

## Gauss Quadrature

Mth 351 Aug 5 1999

Bent E. Petersen

Filename: gauss\_quadrature\_proc.mws

This worksheet introduces some Maple procedures that allow you to experiment with Gauss quadrature on any interval for any number of nodes.

The nodes in Gauss quadrature are the roots of a Legendre polynomial. These can be found to any desired degree of accuracy by using the orthopoly package.

The weights in Gauss quadrature are found by solving a system of linear equations, after the nodes are found. This system is not well conditioned however. Since Gauss quadrature with  $n$  nodes is exact for polynomials of degree up to  $2n-1$  we can find the weights by integrating suitable interpolation polynomials. This method may give better results than solving the system of linear equations directly, but keep in mind finding interpolation polynomials is not all that well conditioned either.

We begin by writing a procedure to calculate the nodes and weights for the interval  $[-1,1]$ . Since Maple remembers previously evaluated expressions we will not worry about multiple evaluations in our code. This simplifies it somewhat. The result is returned as a list of lists, [Gnodes, Gweights].

```
> gauss:=proc(n,d) # n = number of nodes, d = extra precision
> local Gnodes,Gweights,uu,k,u,q,w;
> Digits:=Digits+d;
> Gnodes:=fsolve(orthopoly[P](n,x),x); # P = Legendre
> Gweights:=[]; # initially empty
> uu:=seq(0,k=1..n);
> for k from 1 to n do
>   u:=subsop(k=1,uu); # make kth ordinate 1
>   q:=interp(Gnodes,u,x); # interpolation polynomial
>   w:=int(q,x=-1..1); # integrate to get weight
>   Gweights:=op(Gweights),w; # push onto list of weights
> od;
> Digits:=Digits-d;
> evalf([Gnodes,Gweights]);
> end;
```

Let's test this routine before moving on.

```
> gauss(5,2);
[[-.9061798459, -.5384693101, 0, .5384693101, .9061798459],
 [.2369268851, .4786286705, .5688888889, .4786286705, .2369268851]]
```

That looks alright! Note if you have many nodes you may want to use a larger value for  $d$  in calculating the nodes and weights since calculating the weights is not very well conditioned.

We now write a routine `gint()` to do the actual Gauss quadrature to estimate the integral of a function on an interval  $[a,b]$ . Note `gint()` will generally not produce much error unless the nodes and weights are way off, or the function to be integrated is pretty wild.

```
> gint:=proc(f,a,b,gnw) # gnw list of Gauss nodes and weights
> local ords,g,Gnodes,Gweights;
> g:=t->((b-a)/2)*f(((b-a)*t+(b+a))/2); # rescale to [-1,1]
> Gnodes:=gnw[1];
> Gweights:=gnw[2];
> ords:=map(g,Gnodes); # ordinates
> evalf(linalg[dotprod](ords,Gweights,`orthogonal`));
> end;
```

We now compare Gauss method with Simpson's rule. In order to have Simpson's rule available we must load the student package.

```
> with(student):
Warning, new definition for D
```

Here is a test routine to ease doing our comparisons. We round the errors to 6 significant decimal digits.

```
> gtest:=proc(f,a,b,n)
> local integ,serr,gerr,gwn;
> integ:=evalf(Int(f(x),x=a..b)): # Maple's numeric quadrature
> serr:=integ-evalf(simpson(f(x),x=a..b,n)):
> gwn:=gauss(n,2); gerr:=integ-gint(f,a,b,gwn):
> `integral` = evalf(integ,6), `Simpson error` = evalf(serr,6),
> `Gauss error` = evalf(gerr,6);
> end;
```

Now let's do some actual tests.

```
> Digits:=16: gtest(sin,0,Pi,4);
      integral = 2.00000, Simpson error = -.00455975, Gauss error = .0000157715
> gtest(exp,-1,5,6);
      integral = 148.045, Simpson error = -.733887, Gauss error = .0000212785
> gtest(exp,-1,5,6);
      integral = 148.045, Simpson error = -.733887, Gauss error = .0000212785
> gtest(exp,-1,5,8);
```

```

    integral = 148.045, Simpson error = -.243783, Gauss error = .24036 10-8
> gtest(exp,-1,5,12);
    integral = 148.045, Simpson error = -.0499139, Gauss error = -.1 10-12
> Digits:=20: gtest(exp,-1,5,12);
    integral = 148.045, Simpson error = -.0499139, Gauss error = .9 10-16
> f:=x->1/(1+x^8);
    f := x →  $\frac{1}{1+x^8}$ 
> Digits:=16: gtest(f,-1,1,2);
    integral = 1.84930, Simpson error = .182637, Gauss error = -.126306
> gtest(f,-1,1,4);
    integral = 1.84930, Simpson error = .0211581, Gauss error = .0110689
> gtest(f,-1,1,8);
    integral = 1.84930, Simpson error = -.00538780, Gauss error = -.0000106001
> gtest(f,-1,1,12);
    integral = 1.84930, Simpson error = -.00117206, Gauss error = -.116830 10-5
> gtest(f,-1,1,20);
    integral = 1.84930, Simpson error = -.0000798923, Gauss error = .67 10-13

```

Note Maple is of the opinion that the last Gauss error has only 2 significant figures. Estimates of significant figures are not all that reliable, so the error could be bogus. Let's check at higher precision.

```

> Digits:=17: gtest(f,-1,1,20);
    integral = 1.84930, Simpson error = -.0000798923, Gauss error = -.148998 10-10
> Digits:=18: gtest(f,-1,1,20);
    integral = 1.84930, Simpson error = -.0000798923, Gauss error = -.303857 10-10
> Digits:=19: gtest(f,-1,1,20);
    integral = 1.84930, Simpson error = -.0000798923, Gauss error = -.287553 10-10
> Digits:=28: gtest(f,-1,1,20);
    integral = 1.84930, Simpson error = -.0000798923, Gauss error = -.288580 10-10

```

The conclusion we make is that Gauss quadrature is remarkably accurate (in the cases we tested) but in order to obtain the potential accuracy we must use very high precision in computing the nodes and weights if we have many nodes.

Gauss quadrature also works when we have integrable end-point singularities. In this case the error tends to be larger, but still surprisingly good.

```

> Int(log(x),x=0..1): %=value(%);

```

$$\int_0^1 \ln(x) dx = -1$$

```
> Digits:=16:
```

```
> gnw:=gauss(6,2): gint(log,0,1,gnw); # 6 nodes
```

```
-9849912102623440
```

```
> gnw:=gauss(10,2): gint(log,0,1,gnw); # 10 nodes
```

```
-9942637022162118
```

Here's another example.

```
> Digits:=16:
```

```
> f:=x->x^(-1/2): Int(f(x),x=0..2): %=value(%); evalf(lhs(%));
```

$$\int_0^2 \frac{1}{\sqrt{x}} dx = 2\sqrt{2}$$

```
2.828427124746190
```

```
> gnw:=gauss(10,2): gint(f,0,2,gnw); # 10 nodes
```

```
2.711137826725326
```

```
> gnw:=gauss(20,2): gint(f,0,2,gnw); # 20 nodes
```

```
2.768359173223566
```

```
> gnw:=gauss(40,2): gint(f,0,2,gnw); # 40 nodes
```

```
2.136736116949866
```

```
> Digits:=8:
```

```
> gnw:=gauss(10,2): gint(f,0,2,gnw); # 10 nodes
```

```
2.7111383
```

```
> gnw:=gauss(20,2): gint(f,0,2,gnw); # 20 nodes
```

```
2.7848108
```

```
> gnw:=gauss(40,2): gint(f,0,2,gnw); # 40 nodes
```

```
.97150749 1015
```

The last example above is a disaster. It illustrates how a large number of nodes together with low precision can lead to total nonsense - we are off by 15 orders of magnitude! The error here probably comes from the calculation of the weights - it might be fun to track it down. Consider that a challenge!

Of course one has to compute the nodes and weights only once, for a given number of nodes. However, if you want to use Gauss quadrature once in a while, and just compute the nodes and weights on the fly, then you had better restrict yourself to 10 or 20 nodes at most if you are using normal precision (say 16 digits).

It should now be clear why, in spite of the capabilities of modern computers, tables of Gauss nodes and weights are still published (e.g., by the National Bureau of Standards) and used.