq 1 \times 10^{-8} \]
The while loop

- Another alternative loop iteration is the **while** loop.
- Their functional form is:

```c
while (condition) statement;
```

- The statement is executed until the condition is true.
- The condition is evaluated **before** the statement.

```c
int i = 0;
while (i < 100)
{
    i++;
    std::cout << " i= " << i << std::endl;
}
```
Another alternative loop iteration is the do-while loop. Their functional form is:

```
do {statement;} while(condition);
```

The statement is executed until the condition is true. The condition is evaluated after the statement.

```
do {
solve_for_U(u,y,U);
}
while(residual(x,y,U)<tolerance);
```
Part 1:
- Precision and Errors
- Flow control - loop, if conditions

Part 2
- Program Structure and Functions
- Pointers - dynamic allocation, pass by reference

Aims - week 2:
- Understand precision and errors within a program.
- Use functions, loops and if statement to control a program.
- Understand the concept of a pointer.