Numerical Analysis – Lab 10

1 Adaptive quadrature rules (optional)

Adaptive Simpson's Rule. We give the following implementation.

```
function [q,e,x]=adapsimpson(f,a,b,tol)
%ADAPSIMPSON integrates a function on a bounded interval to a given accuracy
%  using the adaptive Simpson's rule.
%  [q,e,x] = adapmid(f,a,b) returns also the estimated error e and points x

c = (a+b)/2; c1=(a+c)/2; c2=(b+c)/2;
Sab = 1/6*(b-a)*(f(a)+4*f(c)+f(b));
Sac = 1/6*(c-a)*(f(a)+4*f(c1)+f(c));
Scb = 1/6*(b-c)*(f(c)+4*f(c2)+f(b));
e = abs(Sab - Sac - Scb)/15;
if e < tol
   q = Sac + Scb; x = [a;c1;c;c2;b];
   return;
else
    [qac, eac, xac] = adapsimpson(f,a,c,tol/2);
    [qcb, ecb, xcb] = adapsimpson(f,c,b,tol/2);
    q = qac + qcb;
    e = eac + ecb;
    x = [xac(1:(end-1)); xcb];
end
```

Let us try it for approximating some integrals. The first example has the exact value 2/3 so that we compare the estimated error with the actual error.

```
f = @(x) sqrt(x);
[q,e,x] = adapsimpson(f,0,1,1e-10);
format long;
disp('quadrature value: ');
disp(q);
disp('estimated error: ');
disp(e);
disp('actual error: ');
disp(abs(2/3-q));
disp('number of quadrature points: ');
disp(length(x));
```

After running the program, we can see that the error estimator is very close to the actual error. 985 quadrature points are used. For the next example we do not have the exact value. We modify the previous program with the first line to

```
f = @(x) 1+sin(exp(3*x));
```

and the two lines for the actual error commented out. After running the program, it can be seen that the approximate integral is 1.202041491138604, the estimated error is 4.321493192621279e-11 and 2121 quadrature points are used.

2 Gaussian quadrature rules

Use of Gaussian quadrature. As we have learned in the lecture, the Gaussian quadrature with the degree of precision $2m-1$ corresponds to using the roots of degree m Legendre polynomial as the quadrature points, and the quadrature weights are the integral values of the Lagrange basis polynomials associated with the quadrature points. The question now is to find the roots and calculate the weights which may be done by hand when m is small but quickly becomes intractable by analytic methods. Since the quadrature rule is going to be used for numerical purpose, we are also happy with numerical values of the quadrature points and weights within machine precision. To date, the fastest method for getting that was proposed by I. Bogaert in *Iteration-free computation of Gauss–Legendre quadrature nodes and weights*, who also provided a C++ program:

`https://sourceforge.net/projects/fastgausslegendrequadrature/`

A MATLAB implementation can be found:

`https://people.sc.fsu.edu/~jburkardt/m_src/fastgl/fastgl.html`

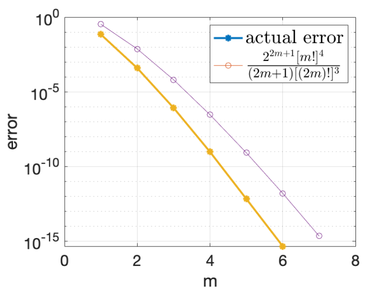
In this lab, we shall use the MATLAB program downloaded from the previous webpage. Based on that, we have the following programs. The first example has the exact integral value $\mathrm{e}-1$.

```
function q = gauss(f,a,b,m)
q = 0;
for k = 1:m
    [~, w, x] = glpair(m,k);
    x = (x*(b-a)+a+b)/2; % [-1,1]->[a,b]
    q = q + w*f(x);
end % function file ends
```

```
f = @(x) exp(x); I = exp(1)-1;
m = 3; % number of quadrature points
a = 0; b = 1; % interval [a, b]
q = gauss(f,a,b,m); format long
disp('quadrature value: '); disp(q);
disp('exact value: ');  disp(I);
disp('actual error: '); disp(abs(I-q));
```

You can try to increase the number of quadrature points m and see how the error decreases. In the following, we plotted the error against m, which shows clearly that the error decreases at least exponentially with m.

```
mrange = 1:7;
e = zeros(numel(mrange),1);
for m = mrange
    q = gauss(f,a,b,m);
    e(m) = abs(I-q);
end
semilogy(mrange,e,'*-');
set(gca,'fontsize',16); grid on;
xlabel('m'); ylabel('error');
hold on;
```



This actually corroborates the theoretical error estimate (not given in our lecture, **optional**)

$$R_m = \frac{2^{2m+1}[m!]^4}{(2m+1)[(2m)!]^3} f^{(2m)}(c)$$

see also `https://encyclopediaofmath.org/wiki/Gauss_quadrature_formula`

Let us try another example. We need only to modify the first two lines with

```
f = @(x) 1 + sin(exp(3*x));
I = gauss(f,a,b,2000);
```

Since the integral cannot be found in closed-form, we use the result with a very large $m = 2000$ as the exact value. We can see that the error still decreases exponentially in m, like e^{-km}, though with a much smaller constant k than before. Do you know why?