Math 128A, Summer 2019

PSET #4 (due Wednesday 7/24/2019)

Problem 1. In class we proved the Euler-Maclaurin summation formula

$$\int_0^1 f(x)\,dx = \frac{1}{2}\left[f(0)+f(1)\right] + \sum_{m=1}^{\infty} b_m \left[f^{(2m-1)}(1) - f^{(2m-1)}(0)\right]$$

for some unknown constants b_m independent of f.

(a) Find a recursive formula for b_m by evaluating both sides for $f(x) = e^{\lambda x}$, where λ is a parameter.

Solution. For $f(x) := e^{\lambda x}$, the above gives

$$\begin{aligned}\int_0^1 e^{\lambda x}\,dx = \frac{e^\lambda - 1}{\lambda} &= \frac{1}{2}\left[e^{\lambda\cdot 0} + e^{\lambda\cdot 1}\right] + \sum_{m+1}^{\infty} b_m \left[\lambda^{2m-1}\cdot e^{1\cdot\lambda} - \lambda^{2m-1}\cdot e^{0\cdot\lambda}\right] \\ &= \frac{1}{2}\left[1+e^\lambda\right] + \sum_{m=1}^{\infty}\left[b_m\lambda^{2m-1}\left(e^\lambda - 1\right)\right] \\ \implies \frac{-\lambda}{e^\lambda - 1} &= -1 + \frac{\lambda}{2} + \sum_{m=1}^{\infty} b_m\lambda^{2m}\end{aligned}$$

As done in lecture on Monday, multiply across to get:

$$\lambda = \left(e^\lambda - 1\right)\left[1 - \frac{\lambda}{2} - \sum_{m=1}^{\infty} b_m\lambda^{2m}\right].$$

Now we write out the Taylor expansion of $(e^t - 1)$, which gives us:

$$\begin{aligned}\lambda &= \left(\sum_{k=0}^{\infty}\frac{\lambda^k}{(k+1)!}\right)\left[1 - \frac{\lambda}{2} - \sum_{m=1}^{\infty} b_m\lambda^{2m}\right] \\ &= \left(1 + \frac{\lambda}{2} + \frac{\lambda^2}{3!} + \cdots\right)\left[1 - \frac{\lambda}{2} - \sum_{m=1}^{\infty} b_m\lambda^{2m}\right]\end{aligned}$$

And define a new series for the RHS expression:

$$\sum_{j=0}^{\infty} a_j\lambda^j := a_0 + a_1\lambda + \sum_{j=1}^{\infty} a_{2j}\lambda^{2j} = 1 - \frac{\lambda}{2} - \sum_{m=1}^{\infty} b_m\lambda^{2m},$$

and we can perform the 'first rule of applied mathematics', according to Strain, which is to swap order of sums to see what it gives. Swapping the summation and dividing through by λ nets us:

$$1 = \sum_{p=0}^{\infty}\sum_{q=0}^{p} a_q\lambda^p \frac{1}{(p-q+1)!}.$$

Notice that when $p = 0$, we have that this double summation equals 1, and the double summation equals 0 when $p \neq 0$. Hence with a_0, a_1 defined, we have:

$$\frac{1}{(2m+1)!} - \frac{1}{2}\cdot\frac{1}{(2m)!} - \sum_{n=1}^{m}\frac{b_n}{[2(m-n)+1]!} = 0.$$

Notice $n := m$ gives a denominator 1!, so pulling that b_m term out of the summation and solving for it, we have:

$$b_m = \frac{1}{(2m+1)!} - \frac{1}{2}\frac{1}{(2m)!} - \sum_{n=1}^{m-1} \frac{b_n}{[2(m-n)+1]!},$$

which is our desired recurrence relation where b_m depends on all b_n with $n =$`1:(m-1)`, with two added terms. This appears correct, as we verify numerically in R (and our steps should be reversible). □

(b) Compute $b_1, b_2, b_3, \ldots, b_{10}$.

Given by Strain, the above results in an infinite system of linear equations for the b_i, so this would be very scary, except that this is a **triangular** system of linear equations. That is, b_0 is determined right away, and b_1 is given by looking at the linear terms, and onwards. Hence we write (wrote above) this neatly as a recurrence relation, with each b_i depending on all $1, 2, \ldots, i-1$.

Solution. First we find the first two Bernoulli coefficients, b_1, b_2, following precisely as Strain suggests. Recall

$$\begin{aligned}\lambda &= \left(1 + \frac{\lambda}{2} + \frac{\lambda^2}{3!} + \cdots\right)\left[1 - \frac{\lambda}{2} - \sum_{m=1}^{\infty} b_m \lambda^{2m}\right] \\ &= \left[\left(1 + \frac{\lambda}{2} + \frac{\lambda^2}{4}\right) - \left(\frac{\lambda}{2} + \frac{\lambda^2}{3!} + \cdots\right) + \cdots\right],\end{aligned}$$

and hence looking at (would-be) linear terms (after dividing out λ), we have

$$\frac{\lambda^2}{3!} - \frac{\lambda^2}{4} + b_1 \lambda^{2\cdot 1} = 0 \cdot \lambda^2 \implies b_1 := \left(\frac{1}{6} - \frac{1}{4}\right) = \frac{-2}{24}.$$

We can continue symbolically by combining terms and using previous b_i; however, Strain mentioned in lectures we are allowed to solve for these Bernoulli coefficients numerically by programming. (Actually, we also could have easily gotten b_1 from the recurrence relation by setting $m := 1$ for our b_m expression.)

□

First we perform some quick sanity checks, and our Bernoulli numbers follow in order $b_1, b_2, \ldots, b_{10}$.

```
> getBern(10)[10] == getBern(200)[10]
[1] TRUE
> getBern(10)
             [,1]         [,2]          [,3]         [,4]          [,5]
[1,] -0.08333333 0.001388889 -3.306878e-05 8.267196e-07 -2.087676e-08
             [,6]          [,7]         [,8]          [,9]         [,10]
[1,] 5.28419e-10 -1.338254e-11 3.38968e-13 -8.586062e-15 2.174869e-16
> getBern(25)[1:10]
 [1] -8.333333e-02  1.388889e-03 -3.306878e-05  8.267196e-07 -2.087676e-08
 [6]  5.284190e-10 -1.338254e-11  3.389680e-13 -8.586062e-15  2.174869e-16
```

The printed values match, and performing element-wise `==` returns true, so we are satisfied. Our code for generating these is very straightforward, simply from the recurrence relation we found above.

```
```{r}
getBern <- function(m) {
  b.init <- function(i) {1/factorial(2*i+1) - 1/(2*factorial(2*i))}
  bvec <- matrix( data = b.init(1:m), nrow = 1 )
  for (k in 2:m) {
    for (n in 1:(k-1)) {
      bvec[k] <- bvec[k] - bvec[n]/factorial(2*(k-n)+1)
    }
  }
  bvec
}
```
```

(c) Compound the formula to show:

$$\int_0^n f(x)\ dx = \frac{1}{2}f(0) + f(1) + f(2) + \cdots + \frac{1}{2}f(n) + \sum_{m=1}^{\infty} b_m \left[f^{(2m-1)}(n) - f^{(2m-1)}(0)\right].$$

Solution. As given in the beginning of the problem statement (and as proven in Lecture), the Euler-Maclaurin summation formula gives:

$$\int_0^1 f(x)\ dx = \frac{1}{2}\left[f(0) + f(1)\right] + \sum_{m=1}^{\infty} b_m \left[f^{(2m-1)}(1) - f^{(2m-1)}(0)\right],$$

for our Bernoulli constants found in part (b) above. Solving for the summation, for interval $[0, 1]$, we have:

$$[0,1] \qquad \sum_{m=1}^{\infty} b_m \left[f^{(2m-1)}(1) - f^{(2m-1)}(0)\right] = \int_0^1 f(x)\ dx - \frac{1}{2}\left[f(0) + f(1)\right],$$

and equivalently, for $[1, 2]$, we have:

$$[1,2] \qquad \sum_{m=1}^{\infty} b_m \left[f^{(2m-1)}(2) - f^{(2m-1)}(1)\right] = \int_1^2 f(x)\ dx - \frac{1}{2}\left[f(1) + f(2)\right],$$

and we continue this for $[0, 1], [1, 2], [2, 3], \ldots, [n-1, n]$. This nets us:

$$[0,1] \cup [1,2] \cup \cdots \cup [n-1,n] \qquad \sum_{k=1}^{n}\sum_{m=1}^{\infty} b_m \left[f^{(2m-1)}(k) - f^{(2m-1)}(k-1)\right] = \sum_{k=1}^{n}\left[\int_{k-1}^{k} f(x)\ dx - \frac{1}{2}\left(f(k-1) + f(k)\right)\right],$$

where the sum of the integral on the RHS is simply $\int_0^n f(x)\ dx$, and the terms on the LHS form a telescoping series, to net:

$$\sum_{m=1}^{\infty} b_m \left[f^{(2m-1)}(n) - f^{(2m-1)}(0)\right] = \int_0^n f(x)\ dx - \left[\frac{1}{2}f(0) + f(1) + f(2) + \cdots + f(n-2) + f(n-1) + \frac{1}{2}f(n)\right],$$

which gives precisely our desired result when solving for $\int_0^n f(x)\ dx$. Recall from above, that is,

$$\int_0^n f(x)\ dx = \frac{1}{2}f(0) + f(1) + f(2) + \cdots + \frac{1}{2}f(n) + \sum_{m=1}^{\infty} b_m \left[f^{(2m-1)}(n) - f^{(2m-1)}(0)\right].$$

□

(d) Use the Euler-Maclaurin formula to show that:

$$\sum_{j=1}^{n} j^k = P_{k+1}(n)$$

is a degree-$(k+1)$ polynomial in n. For example,

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}.$$

Solution. Define $f(x) := x^k$, so that the Euler-Maclaurin summation formula in part (c) gives:

$$\int_0^n x^k \, dx = \left[\frac{1}{2}0^k + 1^k + 2^k + \cdots + (n-1)^k + \frac{1}{2}n^k\right] + \sum_{m=1}^{k} b_m \left[f^{(2m-1)}(n) - f^{(2m-1)}(0)\right],$$

and notice that because f is a degree-k monomial (or polynomial), if $k < (2m-1)$, then the derivatives are exactly zero, but our summation formula goes past $k/2$, up to k, as the summation expressed by sigma notation will include differences of zero. Simplifying across gives:

$$\frac{1}{k+1}n^{k+1} = \underbrace{\frac{1}{2}0^k + 1^k + 2^k + \cdots + (n-1)^k + n^k}_{} + \sum_{m=1}^{k} b_m \left(\frac{k!}{(k-2m+1)!}\left[n^{k-2m+1} - 0^{k-2m+1}\right]\right)$$

We are interested in the underbraced expression as it closely resembles $\sum_{j=1}^{n} j^k$ as we are tasked to show is a degree-$k+1$ polynomial on n. By adding $\frac{0^k}{2} + \frac{n^k}{2}$ across the equation (note that we keep the expression 0^k and do not simplify, because we can possibly have $k := 1$, in which case we define $0^0 := 1$ for our purposes), we precisely have:

$$\sum_{j=0}^{n} j^k = \frac{1}{k+1}n^{k+1} + \frac{n^k}{2} - \sum_{m=1}^{k} b_m \left(\frac{k!}{(k-2m+1)!}\left[n^{k-2m+1} - 0^{k-2m+1}\right]\right),$$

which is surely a degree-$(k+1)$ polynomial in n as was to be shown.

□

(e) Show that the error in the trapezoidal rule satisfies:

$$\int_0^1 f(x)\ dx - h\left(\frac{1}{2}f(0) + f(h) + f(2h) + \cdots + f[(n-1)h] + \frac{1}{2}f[nh]\right) = \sum_{m=1}^{\infty} b_m h^{2m}\left(f^{(2m-1)(1)} - f^{(2m-1)}(0)\right).$$

Solution. Recall that Euler-Maclaurin on Trapezoidal Rule acts on an interval $[0, n]$ with equispaced intervals of size $h := \frac{1}{n}$. The Euler-Maclaurin from earlier gives us:

$$\int_0^n f\left(\frac{x}{n}\right)\ dx = \frac{1}{2}f\left(\frac{0}{n}\right) + f\left(\frac{1}{n}\right) + f\left(\frac{2}{n}\right) + \cdots + \frac{1}{2}f\left(\frac{n-1}{n}\right) + \frac{1}{2}f\left(\frac{n}{n}\right) + \sum_{m=1}^{\infty} b_m\left(f^{(2m-1)}\left(\frac{n}{n}\right) - g^{(2m-1)}(0)\right).$$

Consider the scaling function $g(x) := f(hx)$. Then, in terms of h, our above expression is equivalent to:

$$\int_0^n g(x)\ dx = \frac{1}{2}f(0h) + f(1h) + f(2h) + \cdots + \frac{1}{2}f([n-1]h) + \frac{1}{2}f(nh) + \sum_{m=1}^{\infty} b_m\left(g^{(2m-1)}(n) - g^{(2m-1)}(0)\right).$$

Recall that taking the $(2m-1)$-th derivative of $g(x) = f(hx)$ gives $g^{(2m-1)}(x) = h^{2m-1}f^{2m-1}(hx)$, where h is the constant scaling factor. Hence substituting this into the above, we have:

$$\frac{1}{h}\int_0^1 f(x)\ dx = \frac{1}{2}f(0h) + f(1h) + f(2h) + \cdots + f([n-1]h) + \frac{1}{2}f(nh) + \sum_{m=1}^{\infty} b_m \frac{h^{2m}}{h}\left(f^{(2m-1)}(1) - f^{(2m-1)}(0h)\right).$$

Now multiplying across by h, and rearranging terms, we have:

$$\int_0^1 f(x)\ dx - h\left(\frac{1}{2}f(0) + f(h) + f(2h) + \cdots + f[(n-1)h] + \frac{1}{2}f[nh]\right) = \sum_{m=1}^{\infty} b_m h^{2m}\left(f^{(2m-1)(1)} - f^{(2m-1)}(0)\right),$$

which was to be shown. □

Problem 2.

```
function w = ectr(n, k)
% n : quadrature points are 0 ... n
% k : degree of precision
```

(a) Write a (Matlab) program `ectr.m` of the form as above, which produces the weight vector $(w_0, \ldots, w_n)$ containing endpoint-corrected trapezoidal weights of even order $k = 2, 4, \ldots, 10$, for given $n \geq 2k$. Your code should combine previous codes for the differentiation matrix, the Bernoulli numbers $b_1, \ldots, b_{10}$, and the Euler-Maclaurin summation formula. Use it to complete the following table of endpoint-corrected trapezoidal weights of even order, $k = 2, 4, \ldots, 10$, as below.

k	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	$\cdots$
2	$1/2$	1	1	1	1	1	1	1	$\cdots$
4	$9/24$	$28/24$	$23/24$	1	1	1	1	1	$\cdots$
6						1	1	1	$\cdots$
$\vdots$									

Solution. Recall that in Question 1, we gave the Euler-Maclaurin summation formula for $f(x) = e^{\lambda x}$ and found a recursive formula for the Bernoulli numbers b_m. Because here in Question 2 we are evaluating on the same function ($\int_0^1 e^x \, dx$), we build on existing code.
Further notice that the Euler-Maclaurin summation formula requires us to take the $(2m-1)$-th derivative of e^x. Although this is trivial, we build our program in the more general sense, using the differentiation matrix to give the approximation at 0:

$$f^{(2m-1)}(0) \approx \sum_{k=0}^{n} \delta_{n,k}^{2m-1}(0) f(x_k).$$

Recall that $h := \frac{1}{n}$, so we consider $n = 2^5, 2^6, \ldots, 2^{10}$. This gives our desired integration rule with weights given by our program:

```
> ectr(2**10,10)[1:10]
 [1]  0.28697545  1.58901979  0.03598517  2.24089038 -0.14056437  1.72094384  0.70214534
    1.07249697  0.99210745  1.00000000
> ectr(2**8, 8)[1:10]
 [1] 0.3042245 1.4603836 0.4534640 1.4714286 0.7393932 1.0824735 0.9886326 1.0000000 1.0000000
    1.0000000
> ectr(2**6, 6)[1:10]
 [1] 0.3298611 1.3208333 0.7666667 1.1013889 0.9812500 1.0000000 1.0000000 1.0000000 1.0000000
    1.0000000
> ectr(2**4, 4)[1:10]
 [1] 0.3750000 1.1666667 0.9583333 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
    1.0000000
> ectr(2**10,2)[1:10]
 [1] 0.5 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

Our program is included here:

```r
ectr <- function(n, k) {
# n : quadrature points are 0 ... n
# k : degree of precision = 2,4, ..., 10

  w <- matrix(data=1, nrow = 1, ncol = n+1);
  w[1] <- 0.5; w[length(w)] <- 0.5;

  if (k >= 4) {

  # get b_1, b_2, ..., b_10
  getBern <- function(m) {
    b.init <- function(i) {1/factorial(2*i+1) - 1/(2*factorial(2*i))}
    bvec <- matrix( data = b.init(1:m), nrow = 1 )
    for (k in 2:m) {
      for (n in 1:(k-1)) {
        bvec[k] <- bvec[k] - bvec[n]/factorial(2*(k-n)+1)
      }
    }
    bvec
  }
  bvec <- getBern(10);

  # get deltas, given by HW 3
  M <- k - 3;
  N <- k - 2;
  d.matrix <- matrix(data=0, nrow=N+1, ncol = M);
  d.init <- matrix(data=0, nrow=N+1, ncol = M+2);
  d.init[1,2] <- 1;
  for (h in 1:(N+1)) {
    x <- 0:N;
    x[c(1,h)] <- x[c(h,1)]; #swap elements
    d.copy <- d.init;
    for (j in 2:(M+2)) {
      for (i in 2:(N+1)) {
        m <- j - 2;
        d.copy[i,j] <- (d.copy[i-1, j-1] * m - d.copy[i-1,j] * x[i]) / (x[1] - x[i]);
      }
    }
    d.matrix[h,] <- d.copy[nrow(d.copy), 3:ncol(d.copy)];
  }

  # multiply bernoulli coeffs by delta coeffs
  uwu <- matrix(data=0, nrow = k-1, ncol = 1);
  i.end <- (k/2 - 1) #last term to sum
  for (i in 1:i.end) {
    uwu <- uwu + bvec[i] * d.matrix[,2*i - 1];
  }
  w[1:(k-1)] <- w[1:(k-1)] - t(uwu);
  w[(n-k+3):(n+1)] <- w[(n-k+3):(n+1)] - t(uwu[nrow(uwu):1,])

}
  w
}
```

□

(b) Check the order of accuracy of the weights on the function

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

for $h = \frac{1}{32}, \ldots, \frac{1}{1024}$ and even $k = 2, 4, \ldots, 10$.

Solution. Again we first notice that e^x is simply a special case of what we have via Question 1, where we took $f(x) := e^{\lambda x}$, so our Bernoulli values $b_1, \ldots, b_{10}$ are valid here in this subquestion.
Because our interval is fixed $[0, 1]$, we have that different step sizes $h := \frac{1}{2^5}, \ldots, \frac{1}{2^{10}}$ give $n := 2^5, 2^6, \ldots, 2^{10}$.
To get the exact or theoretical result, we simply take the result from lecture notes (and the 'Euler-Maclaurin and ECTR' ECTR.pdf document Strain posted online), of the fourth-order endpoint corrected trapezoidal rule:

$$\int_0^1 f(x) \, dx = \frac{h}{24} \left[9f(0) + 23f(h) + 28f(2h) + 24f(3h) + 24f(4h) + 24f(5h) + \cdots + 9f(nh) \right].$$

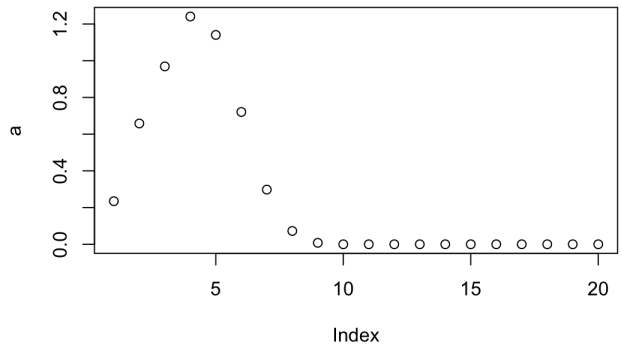
To generate the exact trapezoidal rule weights,

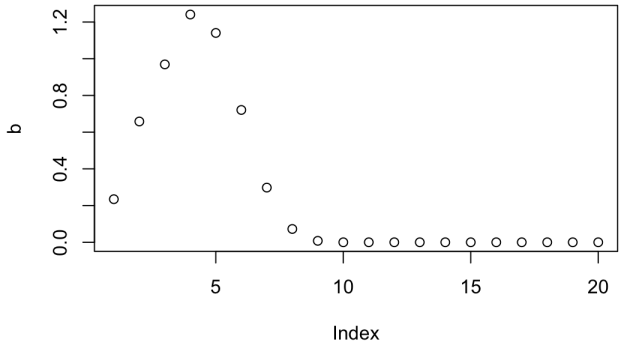
```
```{r}
ec <- function(n) {
  w <- matrix(data=24, nrow = 1, ncol = (n+1))
  # endpoint corrections
  w[1] <- 9; w[n+1] <- 9;
  w[2] <- 23; w[n] <- 23;
  w[3] <- 28; w[n-1] <- 28
  w <- w/24
}
```
```

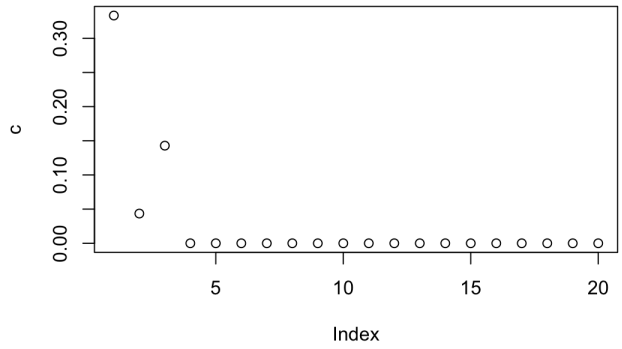
Then running against `ectr.r`, we have relative errors like these:

```
> a <- (abs(ectr(2^10,10) - ec(2^10))/ec(2^10))[1:20]
> b <- (abs(ectr(2^8,10) - ec(2^8))/ec(2^8))[1:20]
> c <- (abs(ectr(2^10,2) - ec(2^10))/ec(2^10))[1:20]
> a
 [1] 0.234732143 0.658107603 0.969155565 1.240890377 1.140564374 0.720943838
 [7] 0.297854663 0.072496969 0.007892554 0.000000000 0.000000000 0.000000000
[13] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
> b
 [1] 0.234732143 0.658107603 0.969155565 1.240890377 1.140564374 0.720943838
 [7] 0.297854663 0.072496969 0.007892554 0.000000000 0.000000000 0.000000000
[13] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
> c
 [1] 0.33333333 0.04347826 0.14285714 0.00000000 0.00000000 0.00000000
 [7] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[13] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
```

Plotting the above relative errors, perturbing each of n, k while holding the other constant, we have:







Notice that the relative error plots for and values of a, b are identical, which suggests that the precision of weights on the function depends on k rather than h, provided $h \leq \frac{1}{32}$.
We also run tests on `ectr` on a diverse subset of the arguments n, k.

```
> ectr(2^10,2)[1:10]
 [1] 0.5 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
> ectr(2^6,2)[1:10]
 [1] 0.5 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

For $k := 2$, our weights are trivial. Trying $k = 4, 6, 8, 10$, we have:

```
> ectr(2^6,4)[1:10]
 [1] 0.3750000 1.1666667 0.9583333 1.0000000 1.0000000 1.0000000
 [7] 1.0000000 1.0000000 1.0000000 1.0000000
> ectr(2^6,6)[1:10]
 [1] 0.3298611 1.3208333 0.7666667 1.1013889 0.9812500 1.0000000
 [7] 1.0000000 1.0000000 1.0000000 1.0000000
> ectr(2^6,8)[1:10]
 [1] 0.3042245 1.4603836 0.4534640 1.4714286 0.7393932 1.0824735
 [7] 0.9886326 1.0000000 1.0000000 1.0000000
> ectr(2^6,10)[1:10]
 [1]  0.28697545  1.58901979  0.03598517  2.24089038 -0.14056437
 [6]  1.72094384  0.70214534  1.07249697  0.99210745  1.00000000
```

We can also check:

```
> ectr(2^10,2)[1:10] - ectr(2^5,2)[1:10]
 [1] 0 0 0 0 0 0 0 0 0 0
> ectr(2^10,4)[1:10] - ectr(2^5,4)[1:10]
 [1] 0 0 0 0 0 0 0 0 0 0
> ectr(2^10,6)[1:10] - ectr(2^5,6)[1:10]
 [1] 0 0 0 0 0 0 0 0 0 0
> ectr(2^10,8)[1:10] - ectr(2^5,8)[1:10]
 [1] 0 0 0 0 0 0 0 0 0 0
> ectr(2^10,10)[1:10] - ectr(2^5,10)[1:10]
 [1] 0 0 0 0 0 0 0 0 0 0
```

□

Problem 3.

(a) Find an exact formula for the quintic (degree 5) polynomial such that:

$$\int_{-1}^{1} P_5(x)q(x)\ dx = 0,$$

for any quartic polynomial q.

Solution. From lecture, we have $P_0(x) := 1, P_1(x) := x$. Further, we continued to construct Legendre polynomials where the symmetric integral (i.e. -1 to 1) equals zero (integration as an inner product of orthogonal functions). We showed the recurrence relation for Legendre polynomials:

$$P_{n+1}(t) = tP_n(t) - c_nP_{n-1}(t); \qquad c_n := \frac{n^2}{4n^2-1}.$$

This iteratively gives us:

$$\begin{aligned}
P_2(x) &:= x\,(x) - \frac{1^2}{4(1)^2-1} = x^2 - \frac{1}{3} \\
P_3(x) &:= x\left(x^2 - \frac{1}{3}\right) - \frac{2^2}{4(2^2)-1}\,(x) = x^3 - \frac{3}{5}x \\
P_4(x) &:= x\left(x^3 - \frac{3}{5}x\right) - \frac{3^2}{4(3^2)-1}\left(x^2 - \frac{1}{3}\right) = x^4 - \left(\frac{6}{7}\right)x^2 + \frac{3}{35} \\
P_5(x) &:= x\left[x^4 - \left(\frac{6}{7}\right)x^2 + \frac{3}{35}\right] - \left(\frac{4^2}{4(4^2)-1}\right)\left[x^3 - \frac{3}{5}x\right] = x^5 - \frac{10}{9}x^3 + \frac{5}{21}x
\end{aligned}$$

We showed the recurrence

□

(b) Find exact formulas for the five roots x_1, x_2, x_3, x_4, x_5 of the equation $P_5(x) = 0$.

Solution. Although this is typically difficult, we notice that our Legendre polynomial $P_5(x)$ is odd, and thus we factor out x with one root, say $x_1 := 0$.

$$\begin{aligned}
P_5(x) = 0 &= x^5 - \frac{10}{9}x^3 + \frac{5}{21}x \\
&= [x^2]^2 - \frac{10}{9}[x^2] + \frac{5}{21},
\end{aligned}$$

which by the quadratic formula, explicitly gives:

$$x^2 = \frac{\frac{10}{9} \pm \sqrt{\left(\frac{10}{9}\right)^2 - 4\left(\frac{5}{21}\right)}}{2},$$

and because we only want proof-of-concept of the exact roots to $P_5(x) = 0$, we just write:

$$x = \pm\sqrt{\frac{\frac{10}{9} \pm \sqrt{\left(\frac{10}{9}\right)^2 - 4\left(\frac{5}{21}\right)}}{2}}.$$

The expression might look better with some algebra– actually, Wolfram Alpha doesn't like the nested `sqrt{}` when pasting the TeX expression, and manually typing it doesn't return a significantly better expression).

Hence we'll leave it in this form. The two instances of $\pm$ give rise to 4 distinct real roots (in addition to 0). Hence we conclude, the roots are:

$$x_3 = 0, x_2 = -\sqrt{\frac{\frac{10}{9} - \sqrt{\left(\frac{10}{9}\right)^2 - 4\left(\frac{5}{21}\right)}}{2}}, x_1 = -\sqrt{\frac{\frac{10}{9} + \sqrt{\left(\frac{10}{9}\right)^2 - 4\left(\frac{5}{21}\right)}}{2}},$$
$$x_4 = +\sqrt{\frac{\frac{10}{9} - \sqrt{\left(\frac{10}{9}\right)^2 - 4\left(\frac{5}{21}\right)}}{2}}, x_5 = +\sqrt{\frac{\frac{10}{9} + \sqrt{\left(\frac{10}{9}\right)^2 - 4\left(\frac{5}{21}\right)}}{2}},$$

where inspection helps us order them in increasing order (square roots yield a positive value). Because we need these values for the next problem, these are approximately:

```
> x <- c(
+     - sqrt( 10/9/2 + sqrt((10/9)^2 - 4*(5/21))/2 ),
+     - sqrt( 10/9/2 - sqrt((10/9)^2 - 4*(5/21))/2 ),
+     0,
+     sqrt( 10/9/2 - sqrt((10/9)^2 - 4*(5/21))/2 ),
+     sqrt( 10/9/2 + sqrt((10/9)^2 - 4*(5/21))/2 ) )
> x
[1] -0.9061798 -0.5384693  0.0000000  0.5384693  0.9061798

```

and these seem plausible to be interlaced between the roots of $P_4(x) = 0$ and are additionally all in the interval $(-1, 1)$. □

(c) Find exact formulas for the integration weights w_1, w_2, w_3, w_4, w_5 such that

$$\int_{-1}^{1} q(x)\, dx = \sum_{j=1}^{5} w_j q(x_j)$$

exactly whenever q is a polynomial of degree 5.

Solution. Recall that from lecture, we can use $q(x) := L_j(x)$ or $q(x) := L_j^2(x)$. That is, we can choose two equations to compute our desired weights:

$$\int_{-1}^{1} L_j(x)\, dx = \sum_{k=1}^{5} w_k L_j(x_k) = w_j \tag{1}$$
$$\int_{-1}^{1} L_j^2(x)\, dx = \sum_{k=1}^{5} w_k L_j(x_k) = w_j, \tag{2}$$

where (1) and (2) give the same result for w_k. In class, we mentioned that $L_j^2(x)$ may be better computationally but relatively more difficult than $L_j(x)$ when solving analytically by hand.
First we express the Lagrange basis:

$$L_1(x) = \frac{(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)(x_1-x_5)}$$
$$L_2(x) = \frac{(x-x_1)(x-x_3)(x-x_4)(x-x_5)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)(x_2-x_5)}$$
$$L_3(x) = \frac{(x-x_1)(x-x_2)(x-x_4)(x-x_5)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)(x_3-x_5)}$$
$$L_4(x) = \frac{(x-x_1)(x-x_2)(x-x_3)(x-x_5)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)(x_4-x_5)}$$
$$L_5(x) = \frac{(x-x_1)(x-x_2)(x-x_3)(x-x_4)}{(x_5-x_1)(x_5-x_2)(x_5-x_3)(x_5-x_4)},$$

and we substitute in our values for x_1, x_2, x_3, x_4, x_5 into the above and bash out the integrals via our favorite CAS (Computer Algebraic System, as permitted by Strain):

$$w_i := \int_{-1}^{1} L_i(x)\ dx,$$

to obtain:

$$\begin{aligned} w_1 &= \frac{322 - 13\sqrt{70}}{900} = w_5 \\ w_2 &= \frac{322 + 13\sqrt{70}}{900} = w_4 \\ w_3 &= \frac{128}{225}, \end{aligned}$$

where w_i corresponds to x_i, with

$$x_1 < x_2 < x_3 < x_4 < x_5,$$

defined above in the previous page.

□

(d) Given any real numbers $a < b$, find exact formulas for points $y_j \in [a, b]$ and weights $u_j > 0$ such that

$$\int_a^b q(x)\ dx = \sum_{j=1}^{5} u_j q(y_j)$$

whenever q is a polynomial of degree 5.

Solution. Because we know how to compute the integral from $[-1, 1]$, we take the transformation:

$$y := \frac{b-a}{2}x + \frac{a+b}{2}$$

Hence defining $g(x) := f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right)$, we still satisfy the $\int_a^b q(x)\ dx = \sum_{j=1}^{5} u_j q(y_j)$ while we have:

$$\int_a^b f(y)\ dy = \frac{b-a}{2}\int_{-1}^{1} g(x)\ dx,$$

as desired, where g matches the degree of f, and we can evaluate the integral of f via g on $[-1, 1]$. The problem asks for weights $u_j > 0$, so via our transformation with $\frac{dy}{dx} = \frac{b-a}{2}$, we can write:

$$\begin{aligned} u_1 &:= \frac{b-a}{2}w_1 = \frac{b-a}{2}\left(\frac{322 - 13\sqrt{70}}{900}\right) = u_5 \\ u_2 &:= \frac{b-a}{2}w_2 = \frac{b-a}{2}\left(\frac{322 + 13\sqrt{70}}{900}\right) = u_4 \\ u_3 &:= \frac{b-a}{2}w_3 = \frac{b-a}{2}\left(\frac{128}{225}\right) \end{aligned}$$

where a, b are arbitrary but fixed and given to us. Similarly, we take our roots $x_1, \ldots, x_5$ into our transformation:

$$y_j := \left(\frac{b-a}{2}\right)x_j + \frac{a+b}{2},$$

where again we have exact formulas and values when $a, b \in \mathbb{R}$ are supplied to us. The ordering may change, depending on the values of a, b. □

(e) Explain why each of the three factors in the error estimate

$$\int_a^b f(x)\ dx - \sum_{j=1}^{5} u_j f(y_j) = \underbrace{C_{10}\ \overbrace{f^{(10)}(\xi)}\ \int_a^b \left[(y-y_1)^2(y-y_2)^2(y-y_3)^2(y-y_4)^2(y-y_5)^2\right]\ dy}$$

is inevitable and determine the exact value of the constant C_{10}.

Solution. The coefficient C_{10} is required because the 10th derivative of

$$\left[(y-y_1)^2(y-y_2)^2(y-y_3)^2(y-y_4)^2(y-y_5)^2\right]$$

gives a factor of 10! (which can be seen by expanding the product). To see the value of C_{10} explicitly, see below.

The 10th derivative on $f(x)$ evaluated at ξ is given by the MVT for integrals as a bound for error; however, interpolating polynomials $(n+1)$ points exactly match polynomials of lower degree (n) points or fewer. Hence we need the error to be zero for any polynomial of degree less than 10 (i.e. degrees $9, 8, 7, ..., 0$).

The integral on the right is due to the fact that Gaussian quadrature is essentially integration on a Hermite polynomial, so we get a similar error expression as we do for Hermite interpolation error, with the added integral on the error polynomial.

Now to see explicitly that $C_{10} = \frac{1}{10!}$, consider the transformation given in OH.
Define $g(x) := f(y)$ with $y := \frac{b-a}{2}x + \frac{a+b}{2}$. It follows that

$$g^{(10)}(x) = f^{(10)}(y)\frac{d^{10}y}{dx^{10}} = f^{(10)}(y)\left(\frac{b-a}{2}\right)^{10}.$$

Hence $y - y_j = \frac{b-a}{2}(x - x_j)$ gives us:

$$\left[(y-y_1)^2(y-y_2)^2(y-y_3)^2(y-y_4)^2(y-y_5)^2\right] = \left(\frac{b-a}{2}\right)^{10}\left[(x-x_1)^2(x-x_2)^2(x-x_3)^2(x-x_4)^2(x-x_5)^2\right]$$

We wish to find some error expression like

$$\int_a^b f(x)\ dx - \sum_{j=1}^{5} u_j f(y_j) = C_{10} f^{(10)}(\xi)\int_a^b \left[(y-y_1)^2(y-y_2)^2(y-y_3)^2(y-y_4)^2(y-y_5)^2\right]\ dy,$$

so we write:

$$\begin{aligned}
\int_a^b f(y)\ dy - \sum_{j=1}^{5} u_j f(y_j) &= \frac{b-a}{2}\left[\int_{-1}^{1} g(x)\ dx - \left(\sum_{j=1}^{5} u_j f(y_j)\right)\right] \\
&= \frac{b-a}{2}\frac{g^{(10)}(\xi)}{10!}\int_{-1}^{1}(x-x_1)^2(x-x_2)^2(x-x_3)^2(x-x_4)^2(x-x_5)^2\ dx \\
&= \frac{f^{(10)}(\hat{\xi})}{10!}\int_{-1}^{1}\left(\frac{b-a}{2}\right)^{10}\left[(x-x_1)^2(x-x_2)^2(x-x_3)^2(x-x_4)^2(x-x_5)^2\right]\underbrace{\frac{b-a}{2}\ dx}_{dy} \\
&= \frac{f^{(10)}(\hat{\xi})}{10!}\int_a^b \left[(y-y_1)^2(y-y_2)^2(y-y_3)^2(y-y_4)^2(y-y_5)^2\right]\ dy.
\end{aligned}$$

Recall that we defined $y := \frac{b-a}{2}x + \frac{a+b}{2}$, so that our value for $f^{(10)}(\hat{\xi})$ is given by $\hat{\xi} := \frac{b-a}{2}\xi + \frac{a+b}{2}$, where $\xi \in (-1, 1)$, and hence $\hat{\xi} \in (a, b)$.

□

Problem 4. Write, test and debug an adaptive 5-point Gaussian integration code `gadap.m` of the form:

```
function [int, abt] = gadap(a, b, f, r, tol)
% a,b: interval endpoints with a < b
% f: function handle f(x, r) to integrate
% r: parameters for f
% tol: User-provided tolerance for integral accuracy
% int: Approximation to the integral
% abt: Endpoints and approximations
```

Build a list `abt` $= \{[a_1, b_1, t_1], \ldots, [a_n, b_n, t_n]\}$ containing n intervals $[a_j, b_j]$. With these, approximate integrals $t_j \approx \int_{a_j}^{b_j} f(x, r)\, dx$, computed with 5-point Gaussian integration. Initialize with $n := 1$ and $[a_1, b_1] := [a, b]$.
At each step $j = 1, 2, \ldots,$ subdivide interval j into left and right half-intervals l and r, and approximate the integrals t_l and t_r over each half-interval by 5 point Gaussian quadrature.
If

$$|t_j - (t_l + t_r)| > \texttt{tol} \cdot \max\{|t_j|, |t_l| + |t_r|\},$$

then add the half-intervals l and r and approximations t_l and t_r to the list. Otherwise, increment `int` by t_j. Guard against infinite loops and floating point issues as you see fit and briefly justify your design decisions in comments and documentation.

Solution. Notice that we use 5-point Gaussian quadrature, so we use our results from 4 (roots and weights). Our code is as follows:

```
  ```{r}
  gadap <- function(a,b,f,p,tol) {
   # a,b : interval endpoints, with a < b
   # f : function handle f(x, p) to integrate (p for user parameters)
   # tol : user-specified tolerance for integral accuracy
   # int : approximation to the integral
   # abt : endpoints and approximations

    num.pts <- 5;
    w <- c(
      (322 - 13 * sqrt(70))/900,
      (322 + 13 * sqrt(70))/900,
      128/225,
      (322 + 13 * sqrt(70))/900,
      (322 - 13 * sqrt(70))/900) # weights from Question 3c; should be length 5
    # roots from Question 3
    x <- c(
        - sqrt( 10/9/2 + sqrt((10/9)^2 - 4*(5/21))/2 ),
        - sqrt( 10/9/2 - sqrt((10/9)^2 - 4*(5/21))/2 ),
        0,
        sqrt( 10/9/2 - sqrt((10/9)^2 - 4*(5/21))/2 ),
        sqrt( 10/9/2 + sqrt((10/9)^2 - 4*(5/21))/2 ) )

    # integrate on [a,b]
    m <- 0.5 * ( a + b );
    l <- 0.5 * (b - a);
    int <- 0;

    for (i in 1:num.pts) {
      int <- int + w[i] * f(m + l*x[i], p);
    }
    int <- l * int

    # initialize
    n <- 1 # n = 1
    abt <- matrix(data = NA)
    abt[1,1] <- a; # a_1 = a
    abt[2,1] <- b; # b_1 = b
```
```

```
    abt[3,1] <- int;


    int <- 0; # initialize int to be summed
    j <- 0; # start at 1, adding 0+1 immediately and looping
    while (j < n) {
      j <- j + 1

      # divide [a,b] into two intervals,
      # [a.left,b.left] and [a.right, b.right]
      a.left <- abt[1,j]
      b.right <- abt[2,j]
      ab.int <- abt[3,j]

      b.left <- 0.5*(a.left + b.right)
      a.right <- b.left # intervals meet at a.right = b.left

      # integrate left interval
      m <- 0.5 * (a.left + b.left)
      l <- 0.5 * (b.left - a.left)
      l.int = 0
      for (i in 1:num.pts) {
        l.int <- l.int + w[i] * f(m + l * x[i], p)
      }
      l.int <- l * l.int;

      # integrate right interval
      m <- 0.5 * (a.right + b.right)
      l <- 0.5 * (b.right - a.right)
      r.int <- 0
      for (i in 1:num.pts) {
        r.int <- r.int + w[i] * f( m + l*x[i], p)
      }
      r.int <- l * r.int

      # if condition met:
      if ( abs(ab.int - (l.int + r.int) ) > tol * abs(ab.int) ) {
        # add half-intervals and approximations to the list
        n <- n + 1
        abt[1,n] <- a.left
        abt[2,n] <- b.left
        abt[3,n] <- l.int
        n <- n + 1
        abt[1,n] <- a.right
        abt[2,n] <- b.right
        abt[3,n] <- r.int
      } else {
        # otherwise increment int by t_j
        int <- int + ab.int
      }
    }
    # output
    c(int, abt)
  }
  ```
```

□

We use our `gadap.r` code in later problems and homeworks.
```

Problem 5.

(a) Show that

$$\int_0^1 x^{-x}\,dx = \sum_{n=1}^{\infty} n^{-n}.$$

Solution. To get this desired result, we first rewrite our function $f(x) = x^{-x} = e^{\ln\left(x^{-x}\right)} = e^{-x\ln x}$. Using a Maclaurin series expansion and defining $du := -\frac{1}{x}\,dx$, we have:

$$\begin{aligned}
\int_0^1 x^{-x}\,dx &= \int_0^1 e^{-x\ln x}\,dx \\
&= \int_0^1 \sum_{n=0}^{\infty} \frac{1}{n!}(-x)^n(\ln x)^n\,dx \\
&= \sum_{n=0}^{\infty} \frac{1}{n!}\left[\int_0^1 (-x)^n(\ln x)^n\,dx\right] \\
&= -\sum_{n=0}^{\infty}\left[\frac{1}{n!}\int_0^{\infty}(-x)^{n+1}(-u)^n\,du\right] \quad (\text{let } u := -\ln x, \text{ so } du := -\frac{1}{x}\,dx) \\
&= \sum_{n=0}^{\infty}\left[\frac{1}{n!}\int_0^{\infty}\left[e^{-u}\right]^{n+1}u^n\,du\right] \quad (x = e^{-u}, \text{ cancel negative signs by parity}) \\
&= \sum_{n=0}^{\infty}\left[\frac{1}{n!(n+1)^n}\int_0^{\infty} e^{-u(n+1)}[(n+1)u]^n\,du\right] \quad (\text{introduce } (n+1)^n \text{ to complete gamma fn expr}) \\
&= \sum_{n=0}^{\infty}\frac{1}{(n+1)!(n+1)^n}\Gamma(n+1) \\
&= \sum_{n=0}^{\infty}\frac{1}{(n+1)!(n+1)^n}(n)! \quad (\text{known fact } \Gamma(n+1) = n!) \\
&= \sum_{n=0}^{\infty}\frac{1}{(n+1)^{n+1}} \\
&= \sum_{n=1}^{\infty}\frac{1}{n^n}
\end{aligned}$$

where we were stuck until we recognized the final integral to be the gamma (Γ) function. □

(b) Use the sum in (a) to evaluate the integral in (a) to 12-digit accuracy.

Solution. Our estimation of the finite integral is an infinite sum, so we need to find some stopping point that ensures an error bounded within our tolerance of 12-digit accuracy, 10^{-12}.
That is, for some N, we have

$$\int_0^1 x^{-x}\,dx = \sum_{n=1}^{N} n^{-n} + \underbrace{\sum_{n=N+1}^{\infty} n^{-n}}_{=:E}.$$

but N also satisfies $E < 10^{-12}$.
We just need to find sufficiently large N, not the least such N that guarantees 12-digit accuracy. It is easy to see such N exists via comparison test of convergent series; however, we are interested in appropriateness of this approximation method of taking a finite sum.
One such liberal bound is

$$E = \sum_{n=N+1}^{\infty} n^{-n} \leq (N+2)[N+1]^{-(N+2)}.$$

Plugging in values, we see:

```
  > E <- function(N) { (N+2)*(N+1)^(-(N+2)) }
> E(9)
[1] 1.1e-10
> E(10)
[1] 3.82357e-12
> E(11)
[1] 1.21503e-13

```

so we conclude that in order to achieve $E < 10^{-12}$, we need:

$$N \geq 11.$$

Hence by our bound, it should suffice to sum 11 terms of $\sum_{n=1}^{N} n^{-n}$.

```
> a <- function(n) { n^{-n} }
> a.vec <- a(1:11)
> a.sum <- sum(a.vec)
> options(digits=15)
> a.sum
[1] 1.2912859970625480965
```

which gives us our approximation to the desired integral:

$$\int_0^1 x^{-x}\,dx = \sum_{n=1}^{N} n^{-n} \approx \sum_{n=1}^{11} n^{-n} \approx 1.2912859970625480965.$$

In the next problem, we find that $N := 14$ is more than sufficient to get full double precision and use this as our 'true' value for comparison against. For a preview,

```
> sum(a(1:14)) == sum(a(1:100000))
[1] TRUE
```

returns true.

□

(c) Evaluate the integral in (a) by `ectr.m` to 1,2, and 3-digit accuracy. Estimate how many function evaluations will be required to achieve p-digit accuracy for $1 \leq p \leq 12$. Explain the agreement or disagreement of your results with theory.

Solution. Running this in R console, we have:

```
> a <- function(n) { n^{-n} }
> sum(a(1:1000)) == sum(a(1:100000)) #returns true, so our value is precise to double precision
[1] TRUE
> a.vec <- a(1:1000)
> sum.5c <- sum(a.vec)
>
> # ectr
> f5 <- function(x) { x^{-x} }
> h <- 2^(-5)
> x <- seq(0,1,by=h)
> fx <- f5(x)
> w.5c <- ectr(1/h,10)
> ectr.5c <- sum(w.5c*fx)*h # result from ectr
> relerr.5c <- abs(sum.5c - ectr.5c)/abs(sum.5c)
> ectr.5c
[1] 1.2912390680663532461
> relerr.5c
[1] 3.6342836844095578156e-05
> abserr.5c <- abs(sum.5c - ectr.5c)
> abserr.5c
[1] 4.6928996310313664253e-05
```

Our approximation on inspection doesn't look too bad, but this relative and absolute error is quite bad (but yet much better than the required $1, 2, 3$-digit accuracy). Toying with different h values, it seems difficult to get worse than 3-digit accuracy. Taking $h = 2^{-10}$ gives relative error: 3.6726×10^{-08}. Our result seems to approximate the true theoretical result.
Now, to get 12-digit accuracy, notice that $h := 2^{-17}$ gives relative error $2.246263e-12$, and $h := 2^{-18}$ gives relative error $5.616088e-13$. Our total number of function evaluations, including the factorial computations and other vectorized operations is bounded by 1000 when we target 12-digit accuracy. □

(d) Approximate the integral $\int_0^1 x^{-x}\, dx$ using your code `gadap.m`. Tabulate the total number of function evaluations required to obtain p-digit accuracy for $1 \leq p \leq 10$. Compare your results with the results and estimates for endpoint-corrected trapezoidal integration obtained in (c).

Solution. For $1 \leq p \leq 10$-digit accuracy, our number of function evaluations is bounded by 20,000. For $p = 10$-digit accuracy, our algorithm needs 18554 function evaluations. This seems costly but makes sense as this is adaptive integration.
Running in R, we have the following results:

```
> f5 <- function(x,p) { x^{-x} }
> f.evals <- fn.count(gadap(0,1,f5,0,10^(-12)))
> f.evals
[1] 18554
> gadap.5d <- gadap(0,1,f5,0,10^(-12))
> gadap.5d
[1] 1.2912859970621564099
> relerr.5d <- abs(sum.5c - gadap.5d)/abs(sum.5c)
> relerr.5d
[1] 3.9274791084423147149e-11
> abserr.5d
[1] 5.0714987764877150767e-11
```

Our adaptive integration scheme `gadap.r` to 10-digit accuracy definitely took more work than `ectr.r`, and takign it to only 10-digit accuracy does not exhibit its precision versus 12-digit accuracy of `ectr.r`. However, the result is as desired, and the absolute and relative errors are acceptable for our given tolerances.

□

Problem 6.

```
function p = pleg(t, n)
% t: evaluation point
% n: degree of polynomial
```

Implement debug and test a (Matlab) function `pleg.m` of the form as above. This function evaluates a single $P_n(t)$ of the monic Legendre polynomial P_n of degree n, at evaluation point t with $|t| \leq 1$. Here $P_0 := 1$, $P_1(t) := t$ and P_n is determined by the recurrence:

$$P_n(t) = tP_{n-1}(t) - c_n P_{n-2}(t),$$

for $n \geq 2$, where $c_n = \frac{(n-1)^2}{4(n-1)^2-1}$. Be sure to iterate forward from $n = 0$ rather than recurse backwards from n, and do not generate any new function handles. Test that your function gives the right values for small n where you know P_n.

Solution. Our function `pleg.r` is extremely simple, as follows.
Recall from Problem 3(a), we already showed:

$$\begin{aligned}
P_2(x) &:= x\,(x) - \frac{1^2}{4(1)^2-1} = x^2 - \frac{1}{3} \\
P_3(x) &:= x\left(x^2 - \frac{1}{3}\right) - \frac{2^2}{4(2^2)-1}\,(x) = x^3 - \frac{3}{5}x \\
P_4(x) &:= x\left(x^3 - \frac{3}{5}x\right) - \frac{3^2}{4(3^2)-1}\left(x^2 - \frac{1}{3}\right) = x^4 - \left(\frac{6}{7}\right)x^2 + \frac{3}{35} \\
P_5(x) &:= x\left[x^4 - \left(\frac{6}{7}\right)x^2 + \frac{3}{35}\right] - \left(\frac{4^2}{4(4^2)-1}\right)\left[x^3 - \frac{3}{5}x\right] = x^5 - \frac{10}{9}x^3 + \frac{5}{21}x
\end{aligned}$$

Testing this to what we know, we have:

```
> # testing pleg versus our calculated values
> p5 <- function(x) { x^5 - 10/9* x^3 + 5/21 * x }
> p5(0.75)
[1] -0.052873883928571452362
> pleg(0.75,5)
[1] -0.052873883928571431545
> p5(0.75) == pleg(0.75, 5)
[1] FALSE
> pleg5.abserr = abs(p5(0.75) - pleg(0.75,5))
> pleg5.relerr = pleg5.abserr/abs(p5(0.75))
> pleg5.abserr
[1] 2.0816681711721685133e-17
> pleg5.relerr
[1] 3.9370441823118990853e-16
```

Our errors are within machine epsilon `.Machine$double.eps`, so we are confident our implementation is correct (within double precision). And of course we check the base cases:

```
> pleg(1,1)
[1] 1
> pleg(5,1)
[1] 5
> pleg(5,0)
[1] 1
> pleg(1,0)
[1] 1

```

□

Problem 7

```
function [w, t] = gaussint(n)
% n : Number of Gauss weights and points
```

Implement a (Matlab) function `gaussint.m` of the form as above, which computes weights w and points t for the n-point Gaussian integration rule:

$$\int_{-1}^{1} f(t)\, dt \approx \sum_{j=1}^{n} w_j f(t_j).$$

(a) Find the points t_j to as high precision as possible, by applying your code `bisection.m` to `pleg.m`. Bracket each t_j initially by the observation that the zeroes of P_{n-1} separate the zeroes of P_n for every n. Thus the single zero of $P_1 = t$ separates the interval $[-1, 1]$ into two intervals, each containing exactly one zero of P_2. The two zeroes of P_2 separate the interval $[-1, 1]$ into three intervals, and so forth. Thus you will find all the zeroes of $P_1, P_2, \ldots, P_{n-1}$ in the process of finding all the zeroes of P_n.

Solution. Our code for `gaussint.r` is as follows:

```
```{r}
gaussint <- function(n) {
  # n : number of Gauss weights and points

  #initalize
  t <- matrix(nrow = n, ncol = n)
  w <- matrix(nrow = n, ncol = n)
  tol <- .Machine$double.eps
  # root-finding of Legendre polynomials via bisection
  for (i in 1:n) {
    for (j in 1:i) {
      if (i == 1) {
        a <- -1; b <- 1;
      } else if (j == 1) {
        a <- -1; b <- t[i-1,1];
      } else if (j == n) {
        a <- t[i-1,i-1]; b <- 1;
      } else {
        a <- t[i-1,j-1]; b <- t[i-1,j];
      }
      t[i,j] <- bisection(a,b,pleg,n,tol)
    }
  }

  # get weights
  p <- t
  for (i in 1:n) {
    for (j in 1:i) {
      p[i] <- i
      p[j] <- j
      x <- seq(-1, 1+8*i,by=1)
      w[i,j] <- dot( L.sqd(x,p), ectr(8*i, 1 + 2*i) )
    }
  }
  #return
  data.frame(w,t)
}

L.sqd <- function(x,p) {
  i <- p[i]; j <-p[j]; t <- p[t];
  L <- matrix(data=1, nrow = nrow(x), ncol = ncol(x) )
  for (u in 1:i) {
    if (u != j) {
      L <- L *(x - t[i,u]) / (t[i,j] - t[i,u])
```
```

```
    }
  }
  L^2
}

# bisection via geo mean, as given by Strain
bisection <- function(a,b,f,p,t) {
  # a : Beginning of interval
  # b : End of interval
  # f : function handle y = f(x,p)
  # p : parameters to pass through to f
  # t : user-provided tolerance for interval width
  h <- c()

  while (true) {
    m <- mid(a,b); # geomean
    f.a <- f(a,p);
    f.b <- f(b,p);
    f.m <- f(m,p);
    # terminate if f vanishes
    if (f.a == 0) {
      r <- a;
      h <- c(h, a,b, f.m)
      break
    } else if (f.b == 0) {
      r <- b;
      h <- c(h, a,b, f.m)
      break
    } else {
      r <- m
      h <- c(h, a,b,f.m)
    }
    # terminate if b-a is small
    if (b - a <= t * min(abs(a), abs(b)) || a ==m || b ==m ) {
      break
    }
    # otherwise, bisect
    if (sign(f.a) != sign(f.m)) {
      b <- m
    } else {
      a <- m
    }
  }
  c(r, matrix(data = h, nrow = 3))
}

mid <- function(a,b) {
  # midpoint via geo mean of ab
  if (a == 0) {
    m <- .Machine$double.xmin
  } else if (b == 0) {
    m <- - .Machine$double.xmin
  } else if (sign(a) != sign(b)) {
    m <- 0
  } else {
    m <- sign(a) * sqrt(abs(a)) * sqrt(abs(b))
  }
  m
}
```

□

(b) Find the weights w_j to as high precision as possible by applying your code to `gadap.m` to

$$w_j = \int_{-1}^{1} [L_j(t)]^2 \, dt$$

where L_j is the jth Lagrange basis polynomial for interpolating $t_1, t_2, \ldots, t_n$.

Solution. We already have the Legendre polynomials computed by hand in a previous question, so we use can find their roots via geometric mean bisection. With these roots, we construct a Lagrange basis easily and then run the polynomial of $[L_j]^2$ through `gadap.m`. This nets us:

```
> w1
[1] 2
> w2
[1] 1 1
> w3
[1] 0.555555555556 0.888888888889 0.555555555556
> w4
[1] 0.347855 0.652145 0.652145 0.347855
> w5
[1] 0.236927 0.478629 0.568889 0.478629 0.236927
> w6
[1] 0.171324 0.360762 0.467914 0.467914 0.360762 0.171324

```

where w_i is the weight corresponding to t_i, for $i =$ `1:n`. □

(c) For `n = 1:20`, test that your weights and points integrate monomials $f(t) = t^j$ exactly for `j = 0:(2n-1)`.

Solution. We check in R console:

```
> abserr <- c()
> for (i in 1:20) {
> w <- gaussint(20)[2*i - 1]
> t <- gaussint(20)[2*i]
>   true <- ( 1^(2*i) - (-1)^(2*i) ) / (2*i)
>   numeric <- sum( w[i,1:i] * t[i,1:i]^(2*i - 1) )
>   abserr <- c(abserr, abs(true - numeric))
> }
> abserr
[1] 0 0 0 0 0 0 0 0 0 0 0 0
[13] 0 0 0 0 0 0 0 0

```

which shows our `gaussint` weights `w,t` exactly integrate monomials of odd degree, up to $2n - 1$. □