Lab4 The application of Romberg Algorithm

Sheng Zhang
2016310200210
2nd class of information and computing science

1 Lab purpose

Given function $y(x) = lsin(tx)$, please calculte approximation value of $\int_a^b lsin(tx)$ by Romberg Algorithm.

2 Lab Requirements

2.1 Definition of Function interface

```
double Integral(double a, double b, double (*f)(double x, double y,
    double z), double TOL, double l, double t)
```

In the definition of this part, a and b are upper and lower bounds of definite integral respectively, and f is an integral function. x is the integral variable, y and z are special parameters which are correspond to l and t of $\int_a^b lsin(tx)$ respectively. TOL is the precision of definite integral. l and t are the input value. Please notice $y(x) = lsin(tx)$ is in centimeters, but the length of output is supposed to be in meters.

The original code is as follow:

```
#include<stdio.h>
#include<math.h>

double f0( double x, double l, double t )
{
  return sqrt(1.0+l*l*t*t*cos(t*x)*cos(t*x));
}

double Integral(double a, double b, double (*f)(double x, double y,
    double z), double TOL, double l, double t);

int main()
{
  double a=0.0, b, TOL=0.005, l, t;
  while (scanf("%lf %lf %lf", &l, &b, &t) != EOF)
    printf("%.2f\n", Integral(a, b, f0, TOL, l, t));
  return 0;
}
```

2.2 Data requirements

Input: 2 100 1
Output: 1.68

3 Method

3.1 Detail of Romberg algorithm

Romberg algorithm is essentially an extrapolation method, which can increase the error order, so that the error term converges faster. More conveniently, we can implement Romberg algorithm easily utilizing iteration.

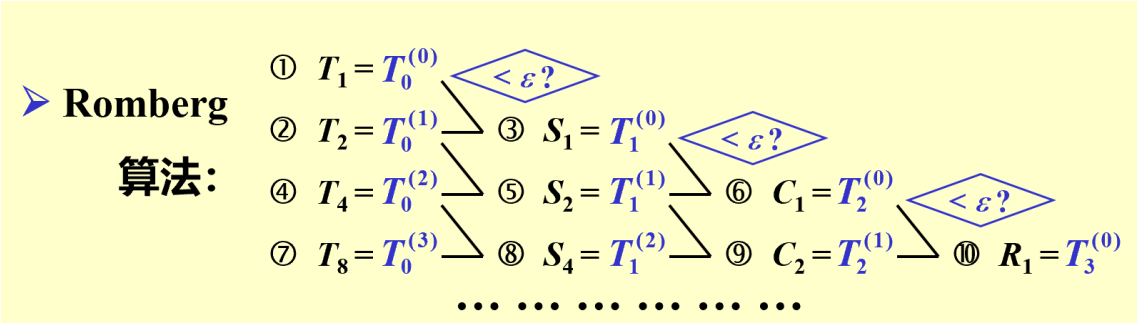

Figure 1: The algorithm flow.

The process of algorithm is as the figure1. The requirement is $|T_k^{(0)} - T_{k-1}^{(0)}| < \epsilon$, so we need figure out how to compute T_k^0 and $T_{k-1}^{(0)}$.

$$T_m^{(k)} = \frac{4^m}{4^m - 1} T_{m-1}^{(k+1)} - \frac{1}{4^m - 1} T_{m-1}^{(k)} \tag{1}$$

The equation(1) provides an iterative algorithm, which can calculate the trapezoid value after accelerating m times and k dichotomies. So in my codes, I compute out the trapezoid value after k dichotomies firstly, then $T_m^{(k)}$ can be computed out by $T_{m-1}^{(k+1)}$ and $T_{m-1}^{(k)}$, which is showed in equation(1).

4 Code analysis and thinking

The trapezoid value after n dichotomies and 0 accelerations ($T_0^{(k)}$) can be computed by following codes.

```
double T(int n, double a, double b, double (*f)(double x, double y,
    double z), double l, double t)
{
    double T0 = 0.5*(f(a,l,t)+f(b,l,t));
    if(n==0)
        return T0;
    double h = (b-a)/2;
    double T1 = 0.5*T0+0.5*f(a+h,l,t);
    if(n==1)
        return T1;
    else
    {
        double Tm;
        int m = 1;
        while(m<n)
        {
            m = m+1;
            h = h/2;
            double sum = 0;
            sum += f(a+h,l,t);
            for(int i=2;i<=pow(2,m-1);i++)
                sum += f(a+h+(i-1)*2*h,l,t);
            Tm = 0.5*T1+ h*sum;
            T1 = Tm;
        }
        return Tm;
    }
}
```

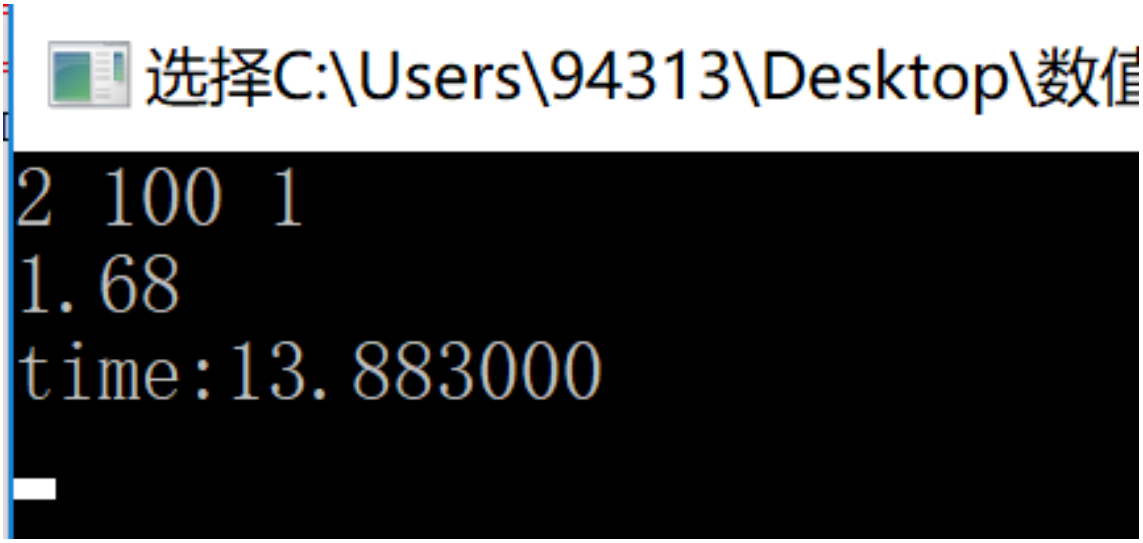

Figure 2: The result of Romberg algorithm.

The result is showed in Figure 2, and the whole code is also pasted below. The computing process cost around 13.883 second, which is not the best result. Obviously, my code is not enough perfect to solve this problem because of too many function call. In order to compute the new trapezoid value, we just need to utilize the equation(1) once. However my codes do it more than once. A good way to reduce the cost time is to reduce the times of function call, more specifically, I can store the $T_m^{(k)}$ in one way array. When I need to compute a new one, I just index the old one. For lack of time, I don't do this step in my code.

5 Code sharing

```
#include<stdio.h>
#include<math.h>
#include<time.h>
double f0( double x, double l, double t )
{
  return sqrt(1.0+l*l*t*t*cos(t*x)*cos(t*x));
}

double T(int n, double a, double b, double (*f)(double x, double y,
    double z), double l, double t)
{
    double T0 = 0.5*(f(a,l,t)+f(b,l,t));
    if(n==0)
        return T0;
    double h = (b-a)/2;
    double T1 = 0.5*T0+0.5*f(a+h,l,t);
    if(n==1)
        return T1;
    else
    {
        double Tm;
        int m = 1;
        while(m<n)
        {
            m = m+1;
            h = h/2;
            double sum = 0;
            sum += f(a+h,l,t);
            for(int i=2;i<=pow(2,m-1);i++)
                sum += f(a+h+(i-1)*2*h,l,t);
            Tm = 0.5*T1+ h*sum;
            T1 = Tm;
```

```
        }
        return Tm;
    }
}

double get_temp(int m, int k, double a, double b, double (*f)(double x,
    double y, double z), double l, double t)
{
    if(m==0)
        return T(k, a, b, f, l, t);
    else return pow(4,m)*get_temp(m-1,k+1,a,b,f,l,t)/(pow(4,m)-1) -
        get_temp(m-1,k,a,b,f,l,t)/(pow(4,m)-1);
}

double Integral(double a, double b, double (*f)(double x, double y,
   double z), double TOL, double l, double t)
{
    int m = 0;
    int k = 0;
    double temp1 = get_temp(m,k,a,b,f,l,t);
    double temp2 = get_temp(m+1,k,a,b,f,l,t);
    while(fabs(temp2-temp1)>=TOL)
    {
        m += 1;
        temp1 = temp2;
        temp2 = get_temp(m+1,k,a,b,f,l,t);
    }
    return 0.01*temp2;
}

int main()
{
  double a=0.0, b, TOL=0.005, l, t;
  while (scanf("%lf %lf %lf", &l, &b, &t) != EOF)
  {
    clock_t start, end;
    start = clock();
    double result = Integral(a, b, f0, TOL, l, t);
    end = clock();
    printf("%.2f\n", result);
    printf("time:%f\n",(double)(end-start)/CLOCKS_PER_SEC);
  }
  return 0;
}
```