Foundations of Computational Math 2

Program 3

Chenchen Zhou

1. Program Description

In this program, we implement two kinds of adaptive Newton-Cotes integration strategies.
First one is just an adaptive method which is accomplished by step halving. The second one is also involving the step halving but it improves the efficiency.

2. Code Description

For first adaptive method, which uses step halving over the entire interval and refinement continues until the estimated error is smaller than a requested tolerance.
For every step, use the relationship between old T and new T

$$T = \frac{T}{2} + \frac{(\sum_{i=1}^{n} f(a+(2i-1)\cdot\frac{(b-a)}{2n}))\cdot(b-a)}{2n}$$

We know every step, we can use the functional values which are computed by the formal step and we just need to calculate the n new points.

For second adaptive method, we need to check the error from left halve and right halve and doing the respective step halving only when they exceed their alloted portions of the error requested.

For the error control, we know the error formula of trapezoidal rule and by using the thought of extrapolation. We derive the

$$I_1 : \frac{h}{2}[f_0 + f_1], \quad E_1 = -\frac{h^3}{12} f^{(2)}(\eta), \text{ trapezoidal rule}$$

$$E_f \approx \hat{E}_f = \frac{1}{(2^r - 1)}(I_f - I_c)$$

And use it to terminate our codes.

Main function

I write four functions.

Trapezoidal.m
This function is used to evaluate the quadrature by applying the trapezoidal rule.

Halving.m

This function is used to refine the quadrature by using the halving step.

Adaptive.m

This function is another adaptive method to calculate the quadrature and it halving the left and right halves by checking the error bound requested.

Test.m
This function is just used to run the Adaptive function. And it gives out the difference between the actual error and estimated error. And the number of function evaluation.

3. Experiments and results.

1. **verify the degree of exactness of the method by integrating an appropriate set of polynomials;**

According to the definition about the degree of exactness, I choose f0=constant, f1=x, f2=x^2 to for testing with the Trapezoidal.m.

	by our code	actual value
f0=2	4	4
f1=x	2	2
f2=x^2	3	3/8

This results in that the degree of the exactness for trapezoidal rule is 1.

2. **Integrate f(x) = ex on [a, b] = [0, 2] and compare the true error, I(f) − In(f), and the error estimate that controls your termination of the adaptive algorithm;**

tolerence	Halving		Adaptive	
0.000001	1.27E-07	4097	7.11E-14	495
0.001	1.30E-04	129	8.11E-08	15
0.1	0.0332	9	1.38E-04	2

This result shows that the actual error is very closely to our estimate error. In generally , it is smaller than the estimate error and is about 1/10 of the estimated error.

3. **compare the number of function evaluations required by the two adaptive forms of Newton Cotes to achieve a similar accuracy over the entire interval**

of integration.

From this result in the table above we can conclude that the second adaptive method, i.e. step halving by left and right halves respectively has much higher efficiency in terms of the number of function evaluations. To achieve a similar accuracy, the second method has much less number of evaluations. The number is almost 1/10 of the number of the method which step halving over the entire interval.

4. Codes

Adaptive.m

```
function [nodes ll]= Adaptive(a,b,n,Abs_Tol)

    inte1=Trapezoidal(a,b,n);
    inte2=Trapezoidal(a,a+(b-a)/2,n)+Trapezoidal(a+(b-a)/2,b,n);
    err=(inte2-inte1)/3;
    if (abs(err)<=Abs_Tol)
        ll=inte2+err;
        nodes=1;
    else
        [n1 inte_left]=Adaptive(a,a+(b-a)/2,n,Abs_Tol/2);
        [n2 inte_right]=Adaptive(a+(b-a)/2,b,n,Abs_Tol/2);
        nodes=n1+n2;
        ll=inte_left+inte_right;
    end

end
```

Halving.m

```
function [Nodes T] = Halving(a,b,Abs_Tol)
n=1;
h=(b-a)/n;
T0=-10000;
T=h*(f(a)+f(b))/2;
while(abs(T-T0)>Abs_Tol )
T0=T;
T=0;
h=h/2;
n=2*n;
T=f(a);
```

```
for i=1:n-1
T=2*f(a+i*h)+T;
end
T=(T+f(b))*h/2;
end
Nodes=n+1;
abs(exp(2)-1-T)
```

Trapezoidal.m

```
function T=Trapezoidal(a,b,n)
h=(b-a)/n;
T=f(a);
for i=1:n-1
T=2*f(a+i*h)+T;
end
T=(T+f(b))*h/2;
```

Test.m

```
function [nodes ll]=test(a,b,n,Abs_Tol)
[nodes ll]= Adaptive(a,b,n,Abs_Tol);
abs(exp(2)-1-ll)
```