A Modified Adaptive Quadrature Scheme for Continuous Piecewise-Smooth Functions

Swagato Chakraborty and Vikram Jandhyala
{swagato,jandhyala}@ee.washington.edu

Dept of EE, University of Washington
Seattle WA, 98195-2500

UWEE Technical Report
Number UWEETR-2002-0015
August 2, 2002

Department of Electrical Engineering
University of Washington
Box 352500
Seattle, Washington 98195-2500
PHN: (206) 543-2150
FAX: (206) 543-3842
URL: http://www.ee.washington.edu

An Adaptive Quadrature Scheme for Continuous Piecewise-Smooth Functions

Swagato Chakraborty and Vikram Jandhyala

Applied Computational Electromagnetics Lab
Department of Electrical Engineering
University of Washington
Seattle WA 98195
Phone: 206-543-2186
Fax: 206-543-3842
Email: jandhyala@ee.washington.edu

An adaptive quadrature routine is presented here, that uses a dynamically updated estimate of the total integral value for convergence. For continuous piecewise-smooth functions, this enables comparable accuracies with existing schemes that have a coarse estimate of the total integral value with fewer number of quadrature points.

A pseudo-code to integrate a continuous and piece-wise smooth function within given limits is discussed in the report. Let a continuous and piece-wise smooth function $f(\rho)$ has to be integrated within given limits $(\rho_{\min}, \rho_{\max})$. Also there are L smooth segments $f_i(\rho)$ given by (ρ_i, ρ_{i+1}), where $i \in \{0,1,2,.....L-1\}$, $\rho_0 = \rho_{\min}$, and $\rho_K = \rho_{\max}$, the total integral is written as

$$I = \sum_{i=1}^{L} \int_{\rho_{i-1}}^{\rho_i} f_i(\rho)\, d\rho \tag{1.1}$$

Initially a coarse estimate of I^{est} is obtained by using a 5 point Newton-Cotes formula based on the Bodé rule [1], over each of the individual L segments. The rule computes an integral $Q_{ab} = \int_a^b f(x)\, dx$ with the approximation

$$Q_{ab} \cong \frac{h}{90}\{7f(a) + 32f(a+\frac{h}{4}) + 12f(a+\frac{h}{2}) + 32f(a+\frac{3h}{4}) + 7f(a+h)\} \tag{1.2}$$

where $h = b-a$. Thus the total integral is initially estimated as

$$I^{est} = \sum_{i=1}^{L} Q_{\rho_{i-1}\rho_i} \tag{1.3}$$

The quadrature is refined recursively by using an adaptive integration method. For efficiency in terms of number of sample points, non-uniform sampling of the function $f(\rho)$ is required within each subinterval (ρ_{i-1}, ρ). At a particular level of recursion, for a given subinterval (a,b), an estimate of the integral Q_{ab} is obtained using Eqn.(1.2). Subsequently, a binary split is performed on (a,b), and $Q'_{ab} = Q_{ac} + Q_{cb}$ is obtained where $c = (a+b)/2$. The estimate of $I^{init}_{\varphi\chi}$ is dynamically refined as

$$I_{\varphi\chi}^{init} \leftarrow I_{\varphi\chi}^{init} + (Q_{ac} + Q_{cb} - Q_{ab}) \tag{1.4}$$

at each level of recursion. If the change in the integration result due to the binary split,

$$\Delta = Q_{ac} + Q_{cb} - Q_{ab} \tag{1.5}$$

relative to the total integral $I_{\varphi\chi}^{init}$ is smaller than a pre-specified threshold tolerance $tol1$, then the contribution of that split to the over-all integral is ignored. The exact stopping criteria used for the convergence test are given by

$$\lfloor \Delta \rfloor \leq tol1 \cdot \lfloor I_{\varphi\chi}^{init} \rfloor \tag{1.6a}$$

or

$$\lfloor \Delta \rfloor \leq tol2 \cdot \lfloor Q_{ab} \rfloor \tag{1.6b}$$

where the criterion in Eqn.(6.6b) using the pre-specified tolerance $tol2$ is required in order to expedite the convergence in the case of a nearly linear function segment.

The method described above is similar in approach to the existing popular Matlab-based adaptive quadrature method *quad* [2], with some important differences. it uses a different order rule to evaluate Q_{ab} in Eqn.(1.2). Also it uses an improved dynamic refinement of the estimate of the total integral Eqn.(1.4), and leads to rules with smaller number of samples for a given approximation error for the integrals in this paper.

A Matlab pseudo-routine *IntAdaptive* implementing the above method is presented. Note that this routine uses a different integration rule from the existing popular Matlab-based adaptive quadrature method *quad*[1], and has an improved dynamic refinement of the estimate of the total integral Eqn.(1.4), and leads to rules with smaller number of samples for a given approximation error for the integrals in this paper. In terms of the presented routine IntAdaptive, the final estimate of the required integral over all segments is given by

$$I_{\varphi\chi}^{final} = \sum_{i=1}^{K} IntAdaptive(f, \rho_{i-1}, \rho_i, tol, vArg\{:\}) \tag{1.7}$$

```
function Integral = IntAdaptive(f,a,b,tol,vArg{:})
% IntAdaptive (f,a,b,tol,vArg{:}) returns the integration of function f with the arguments
% given by vArg{:}, using a three point Simpson's rule a 5 point double Simpson's rule
% and one level of Romberg refinement.

% The over all segment length
h=b-a;

% The respective weights  are defined based on eqn.( )
weight=[7 32 12 32 7]./90;

%The initial 5 points where the samples are computed, eps is a very small number
%(machine precision) ,   which is used so that the sample points are never beyond the
%boundary, due to machine precision error
    xPoints=[(a+eps)  (a+h/4) (a+h/2) (b-h/2) (b-eps)];

%Sample value at the designated sample points
```

```
yValue=feval(f,xPoints,vArg{:});

%Obtaining the initial estimate of the integral
initialIntegral = sum(yValue.*weight)*(b-a);

%Initializing Level to zero
level=0;

%Invoking the core adaptive routine
Integral=adaptation(f,(a+eps),(b-eps),tol,yValue,initialIntegral,vArg,level);

%Function ends
return;



%adaptation(f,a,b,tol,yValue,integralParent,vArg,level) is the core adaptive integration
routine
function Integral = adaptation(f,a,b,tol,yValue,integralParent,vArg,level)

% Error tolerance to compare the integration values before and after a binary split
global running_tol;

%Initially estimated and refined toal integral value
global I_init;
%Number of recursions
global LEVEL_THRESHOLD;
weight =[7 32 12 32 7]./90;
h=b-a;
xPointsLeft=[a ,a+h/8 ,a+h/4 ,a+3h/8 ,a+h/2];
xPointsRight=[b-h/2, b-3h/8 ,b-h/4, b-h/8, b];
yValueLeft(1:2:5)=y(1:3);
yValueRight(1:2:5)=y(3:5);

%Inserting new function values
yValueLeft(2:2:4)=feval(f,xPointsLeft(2:2:4),vArg{:});
yValueRight(2:2:4)=feval(f,xPointsRight(2:2:4),vArg{:});

%Splitting the interval , and integrating them seperately
h=h/2;
int1=sum(yValueLeft.*weight)*h;
int2=sum(yValueRight.*weight).*h;

%Change due to binary split
delta=int1+int2-int;
```

```
%Refining the global integral value
I_init=I_init+delta;

%Checking if the change in integration value small compared to the global integral
cond1=abs(delta)<tol*abs(I_init);

% Checking if the change in integration value small compared to the parent level integral
cond2=abs(delta)<running_tol*abs(int1+int2);

% If too many recursions
cond3=level>=LEVEL_THRESHOLD;
if(cond1 | cond2 | cond3)
        Integral=int1+int2;
        %Recursion Ends
        return;
end

%Updating number of recursion
level=level+1;

%Binary Split
int_left= adaptation(f,a,a+h,tol,y1,int1,vArg,level);
int_right= adaptation(f, a+h,b,tol,y2,int2,vArg,level);
integral=int_left+int_right;
return;
```

References

[1] M. Abramowitz and I. Stegun, Chapter 25, *Handbook of Mathematical Functions*, Dover, New York, 1970.

[2] W.Gander and W.Gautschi, "Adaptive quadrature - revisited," Technical Report 306, Department Informatik, ETH Zürich, August 1998.