Case study:

Implementing Numerical Algorithms as C++ functions

- Case Study Numerical Integration
- Romberg Algorithm
  - The Romberg algorithm produces a triangular array of numbers, all of which are numerical estimates of the definite integral

$$\int_a^b f(x)dx$$

For example $\pi \approx \int_0^1 \frac{4}{1+x^2}dx$

Rombergs Method and Triangular Array for Pi

- Estimate π $with$ Romberg n = 5

where n is the number of intervals

$$R(0,0) = \frac{1}{2}(b-a)(f(a)+f(b))$$

$$R(n,0) = \frac{1}{2}R(n-1,0) + h_n \sum_{k=1}^{2^{n-1}} f(a+(2k-1)h_n)$$

$$R(n,m) = R(n,m-1) + \frac{1}{4^m-1}(R(n,m-1) - R(n-1,m-1))$$

3.00000 00000 000				
3.09999 99046 326	3.13333 32061 768			
3.13117 64717 102	3.14156 86607 361	3.14211 77387 238		
3.13898 84948 730	3.14159 25025 940	3.14159 41715 240	3.14158 58268 738	
3.14094 16198 730	3.14159 27410 126	3.14159 27410 126	3.14159 27410 126	3.14159 27410 126

Pseudo code

procedure *Romberg*$(f, a, b, n, (r_{ij}))$
integer i, j, k, n; **real** a, b, h, sum; **real array** $(r_{ij})_{0:n \times 0:n}$
external function f
$h \leftarrow b - a$
$r_{00} \leftarrow (h/2)[f(a) + f(b)]$
for $i = 1$ **to** n **do**
    $h \leftarrow h/2$
    $sum \leftarrow 0$
    for $k = 1$ **to** $2^i - 1$ **step** 2 **do**
        $sum \leftarrow sum + f(a + kh)$
    end for
    $r_{i0} \leftarrow \frac{1}{2} r_{i-1,0} + (sum)h$
    for $j = 1$ **to** i **do**
        $r_{ij} \leftarrow r_{i,j-1} + (r_{i,j-1} - r_{i-1,j-1})/(4^j - 1)$
    end for
end for
end procedure *Romberg*

Cheney and Kincaid, Numerical Mathematics and Computing. Sixth edition, 2008

http://en.wikipedia.org/wiki/Romberg's_method

C++ implementation: user-defined functions for estimation of Pi using Romberg's method romberg_pi.cpp

```
#include <iostream>
#include <cmath>
using namespace std;
double f(double x)
{
  double fx;
  fx = 4.0f/(1 + x*x);
  return fx;
}

void romberg(double **r,  double a, double b, int n)
{
        int i,j,k;
         double sum, h;
         h=b-a;
         r[0][0]= h/2.0 * ( f(a) + f(b) );
        cout << r[0][0] <<endl;
        cout.precision(16);
        cout.setf(ios::fixed,ios::floatfield);
```

```
        for (i=1; i<n;i++)
        {
         h = h/2.0;
         sum=0.0;
        for (k=1; k <=  pow(2.0,i)  ;  k+=2)
         {
           sum = sum + f( a+ k*h);
         }
         r[i][0]= ( 0.5f * r[ i-1 ][0]) + sum*h;
         cout << r[i][0];
         for (j=1; j < i; j++)
         {
          r[i][j] =  r[i][j-1] + (r[i][j-1] - r[i-1][j-1])/(pow(4.0,j)-1);
          cout <<" "<<r[i][j] ;
          r[n-1][n-1]=  r[i][j] ;
         }
        cout << endl;
        }
         cout <<"Best estimate = "<< r[n-1][n-1] << endl;
}
```

C++ implementation: Main function for estimation of Pi using Romberg’s method romberg_pi.cpp

```
int main (int argc , char **argv)
{
	cout << "Enter the Number of Intervals N: \n";
	int N=5, n=5;
	cin >>  N ;
	if ( N<=0) n=5;
	else n=N;
	int i;
	double a, b,  sum, **r;
	b=1.0;
	a=0.0;
	r=new double *[n];
	for (i=0;i<n; i++) r[i]=new double  [n];
	romberg(r,a,b,n);
	return 0;
}
```

Exercises

- Use Romberg function template from the pi code to estimate (use romberg_gaussian.cpp as template)

  the Gaussian function that is integrated from 0 to 1, i.e. the error function

  – erf(1) ≈ 0.842700792949715

  –

$$erf(1) \approx \frac{2}{\sqrt{\pi}} \int_0^1 e^{-t^2} dt$$