Pat O'Sullivan

Mh4718 Week 6

Week 6

If we use a program to calculate a Riemann sum then there is a trade off between the mathematical theory and the problem of round off error. In particular cancellation error puts a limit on how accurate a Riemann Sum can be.

If f is a function integrable over the interval $[a, b]$ a Riemann Sum is determined by first deciding on a partition of the interval. Very often we choose an equal subdivision of the interval into n sub-intervals of equal size. If we decide on n equal sized sub-intervals then each sub-interval will have length $h = \frac{b-a}{n}$. We thus get a Riemann sum:

$$f(a)h + f(a+h)h + f(a+2h)h + \ldots f(a+(n-1)h)h$$

which is an estimate for $\int_a^b f(x)dx$ We can prove that the finer the partition (the smaller h is here) then the closer the Riemann Sum will be to the exact value of the interval. However, if we make h too small in a computer program then there is a danger that we cause greater rounding error in the adding up process. The following program illustrates this:

```
#include <iostream>
#include<cmath>
#include<fstream>
using namespace std;
float f(float x)
{
    return x;
}
void main()
{
ofstream fout("numerics.txt");
```

```
    float sum;
    float a=0,b=0.125;

    float h;

    for(int n = 10;n<pow(2.0,27);n*=2)
    {
        sum=0;
        h=(b-a)/n;
        for(int i =0;i<n;i++)
        {
            sum+=f(a+i*h)*h;
        }
fout.precision(20);
fout<<n<<" \t "<<fabs(sum-0.0078125)<<endl;

     }
}
```

You will see here that the Riemann Sum becomes closer to the exact value of the integral as n increases up to a certain value but then as n increases further the Riemann Sum moves away from the integral value except for some exceptional values. If you changed all the float types to double types then you will notice that the Riemann Sums become consistently more accurate as n increases because the extra space available for the mantissa will reduces the influenc of roundoff error.

0.1 Estimating Functions

The only mathematical operations that a computer can carry out are addition, multiplication, subtraction and division. It follows that the only type of functions that a computer can directly evaluate are polynomials. All other types of function values have to be estimated by using suitable polynomials.
The "natural" process for evaluating polynomials actually uses unnecessarily many multiplication operations.

Example 0.1

How many operations are needed to evaluate this for a particular value of x using the "natural" way to evaluate the polynomial?
Consider the polynomial

$$p(x) = 2x^4 + 7x^3 + 6x^2 + 8x + 12.$$

To make the operations count easier, write it as

```
p(x) = 2*x*x*x*x + 7*x*x*x + 6*x*x + 8*x + 12
```

and now counting operation signs gives 10 multiplications and 4 additions.

Horner's method gives us a means of evaluating polynomials using fewer multiplications and so possibly fewere round-off errors:

0.1.1 Horner's method

Thm: (Horner's method)
Let $P(x) = a_n x^n + a_{n-1}x^{n-1} + \ + a_1 x + a_0$
Let $b_n = a_n$ and $b_k = a_k + b_{k+1}x_0 \ \ k = n-1, \ldots 0$
If

$$Q(x) = b_n x^{n-1} + b_{n-1}x^{n-2} + \cdots + b_2 x + b_1$$

then

$$P(x) = Q(x)(x - x0) + b_0$$

Pf:

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots a_1 x + a_0$$

$= b_n x^n + (b_{n-1} - b_n x_0)x^{n-1} + (b_{n-2} - b_{n-1}x_0)x^{n-2} + \cdots + (b_1 - b_2 x_0)x + b_0 - b_1 x_0$

$= b_n x^n + b_{n-1}x^{n-1} - b_n x_0 x^{n-1} + b_{n-2}x^{n-2} - b_{n-1}x_0 x^{n-2} + \cdots + b_1 x - b_2 x_0 x + b_0 - b_1 x_0$

$= b_n x^n + b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1 x + b_0 - b_n x_0 x^{n-1} - b_{n-1}x_0 x^{n-2} - \cdots - - b_2 x_0 x - b_1 x_0$

$$= Q(x)(x - x_0) + b_0$$

QED
Corrolary: $P(x_0) = b_0$.

Proof: $P(x) = Q(x)(x - x0) + b0 \Rightarrow P(x_0) = Q(x_0)(x_0 - x_0) + b_0 \Rightarrow P(x_0) = b_0$.
QED

Example 0.2

Now determine the number of multiplications and additions in using Horner's method to evaluate, say $p(2)$ for the above polynomial:

2	2	7	6	8	12
+		$+2\times 2$	$+2\times 11$	$+2\times 28$	$+2\times 64$
	2	11	28	64	140

So we see that the value $p(2) = 140$ and we did 4 multiplications and 4 additions.

We can also see from Horner's method that

$$p(x) = (2x^3 + 11x^2 + 28x + 64)(x - 2) + 140.$$

Code for Horner's method (assuming a degree 10 polynomial):

```
double a[11]={.............};
double b[11];
b[10]=a[10];

for(int i=9;i>=0;i--)
{
b[i]=a[i]+0.8*b[i+1];
}
cout<<b[0]<<endl;
}
```

The importance of Horner's method for programming is that it uses fewer operations to calculate the value of a polynomial and so is liable to have less round off error.