

Gauss Quadrature

Mth 351 Aug 5 2002 Maple 6

Bent E. Petersen

Filename: 351u2002_gauss_quadrature.mws

```
> restart;
```

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. Since Gauss quadrature with n nodes is exact for polynomials of degree up to $2n-1$ we can also find the weights by integrating suitable interpolation polynomials. The second procedure is the one we will use.

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, $[Gnodes, Gweights]$.

Note we use an optional parameter to specify the precision for intermediate calculations. The answer however is rounded to "Digits" decimal digits. Note also if Digits is changed inside a procedure it reverts to its previous value when the procedure returns.

```
> gauss:=proc(n) # n is number of nodes
> local Gnodes,Gweights,uu,k,u,q,w;
> if nargs > 1 then
>   Digits:=args[2];
> fi;
> Gnodes:=[fsolve(orthopoly[P](n,x),x)]; # P = Legendre
> Gweights:=[];
> uu:= [seq(0,k=1..n)];
> for k from 1 to n do
>   u:=subsop(k=1.0,uu); # make kth ordinate 1.0
>   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;
> evalf([Gnodes,Gweights]);
> [Gnodes,Gweights];
```

```
> end:
```

Let's test this routine before moving on.

```
> gauss (5) ;  
[[-.9061798459, -.5384693101, 0., .5384693101, .9061798459],  
 [.2369268858, .4786286687, .5688888891, .4786286699, .2369268851]]
```

That looks alright! Let's try specifying the precision.

```
> gauss (5, 4) ;  
[[-.9062, -.5385, 0., .5385, .9062], [.2386, .4775, .5689, .4783, .2371]]
```

We now write a routine `gint()` to do the actual Gauss quadrature to estimate the integral of a function on an interval `[a,b]`. Again we use an optional parameter to specify the precision.

```
> gint:=proc(f,R,gnw) # gnw list of Gauss nodes and weights  
> local a,b,ords,g,Gnodes,Gweights,k,ans;  
> a:=lhs(R); b:=rhs(R);  
> g:=t->((b-a)/2)*f((b-a)*t+(b+a)/2); # rescale to [-1,1]  
> if nargs > 3 then  
> Digits:=args[4];  
> fi;  
> Gnodes:=gnw[1];  
> Gweights:=gnw[2];  
> ords:=map(g,Gnodes);  
> ans:=0;  
> for k from 1 to nops(Gweights) do  
> ans:=ans + ords[k]*Gweights[k];  
> od;  
> evalf(ans);  
> end:
```

Here's a quick test.

```
> Int(sin(x),x=0..Pi):% = evalf(%); Gauss =  
gint(sin,0..Pi,gauss(4));
```

$$\int_0^{\pi} \sin(x) dx = 2.$$

Gauss = 1.999984228

That's pretty impressive for just 4 nodes.

We now compare Gauss method with Simpson's rule. In order to have Simpson's rule available we must load the student package, or at least `simpson()` command

```
> with(student, simpson):
```

Here is a test routine to ease doing our comparisons. Note n is the number of nodes in the Gauss quadrature and n is the number of subintervals in Simpson's rule so we really have $n+1$ nodes and n must be even. We round the errors to 6 significant decimal digits.

```
> gtest:=proc(f,R,n)
> local integ,serr,gerr,gwn;
> integ:=evalf(Int(f(x),x=R)): # Maple's numeric quadrature
> serr:=integ-evalf(simpson(f(x),x=R,n)):
> gwn:=gauss(n,Digits+2); gerr:=integ-gint(f,R,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,2);
      integral = 2.00000, Simpson error = -.0943951, Gauss error = .0641804
> gtest(sin,0..Pi,4);
      integral = 2.00000, Simpson error = -.00455975, Gauss error = .0000157715
> gtest(sin,0..Pi,6);
      integral = 2.00000, Simpson error = -.000863190, Gauss error = .522729 10-9
> Digits:=24: gtest(sin,0..Pi,8);
      integral = 2.00000, Simpson error = -.000269170, Gauss error = .463957 10-14
> gtest(sin,0..Pi,10);
      integral = 2.00000, Simpson error = -.000109517, Gauss error = .1529 10-19
> Digits:=30: gtest(sin,0..Pi,12);
      integral = 2.00000, Simpson error = -.0000526243, Gauss error = .2306 10-25
```

We see that Gauss quadrature is remarkably accurate (in the cases we tested above). However in order to obtain the potential accuracy we must use very high precision in computing the nodes and weights if we have many nodes. Otherwise we may see a great deal of loss of significance (roundoff).

Error estimates for Gauss quadrature suggest that smooth (many times continuously differentiable) integrands will yield the best results.

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
```

Note we can not use Simpson's rule (directly) here since the logarithm has a singularity at the origin.

Here's another example.

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

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

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

$$2\sqrt{2} = 2.82842712474619009760337744842$$

```
> gnw:=gauss(10): gint(f,0..2,gnw); # 10 nodes  
2.71113782672533069701473621114
```

```
> gnw:=gauss(20): gint(f,0..2,gnw); # 20 nodes  
2.76835917344199410134342548135
```

```
> gnw:=gauss(40): gint(f,0..2,gnw); # 40 nodes  
2.79802329999083511831390920549
```

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

```
> gnw:=gauss(10): gint(f,0..2,gnw); # 10 nodes  
2.7111520
```

```
> gnw:=gauss(20): gint(f,0..2,gnw); # 20 nodes  
-.59288221
```

```
> gnw:=gauss(40): gint(f,0..2,gnw); # 40 nodes  
-.18106783 1019
```

The last two examples above are a disaster. They illustrates how a large number of nodes together with low precision can lead to total nonsense - we are off by 19 orders of magnitude in the last example!

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, high precision tables of Gauss nodes and weights are still published (e.g., by the National Bureau of Standards) and used.