

Homework 4

Math 128A

Due Monday, 11/24/08, 11:00 a.m.

- Write a composite trapezoidal rule routine to compute an approximation to the integral of a function, f , over an interval $[a, b]$. Your routine should take as inputs: the integrand f , the endpoints a and b , and the number the number of subintervals n .
 - Write a composite Simpson's rule integration routine that takes the same inputs as your composite trapezoidal rule routine.
 - Write a composite 3-point Gaussian quadrature routine that takes the same inputs as your composite trapezoidal rule routine.
 - Apply each of these composite quadratures to approximate

$$\int_0^1 e^x dx.$$

Make a table of the results for $n = 2, 4, 8, 16, 32$, and a table of the errors. How is the order of accuracy demonstrated in the table of errors? Comment on your results.

- Consider the integral

$$\int_0^1 x^{-2/3} \exp(-x^2) dx.$$

The integrand is singular, although it is integrable. The value of this integral is approximately 2.6647910962558108.

- Use MATLAB's built-in quadrature, `quad`, to compute this integral, as written, with error tolerances of 10^{-8} , 10^{-10} , 10^{-12} . What values of the approximate integral does this give? What are the actual errors? How many function evaluations are required?
 - Make the change of variables $x = u^\alpha$ in this integral. What values of α give a transformed integral that is appropriate for numerical quadrature?
 - Repeat part (a) with your transformed integral, and comment on the results.
- On the last page of this assignment a MATLAB code for adaptive Simpson quadrature is given.
 - Briefly describe each of the six blocks of code labeled 1 through 6.
 - This code is written for clarity, not for efficiency. Explain why this program is not an efficient implementation of adaptive Simpson's rule.
 - Change this code to make an adaptive 3-point Gaussian quadrature. Is this an efficient (not necessarily optimal) implementation?
 - Use the given adaptive Simpson's rule routine, your adaptive Gaussian quadrature routine, and MATLAB's build-in adaptive Simpson's quadrature, `quad`, to evaluate the the integral below with an error tolerance of 10^{-10} .

$$\int_0^1 \frac{1}{x + 0.01} dx$$

What are the actual errors in each case? How many function evaluations are necessary in each routine. Comment on your results.

- (e) Analytically derive a bound for the number of points needed to guarantee that the error is below 10^{-10} for the composite Trapezoidal rule and for the composite Simpson's rule each using equally spaced points for the integral above.
- (f) Experiment with your composite Simpson's rule code to estimate (within 1000 points) the actual number of points needed to get an error below 10^{-10} . Comment on your results taking into account the analytic bound on the number of points and the number of function evaluations used by the (efficient) adaptive routines.

```

% adaptive quadrature using Simpson's rule
%
% input: f      - integrand, scalar function that takes one input
%          a,b   - integration is over interval [a,b]
%          ep    - error tolerance
%          count - number of times the integrand is evaluated
%                  this should not be passed in by the user, it
%                  is only used in recursive calls
%
% output: q      - the approximation to the integral
%          new_count - number of times the integrand was evaluated
%
% note: this function requires that the quadrature rule S be
%       defined in the same file
%
function [q,new_count]=adapt(f,a,b,ep,count);

% if count was not passed in, initialize it to zero
%
if( nargin < 5 )
    count = 0;
end

new_count = count + 9;           %% 1

sab = S(f,a,b);                 %% 2

c   = 0.5*(a+b);                %% 3
sac = S(f,a,c);                 %%
scb = S(f,c,b);                 %%

factor = 15;                     %% 4

if( abs(sac+scb-sab) < factor*ep ) %% 5
    q = sac + scb;               %%
else
    [q1,new_count]=adapt(f,a,c,ep/2,new_count); %% 6
    [q2,new_count]=adapt(f,c,b,ep/2,new_count); %%
    q =q1+q2;                    %%
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Simpson's Rule
% input: f      - integrand, scalar function that takes one input
%          a,b   - integration is over [a,b]
%
% output: q      - simpson's rule approximation to integral of f over [a,b]
%
function q=S(f,a,b);
q = (b-a)/6*( feval(f,a) + 4*feval(f,(a+b)/2) + feval(f,b));

```