Adaptive Quadrature

1. Let f be continuously differentiable on the interval $[a, b]$ and denote the integral and trapezoid formula, respectively, by

$$I_{ab} = \int_a^b f(x)dx \qquad \text{and} \qquad T_{ab} = \frac{f(a)+f(b)}{2}(b-a).$$

It is known that

$$I_{ab} = T_{ab} - \frac{f''(\xi_1)}{12}(b-a)^3 \qquad \text{for some} \qquad \xi_1 \in [a,b]. \tag{1}$$

Denote the midpoint of the interval $[a, b]$ by $c = (a + b)/2$. Show that

$$I_{ab} = T_{ac} + T_{cb} - \frac{f''(\xi_2)}{48}(b-a)^3 \qquad \text{for some} \qquad \xi_2 \in [a,b]. \tag{2}$$

By equation (1) applied to the intervals $[a, c]$ and $[c, b]$ we obtain $\xi_1' \in [a, c]$ and $\xi_1'' \in [c, b]$ such that

$$I_{ac} = T_{ac} - \frac{f''(\xi_1')}{12}(c-a)^3 \qquad \text{and} \qquad I_{cb} = T_{cb} - \frac{f''(\xi_1'')}{12}(b-c)^3.$$

Since

$$(c-a)^3 = \Big(\frac{a+b}{2} - a\Big)^3 = \frac{(b-a)^3}{8} \qquad \text{and} \qquad (b-c)^3 = \frac{(b-a)^3}{8}$$

it follows that

$$I_{ab} = I_{ac} + I_{cb} = T_{ac} + T_{cb} - \frac{f''(\xi_1') + f''(\xi_1'')}{96}(b-a)^3.$$

Let $m = (f''(\xi_1') + f''(\xi_1''))/2$. By assumption f'' is a continuous function. Since the average m lies somewhere between $f''(\xi_1')$ and $f''(\xi_1'')$, then the intermediate value theorem implies there is $\xi_2 \in [\xi_1', \xi_1'']$ such that $m = f''(\xi_2)$. It follows for this value of ξ_2 that equation (2) holds.

2. Find I_{ab} for $a = 1$, $b = 2$ and $f(x) = x \cos x$ using integration by parts. Express your answer exactly in terms of sine and cosine functions.

Using integration by parts we obtain

$$\begin{aligned}\int_a^b f(x)dx = \int_1^2 x \cos x \, dx = \int_1^2 x \, d \sin x = x \sin x \Big|_1^2 - \int_1^2 \sin x \, dx \\ = \big(x \sin x + \cos x\big)\Big|_1^2 = 2 \sin 2 + \cos 2 - \sin 1 - \cos 1.\end{aligned}$$

3. Simpson's formula is

$$S_{ab} = \frac{4T_{ac} + 4T_{cb} - T_{ab}}{3} = \frac{f(a) + 4f(c) + f(b)}{6}(b-a)$$

Write a program to calculate

$$\Big|I_{ab} - S_{ab}\Big|, \quad \Big|I_{ab} - T_{ac} - T_{cb}\Big| \quad \text{and} \quad \Big|S_{ab} - T_{ac} - T_{cb}\Big|$$

for $a = 1$, $b = 2$ and $f(x) = x\cos x$.

The program

```
1  #include <stdio.h>
2  #include <math.h>
3
4  double f(double x){
5      return x*cos(x);
6  }
7  double g(double x){
8      return cos(x)+x*sin(x);
9  }
10 double exact(double a,double b){
11     return g(b)-g(a);
12 }
13 double trap(double a,double b){
14     return (b-a)*(f(a)+f(b))/2;
15 }
16 double simp(double a,double b){
17     double c=(a+b)/2;
18     return (4*(trap(a,c)+trap(c,b))-trap(a,b))/3;
19 }
20
21 int main(){
22     double a=1, b=2, c=(a+b)/2;
23     double Iab=exact(a,b);
24     double Sab=simp(a,b);
25     double Tac=trap(a,c), Tcb=trap(c,b);
26     printf("%24s =%24.15e\n","|Iab-Sab|",fabs(Iab-Sab));
27     printf("%24s =%24.15e\n","|Iab-Tac-Tcb|",fabs(Iab-Tac-Tcb));
28     printf("%24s =%24.15e\n","|Sab-Tac-Tcb|",fabs(Sab-Tac-Tcb));
29     return 0;
30 }
```

produced the output

```
                |Iab-Sab| =   1.397247368493910e-03
            |Iab-Tac-Tcb| =   4.061966698394390e-02
            |Sab-Tac-Tcb| =   4.201691435243782e-02
```

4. It is known that

$$I_{ab} = S_{ab} - \frac{f''''(\xi_3)}{2880}(b-a)^5 \qquad \text{for some} \qquad \xi_3 \in [a,b].$$

If $b-a$ is small and f'''' is reasonably behaved, then the error in S_{ab} is much smaller than the error in $T_{ac}+T_{cb}$. In particular

$$\Big|I_{ab} - S_{ab}\Big| \ll \Big|I_{ab} - T_{ac} - T_{cb}\Big| \approx \Big|S_{ab} - T_{ac} - T_{cb}\Big|. \tag{3}$$

Are the calculations for $a=1$, $b=2$ and $f(x) = x\cos x$ in part 3 consistent with the above inequality. Repeat and comment on this inequality when $b=1.1$, when $b=1.01$ and when $b=10$.

The output of the program in part 3 shows that

$$|I_{ab} - T_{ac} - T_{cb}| \approx |S_{ab} - T_{ac} - T_{cb}| \approx 4\times 10^{-2}.$$

Since $|I_{ab} - S_{ab}| \approx 1.4\times 10^{-3} \ll 4\times 10^{-2}$, then the calculations are consistent with inequality (3). To check the inequality for other values of b the program in part 3 was modified by introducing a loop in the main routine as follows:

```
21 int main(){
22     double B[] = {2, 1.1, 1.01, 10};
23     int n=sizeof(B)/sizeof(double);
24     printf("#%4s %5s %20s %20s %20s\n","a","b",
25         "|Iab-Sab|","|Iab-Tac-Tcb|","|Sab-Tac-Tcb|");
26     for(int i=0;i<n;i++){
27         double a=1, b=B[i], c=(a+b)/2;
28         double Iab=exact(a,b);
29         double Sab=simp(a,b);
30         double Tac=trap(a,c), Tcb=trap(c,b);
31         printf("%5g %5g %20.12e %20.12e %20.12e\n",a,b,
32             fabs(Iab-Sab),fabs(Iab-Tac-Tcb),fabs(Sab-Tac-Tcb));
33     }
34     return 0;
35 }
```

The modified program then produced the output

```
#   a     b            |Iab-Sab|         |Iab-Tac-Tcb|         |Sab-Tac-Tcb|
    1     2   1.397247368494e-03   4.061966698394e-02   4.201691435244e-02
    1   1.1   1.385923877745e-08   4.699581689749e-05   4.700967613627e-05
    1  1.01   1.360205351131e-13   4.639745631446e-08   4.639759233500e-08
    1    10   1.927153900291e+01   7.537203626338e+00   1.173433537658e+01
```

which shows that inequality (3) holds when $b = 2, 1.2$ and 1.01 but not when $b=10$. We comment that $b-a=9$ when $b=10$ which is too large for the error term of the higher order method S_{ab} to be small.

5. Fix $n \in \mathbf{N}$, let $h = (b-a)/n$ and let $x_i = a + ih$. Denote

$$T_i = T_{x_i x_{i+1/2}} + T_{x_{i+1/2}, x_{i+1}} \qquad \text{and} \qquad S_i = S_{x_i, x_{i+1}}.$$

Let

$$S = \sum_{i=0}^{n-1} S_i \qquad \text{and} \qquad E = \sum_{i=0}^{n-1} \big|S_i - T_i\big|$$

and use part 4 to show that $|I_{ab} - S| \ll E$.

Define $I_i = I_{x_i, x_{i+1}}$. Then inequality (3) may be written as

$$|I_i - S_i| \ll |I_i - T_i| \approx |S_i - T_i|.$$

It follows, upon using the triangle inequality, that

$$\begin{aligned} |I_{ab} - S| = \Big| \sum_{i=0}^{n-1} (I_i - S_i) \Big| &\le \sum_{i=0}^{n-1} |I_i - S_i| \\ &\ll \sum_{i=0}^{n-1} |I_i - T_i| \approx \sum_{i=0}^{n-1} |S_i - T_i| = E, \end{aligned}$$

which is the desired result.

6. Write a C computer program to compute S, $|I_{ab} - S|$ and E for $a = 1$, $b = 10$ and $f(x) = x \cos x$ when $n = 10, 20, 50$ and 100.

The main routine of the code in part 3 was modified as

```
21 int main(){
22     int N[] = {10, 20, 50, 100};
23     double K=sizeof(N)/sizeof(int);
24     double a=1, b=10, Iab=exact(a,b);
25     printf("#%4s %20s %20s %20s\n","n","S","|Iab-S|","E");
26     for(int k=0;k<K;k++){
27         int n=N[k];
28         double h=(b-a)/n, S=0, E=0;
29         for(int i=0;i<n;i++){
30             double xi=a+i*h, xip1=a+(i+1)*h, c=(xi+xip1)/2;
31             double Si=simp(xi,xip1);
32             double Ti=trap(xi,c)+trap(c,xip1);
33             S+=Si;
34             E+=fabs(Si-Ti);
35         }
36         printf("%5d %20.12e %20.12e %20.12e\n",n,
37             S,fabs(Iab-S),E);
38     }
39     return 0;
40 }
```

and produced the output

```
#   n                    S              |Iab-S|                    E
   10  -7.661540470735e+00   4.845420889383e-04   5.821134088989e-01
   20  -7.661086394921e+00   3.046627511250e-05   1.498443716033e-01
   50  -7.661056709757e+00   7.811105993127e-07   2.390895971802e-02
  100  -7.661055977476e+00   4.882960791264e-08   5.983358624841e-03
```

Note that $|I_{ab} - S| \ll E$ for all values of n tested.

7. To guarantee $E < \epsilon$ it is sufficient that $|S_i - T_i| < \epsilon/n$ for $i = 0, \ldots, n-1$. Write a recursive function $\text{quad}(a, b, \epsilon)$ defined as

$$\text{quad}(a,b,\epsilon) = \begin{cases} S_{ab} & \text{if } |S_{ab} - T_{ac} - T_{cb}| < \epsilon \\ \text{quad}(a,c,\epsilon/2) & \\ \qquad + \text{quad}(c,b,\epsilon/2) & \text{otherwise} \end{cases}$$

and then use the formula

$$\sum_{i=0}^{n-1} \text{quad}(x_i, x_{i+1}, \epsilon/n)$$

to approximate I_{ab} to precision $\epsilon = 10^{-7}$ for $a = 1$, $b = 10$ and $f(x) = x \cos x$. Run your program with $n = 10$. How does the initial choice of n affect your results? What is a good way of choosing n?

For convenience of notation we write

$$Q = \sum_{i=0}^{n-1} Q_i \qquad \text{where} \qquad Q_i = \text{quad}(x_i, x_{i+1}, \epsilon/n).$$

The program

```
1  #include <stdio.h>
2  #include <math.h>
3
4  double f(double x){
5      return x*cos(x);
6  }
7  double g(double x){
8      return cos(x)+x*sin(x);
9  }
10 double exact(double a,double b){
11     return g(b)-g(a);
12 }
13 double trap(double a,double b){
14     return (b-a)*(f(a)+f(b))/2;
15 }
16 double simp(double a,double b){
17     double c=(a+b)/2;
18     return (4*(trap(a,c)+trap(c,b))-trap(a,b))/3;
19 }
20 double quad(double a,double b,double epsilon){
21     double c=(a+b)/2;
22     double Sab=simp(a,b), Tac=trap(a,c), Tcb=trap(c,b);
```

```
23      if(fabs(Sab-Tac-Tcb)<epsilon) return Sab;
24      return quad(a,c,epsilon/2)+quad(c,b,epsilon/2);
25  }
26
27  int main(){
28      int N[] = {10, 20, 50, 100};
29      double K=sizeof(N)/sizeof(int);
30      double a=1, b=10, Iab=exact(a,b);
31      double epsilon=1e-07;
32      printf("#%4s %24s %20s %12s\n","n",
33          "Q","|Iab-Q|","epsilon");
34      for(int k=0;k<K;k++){
35          int n=N[k];
36          double h=(b-a)/n, Q=0;
37          for(int i=0;i<n;i++){
38              double xi=a+i*h, xip1=a+(i+1)*h;
39              double Qi=quad(xi,xip1,epsilon/n);
40              Q+=Qi;
41          }
42          printf("%5d %24.15e %20.12e %12.4e\n",n,
43              Q,fabs(Iab-Q),epsilon);
44      }
45      return 0;
46  }
```

produced the output

```
#   n                        Q              |Iab-Q|      epsilon
   10   -7.661055928646188e+00   1.776356839400e-15   1.0000e-07
   20   -7.661055928646188e+00   1.776356839400e-15   1.0000e-07
   50   -7.661055928646091e+00   9.503509090791e-14   1.0000e-07
  100   -7.661055928646091e+00   9.503509090791e-14   1.0000e-07
```

Each choice of n produced a nearly identical estimate of I_{ab} accurate to 12 digits. As long as the initial inequality (3) holds on each of the intervals $[x_i, x_{i+1}]$ used to define the Q_i in the main loop, then the value of n shouldn't matter.

When $n = 1$ inequality (3) doesn't hold as seen part 4. Even in this case, the method is likely to work as the following output demonstrates:

```
#   n                        Q              |Iab-Q|      epsilon
    1   -7.661055928647239e+00   1.053379605764e-12   1.0000e-07
    2   -7.661055928647239e+00   1.053379605764e-12   1.0000e-07
    3   -7.661055928646181e+00   4.440892098501e-15   1.0000e-07
```

8. [Extra Credit and Math/CS 666]. Show that

$$R_{ab} = \frac{16S_{ac} + 16S_{cb} - S_{ab}}{15} = I_{ab} + \mathcal{O}\big((b-a)^7\big) \tag{4}$$

and use the error estimate $|I_{ab} - R_{ab}| \ll |R_{ab} - S_{ac} - S_{cb}|$ to create a higher order adaptive quadrature routine. Write your program in a way to minimize the number of function calls by reusing previously computed values of $f(x)$ where possible. Test your routine under the same conditions as part 7 and determine its efficiency by comparing how many function calls are used by each method to achieve the same accuracy. Make up an additional test problem by choosing new values for f, a, b, ϵ and n and compare the results.

To show relation (4) use Taylor's theorem with the help of a computer algebra system to avoid errors. Define

$$g(x) = \int f(x)dx$$

to be an anti-derivative of f. The fundamental theorem of calculus implies

$$I_{ab} = g(b) - g(a) \qquad \text{and} \qquad f(x) = g'(x).$$

Let $b = a + h$ and then develop $I_{ab} - R_{ab}$ as a Taylor series in h around $h = 0$. The script for Reduce which does this is

```
1  load_package(dfpart);
2  load_package(taylor);
3  generic_function g(x);
4  operator f;
5  let f(~arg1) => sub(x=arg1,df(g(x),x));
6  Iab:=g(a+h)-g(a);
7  Tab:=(f(a)+f(a+h))/2*h;
8  taylor(Tab-Iab,h,0,3);
9  Sab:=(f(a)+4*f(a+h/2)+f(a+h))/6*h;
10 taylor(Sab-Iab,h,0,5);
11 Sac:=(f(a)+4*f(a+h/4)+f(a+h/2))/6*(h/2);
12 Scb:=(f(a+h/2)+4*f(a+3*h/4)+f(a+h))/6*(h/2);
13 Rab:=(16*Sac+16*Scb-Sab)/15;
14 taylor(Rab-Iab,h,0,7);
15 quit;
```

with output

```
Reduce (Free PSL version), 25-Sep-2014 ...

1:
2:
```

```
3:
4:
5:
6:
iab := g(a + h) - g(a)

7:
        h*(g (a + h) + g (a))
            x           x
tab := ----------------------
                2

8:
 g   (a)
  xxx      3      4
---------*h  + O(h )
   12

9:
                           2*a + h
        h*(g (a + h) + 4*g (---------) + g (a))
            x              x     2         x
sab := ----------------------------------------
                         6

10:
 g     (a)
  xxxxx      5      6
-----------*h  + O(h )
   2880

11:
                 4*a + h         2*a + h
        h*(4*g (---------) + g (---------) + g (a))
              x     4         x     2         x
sac := ----------------------------------------------
                          12

12:
                            4*a + 3*h         2*a + h
        h*(g (a + h) + 4*g (-----------) + g (---------))
            x              x      4          x     2
scb := ------------------------------------------------
                             12

13:
                               4*a + 3*h            4*a + h
rab := (h*(7*g (a + h) + 32*g (-----------) + 32*g (---------)
              x              x      4             x     4
```

```
                  2*a + h
          + 12*g (---------) + 7*g (a)))/90
                x     2           x
```

```
14:
 g       (a)
  xxxxxxx      7      8
-------------*h  + O(h )
   1935360

15:
Quitting
```

The last line of output above shows that error in the R_{ab} method is

$$R_{ab} = I_{ab} + \frac{f^{(6)}(\xi)}{1935360}(b-a)^7 \qquad \text{for some} \qquad \xi \in [a,b]$$

which verifies (4). The same computation can be done using the Maple script

```
1  restart;
2  f:=unapply(diff(g(x),x),x);
3  Iab:=g(a+h)-g(a);
4  Tab:=(f(a)+f(a+h))/2*h;
5  series(Tab-Iab,h=0,4);
6  Sab:=(f(a)+4*f(a+h/2)+f(a+h))/6*h;
7  series(Sab-Iab,h=0,6);
8  Sac:=(f(a)+4*f(a+h/4)+f(a+h/2))/6*(h/2);
9  Scb:=(f(a+h/2)+4*f(a+3*h/4)+f(a+h))/6*(h/2);
10 Rab:=(16*Sac+16*Scb-Sab)/15;
11 series(Rab-Iab,h=0,8);
12 quit;
```

with output

```
                        f := D(g)

                   Iab := g(a + h) - g(a)

               Tab := 1/2 (D(g)(a) + D(g)(a + h)) h

                        (3)         3      4
                   1/12 (D   )(g)(a) h  + O(h )

            Sab := 1/6 (D(g)(a) + 4 D(g)(a + h/2) + D(g)(a + h)) h

                        (5)         5      6
                  1/2880 (D   )(g)(a) h  + O(h )

          Sac := 1/12 (D(g)(a) + 4 D(g)(a + h/4) + D(g)(a + h/2)) h
```

```
                  /                            3 h               \
      Scb := 1/12 |D(g)(a + h/2) + 4 D(g)(a + ---) + D(g)(a + h)| h
                  \                             4                /

Rab := 4/45 (D(g)(a) + 4 D(g)(a + h/4) + D(g)(a + h/2)) h

           /                            3 h               \
     + 4/45 |D(g)(a + h/2) + 4 D(g)(a + ---) + D(g)(a + h)| h
           \                             4                /

     - 1/90 (D(g)(a) + 4 D(g)(a + h/2) + D(g)(a + h)) h

                                  (7)         7      8
                     1/1935360 (D   )(g)(a) h  + O(h )
```

We also simplify the expression for R_{ab} using a computer algebra system. The reduce script

```
1  operator f;
2  operator T;
3  let T(~a,~b) => (b-a)*(f(a)+f(b))/2;
4  operator S;
5  let S(~a,~b) => (4*T(a,(a+b)/2)+4*T((a+b)/2,b)-T(a,b))/3;
6  S(a,a+h);
7  operator R;
8  let R(~a,~b) => (16*S(a,(a+b)/2)+16*S((a+b)/2,b)-S(a,b))/15;
9  R(a,a+h);
10 quit;
```

produced the output

```
Reduce (Free PSL version), 25-Sep-2014 ...

1:
2:
3:
4:
5:
6:
                    2*a + h
 h*(f(a + h) + 4*f(---------) + f(a))
                       2
---------------------------------------
                 6

7:
8:
9:
                       4*a + 3*h           4*a + h           2*a + h
(h*(7*f(a + h) + 32*f(-----------) + 32*f(---------) + 12*f(---------) + 7*f(a))
                           4                 4                 2

 )/90

10:
Quitting
```

Equivalently, the Maple script

```
restart;
T:=(a,b)->(b-a)*(f(a)+f(b))/2;
S:=(a,b)->(4*T(a,(a+b)/2)+4*T((a+b)/2,b)-T(a,b))/3;
simplify(S(a,a+h));
R:=(a,b)->(16*S(a,(a+b)/2)+16*S((a+b)/2,b)-S(a,b))/15;
simplify(R(a,a+h));
quit;
```

produces the output

```
                    T := (a, b) -> 1/2 (b - a) (f(a) + f(b))

 S := (a, b) -> 4/3 T(a, 1/2 a + 1/2 b) + 4/3 T(1/2 a + 1/2 b, b) - 1/3 T(a, b)

                 1/6 h f(a) + 2/3 h f(a + h/2) + 1/6 h f(a + h)

               16                        16
 R := (a, b) -> -- S(a, 1/2 a + 1/2 b) + -- S(1/2 a + 1/2 b, b) - 1/15 S(a, b)
               15                        15

             16                                      16         3 h
7/90 h f(a) + -- h f(a + h/4) + 2/15 h f(a + h/2) + -- h f(a + ---)
             45                                      45          4

    + 7/90 h f(a + h)
```

In either case we see that

$$R_{ab} = \frac{7f(a) + 32f(c_1) + 12f(c_2) + 32f(c_3) + 7f(b)}{90}(b-a)$$

where $c_k = a + k(b-a)/4$. Using this formula along with the trick of passing the values of f at endpoints of each interval as arguments to the recursive function we obtain the optimized program

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

static int cf;

double f(double x){
    cf++;
    return x*cos(x);
}
double g(double x){
    return cos(x)+x*sin(x);
}
double exact(double a,double b){
```

```
14      return g(b)-g(a);
15  }
16  double quad(double a,double fa,double b,double fb,double epsilon){
17      double c=(a+b)/2, fc=f(c);
18      double Sab=(b-a)*(fa+4*fc+fb)/6;
19      double Tac=(c-a)*(fa+fc)/2;
20      double Tcb=(b-c)*(fc+fb)/2;
21      if(fabs(Sab-Tac-Tcb)<epsilon) return Sab;
22      return quad(a,fa,c,fc,epsilon/2)+quad(c,fc,b,fb,epsilon/2);
23  }
24  double Rquad(double a,double fa,double b,double fb,double epsilon){
25      double c=(a+b)/2, fc=f(c);
26      double c1=(a+c)/2, fc1=f(c1);
27      double c3=(c+b)/2, fc3=f(c3);
28      double Rab=(b-a)*(7*fa+32*fc1+12*fc+32*fc3+7*fb)/90;
29      double Sac=(c-a)*(fa+4*fc1+fc)/6;
30      double Scb=(b-c)*(fc+4*fc3+fb)/6;
31      if(fabs(Rab-Sac-Scb)<epsilon) return Rab;
32      return Rquad(a,fa,c,fc,epsilon/2)+Rquad(c,fc,b,fb,epsilon/2);
33  }
34
35  int main(){
36      int N[] = {1, 2, 3, 10, 20, 50, 100};
37      double K=sizeof(N)/sizeof(int);
38      double a=1, b=10, Iab=exact(a,b);
39      double epsilon=1e-07;
40      printf("#Simpson/Trapezoid Adaptive Quadrature\n");
41      printf("#%4s %24s %20s %12s %7s\n","n",
42          "Q","|Iab-Q|","epsilon","calls");
43      for(int k=0;k<K;k++){
44          cf=0;
45          int n=N[k];
46          double h=(b-a)/n, Q=0;
47          double xi=a, fxi=f(a);
48          for(int i=1;i<=n;i++){
49              double xip1=a+i*h, fxip1=f(xip1);
50              double Qi=quad(xi,fxi,xip1,fxip1,epsilon/n);
51              Q+=Qi;
52              xi=xip1; fxi=fxip1;
53          }
54          printf("%5d %24.15e %20.12e %12.4e %7d\n",n,
55              Q,fabs(Iab-Q),epsilon,cf);
56      }
57      printf("\n\n#Richardson/Simpson Adaptive Quadrature\n");
58      printf("#%4s %24s %20s %12s %7s\n","n",
59          "R","|Iab-R|","epsilon","calls");
60      for(int k=0;k<K;k++){
61          cf=0;
62          int n=N[k];
63          double h=(b-a)/n, R=0;
```

```
64          double xi=a, fxi=f(a);
65          for(int i=1;i<=n;i++){
66              double xip1=a+i*h, fxip1=f(xip1);
67              double Ri=Rquad(xi,fxi,xip1,fxip1,epsilon/n);
68              R+=Ri;
69              xi=xip1; fxi=fxip1;
70          }
71          printf("%5d %24.15e %20.12e %12.4e %7d\n",n,
72              R,fabs(Iab-R),epsilon,cf);
73      }
74      return 0;
75  }
```

Note that this program counts the number of times f is called by using a global counter delared at the beginning of the file.

The resulting output is

```
#Simpson/Trapezoid Adaptive Quadrature
#   n                       Q              |Iab-Q|      epsilon   calls
    1   -7.661055928647239e+00   1.053379605764e-12   1.0000e-07   64527
    2   -7.661055928647239e+00   1.053379605764e-12   1.0000e-07   64527
    3   -7.661055928646181e+00   4.440892098501e-15   1.0000e-07   66833
   10   -7.661055928646189e+00   2.664535259100e-15   1.0000e-07   63105
   20   -7.661055928646188e+00   1.776356839400e-15   1.0000e-07   63105
   50   -7.661055928646091e+00   9.503509090791e-14   1.0000e-07   64927
  100   -7.661055928646092e+00   9.414691248821e-14   1.0000e-07   64927


#Richardson/Simpson Adaptive Quadrature
#   n                       R              |Iab-R|      epsilon   calls
    1   -7.661055928539009e+00   1.071773780836e-10   1.0000e-07     713
    2   -7.661055928539009e+00   1.071773780836e-10   1.0000e-07     711
    3   -7.661055928645403e+00   7.833733661755e-13   1.0000e-07     697
   10   -7.661055928672496e+00   2.630962114836e-11   1.0000e-07     779
   20   -7.661055928672494e+00   2.630784479152e-11   1.0000e-07     759
   50   -7.661055928654511e+00   8.324896327849e-12   1.0000e-07     561
  100   -7.661055928647698e+00   1.512567848749e-12   1.0000e-07     479
```

Computing using R_{ab} is about 100 times more efficient, in terms of number of function calls used, for this test problem. First, a high order method is being used. Second, since the error estimates for the adaptive steps are also more accurate, not as much extra work was done because of overestimating the error. This last observation is reflected in the fact that the actual error $|I_{ab} - R|$ is much closer to the desired value of ϵ than the values of $|I_{ab} - Q|$ given by the previous method.

The additional test problems will vary. An interesting example might be found by integrating a non-smooth function such as the Weierstrass function

$$f(x) = \sum_{n=0}^{\infty} a^n \cos(b^n \pi x)$$

where $0 < a < 1$, b is a positive odd integer and $ab > 1 + 3\pi/2$. Additional extra credit is available for this particular function if turned in by the end of the semester.