Math/CS 467/667 **Programming Assignment 01**

Adaptive Gauss Quadrature

1. Find an orthogonal polynomial p_4 of degree 4 such that

$$\int_{-1}^{1} q(x)p_4(x) = 0$$

for every polynomial $q(x)$ of degree 3 or less. You may use Maple and the Gram–Schmidt process as done in class. Alternatively, use your differential equation skills to find a polynomial solution to the differential equation

$$(1-x^2)y'' - 2xy' + 20y = 0.$$

The Maple worksheet

```
restart;
kernelopts(printbytes=false);
dp:=(f,g)->int(f*g,x=-1..1);
nm:=f->sqrt(dp(f,f));
N:=5;
U:=[seq(x^k,k=0..N-1)];
for i from 1 to N
do
   tV[i]:=U[i];
   for j from 1 to i-1
   do
      tV[i]:=tV[i]-V[j]*dp(V[j],U[i]);
   od;
   V[i]:=tV[i]/nm(tV[i]);
od;
latex(V[5],"p4.tex");
with(CodeGeneration);
codeoptions:=declare=[x::double],optimize=true,output="p4.i";
a:=coeff(V[5],x^4);
t1:=simplify(V[5]/a);
p4:=unapply(t1,x);
C(p4,codeoptions);
dp4:=unapply(diff(t1,x),x);
C(dp4,codeoptions);
```

produces the output

```
    |\^/|     Maple 9.5 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2004
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
```

```
      |       Type ? for help.
> restart;
> kernelopts(printbytes=false);
                                      true

> dp:=(f,g)->int(f*g,x=-1..1);
                                             1
                                            /
                                           |
                          dp := (f, g) ->  |   f g dx
                                           |
                                          /
                                            -1

> nm:=f->sqrt(dp(f,f));
                           nm := f -> sqrt(dp(f, f))

> N:=5;
                                     N := 5

> U:=[seq(x^k,k=0..N-1)];
                                         2   3   4
                            U := [1, x, x , x , x ]

> for i from 1 to N
> do
>    tV[i]:=U[i];
>    for j from 1 to i-1
>    do
>       tV[i]:=tV[i]-V[j]*dp(V[j],U[i]);
>    od;
>    V[i]:=tV[i]/nm(tV[i]);
> od;
                                  tV[1] := 1

                                          1/2
                                         2
                                 V[1] := ----
                                          2

                                  tV[2] := x

                                             1/2
                                         x 6
```

```
                   V[2] := ------
                             2

                              2
                    tV[3] := x

                         2          1/2
                     3 (x  - 1/3) 10
             V[3] := -----------------
                             4

                              3
                    tV[4] := x

                        3            1/2
                    5 (x  - 3/5 x) 14
            V[4] := --------------------
                             4

                              4
                    tV[5] := x

                     4               2   1/2
                105 (x  + 3/35 - 6/7 x ) 2
      V[5] := -----------------------------
                            16

> latex(V[5],"p4.tex");
> with(CodeGeneration);
Warning, the protected name Matlab has been redefined and unprotected
[C, Fortran, IntermediateCode, Java, LanguageDefinition, Matlab, Names, Save,

    Translate, VisualBasic]

> codeoptions:=declare=[x::double],optimize=true,output="p4.i";
     codeoptions := declare = [x::double], optimize = true, output = "p4.i"

> a:=coeff(V[5],x^4);
                                  1/2
                             105 2
                        a := --------
                               16

> t1:=simplify(V[5]/a);
```

```
                         4              2
        t1 := x  + 3/35 - 6/7 x

> p4:=unapply(t1,x);
                              4              2
                p4 := x -> x  + 3/35 - 6/7 x

> C(p4,codeoptions);
Warning, cannot translate type double, using default type
Warning, procedure/module options ignored
> dp4:=unapply(diff(t1,x),x);
                                    3
                dp4 := x -> 4 x  - 12/7 x

> C(dp4,codeoptions);
Warning, cannot translate type double, using default type
Warning, procedure/module options ignored
> quit
bytes used=2414488, alloc=2031244, time=0.08
```

and uses the Gram–Schmidt process to obtain

$$p_4(x) = \frac{105}{16}\left(x^4 + \frac{3}{35} - 6/7\,x^2\right)\sqrt{2}$$

Note that line 16 in the Maple script converts to polynomial to latex format and writes the output into the file `p4.tex` for inclusion in this document while lines 22 and 24 create the file `p4.i` containing C code for use in the next problem. Since the roots are the same for any rescaled versions of the polynomial, we divide out by the coefficient of x^4 for convenience to obtain a monic polynomial for use in the C code.

Before continuing to the next problem, let's also find the fourth degree orthogonal polynomial using the differential equation. As we know the solution will be a fourth degree polynomial we substitute $y = ax^4 + bx^3 + cx^2 + dx + e$ into the differential equation and solve for the coefficients. Since

$$y' = 4ax^3 + 3bx^2 + 2cx + d \qquad \text{and} \qquad y'' = 12ax^2 + 6bx + 2c,$$

we obtain

$$\begin{aligned}(1-x^2)y'' &= -12ax^4 - 6bx^3 + (12a-2c)x^2 + 6bx + 2c\\ -2xy' &= -8ax^4 - 6bx^3 - 4cx^2 - 2dx\\ 20y &= 20ax^4 + 20bx^3 + 20cx^2 + 20dx + 20e\\ \hline 0 &= 8bx^3 + (12a+14c)x^2 + (6b+18d)x + 2c + 20e\end{aligned}$$

It follows that

$$8b = 0, \qquad 12a + 14c = 0, \qquad 6b + 18d = 0 \qquad \text{and} \qquad 2c + 20e = 0.$$

We immediately deduce from the first and third equations that $b = 0$ and $d = 0$. This leaves two equations in three unknowns. Setting $a = 1$ arbitrarily yields $c = -6/7$ and $e = 3/35$. The resulting orthogonal polynomial is

$$x^4 + \frac{-6}{7}x^2 + \frac{3}{35}.$$

Note that, aside from the normalization $105\sqrt{2}/16$, this is the same polynomial as found using the Gram–Schmidt process.

2. The roots of $p_4(x)$ are real and lie in the interval $[-1, 1]$. Use Newton's method with suitable starting points to find all four roots x_0, x_1, x_2 and x_3 as accurately as possible. Compute the residuals and the derivatives

$$p_4(x_j) \quad \text{and} \quad p_4'(x_j) \qquad \text{for} \qquad j = 1, 2, 3, 4$$

and comment on the accuracy of your roots.

The C code for the polynomial p_4 and its derivative generated in the previous step is

```
double p4 (double x)
{
  double t2;
  double t3;
  t2 = x * x;
  t3 = t2 * t2;
  return(t3 + 0.3e1 / 0.35e2 - 0.6e1 / 0.7e1 * t2);
}
double dp4 (double x)
{
  double t2;
  t2 = x * x;
  return(0.4e1 * t2 * x - 0.12e2 / 0.7e1 * x);
}
```

We now include these functions into a C program to compute the roots of the polynomial using Newton's method. The C code

```
#include <stdio.h>
#include <math.h>

#include "p4.i"

double g(double x){
    return x-p4(x)/dp4(x);
}

double guess[] = {-.7,-.3,.3,.7};
#define nguess (sizeof(guess)/sizeof(double))

int main(){
    printf("double roots[]={\n");
    for(int i=0;i<nguess;i++){
        double x=guess[i];
        for(int k=0;k<20;k++) x=g(x);
        printf("\t%24.15e%c // p4(x)=%g\n",
            x,i<nguess-1?',':' ',p4(x));
```

```
    }
    printf("};\n");
    return 0;
}
```

produces the output

```
double roots[]={
      -8.611363115940526e-01, // p4(x)=5.51317e-17
      -3.399810435848563e-01, // p4(x)=1.31798e-17
       3.399810435848563e-01, // p4(x)=1.31798e-17
       8.611363115940526e-01  // p4(x)=5.51317e-17
};
```

In order to comment on the accuracy of our solutions we have computed the residual by plugging each of the roots into p_4 to see how close the result is to 0. In each of the cases the residual is on the order 10^{-17}, which indicates that the solutions are accurate.

3. Find weights w_k for $k = 0, 1, 2, 3$ such that

$$\int_{-1}^{1} x^j \, dx = \sum_{k=0}^{3} w_k x_k^j \qquad \text{for} \qquad j = 0, 1, 2, 3.$$

Verify that

$$\int_{-1}^{1} x^j \, dx = \sum_{k=0}^{3} w_k x_k^j \qquad \text{for} \qquad j = 4, 5, 6, 7.$$

The Maple script

```
restart;
Digits:=15;
kernelopts(printbytes=false);
r:=[-8.611363115940526e-01,
    -3.399810435848563e-01,
    3.399810435848563e-01,
    8.611363115940526e-01];
n:=nops(r);
approx:=add(w[k]*f(r[k]),k=1..n);
eq:=int(f(x),x=-1..1)=approx;
eqf:=unapply(eq,f);
eqs:={seq(eqf(x->x^k),k=0..n-1)};
vbls:=seq(w[k],k=1..n);
t1:=solve(eqs,{vbls});
t2:=subs(t1,[vbls]);
with(CodeGeneration);
C(t2,resultname="weights",output="w.i");
t3:=subs(t1,rhs(eq))-lhs(eq);
err4:=unapply(t3,f);
seq([j,err4(x->x^j)],j=n..2*n);
```

with output

```
    |\^/|     Maple 9.5 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2004
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> restart;
> Digits:=15;
                                  Digits := 15

> kernelopts(printbytes=false);
                                      true
```

```
> r:=[-8.611363115940526e-01,
>   -3.399810435848563e-01,
>   3.399810435848563e-01,
>   8.611363115940526e-01];
r := [-0.8611363115940526, -0.3399810435848563, 0.3399810435848563,

    0.8611363115940526]

> n:=nops(r);
                                     n := 4

> approx:=add(w[k]*f(r[k]),k=1..n);
approx := w[1] f(-0.8611363115940526) + w[2] f(-0.3399810435848563)

     + w[3] f(0.3399810435848563) + w[4] f(0.8611363115940526)

> eq:=int(f(x),x=-1..1)=approx;
         1
        /
       |
eq :=  |   f(x) dx = w[1] f(-0.8611363115940526) + w[2] f(-0.3399810435848563)
       |
      /
        -1

     + w[3] f(0.3399810435848563) + w[4] f(0.8611363115940526)

> eqf:=unapply(eq,f);
               1
              /
             |
eqf := f ->  |   f(x) dx = w[1] f(-0.8611363115940526)
             |
            /
              -1

     + w[2] f(-0.3399810435848563) + w[3] f(0.3399810435848563)

     + w[4] f(0.8611363115940526)

> eqs:={seq(eqf(x->x^k),k=0..n-1)};
eqs := {2 = 1. w[1] + 1. w[2] + 1. w[3] + 1. w[4], 0 = -0.8611363115940526 w[1]

     - 0.3399810435848563 w[2] + 0.3399810435848563 w[3]
```

```
     + 0.8611363115940526 w[4], 2/3 = 0.741555747145810 w[1]

     + 0.115587109997048 w[2] + 0.115587109997048 w[3] + 0.741555747145810 w[4]

    , 0 = -0.638580580938515 w[1] - 0.0392974262817538 w[2]

     + 0.0392974262817538 w[3] + 0.638580580938515 w[4]}

> vbls:=seq(w[k],k=1..n);
                         vbls := w[1], w[2], w[3], w[4]

> t1:=solve(eqs,{vbls});
t1 := {w[4] = 0.347854845137453, w[3] = 0.652145154862547,

    w[2] = 0.652145154862547, w[1] = 0.347854845137453}

> t2:=subs(t1,[vbls]);
t2 := [

    0.347854845137453, 0.652145154862547, 0.652145154862547, 0.347854845137453]

> with(CodeGeneration);
Warning, the protected name Matlab has been redefined and unprotected
[C, Fortran, IntermediateCode, Java, LanguageDefinition, Matlab, Names, Save,

    Translate, VisualBasic]

> C(t2,resultname="weights",output="w.i");
> t3:=subs(t1,rhs(eq))-lhs(eq);
t3 := 0.347854845137453 f(-0.8611363115940526)

     + 0.652145154862547 f(-0.3399810435848563)

     + 0.652145154862547 f(0.3399810435848563)

                                                    1
                                                   /
                                                  |
     + 0.347854845137453 f(0.8611363115940526) -  |   f(x) dx
                                                  |
                                                 /
                                                   -1
```

```
> err4:=unapply(t3,f);
err4 := f -> 0.347854845137453 f(-0.8611363115940526)

     + 0.652145154862547 f(-0.3399810435848563)

     + 0.652145154862547 f(0.3399810435848563)

                                                    1
                                                   /
                                                  |
     + 0.347854845137453 f(0.8611363115940526) -  |   f(x) dx
                                                  |
                                                 /
                                                   -1

> seq([j,err4(x->x^j)],j=n..2*n);
                 -14
      [4, -0.2 10   ], [5, 0.], [6, 0.], [7, 0.], [8, -0.011609977324264]

> quit
bytes used=2611740, alloc=2096768, time=0.07
```

creates the C include file which displays the weights

```
weights[0] = 0.347854845137453e0;
weights[1] = 0.652145154862547e0;
weights[2] = 0.652145154862547e0;
weights[3] = 0.347854845137453e0;
```

and also computes the error in the resulting quadrature formula against x^j for $j = 4, 5, 6, 7$ as well as $j = 8$. The output from line 20 indicates that $j = 4$ has a residual error of -0.2×10^{-14} while $j = 5, 6, 7$ the residual error is exactly zero. Even the $j = 4$ value is close enough to zero to verify that the quadrature formula is exact for $j = 4, 5, 6, 7$. Unsurprisingly, there is significant error of about -0.0116 when $j = 8$.

4. Prove the equality

$$\int_a^b f(t)dt = \frac{b-a}{2}\int_{-1}^{1} f\Big(a + \frac{b-a}{2}(x+1)\Big)dx.$$

This is a change of variables or u-substitution such that

$$t = a + \frac{b-a}{2}(x+1) \qquad \text{with} \qquad dt = \frac{b-a}{2}dx.$$

The limits of integration are correct since

$$t\Big|_{x=-1} = a + \frac{b-a}{2}(-1+1) = a$$

and

$$t\Big|_{x=1} = a + \frac{b-a}{2}(1+1) = a + b - a = b.$$

5. Define

$$G_4(a,b,f)=\frac{b-a}{2}\sum_{k=0}^{3} w_k f\Big(a+\frac{b-a}{2}(x_k+1)\Big)$$

We know from the verification in question 3 as well as the general theory of Gauss quadrature that

$$\Big|\int_a^b f(t)dt-G_4(a,b,f)\Big|=\mathcal{O}\big((b-a)^9\big)\qquad\text{as}\qquad b-a\to 0.$$

Let $c=(a+b)/2$ and use Richardson extrapolation to find α and β such that

$$R(a,b,f)=\alpha G_4(a,b,f)+\beta\big(G_4(a,c,f)+G_4(c,b,f)\big)$$

satisfies

$$\Big|\int_a^b f(t)dt-R(a,b,f)\Big|=\mathcal{O}\big((b-a)^{10}\big)\qquad\text{as}\qquad b-a\to 0.$$

Suppose that K is the constant so that

$$\int_a^b f(t)dt-G_4(a,b,f)\approx K(b-a)^9$$

Then

$$\begin{aligned}\int_a^b f(t)dt-R(a,b,f)&=\int_a^b f(t)dt-\alpha G_4(a,b,f)-\beta\big(G_4(a,c,f)+G_4(c,b,f)\big)\\&=\alpha\Big(\int_a^b f(t)dt-G_4(a,b,f)\Big)\\&\qquad+\beta\Big(\int_a^c f(t)dt-G_4(a,c,f)\Big)+\beta\Big(\int_c^b f(t)dt-G_4(c,b,f)\Big)\\&\qquad\qquad+(1-\alpha-\beta)\int_a^b f(t)dt\\&=\alpha K(a-b)^9+\beta K(a-c)^9+\beta K(c-b)^9+(1-\alpha-\beta)\int_a^b f(t)dt.\end{aligned}$$

To make the last term vanish we require $\alpha+\beta=1$. Since

$$a-c=a-(a+b)/2=(a-b)/2\quad\text{and}\quad c-b=(a+b)/2-b=(a-b)/2$$

it follows that

$$\int_a^b f(t)dt-R(a,b,f)=\alpha K(a-b)^9+2(1-\alpha)K(a-b)^9/2^9.$$

The right side of the above equality vanishes when

$$\alpha+(1-\alpha)/2^8=0\qquad\text{or equivalently}\qquad\alpha=\frac{-1}{2^8-1}=\frac{-1}{255}.$$

Therefore $\alpha=-1/255$ and $\beta=256/255$.

6. Consider the adaptive quadrature rule given by

$$Q(a,b,f,\varepsilon) = \begin{cases} R(a,b,f) & \text{if } \left|G_4(a,b,f) - R(a,b,f)\right| < \varepsilon \\ Q(a,c,f,\varepsilon/2) & \\ \quad +Q(c,b,f,\varepsilon/2) & \text{otherwise} \end{cases}$$

and use this rule to approximate the improper integral

$$\int_0^1 f(t)dt \qquad \text{where} \qquad f(t) = \frac{1}{t}e^{-(\log t)^2}.$$

Since the exact value of the integral is $\sqrt{\pi}/2$ check that your approximation satisfies

$$\text{error} = \left|Q(0,1,f,\varepsilon) - \frac{\sqrt{\pi}}{2}\right| \le \varepsilon \qquad \text{when} \qquad \varepsilon = 10^{-7}.$$

What happens to the above approximation when $\varepsilon = 10^{-p}$ for $p = 8, 9, 10, \ldots 15$?

Recall the include file `r.i` with the roots generated by Newton's method that was created in answer to question 2 of this project given by

```
double roots[]={
      -8.611363115940526e-01, // p4(x)=5.51317e-17
      -3.399810435848563e-01, // p4(x)=1.31798e-17
       3.399810435848563e-01, // p4(x)=1.31798e-17
       8.611363115940526e-01  // p4(x)=5.51317e-17
};
```

and the include file `w.i` with the weights generated by solving the resulting linear systems of equations in question 3 given by

```
weights[0] = 0.347854845137453e0;
weights[1] = 0.652145154862547e0;
weights[2] = 0.652145154862547e0;
weights[3] = 0.347854845137453e0;
```

These two files are used by the following C code which implements the quadrature rule:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "r.i"
double weights[4];
typedef double (*function)(double x);

double G4(double a,double b,function f){
```

```
    double h=(b-a)/2,r=0;
    for(int k=0;k<4;k++){
        double x=a+h*(roots[k]+1);
        r+=weights[k]*f(x);
    }
    return h*r;
}

double alpha=-1.0/255, beta=256.0/255;
double R(double a,double b,function f){
    double c=(a+b)/2;
    return alpha*G4(a,b,f)+beta*(G4(a,c,f)+G4(c,b,f));
}

double Q(double a,double b,function f,double epsilon){
    double t1=G4(a,b,f),t2=R(a,b,f);
    if(fabs(t1-t2)<epsilon) return t2;
    double c=(a+b)/2;
    return Q(a,c,f,epsilon/2)+Q(c,b,f,epsilon/2);
}

double f(double t){
    double t1=log(t);
    return exp(-t1*t1)/t;
}

int main(){
    #include "w.i"
    printf(
"Adaptive Guassian Quadrature Using Richardson Extrapolation.\n\n");
    printf("%24s %24s\n","roots","weights");
    for(int k=0;k<4;k++){
        printf("%24.15e %24.15e\n",roots[k],weights[k]);
    }
    printf("\n%5s %18s %24s %24s\n","p","epsilon","Q","|Q-sqrt(pi)/2|");
    double exact=sqrt(M_PI)/2;
    for(int p=7;p<16;p++){
        double epsilon=pow(10.0,-(double)p);
        double z=Q(0,1,f,epsilon);
        printf("%5d %18.10e %24.15e %24.15e\n",
            p,epsilon,z,fabs(z-exact));
    }
    return 0;
}
```

Output from this program is

```
Adaptive Guassian Quadrature Using Richardson Extrapolation.

                 roots                    weights
 -8.611363115940526e-01    3.478548451374530e-01
 -3.399810435848563e-01    6.521451548625470e-01
  3.399810435848563e-01    6.521451548625470e-01
  8.611363115940526e-01    3.478548451374530e-01

  p           epsilon                         Q          |Q-sqrt(pi)/2|
  7    1.0000000000e-07    8.862269253240691e-01    1.286888373641659e-10
  8    1.0000000000e-08    8.862269254533175e-01    5.595524044110789e-13
  9    1.0000000000e-09    8.862269254526169e-01    1.409983241273949e-13
 10    1.0000000000e-10    8.862269254527707e-01    1.276756478318930e-14
 11    1.0000000000e-11    8.862269254527578e-01    1.110223024625157e-16
 12    1.0000000000e-12    8.862269254527579e-01    0.000000000000000e+00
 13    1.0000000000e-13    8.862269254527579e-01    0.000000000000000e+00
 14    1.0000000000e-14    8.862269254527579e-01    0.000000000000000e+00
 15    1.0000000000e-15    8.862269254527579e-01    0.000000000000000e+00
```

which indicates the method works for all values of ε. Note that we are lucky in this case that the quadrature method converges to exactly the same value as computed for `sqrt(M_PI)/2` by the built-in functions. I had expected that the program would abort with a stack overflow when the tolerance ε was specified to be less than the machine precision of the floating point variables. I found it interesting that this didn't happen and in retrospect am now not sure that it ever could. It would be interesting to know whether there is a function f and a choice for $\varepsilon > 0$ for which the program fails to run.