0. Annotated slides from Friday

CS319: Scientific Computing

Arrays, Memory Allocation (and quadrature again)

Dr Niall Madden

Week 6: 11th and 13th of February, 2026

Slides and examples:
https://www.niallmadden.ie/2526-CS319

0. The array of topics this week:

1. Arrays
   - Basics
   - Initialization
   - Base Address
2. Timing Code
   - ctime
   - Example
   - chrono

Wed
nes
day

3. Pointers
   - Pointer arithmetic
   - Warning!
4. Dynamic Memory Allocation
   - `new`
   - `delete`
5. Example: Quadrature 1
6. Quadrature 2: Simpson's Rule
7. Analysis
8. Juputer: lists and NumPpy

Slides and examples: https://www.niallmadden.ie/2526-CS319

4. Pointers

To properly understand how to use arrays, we need to study **Pointers**.

- We already learned that if, say, `x` is a variable, then `&x` is its memory address.
- A **pointer** is a special type of variable that can store memory addresses. We use the `*` symbol before the variable name in the declaration.
- For example, if we declare

```
int i;
int *p;
```

then we can set `p=&i`.

int* p, q, j

⟺ int *p, q, j

I usually think of "int *p" as "int* p".

4. Pointers

02Pointers.cpp

```
int a=-3, b=12;
int *where;

std::cout << "The variable ’a’ stores " << a << std::endl;
std::cout << "The variable ’b’ stores " << b << std::endl;
std::cout << "’a’ is stored at address " << &a << std::endl;
std::cout << "’b’ is stored at address " << &b << std::endl;

where = &a;
std::cout << "The variable ’where’ stores "
          << (void *) where << std::endl;
std::cout << "... and that in turn stores " <<
  *where << ’\n’;
```

Here * is a "derefercer". Technically a mem address is a "reference".

If p stores a reference then *p is the value stored at that refence.

~~One can actually do calculations on memory addresses. This is called **pointer arithmetic**. One can't (for example) add two addresses, or compute their product, but you can, for example, increment them.~~

```
The variable 'a' stores -3
The variable 'b' stores 12

'a' is stored at address 0x7ffc6c0c4894
'b' is stored at address 0x7ffc6c0c4890

The variable 'where' stores 0x7ffc6c0c4894
... and that in turn stores -3
```

where = &a.

*where.

One can actually do calculations on memory addresses. This is called **pointer arithmetic**. One can't (for example) add two addresses, or compute their product, but you can, for example, increment them.

Note: we can think of

& (get address of a variable)

and

* (get value stored at an address).

as inverses of each other

E.g.,

```
a ==  *(&a)     but    a != &(*a)
```

also (I think) p != &(*p)

03PointerArithmetic.cpp

```
int vals[3];
vals[0]=10;  vals[1]=8;  vals[2]=-4;

int *p;
p = vals;

for (int i=0; i<3; i++)
{
  std::cout << "p=" << p << ", *p=" << *p << "\n";
  p++;
}
```

This would be the same as

```
for (int *p=vals; p<=&vals[2]; p++)
   std::cout << "p=" << p << ", *p=" << *p << "\n
```

Warning!

Being able to manipulate memory addresses is one of the reasons C++ is considered a very **powerful** language. It is possible to preform (low-level) operations in C++ that are impossible in, say, Python.

(For many years, the US Cybersecurity and Infrastructure Security Agency, CISA, has argued that C and C++ are inherently unsafe).

But it is also possible to write programmes that will crash, or even crash your computer, since memory addresses are not well protected.

5. Dynamic Memory Allocation

In all examples we've had so far, we've specified the size of an array at the time it is defined.

In many practical cases, we don't have that information. For example, we might need to read data from a file, but not know the file size in advance.

It would be useful if, on the fly, we could set the size of an array.

Furthermore, for efficiency, we may want to free up memory allocated.

To add this functionality, we will use two new (to us) C++ operators for dynamic memory allocation and deallocation:

- `new` and
- `delete`.

(There are also functions `malloc()`, `calloc()` and `free()` inherited from C, but we won't use them).

5. Dynamic Memory Allocation new

The `new` operator is used in C++ to allocate memory. The basic form is

var = new *type*; → eg x = new int;

Same as int x;

where *type* is the specifier of the object for which you want to allocate memory and *var* is a pointer to that type.

If insufficient memory is available then `new` will return a *NULL* pointer or generate an exception.

To dynamically allocate an array:

- First declare a pointer of the right type:

  ```
  int *data;
  ```

- Then use `new`

  ```
  data = new int [MAXSIZE];
  ```

5. Dynamic Memory Allocation `delete`

When it is no longer needed, the operator `delete` releases the memory allocated to an object.

To "delete" an array we use a slightly different syntax:

```
   delete [] array;
```

where *array* is a pointer to an array allocated with `new`.

6. Example: Quadrature 1

Previously, we introduced the idea of **numerical integration** or **quadrature**.

We computed estimates for $\int_a^b f(x)dx$ by applying the Trapezium Rule:

- Choose the number of intervals N, and set $h = (b-a)/N$.
- Define the quadrature points $x_0 = a$, $x_1 = a + h$, $\dots x_N = b$. In general, $x_i = a + ih$.
- Set $y_i = f(x_i)$ for $i = 0, 1, \dots, N$.
- Compute $\displaystyle\int_a^b f(x)dx \approx Q_1(f) := h\big(\frac{1}{2}y_0 + \sum_{i=1}^{N-1} y_i + \frac{1}{2}y_N\big).$

[Take notes for the next few slides]

6. Example: Quadrature 1

We'll have a new function

double Quad1(double *x, double *y, unsigned int N);

The function Quad1

- take 2 pointes as inputs, x & y, along with a int, N
- x, y will store the base addresses for the arrays of points & values
- N is the number of intervals.

6. Example: Quadrature 1

04TrapeziumRule.cpp

```
#include <iostream>
#include <cmath>   // For exp()
#include <iomanip>

double f(double x) {  return(exp(x)); } // definition
double ans_true = exp(1.0)-1.0; // true value of integral

double Quad1(double *x, double *y, unsigned int N);
```

usual #include lines. Also use C-math library.

$$\exp(x) = e^x$$

Recall $$\int_0^1 f(x)\,dx = e^x\Big|_0^1 = e^1 - e^0 = e^1 - 1$$

6. Example: Quadrature 1

Next we skip to the function code...

04TrapeziumRule.cpp

```
double Quad1(double *x, double *y, unsigned int N)
{
  double h = (x[N]-x[0])/double(N);
  double Q = 0.5*(y[0] + y[N]);
  for (unsigned int i=1; i<N; i++)
    Q += y[i];
  Q *= h;
  return(Q);
}
```

Source of confusion: * is used in two very different contexts here.

Note that, if x is a pointer,
$x[0]$, $x[1]$ etc are valid
($x[i] \Leftrightarrow *(x+i)$))

6. Example: Quadrature 1

Back to the main function: declare the pointers, input N, and allocate memory.

04TrapeziumRule.cpp

```
int main(void )
{
  unsigned int N;
  double a=0.0, b=1.0; // limits of integration
  double *x; // quadrature points
  double *y; // quadrature values

  std::cout << "Enter the number of intervals: ";
  std::cin >> N; // not doing input checking

  x = new double[N+1];
  y = new double[N+1];
```

This is **not** the same as if we had set , e.g.,
x = int[N];

6. Example: Quadrature 1

Initialise the arrays, compute the estimates, and output the error.

04TrapeziumRule.cpp

```
  double h = (b-a)/double(N);
  for (unsigned int i=0; i<=N; i++)
  {
    x[i] = a+i*h;
    y[i] = f(x[i]);
  }
  double Est1 = Quad1(x, y, N);
  double error = fabs(ans_true - Est1);
  std::cout << "N=" << N << ", Trap Rule="
            << std::setprecision(6) << Est1
            << ", error=" <<  std::scientific
            << error << std::endl;
```

6. Example: Quadrature 1

Finish by de-allocating memory (optional, in this instance).

04TrapeziumRule.cpp

```
  delete [] x;
  delete [] y;
  return(0);
}
```

6. Example: Quadrature 1

Although this was presented as an application of using arrays in C++, some questions arise...

1. What value of N should we pick the ensure the error is less than, say, 10^{-6}?
2. How could we predict that value if we didn't know the true solution?
3. What is the smallest error that can be achieved in practice? Why?
4. How does the time required depend on N? What would happen if we tried computing in two or more dimensions?
5. **Are there any better methods? (And what does "better" mean?)**

6. Example: Quadrature 1

(FInished here Fri)

Some answers to some of those questions

For 1+2:
Ideas for estimating the error include seeing how much the answer changes... That is, compute the approximations for both N and 2*N and compare. (Error estimation).

3: min error is related to machine epsilon

4: Time is proportional to N. That is
T=C*N. And we can estimate C.