Math 128A, Summer 2019

PSET #6 (due Wednesday 8/7/2019)

```
function [p,q] = pcoeff(t, n, k)
% t : solution times t(1) < t(2) < ...< t(n) < t(n+1)
% n+1 : new time step
% k : number of previous steps t(n-k+1) ... t(n)
```

Problem 1. Write, test and debug a matlab function which computes coefficients p and q for the k-step predictor-corrector method

$$v_{n+1} = u_n + \int_{t_n}^{t_{n+1}} p(t)\,dt = u_n + p_1 f_n + p_2 f_{n-1} + \cdots + p_n f_{n-k+1},$$
$$u_{n+1} = u_n + \int_{t_n}^{t_{n+1}} q(t)\,dt = u_n + q_1 f(t_{n+1}, v_{n+1}) + q_2 f_n + \cdots + q_k f_{n-k+2}$$

Here $p(t)$ is the degree $k-1$ polynomial which interpolates the values $f_j = f(t_j, u_j)$ for $n-k+1 \le j \le n$ and $q(t)$ is the degree $k-1$ polynomial which interpolates the values f_j for $n-k+2 \le j \le n$ and also the predicted slope $f(t_{n+1}, v_{n+1})$ at t_{n+1}. Thus

$$p_j = \int_{t_n}^{t_{n+1}} \prod_{i\neq j} \frac{t - t_{n-i+1}}{t_{n-j+1} - t_{n-i+1}}, \quad \text{and}$$
$$q_j = \int_{t_n}^{t_{n+1}} \prod_{i\neq j} \frac{t - t_{n-i+2}}{t_{n-j+2} - t_{n-i+2}}$$

for $1 \le j \le k$. Tabulate the coefficients p, q with constant step size $h = 1$ and $k \le 5$ and verify against Adams-Bashforth and Adams-Moulton methods. (Hint: the integrands are polynomials of degree $k-1$ for which `ceiling`$(k/2)$ Gaussian integration points and weights will give an **exact** result.)

Solution. Strain reminds that Adams-Bashforth is explicit Adams, and Adams-Moulton is implicit Adams. Taking the hint, we take Gaussian integration points given by

`https://keisan.casio.com/exec/system/1329114617` ,

which gives Legendre-Gaussian points and weights each to at least 50-digit accuracy (reflected in the appended code).
Testing results gives (first list element is p, second is q):

```
> pcoeff(0:42,42,1)
[[1]]
[1] 1
[[2]]
[1] 1

> pcoeff(0:42,42,2)
[[1]]
[1]  1.5 -0.5
[[2]]
[1] 0.5 0.5

> pcoeff(0:42,42,3)
[[1]]
[1]  1.9166666666666666074548 -1.3333333333333332593185
[3]  0.4166666666666666185570
[[2]]
[1]  0.4166666666666661855700  0.6666666666666676288600
[3] -0.0833333333333337866744

> pcoeff(0:42,42,4)
[[1]]
[1]  2.2916666666666656304585 -2.4583333333333303727386
```

```
[3]  1.541666666666666640761463  -0.374999999999999993893773
[[2]]
[1]  0.374999999999999933386619   0.791666666666666829499377
[3] -0.208333333333333448056379   0.041666666666666690027609

> pcoeff(0:42,42,5)
[[1]]
[1]  2.640277777777777810541693  -3.852777777777777876266896
[3]  3.633333333333333439618684  -1.769444444444444501485236
[5]  0.348611111111111123150863
[[2]]
[1]  0.348611111111111231508630   0.897222222222221903464856
[3] -0.366666666666666369867045   0.147222222222222103305000
[5] -0.026388888888888867370608
```

Checking against wikipedia (`https://en.wikipedia.org/wiki/Linear_multistep_method`) suggests that our results are accurate, modulo some error.

$$\begin{aligned}
y_{n+1} &= y_n + hf(t_n, y_n), \qquad \text{(This is the Euler method)}\\
y_{n+2} &= y_{n+1} + h\left(\frac{3}{2}f(t_{n+1}, y_{n+1}) - \frac{1}{2}f(t_n, y_n)\right),\\
y_{n+3} &= y_{n+2} + h\left(\frac{23}{12}f(t_{n+2}, y_{n+2}) - \frac{16}{12}f(t_{n+1}, y_{n+1}) + \frac{5}{12}f(t_n, y_n)\right),\\
y_{n+4} &= y_{n+3} + h\left(\frac{55}{24}f(t_{n+3}, y_{n+3}) - \frac{59}{24}f(t_{n+2}, y_{n+2}) + \frac{37}{24}f(t_{n+1}, y_{n+1}) - \frac{9}{24}f(t_n, y_n)\right),\\
y_{n+5} &= y_{n+4} + h\left(\frac{1901}{720}f(t_{n+4}, y_{n+4}) - \frac{2774}{720}f(t_{n+3}, y_{n+3}) + \frac{2616}{720}f(t_{n+2}, y_{n+2}) - \frac{1274}{720}f(t_{n+1}, y_{n+1}) + \frac{251}{720}f(t_n, y_n)\right).
\end{aligned}$$

Figure 1: Adams-Bashforth

$$\begin{aligned}
y_{n+2} &= y_{n+1} + h\left(\frac{5}{12}f(t_{n+2}, y_{n+2}) + \frac{2}{3}f(t_{n+1}, y_{n+1}) - \frac{1}{12}f(t_n, y_n)\right),\\
y_{n+3} &= y_{n+2} + h\left(\frac{9}{24}f(t_{n+3}, y_{n+3}) + \frac{19}{24}f(t_{n+2}, y_{n+2}) - \frac{5}{24}f(t_{n+1}, y_{n+1}) + \frac{1}{24}f(t_n, y_n)\right),\\
y_{n+4} &= y_{n+3} + h\left(\frac{251}{720}f(t_{n+4}, y_{n+4}) + \frac{646}{720}f(t_{n+3}, y_{n+3}) - \frac{264}{720}f(t_{n+2}, y_{n+2}) + \frac{106}{720}f(t_{n+1}, y_{n+1}) - \frac{19}{720}f(t_n, y_n)\right).
\end{aligned}$$

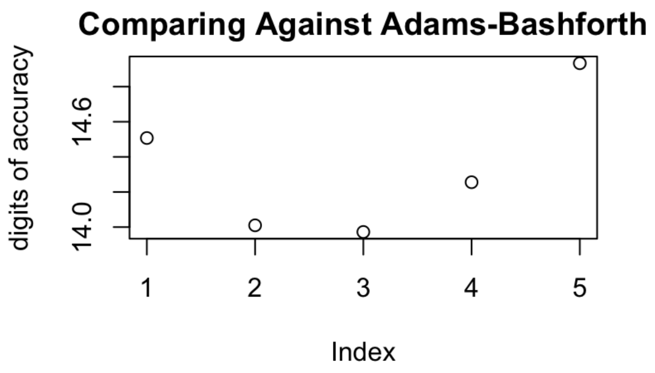
Figure 2: Adams-Moulton

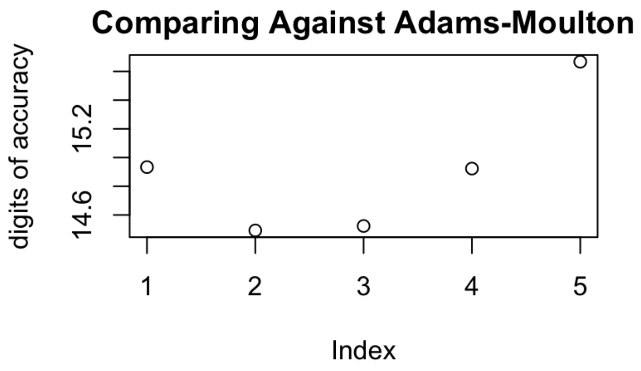
This gives the following errors:

```
## ADAMS BASHFORTH
> err.ab1
[1] 0
> err.ab2
[1] 0 0
> err.ab3
[1] -4.44089209850e-16  6.66133814775e-16 -4.44089209850e-16
> err.ab4
[1] -8.88178419700e-16  2.66453525910e-15 -2.44249065418e-15
[4]  6.10622663544e-16
> err.ab5
[1]  3.10862446895e-15 -9.76996261670e-15  1.06581410364e-14
[4] -5.55111512313e-15  1.16573417586e-15

## ADAMS MOULTON
> err.am1
[1] 0
> err.am2
[1] 0 0
> err.am3
[1] -4.44089209850e-16  9.99200722163e-16 -4.57966997658e-16
```

```
> err.am4
[1] -6.66133814775e-16  1.66533453694e-15 -1.16573417586e-15
[4]  2.35922392733e-16
> err.am5
[1]  1.16573417586e-15 -3.21964677141e-15  2.99760216649e-15
[4] -1.19348975147e-15  2.15105711021e-16
```

Plotting the interesting ($k = 5$) results, we have the following digits of accuracy:





and we conclude this is close enough to floating-point accuracy. To generate these plots:

```
# ab: adams-bashforth, explicit ( *h=1 )
ab1 <- c(1)
ab2 <- 1/2 * c(3, -1)
ab3 <- 1/12 * c(23, -16, 5)
ab4 <- 1/24 * c(55, -59, 37, -9)
ab5 <- 1/720 * c(1901, -2774, 2616, -1274, 251)

# adams-moulton, implicit ( *h=1 )
am1 <- c(1)
am2 <- 1/2* c(1, 1)
am3 <- 1/12 * c(5, 8, -1) # compare to q of 3
am4 <- 1/24 * c(9, 19, -5, 1)
am5 <- 1/720* c(251, 646, -264, 106, -19)

pq1 <- pcoeff(0:42, 42, 1)
pq2 <- pcoeff(0:42, 42, 2)
pq3 <- pcoeff(0:42, 42, 3)
pq4 <- pcoeff(0:42, 42, 4)
pq5 <- pcoeff(0:42, 42, 5)

# errors without absolute value
err.ab1 <- pq1[[1]] - ab1
err.ab2 <- pq2[[1]] - ab2
err.ab3 <- pq3[[1]] - ab3
err.ab4 <- pq4[[1]] - ab4
err.ab5 <- pq5[[1]] - ab5

err.am1 <- pq1[[2]] - am1
err.am2 <- pq2[[2]] - am2
err.am3 <- pq3[[2]] - am3
err.am4 <- pq4[[2]] - am4
err.am5 <- pq5[[2]] - am5

plot(-log(abs(err.ab5), base=10), main='Comparing Against Adams-Bashforth', ylab='digits of
    accuracy')
plot(-log(abs(err.am5), base=10), main='Comparing Against Adams-Moulton', ylab='digits of accuracy
    ')
```

And of course, our code for `pcoeff` is as follows:

```
pcoeff <- function(t,n,k) {
  # output list.p.q <- list(p,q)
  # t : solution times, t(1) < t(2) < ... < t(n) < t(n+1)
  # n+1 : new time step
  # k : = 1,2,3,4,5 : number of previous steps, t(n-k+1) ... t(n)

  # gauss-legendre points and weights to 50-digit accuracy for fun
  t1 <- c(0);
  w1 <- c(2);
  t2 <- c(-0.57735026918962576450914878050195745564760175127013,
    0.57735026918962576450914878050195745564760175127013);
  w2 <- c(1,1); t3 <- c(-0.77459666924148337703585307995647992216658434105832, 0,
    0.77459666924148337703585307995647992216658434105832);
  w3 <- c(0.55555555555555555555555555555555555555555555555556,
    0.88888888888888888888888888888888888888888888888889,
    0.55555555555555555555555555555555555555555555555556);
  t4 <- c(-0.86113631159405257522394648889280950509572537962972,
    -0.33998104358485626480266575910324468720057586977091,
    0.33998104358485626480266575910324468720057586977091,
    0.86113631159405257522394648889280950509572537962972);
  w4 <- c(0.347854845137453857373063949221999407235348695 8339,
    0.65214515486254614262693605077800059276465130416611,
    0.6521451548625461426269360507780005927646513041661,
    0.34785484513745385737306394922199940723534869583389);
  t5 <- c(-0.90617984593866399279762687829939296512565191076253,
    -0.53846931010568309103631442070020880496728660690556, 0,
    0.53846931010568309103631442070020880496728660690556,
    0.90617984593866399279762687829939296512565191076253);
  w5 <- c(0.23692688505618908751426404071991736264326000221241,
    0.47862867049936646804129151483563819291229555334314,
    0.5688888888888888888888888888888888888888888888889,
    0.47862867049936646804129151483563819291229555334314,
    0.23692688505618908751426404071991736264326000221241);
  t6 <- c(-0.93246951420315202781230155449399460913476573771229,
    -0.66120938646626451366139959501990534700644856439517,
    -0.2386191860831969086305017216807119354186106301 4,
    0.23861918608319690863050172168071193541861063014002,
    0.66120938646626451366139959501990534700644856439517,
    0.93246951420315202781230155449399460913476573771229);
  w6 <- c(0.17132449237917034504029614217273289352682250148404,
    0.3607615730481386075698335138377161116615218927467,
    0.46791393457269104738987034398955099481165560576921,
    0.4679139345726910473898703439895509948116556057692,
    0.3607615730481386075698335138377161116615218927467,
    0.17132449237917034504029614217273289352682250148404);

  h <- 1/2 * (t[n+1] - t[n]) # half-step length
  m <- 1/2 * (t[n+1] + t[n]) # midpoint to set center weight

  ## lazy case-work handling ceil(k/2) assignment
  ceilk2 <- ceiling(k/2)
  if ( k == 1) {
    return(list(h*2, h*2)) # p,q
  } else if (k == 2) {
    t.gauss <- t1 * h + m
    w.gauss <- w1 * h
  } else if (ceilk2 == 2) {
    t.gauss <- t2 * h + m
    w.gauss <- w2 * h
  } else if (ceilk2 == 3) {
    t.gauss <- t3 * h + m
    w.gauss <- w3 * h
  } else if (ceilk2 == 4) {
    t.gauss <- t4 * h + m
    w.gauss <- w4 * h
  } else if (ceilk2 == 5) {
    t.gauss <- t5 * h + m
    w.gauss <- w5 * h
  } else if (ceilk2 == 6) {
    t.gauss <- t6 * h + m
    w.gauss <- w6 * h
  }

  # p <- matrix(data = 42, nrow = 1, ncol = k)
  # q <- matrix(data = 42, nrow = 1, ncol = k)
  p <- c()
  q <- c()
  for (j in 1:k) { # go through k previous steps
    p.total <- 0
    q.total <- 0
    for (order in (1 : ceiling(k/2))) { # gauss int t_n to t_{n+1}
      t.eval <- t.gauss[order]
      p.prod <- w.gauss[order]
      q.prod <- w.gauss[order]
      for (i in 1:k) { # eval \prod expression
        if (i != j) {
          p.prod <- p.prod * (t.eval - t[n-i+1]) / (t[n-j+1] - t[n-i+1])
          q.prod <- q.prod * (t.eval - t[n-i+2]) / (t[n-j+2] - t[n-i+2])
        }
      }
    p.total <- p.total + p.prod
    q.total <- q.total + q.prod
    }
  p <- c(p, p.total)
  q <- c(q, q.total)
  }
  return(list(p,q))
}
```

□

Problem 2. Write, test and debug a matlab function which uses `pcoeff` to approximate the solution vector $y(t)$ of the vector initial value problem

$$y' = f(t, y, r)$$
$$y(a) = y_a$$

by the family of methods you derive in Problem 1, with $u_1 = y_a$. Start with $k_1 = 1$ and a tiny step size

$$h_1 = (b-a)\left(\frac{h}{b-a}\right)^{k/2}$$

which brings the one-step error in line with the $O(h^k)$ error. Increase the step size smoothly (e.g. by $h_1 \leftarrow \left(1+\frac{1}{k}\right)h_1$) and increase k_1 (e.g. by steps of 1 up to k) until $h_1 \geq h = \frac{b-a}{N-1}$ and then continue with uniform step sizes.
(Notice that to save CPU time,
(i) when the most recent k step sizes are uniform, the predictor-corrector coefficients p, q can be frozen and
(ii) many values of f can be saved rather than re-evaluated.)

(a) Use `pcode.m` with odd $k = 1, 3, 5, \ldots, 11$ (requires 6-point Gaussian Quadrature, or Strain says we can go to 9 and call it a day) and $N = 10000, 20000, 40000, 80000, 160000$ to approximate the final solution vector $u(T)$ of the intial value problem derived in Problem 4 of PSET 5 (satellite orbit around the earth and moon). Tabulate the errors

$$E_{kN} = \max_{1\leq j\leq 4} |u_j(T) - u_j(0)|.$$

Estimate the constant C_k such that the error behaves like $C_k h^k$.

Solution. From the following errors, we estimate C_k via $h = -\log n$ linear slope:

$$C_1 = \frac{1.36891 - 0.33039}{\log_1 0160000 - \log_1 010000} = 0.862472 \text{ seconds}$$
$$C_3 = 8.62472 \times 10^3 \text{ seconds}$$
$$C_9 = 8.62472 \times 10^{17} \text{ seconds}$$

Our errors for $k = 1$ across N are:

```
> errors.2a # k = 1
[1,] 1.36891
[2,] 2.48831
[3,] 1.48563
[4,] 0.79838
[5,] 0.33039
```

Our errors for $k = 3$ across N are:

```
>> errors.2a # k = 3
          [,1]
[1,]   2.00631
[2,]   1.94371
[3,]   1.76108
[4,]   1.41130
[5,]   1.23706
```

Our errors for $k = 9$ across N are:

```
> errors.2a # k = 9
[1,] 1.39840
[2,] 1.09492
[3,] 0.60355
[4,] 0.29218
[5,] 0.14075
```

To generate plots and tabulate errors and CPU cost:

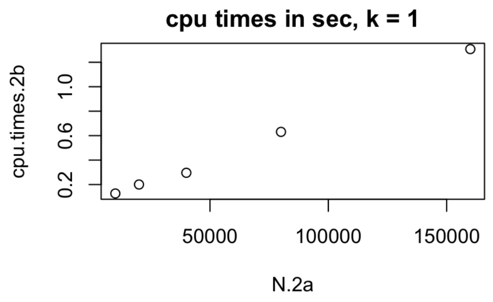
```
period.T <- 17.06521656015796
ua.2a <- c(0.994, 0, 0, -2.00158510637908) # starting pos
N.2a <- 1e4*c(1,2,4,8,16)
errors.2a <- matrix(data=NA, nrow = length(N.2a), ncol = 1)

k = 9 # set k

cpu.times.2b <- c()
for (N.index in 1:length(N.2a)) {
  N <- N.2a[N.index]
  time.start <- Sys.time()
  tu <- pcode(0, period.T, ua.2a, sat.orbit, r, k, N) # set k
  u.all <- tu[[2]]
  u.final <- u.all[,ncol(u.all)]
  time.stop <- Sys.time()
  cpu.times.2b <- c(cpu.times.2b, time.stop - time.start)
  errors.2a[N.index] <- max(abs(u.final - u.all[,1]))
  plot(u.all[1,], u.all[3,], type='l',
       main=paste('Predictor Corrector ODE \n k=',k,'N=',N.2a[N.index]))
}
log.N.2a <- log(N.2a,base=10)
plot(log.N.2a, errors.2a)
plot(log.N.2a, cpu.times.2b)
```

□

(b) Measure the CPU time for each run and estimate the total CPU time necessary to obtain an orbit that is periodic to $3, 6, 12$-digit accuracy.

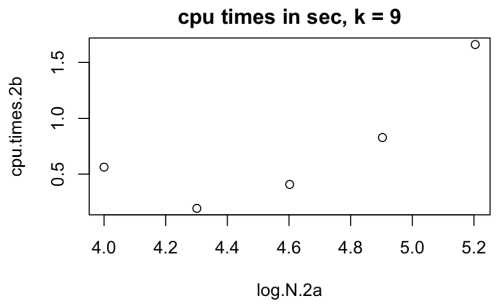
Solution. For $k = 1$, our cpu times across N are:

```
> cpu.times.2b
[1] 0.317155 0.319249 0.529587 0.792668 1.193514
```



For $k = 9$, our cpu times across N are:

```
> cpu.times.2b
[1] 0.562517 0.193649 0.408006 0.827510 1.659895
```



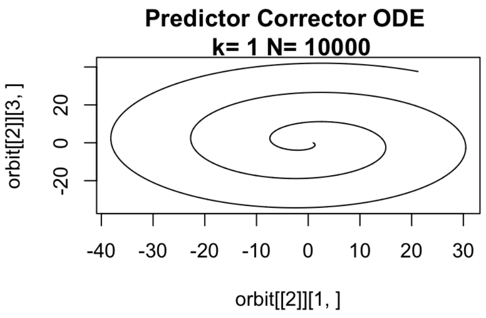
Now, our cpu times are extremely erratic as can be seen by the plots. However, assuming a linear log relationship as we have in homework 5, we can get a (very bad) set of estimates for $k = 9$:

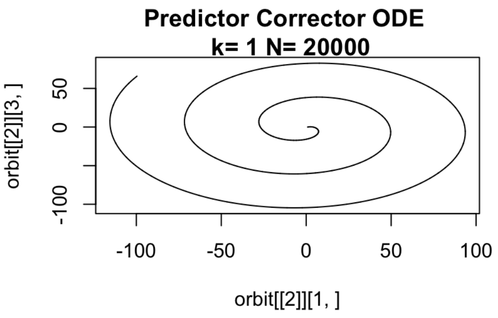
$$\begin{aligned} \text{3-digit accuracy} &= 0.671929 \\ \text{6-digit accuracy} &= 1.44763 \\ \text{12-digit accuracy} &= 7.62813 \end{aligned}$$

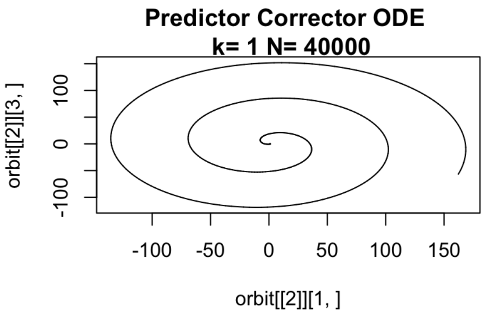
$\square$

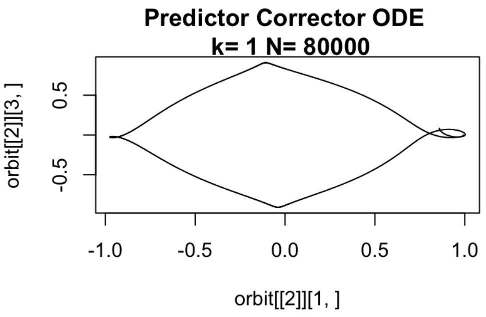
(c) Plot some inaccurate solutions and some accurate solutions, and draw conclusions about values of k that give $3, 6, 12$-digit accuracy for minimal CPU time.

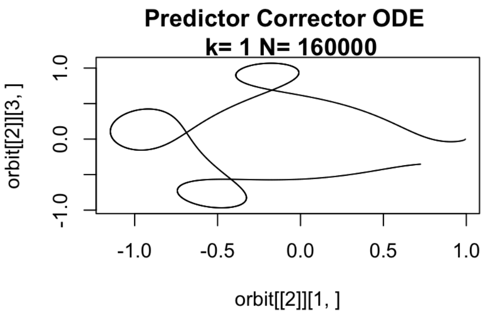
Solution. To exhibit some (very) inaccurate solutions, we first show the results for $k = 1$ across $N = 10000, 20000, 40000, 80000, 16000$, then $k = 3$ across N, and finally $k = 9$ to show the accuracy, and then we call it a day (and give the cpu a rest). It appears that $k = 3$ gives the best spread of accuracy for $3, 6, 12$-digit accuracy. At $k = 9$, it starts to look the same across N, so $k = 9$ may be unnecessarily high.

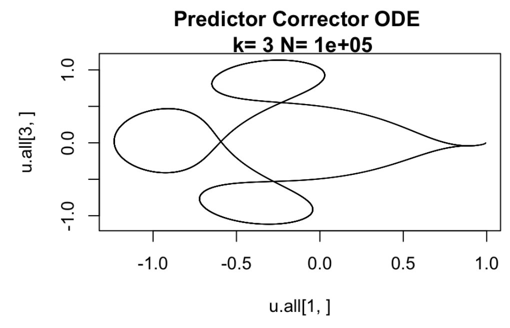


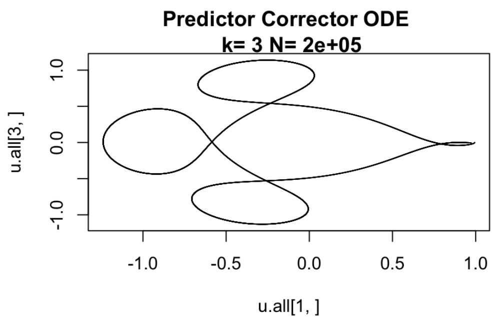


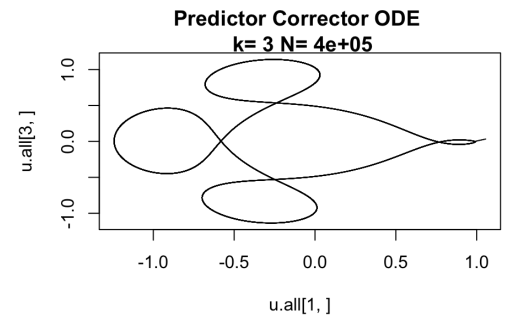


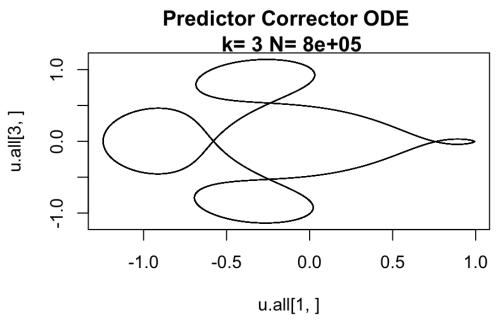


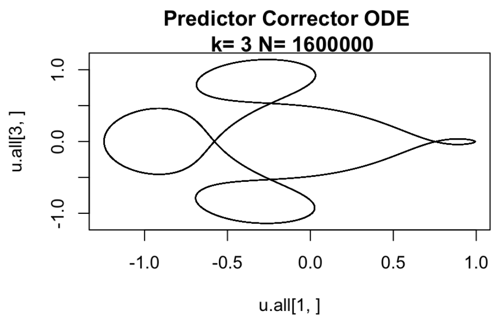


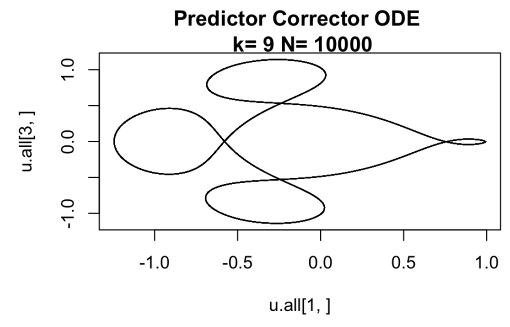


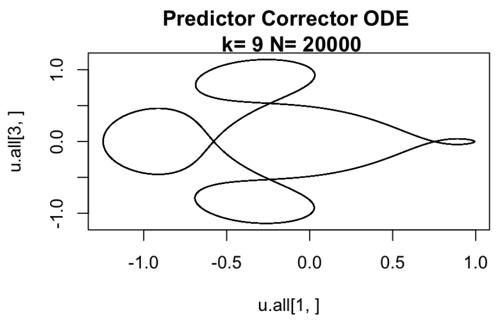


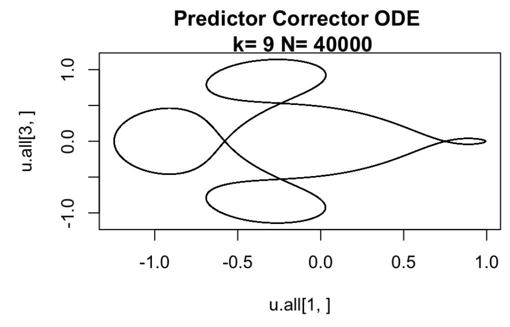


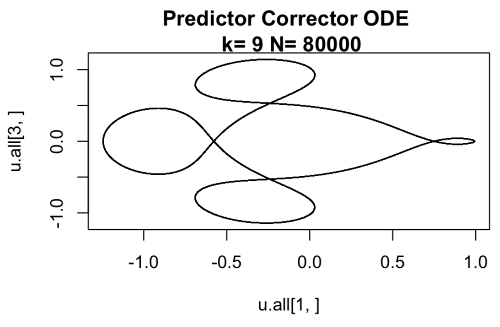










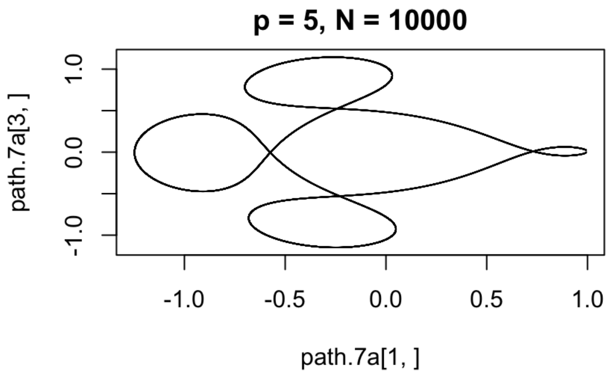


and we call it a day (yeehaw).

$\square$

(d) Compare to the results of `euler.m` and `idec.m`.

Solution. Recall that `idec` with $p = 1$ is simply `euler`, so inherently, we showed that given the same step size, `idec` generally (for $p > 1$) gives more accurate results. It is true that we may approach the exact solution by spamming smaller step sizes in `euler`, but cpu cost explodes to catch up to `idec` performance.
Personally and objectively, it appeared that `idec` gave better results than predictor-corrector, but it may be because I programmed `pcode` inoptimally. However, I will say that for $k \geq 3$, our predictor-corrector scheme `pcode` appears to perform more consistently across the N number of steps that we tested, as opposed to the largely varying results for a fixed p in `idec`.
To compare to our resulting plots in part (c) on the previous page, we include some plots from `idec` that we generated for Homework 5:



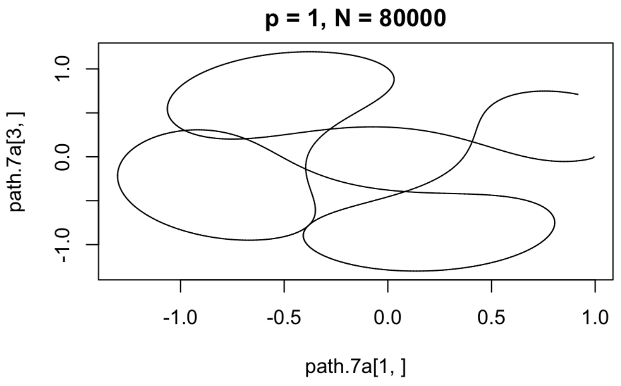


Figure 3: Equivalent to `euler`, very bad

□

Our `pcode` is as follows:

```
pcode <- function(a, b, ua, f, r, k, N) {
  # Predictor Code ODE
  # a,b : interval endpoints with a < b
  # ua : vector u_1 = y(a) of initial conditions # c(x,x,x,x)
  # function handle f(t, u, r) to integrate
  # r : parameters to f
  # k : number of previous steps to use at each regular time step
  # N : total number of time steps
  # t : output times for numerical solution ,   u_n : y(t_n), t(1) = a, t(N) = b
  # u : numerical solution at times t

t <- matrix(data=0, nrow = N, ncol = 1); t[1] <- a;
u <- matrix(data=0, nrow = length(ua), ncol = N); f.eval <- u;
h <- (b - a)/(N - 1)
store.u <- matrix(data=0, nrow = length(ua), ncol =  1)
u[,1] <- matrix(data = ua, nrow = length(ua), ncol = 1)

if (k == 1){ # degenerate, just use 1 step
   for (i in 1:(N-1)) {
     t[i+1] <- t[i] + h
     store.u <- u[,i]
     store.u <- store.u + h * f(t[i], store.u, r)
     u[,i+1] <- u[,i] + h * f( t[i+1], store.u, r )
   }
} else {
  # increase step size smoothly
  t[1] <- a
  theta <- 1 + 1/k # given scaling
  h1 <- (b-a) * ((h/(b-a))**(k/2))
  i <- 1 # init index
  while (h1 < h) { # specified condition
    t[i+1] <- t[i] + h1
      i <- i + 1 # continue
    h1 <- theta * h1 # scaling factor
  }
  # h1 >= h, so resume uniform stepsize
  h <- (b - t[i])/(N-i) # new step size
  for (j in (i+1):N){
    t[j] <- t[i] + h * (j-i)
  }
  u[,1] <- matrix(data=ua, nrow = length(ua), ncol = 1)
  f.eval[,1] <- f( t[1], ua, r)
  freeze.idx <- i + 2 * k # freeze before indexing through again
  for (i in 1:(N-1)) {
    k.min <- min(i,k) # correction steps
    if ( i < freeze.idx ) {
      pq <- pcoeff(t,i,k.min)
      p <- pq[[1]]
      q <- pq[[2]]
    }
    ## better to do 2 separate loops, suggested by Strain
    # Predictor
    store.pred <- u[,i] + p[1] * f.eval[,i]
    for (x in 1:(k.min-1)) {
      store.pred <- store.pred + p[x+1] * f.eval[, i-x]
    }
    # Corrector
    store.corr <- u[,i] + q[1] * f( t[i+1], store.pred, r )
    for ( x in 1:(k.min-1)) {
      store.corr <- store.corr + q[x+1] * f.eval[, i + 1 - x]
    }
    u[,i+1] <- store.corr
    f.eval[,i+1] <- f( t[i+1], store.corr, r )
  }
}
list(t,matrix(data=u, nrow = length(ua)))
}
```

Problem 3. Consider a differential equation

$$y'(t) = f(t, y(t)),$$

where f satisfies the condition

$$(u - v)[f(t, u) - f(t, v)] \leq 0$$

for all u, v.

(a) Suppose $U(t)$ and $V(t)$ are exact solutions. Show that

$$|U(t) - V(t)| \leq |U(0) - V(0)|$$

for all $t \geq 0$.

Solution. Here we want to show that the distances between two exact solutions does not increase over time. Given that $U(t), V(t)$ are exact solutions, then $U'(t) = f(t, U(t))$ and $V'(t) = f(t, V(t))$. Consider:

$$\begin{aligned}(u - v)\,[f(t, u) - f(t, v)] &\leq 0, \quad \forall u, v \quad \text{(given)}\\ (u - v)\,[u' - v'] &\leq 0 \quad \text{(substituting)}\\ \int (u - v) d[u - v] &\leq 0\\ \frac{1}{2}(u - v)^2 &\leq C \qquad \text{(not good enough)},\end{aligned}$$

and although this does not directly give us the desired result (we don't know C), it gives us intuition to work in the opposite direction.

Now define $g(t) := \sqrt{(u(t) - v(t))^2}$ (coercing a positive result–although we only consider $g(t)^2$ anyway), and notice if we show $g(t)$ is monotonic nonincreasing, then we are done. To supply the desired product gg' with the above intuition, take:

$$\begin{aligned}\frac{d}{dt}[g(t)]^2 &= \frac{d}{dt}[u(t) - v(t)]^2\\ &= \frac{d}{dt}\left(\sqrt{(u(t) - v(t))^2}\right)^2\\ &= 2[u(t) - v(t)][u'(t) - v'(t)] \qquad \text{(chain rule)}\\ &= 2(u - v)[f(t, u) - f(t, v)] \qquad \text{(substituting)}\\ &\leq 0 \quad \text{(given dissipative condition)},\end{aligned}$$

which means that the square function $g(t)^2$ is monotonically nonincreasing. Because $g(t) = \sqrt{(u(t) - v(t))^2} = \sqrt{(v(t) - u(t))^2}$ (symmetric), we conclude that $g(t)$ is monotonically nonincreasing. This gives for all $t \geq 0$, $g(t) \leq g(0)$, which equivalently is:

$$\boxed{\forall_{t \geq 0} \quad |U(t) - V(t)| \leq |U(0) - V(0)|},$$

precisely our desired result.

□

(b) Suppose W satisfies a perturbed differential equation

$$W'(t) = f(t, W(t)) + r(t)$$

for $t \geq 0$. Show that for $t \geq 0$,

$$|U(t) - W(t)| \leq |U(0) - W(0)| + \int_0^t |r(s)|\, ds$$

Solution. We want some expression or inequality for the perturbing term $r(t)$ with respect to U, W, so that we can derive the desired inequality. Although the structure here is similar to that in part (a), notice that W is not an exact solution to the original differential equation in (a). However, we do have:

$$[U(t) - W(t)][f(t, U(t)) - f(t, W(t))] = [U(t) - W(t)][U'(t) - W'(t) + r(t)] \leq 0 \tag{1}$$

from the condition given in part (a) which holds for all u, v, namely with $v = W$ with $r(t) = 0$.
Now, we already know from (a) we can get this expression via some $\frac{d}{dt}g(t)^2$, so again let $g(t) := \sqrt{(U(t) - W(t))^2}$ to coerce nonnegative values. Then we have:

$$\begin{aligned}
\frac{d}{dt}g(t)^2 &= \frac{d}{dt}\left(\sqrt{(U(t) - W(t))^2}\right)^2 \\
&= 2g(t)g'(t) \\
&= 2[U(t) - W(t)][U'(t) - W'(t)] \\
&= 2[U(t) - W(t)][f(t, U(t)) - \underbrace{f(t, W(t)) - r(t)}_{W'(t)}] \\
&= \underbrace{2[U(t) - W(t)][f(t, U(t)) - f(t, W(t))]}_{\leq 0 \text{ by (1) above}} - 2[U(t) - W(t)]r(t) \\
&\leq -2[U(t) - W(t)]r(t) \leq |-2[U(t) - W(t)]r(t)| \\
&\leq 2g(t)|r(t)|
\end{aligned}$$

and because our construction for $g(t)$ asserts it is nonnegative (handling the $g(t) = 0$ case separately below), we perform separation of variables on our result above, which gives:

$$\frac{1}{2g(t)}[g(t)^2]' \leq |r(t)|$$

and integrating both sides from 0 to t (and changing the variable name of the integrands as Strain conditions us to not forget),

$$\begin{aligned}
\int_0^t \frac{1}{\cancel{2g(s)}}[\cancel{2g(s)}g'(s)]\, ds &= \int_0^t |r(s)|\, ds \\
g(t) - g(0) &\leq \int_0^t |r(s)|\, ds \\
g(t) &\leq g(0) + \int_0^t |r(s)|\, ds \\
\sqrt{(U(t) - W(t))^2} &\leq \sqrt{(U(0) - W(0))^2} + \int_0^t |r(s)|\, ds
\end{aligned}$$

which brings us precisely to our desired expression:

$$\boxed{|U(t) - W(t)| \leq |U(0) - W(0)| + \int_0^t |r(s)|\, ds}$$

Now we consider the case when $g(t) = 0$ and we cannot perform the separation of variables above by dividing across by $g(t)$. Notice that $|g(t)| \leq |g(0)| + \int_0^t |r(s)|\, dt$ is trivially true when $g(t) = g(0) = 0$. $\square$

(c) Show that two numerical solutions u_n and v_n generated by implicit Euler (e.g. with different initial values) satisfy, for all $n \geq 0$,

$$|u_n - v_n| \leq |u_0 - v_0|.$$

Solution. Consider the set $U \subset \mathbb{N}$ of natural number indices i for which our implicit Euler iteration solutions has this desired property. Trivially, $0 \in U$ because $|u_0 - v_0| \leq |u_0 - v_0|$ in the $i = 0$ case. Now we assume (induction hypothesis) that $n-1 \in U$ and wish to show this implies $n \in U$.
Recall that implicit euler is simply:

$$u_n = u_{n-1} + hf(t_n, u_n), \qquad v_n = v_{n-1} + hf(t_n, v_n),$$

Consider:
We are given the condition

$$(u_n - v_n)[f(t, u_n) - f(t, v_n)] \leq 0, \tag{2}$$

(f dissipative) where this holds for all u, v and hence must hold for $u := u_n$ and $v := v_n$.
Intuitively, in the end (or so I thought) we want to use the condition the product of two elements, which we can write as:

$$2ab = (a^2 + b^2) - (a-b)^2.$$

Now we set $a := (u_n - v_n)$ and $b := h[f(t, u_n) - f(t, v_n)]$, so that

$$a - b = (u_n - v_n) - h[f(t, u_n) - f(t, v_n)] = u_{n-1} - v_{n-1},$$

which by $n-1 \in U$ (strong induction induction assumption), we have $|a - b| = |u_{n-1} - v_{n-1}| \leq |u_0 - v_0|$. This looks promising! Now plugging into our expression gives:

$$\begin{aligned}
0 &\geq 2ab = a^2 + b^2 - (a-b)^2 \\
&= \underbrace{(u_n - v_n)^2}_{a^2} + \underbrace{h^2[f(t, u_n) - f(t, v_n)]^2}_{b^2} - \underbrace{(u_{n-1} - v_{n-1})^2}_{(a-b)^2} \\
\implies (u_n - v_n)^2 &\leq (u_{n-1} - v_{n-1})^2 - h^2[f(t, u_n) - f(t, v_n)]^2 \\
&\leq |u_{n-1} - v_{n-1}|^2 \quad \text{(trivial inequality)} \\
&\leq |u_0 - v_0|^2 \quad (\text{induction hypothesis}, n-1 \in U)
\end{aligned}$$

which by taking the positive square root (via absolute value) precisely gives us:

$$|u_n - v_n| \leq |u_0 - v_0|,$$

so we conclude $n - 1 \in U \implies n \in U$, so by induction, $U = \mathbb{N}$ and we have $\boxed{|u_n - v_n| \leq |u_0 - v_0|, \forall n \in \mathbb{N}}$.

□

This was my first solution, but further review led me to believe I found a neater solution that did not use the given condition and thus publicly shamed the above solution. It turns out that I invented an incorrect triangle inequality, and hence the following is incorrect:

$$\begin{aligned}
|u_n(t) - v_n(t)| - |u_{n-1}(t) - v_{n-1}(t)| &= |[u_{n-1}(t) + hf(t, u_n)] - [v_{n-1}(t) + hf(t, v_n)]| - |u_{n-1}(t) - v_{n-1}(t)| \\
&\leq \cancel{|u_{n-1}(t) - v_{n-1}(t)|} - \underbrace{|hf(t, u_n) - hf(t, v_n)|}_{\geq 0 \text{ abs val}} - \cancel{|u_{n-1}(t) - v_{n-1}(t)|} \quad \leq 0 \\
\implies \underbrace{|u_n - v_n|} &\leq |u_{n-1} - v_{n-1}| \leq \underbrace{|u_0 - v_0|},
\end{aligned}$$

and we have $0, n-1 \in U \implies n \in U$, so by induction we have $U = \mathbb{N}$ and we are finished (not really).

(d) Show that the local truncation error τ_{n+1} of the implicit Euler method

$$u_{n+1} = u_n + hf(t_{n+1}, u_{n+1})$$

is given by

$$\tau_{n+1} = \frac{y_{n+1} - y_n}{h} - f(t_{n+1}, y_{n+1}) = -\frac{h}{2}y''(\xi)$$

where $y_n = y(t_n)$ is the exact solution and ξ is an unknown point.

Solution. Recall from lecture (Lecture 18, Tuesday 7/23), we defined the local truncation error **for the explicit Euler method** ($u_{n+1} = u_n + hf(t_n, u_n)$) as:

$$\tau_n := \frac{y_{n+1} - y_n}{h} - f(t_n, y_n),$$

where we have $f(t_n, y_n) = y'_n$, so Taylor expanding gives:

$$\begin{aligned}\tau_n &= \frac{y_{n+1} - y_n}{h} - y'_n \\ &= \frac{y_n + hy'_n + \frac{1}{2}h^2 y''(\xi_n) - y_n}{h} - y'_n \\ &= \frac{h}{2}y''(\xi_n) = O(h).\end{aligned}$$

Now we do something similar for our given **implicit Euler method** of the form $u_{n+1} = u_n + hf(t_{n+1}, u_{n+1})$. Consider:

$$\begin{aligned}\tau_{n+1} &= \frac{y_{n+1} - y_n}{h} - f(t_{n+1}, y_{n+1}) \\ &= \frac{y_{n+1} - y_n}{h} - y'(t_{n+1}) \\ &= \frac{y_{n+1} - \left[y_{n+1} - hy'(t_{n+1}) + \frac{h^2}{2}y''(\xi)\right]}{h} - y'(t_{n+1}) \\ &= -\frac{h}{2}y''(\xi),\end{aligned}$$

which was to be shown.

□

(e) Show that the numerical solution u_n generated by implicit Euler with $u_0 := y_0$ satisfies:

$$|u_n - y_n| \leq nh\tau$$

for $0 \leq nh < \infty$, where $\tau = \frac{Mh}{2}$ and $|y''| \leq M$.

Solution. First, we explicitize a few things. From the given, $|y''| \leq M$ so y is twice-differentiable and Lipschitz, and M is nonnegative (as is h), so $\tau = \frac{Mh}{2}$ is also nonnegative. Additionally, $0 \leq nh < \infty$ so n is finite and positive. Again, we formalize via induction for all $n \in \mathbb{N}$. Consider the set $U \subset \mathbb{N}$ of indices i for which $|u_n - y_n| \leq nh\tau$. We established that $nh\tau \geq 0$. For $i = 0$, we have $|u_0 - y_0| = 0 = 0 \cdot n\tau$, so $0 \in U$. Now suppose (induction) $n-1 \in U$. That is, assume that we have:

$$|u_{n-1} - y_{n-1}| \leq (n-1)h\tau = (n-1)\frac{Mh^2}{2}.$$

Now, consider:

$$\begin{aligned}
|u_n - y_n| &= \left|[u_{n-1} + \cancel{hf(t_n, u_n)}] - [y_{n-1} + \cancel{hf(t_n, y_n)} - \frac{h^2}{2}y''(\xi)]\right| \\
&\leq \left|u_{n-1} - y_{n-1} + \frac{h^2}{2}y''(\xi)\right| \\
&\leq \underbrace{|u_{n-1} - y_{n-1}|}_{\leq (n-1)\frac{Mh^2}{2}} + \left|\frac{h^2}{2}M\right| \quad (y'' \text{ is Lipschitz bounded by } M) \\
&= n\frac{Mh^2}{2},
\end{aligned}$$

where in the first line we cross these out in absolute value inequality to the next line because we're given the condition $(u-v)[f(t,u) - f(t,v)] \leq 0$ for all u, v, and this argument is identical by multiplying across by $|u_n - y_n|$. Notice that if $u_n = y_n$, then a division across by $|u_n - y_n| = 0$ is not legal; however, in this case $|u_n - y_n| = 0 \leq nh\tau$, as desired.

We conclude $n - 1 \in U \implies n \in U$, so by induction, $U = \mathbb{N}$ and we have:

$$\boxed{|u_n - y_n| \leq nh\tau, \qquad \forall_{n \in \mathbb{N}}},$$

as required. ☐

Problem 4. Consider the linear initial value problem

$$\begin{aligned} y' &= -L(y(t) - \varphi(t)) + \varphi'(t) \\ y(0) &= y_0 \end{aligned}$$

where $\varphi(t) := \cos(30t)$.

(a) Solve the initial value problem exactly.

Solution. First notice that this is essentially the same problem presented in Lecture 23 (Wednesday July 31) in examining stiff equations, linear stability and nonlinear stability:

$$y' = \lambda(y - \phi(t)) + \phi'(t),$$

where we inserted the forcing term $\phi'(t)$ so that our problem is solvable. Strain notes that $-L \in \mathbb{C}$ is the dampening constant if it's negative, with $\text{Re}(-L) << 0$, and $\varphi(t)$ describes something that oscillates smoothly (like a massage chair). Essentially, $-L$ describes how we want to get to that smooth oscillation.
We proceed directly as given by Strain (changing $\lambda \to -L$ and $\phi \to \varphi$ as needed).

$$\begin{aligned} y'(t) &= -L(y(t) - \varphi(t)) + \varphi'(t) \\ (y(t) - \varphi)' &= -L(y(t) - \varphi(t)) \\ y(t) - \varphi &= e^{-Lt}[y(0) - \varphi(0)] \\ y(t) &= \varphi(t) + e^{-Lt}\underbrace{[y(0) - \varphi(0)]}_{transient}, \end{aligned}$$

and we insert our initial conditions $\varphi(0) = \cos(30 \cdot 0) = 1$ and $y(0) = y_0$ and $\varphi(t) := \cos(30t)$ to get:

$$\boxed{y(t) = \cos(30t) + (y_0 - 1)e^{-Lt}}$$

as our exact solution to the IVP. $\square$

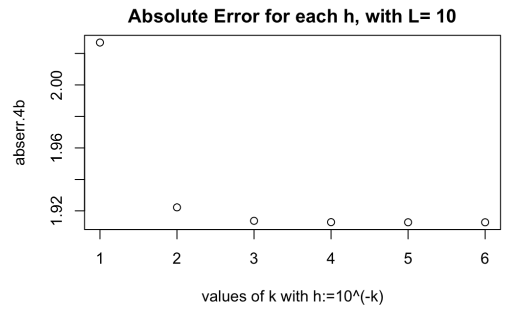
(b) Use `euler.m` to solve the initial value problem with $y(0) = 2$ for $0 \le t \le 1$ with $L := 10^k$ for $k :=$ `1:5`. For each L use $h = 10^{-j}$ with $j =$ `1:6`. Tabulate the errors.

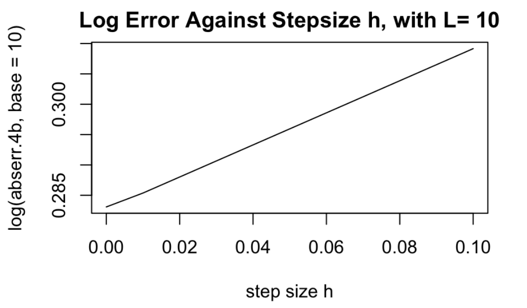
Solution. Here we use the familiar `euler.m` that we made and used in a previous problem set (HW 5). Our errors are as follows:

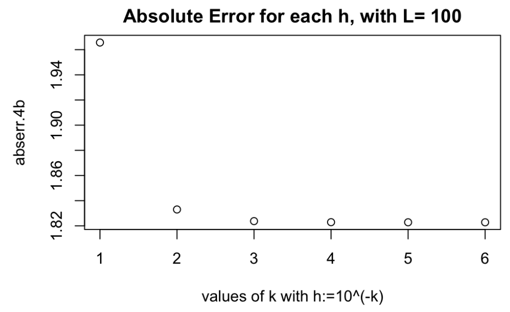
```
> # h=10e-1, 10e-2,  10e-3,  10e-4,  10e-5,  10e-6 :
[1] 2.02695 1.92223 1.91369 1.91286 1.91277 1.91277 # T = 10
[1] 1.96567 1.83300 1.82375 1.82291 1.82282 1.82282 # T = 10^2
[1] 1.96566 1.82715 1.81482 1.81391 1.81382 1.81382 # T = 10^3
[1] 1.96566 1.82714 1.81424 1.81301 1.81292 1.81292 # T = 10^4
[1] 1.96566 1.82714 1.81424 1.81296 1.81283 1.81283 # T = 10^5
```

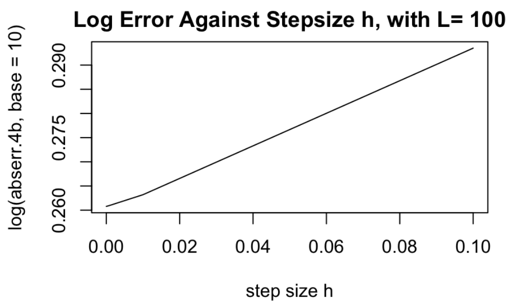
Our code to tabulate data and generate plots:

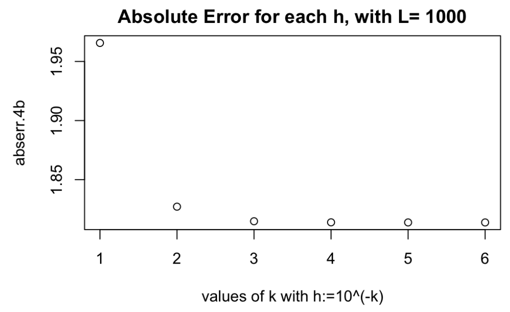
```
L.4b <- 10^c(1,2,3,4,5)
h.4b <- 10^(-1*c(1,2,3,4,5,6))
ya.4b <- 2;

for (L in L.4b) {
  abserr.4b <- c()
  for (h in h.4b) {
    n <- 1/h
    t.4b <- seq(0,1,by=h)
    # create L,n-specific vibrator function
    f.vibrator <- function(t,y,r) {
      # sum small before larger
      (y - 1) * exp(-L * t) + cos(30*t)
    }

    path.vib <- eulerfast(0,1,ya.4b, f.vibrator, r=1, n+1)
    true.4b <- f.vibrator(1,ya.4b,0)
    abserr.4b <- c(abserr.4b, abs(true.4b - path.vib))
    # plot(t.4b, path.vib, main=paste('h=',h,'L=',L))
  }
  plot(abserr.4b, main=paste('Absolute Error for each h, with L=',L), xlab='values of k with h
    :=10^(-k)')
  plot( h.4b , log(abserr.4b,base=10), main=paste('Log Error Against Stepsize h, with L=',L), xlab
    ='step size h', type='l')
  print(abserr.4b,digits=6) # display
}

# from PSET 5
eulerfull <- function(a, b, ya, f, r, n) {
  # RETURNS FULL HISTORY UP TO SOLUTION
  h <- (b - a)/n # fixed time-step
  u <- ya # initial conditions
  ### u.store used to generate movement plot along unit circle
  u.store <- matrix(data=NA, nrow = length(ya), ncol = n+1)
  u.store[,1] <- ya
  t <- a # start of interval
  for (j in 1:n) {
    u <- u + h * f(t,u,r)
    u.store[,j+1] <- u
    t <- t + h
  }
  u.store # outputs matrix of path, used for plotting later
}

eulerfast <- function(a, b, ya, f, r, n) {
  # init
  h <- (b - a)/n # fixed time-step
  u <- ya # initial conditions
  t <- a # start of interval
  for (j in 1:n) {
    u <- u + h * f(t,u,r)
    t <- t + h
  }
  u # outputs u without path
}
```

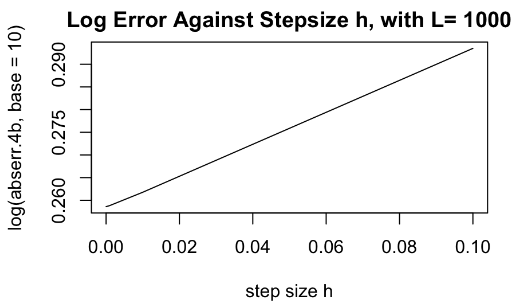

Absolute Error for each h, with L= 10
abserr.4b
1.92
1.96
2.00
1 2 3 4 5 6
values of k with h:=10^(-k)

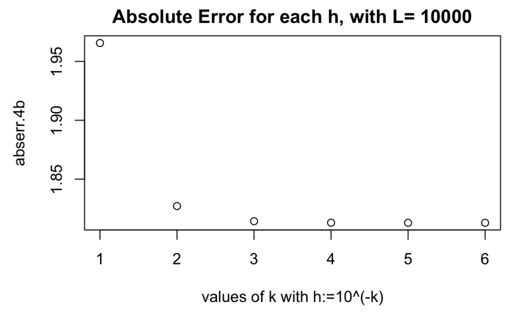

Log Error Against Stepsize h, with L= 10
log(abserr.4b, base = 10)
0.285
0.300
0.00 0.02 0.04 0.06 0.08 0.10
step size h

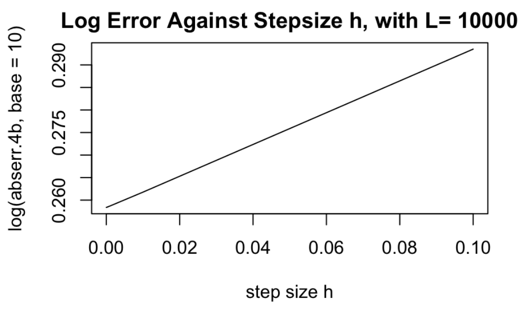

Absolute Error for each h, with L= 100
abserr.4b
1.82
1.86
1.90
1.94
1 2 3 4 5 6
values of k with h:=10^(-k)

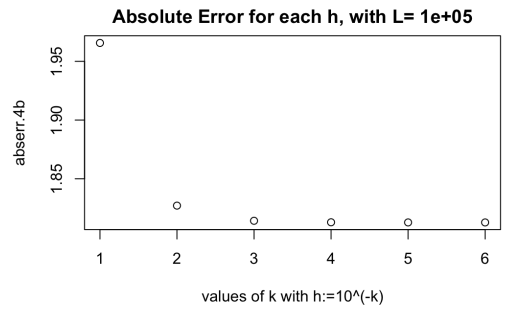

Log Error Against Stepsize h, with L= 100
log(abserr.4b, base = 10)
0.260
0.275
0.290
0.00 0.02 0.04 0.06 0.08 0.10
step size h

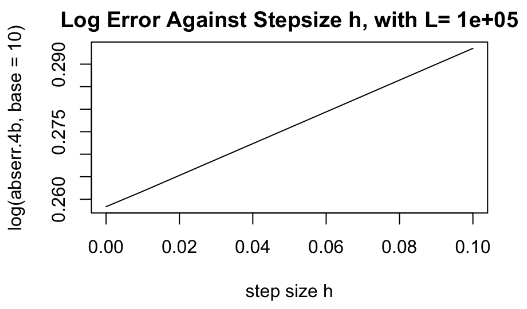

Absolute Error for each h, with L= 1000
abserr.4b
1.85
1.90
1.95
1 2 3 4 5 6
values of k with h:=10^(-k)


Log Error Against Stepsize h, with L= 1000
log(abserr.4b, base = 10)
0.260
0.275
0.290
0.00 0.02 0.04 0.06 0.08 0.10
step size h


Absolute Error for each h, with L= 10000
abserr.4b
1.85
1.90
1.95
1 2 3 4 5 6
values of k with h:=10^(-k)


Log Error Against Stepsize h, with L= 10000
log(abserr.4b, base = 10)
0.260
0.275
0.290
0.00 0.02 0.04 0.06 0.08 0.10
step size h


Absolute Error for each h, with L= 1e+05
abserr.4b
1.85
1.90
1.95
1 2 3 4 5 6
values of k with h:=10^(-k)


Log Error Against Stepsize h, with L= 1e+05
log(abserr.4b, base = 10)
0.260
0.275
0.290
0.00 0.02 0.04 0.06 0.08 0.10
step size h

$\square$

(c) Write a matlab script `ieuler` which uses the implicit euler method to solve the initial value problem with $y(0) = 2$ for $0 \leq t \leq 1$ with $L := 10^k$ for $k :=$ `1:5`. For each L, use $h := 10^{-j}$ with $j :=$ `1:6`. Tabulate the errors.

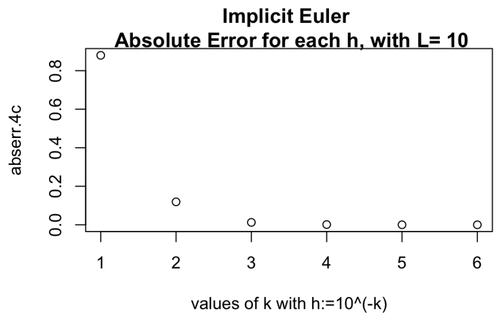
Solution. As expected, for each value of L, the smallest stepsize $h = 10^{-6}$ gives the best results. Our errors are, for `h <- 10**(-1*c(1:6))` :

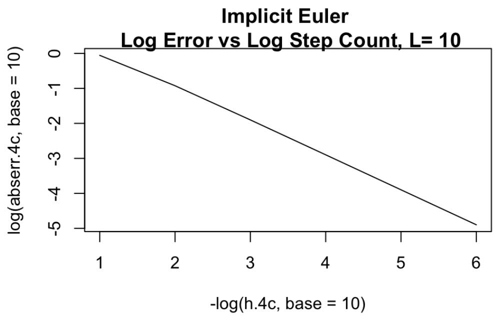
```
> # h = 10e-1,   h = 10e-2,   h = 10e-3,   h = 10e-4,   h = 10e-5,   h = 10e-6 :
 [1] 8.79621e-01 1.19086e-01 1.25677e-02 1.26388e-03 1.26460e-04 1.26469e-05 # L = 10
 [1] 2.12317e-01 9.05436e-03 6.21552e-04 5.90387e-05 5.87235e-06 5.87152e-07 # L = 10^2
 [1] 2.47184e-02 1.13853e-04 5.15422e-05 5.55761e-06 5.59851e-07 5.57860e-08 # L = 10^3
 [1] 2.51304e-03 2.32978e-05 6.36230e-06 6.76338e-07 6.80910e-08 6.57281e-09 # L = 10^4
 [1] 2.51724e-04 2.44988e-06 6.48281e-07 6.88378e-08 6.98033e-09 4.58059e-10 # L = 10^5
```

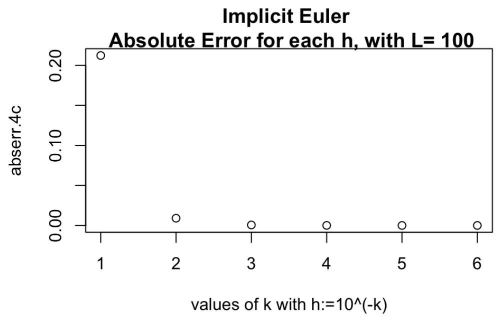
To tabulate data and generate plots, we use:

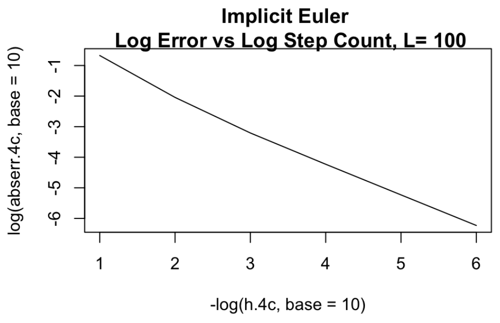
```
 ## Testing ieuler
 # same conditions as above in 4b
 ya.4c <- 2
 L.4c <- 10^c(1:5)
 h.4c <- 10^(-1*c(1:6))

 ## tabulate errors
 for (L in L.4c) {
   abserr.4c <- c()
   for (h in h.4c) {
     n <- 1/h
     t.4c <- seq(0,1,by=h)
     # create L,n-specific vibrator function
     f.vibrator <- function(t,y,r) {
       - 30 * sin(30*t) - L * (y - cos(30*t))
     }
     # df.vibrator <- function(t,y,r) {
     #   -L * (y - cos(30*t)) - 30 * sin(30*t)
     # }
     df.vibrator <- function(t,y,r) { -L }
     path.vib <- ieulerfast(0,1,ya.4c, f.vibrator, df.vibrator, r=1, n+1)
     true.4c <- exp(-L) + cos(30)
     abserr.4c <- c(abserr.4b, abs(true.4c - path.vib))
   }
   plot(abserr.4c, main=paste('Implicit Euler \n Absolute Error for each h, with L=',L), xlab='
   values of k with h:=10^(-k)')
   plot( log(h.4c) , log(abserr.4c,base=10), main=paste('Implicit Euler \n Log Error Against Log
   Stepsize h, with L=',L), type='l')
   print(abserr.4c,digits=6) # display
 }

 ieulerfast <- function(a, b, ya, f, df, r,  n) {
 h <- (b - a)/n # fixed time-step
 u <- ya # initial conditions
 t <- a # start of interval
 # tol <- 10^(-12)
 for (j in 1:n) {
   u0 <- u
   u1 <- u0 - (u0 - u - h * f(t + h, u0, r) ) /
               (1 - h * df(t + h, u0, r) )
   for (spam in 1:(3*n/4)) {
   # while (  abs(u0^2 - u1^2)  > tol ) {
     u0 <- u1
     u1 <- u0 - (u0 - u - h * f(t + h, u0, r) ) /
               (1 - h * df(t + h, u0, r) )
   # }
   u <- u1
   t <- t + h
   }
 }
 return(u) # outputs u without path
}
```

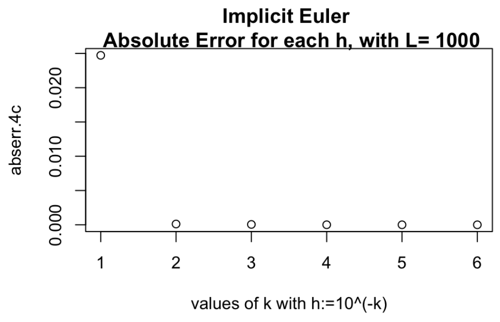
And our plots (combining per L value) are:

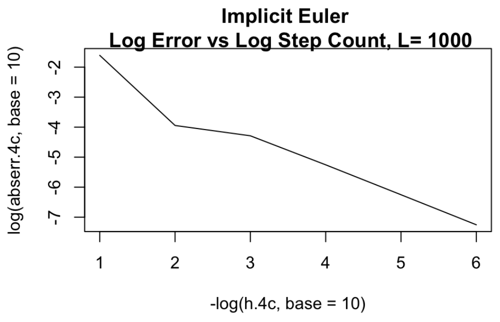


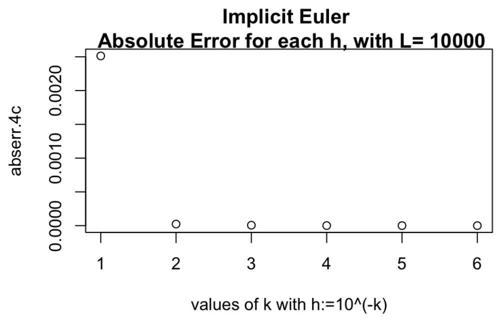


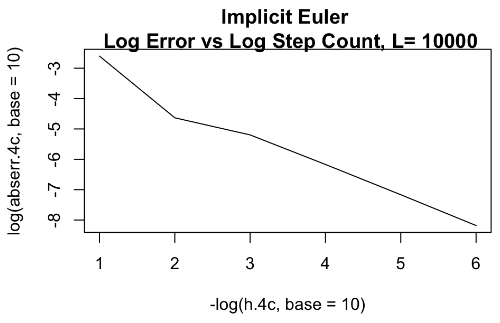


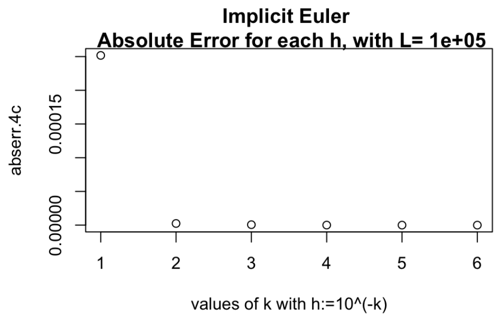


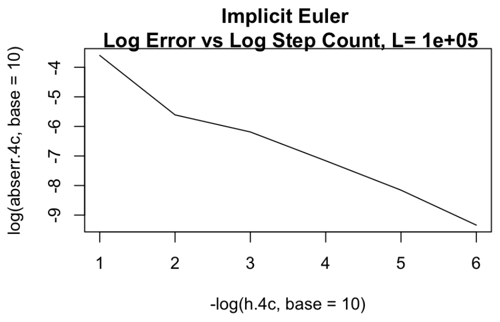












```
function [t, u] = solveinteq( a, b, kernel, rhs, p, n )
% a, b: endpoints of interval
% kernel: function handle for kernel K = kernel( t, s ) of integral equation
% rhs: function handle for right-hand side f = rhs( t, p ) of integral equation
% p: parameters for rhs
% n: number of quadrature points and weights
% t: evaluation points in [a,b]
% u: solution values at evaluation points
```

Problem 5. Write, test and debug a matlab code `solveinteq` which uses n-point Gaussian quadrature points t_i and weights w_i on $[a,b]$ (generated by `gaussint.m`) to approximate the solution $y(t)$ of the integral equations

$$y(t) + \int_a^b K(t,s)y(s)\ ds = f(t,p) \tag{3}$$

on the interval $a \le t \le b$. Your code should set up the $n \times n$ linear system

$$u_i + \sum_{j=1}^{n} K(t_i,t_j)w_j u_j = f(t_i,p) \tag{4}$$

for approximate values $u_i \approx y(t_i)$ and solve it by Gaussian elimination with partial pivoting.

Aside. Recall from Lecture 28, Thursday August 8, that Strain considers

$$u(x) + \int_0^1 \underbrace{K(x,y)}\, u(y)\ dy = g(x)$$

as a rank 1 perturbation of the identity, as opposed to the Volterra equation

$$y(t) = y(0) + \int_0^t f(s,y(s))\ ds$$

where $x_i + \sum_{j=1}^n A_{ij} = b_i$ gives $(I+A)x = b$, so that $x = b - Ax$. Strain notes that Fredholm equations are generally harder than Volterra, but this problem is about having a special kernel

$$K(x,y) = \cos(x)\sin(y)$$

so that

$$\int_0^1 \cos(x)\sin(y)\ u(y)\ dy = \cos(x)\int_0^1 \sin(y)\ u(y)\ dy.$$

(a) Suppose $[a,b]{:=}[0,1]$ and the kernel K is given by $K(t,s){:=}\cos(t)\sin(s)$. For any positive real number m, find a right-hand side $f(t,m)$ such that the exact solution $y(t)$ of the integral equation (3) above is given by $y_m(t) = \cos(mt)$.

Solution. Before solving this rank 1 perturbation problem via matrices, we analytically solve for the case $[a,b] := [0,1]$ and $K(t,s) := \cos(t)\sin(s)$, finding a RHS

$$f(t,m)$$

so that the exact solution $y(t)$ to the equation

$$y(t) + \int_0^1 \cos(t)\sin(s)y(s)\ ds = f(t,p)$$

gives $y_m(t) = \cos(mt)$.

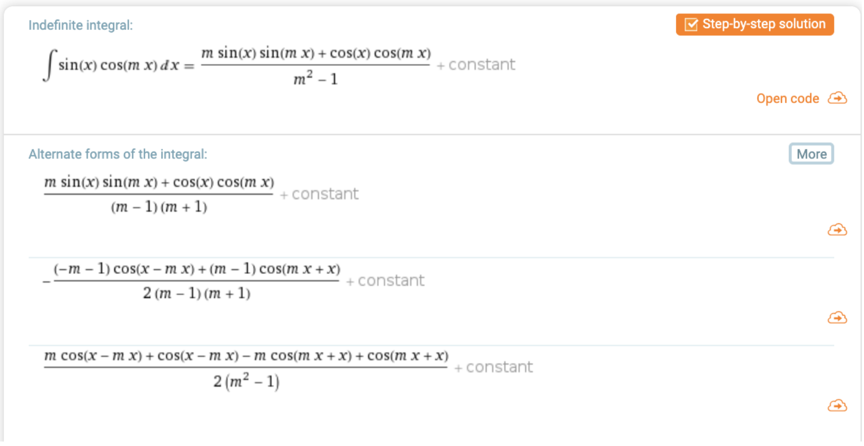
That is, setting the condition $y(t) := \cos(mt), y(s) := \cos(ms)$ into (3) above, we want some $f(t,m)$ that satisfies:

$$y(t) + \int_a^b K(t,s)y(s)\ ds = f(t,p)$$
$$\underbrace{\cos(mt)}_{y(t)} + \int_0^1 \underbrace{\cos(t)\sin(s)}_{K(t,s)}\underbrace{\cos(ms)}_{y(s)}\ ds = f(t,p)$$
$$\cos(mt) + \cos(t)\underbrace{\int_0^1 \cos(ms)\sin(s)\ ds}_{} = f(t,p),$$

where we first treat the case $m = 1$ (recall we're given $m > 0$ in the problem) and then use Wolfram Alpha (I have a small brain) to compute this integral otherwise $(m \neq 1)$:
First suppose we have positive real number $m = 1$, in which case this equation simply becomes (using $2\cos(s)\sin(s) = \sin(2s)$):

$$f(t,p) = \cos(t) + \cos(t)\int_0^1 \frac{1}{2}\sin(2s)\ ds = \cos(t) + \cos(t)\left[-\frac{1}{4}\cos(2s)\right]_0^1 = \left[1 + \frac{1-\cos(2)}{4}\right]\cos(t)$$

Now suppose $m \neq 1$:


Indefinite integral:

Step-by-step solution

$$\int \sin(x)\cos(m\,x)\,dx = \frac{m\sin(x)\sin(m\,x) + \cos(x)\cos(m\,x)}{m^2 - 1} + \text{constant}$$

Open code

Alternate forms of the integral:

More

$$\frac{m\sin(x)\sin(m\,x) + \cos(x)\cos(m\,x)}{(m-1)(m+1)} + \text{constant}$$

$$-\frac{(-m-1)\cos(x - m\,x) + (m-1)\cos(m\,x + x)}{2(m-1)(m+1)} + \text{constant}$$

$$\frac{m\cos(x - m\,x) + \cos(x - m\,x) - m\cos(m\,x + x) + \cos(m\,x + x)}{2(m^2-1)} + \text{constant}$$

Taking the first form and using the trigonometric angle-addition identity $\cos(a-b) = \cos(b-a) = \cos(a)\cos(b) - \sin(a)\sin(b)$, we have:

$$f(t,p) = \cos(mt) + \cos(t)\left[\frac{m\sin(s)\sin(ms) + \cos(s)\cos(ms)}{m^2-1}\right]_0^1$$
$$= \cos(mt) + \cos(t)\left[\frac{m\sin(1)\sin(m) + \cos(1)\cos(m)}{m^2-1} - \frac{\cancel{m\sin(0)\sin(0)}^{=0} + \cancel{\cos(0)\cos(0)}^{=1}}{m^2-1}\right]$$
$$= \cos(mt) + \left[\frac{(m-1)\sin(1)\sin(m) + \cos(m-1) - 1}{m^2-1}\right]\cos(t)$$

So in total, we conclude:

$$\boxed{f(t,p) = \begin{cases} \left[1 + \frac{1-\cos(2)}{4}\right]\cos(t), & m = 1 \\ \cos(mt) + \left[\frac{(m-1)\sin(1)\sin(m)+\cos(m-1)-1}{m^2-1}\right]\cos(t), & m \in (0,1)\cup(1,\infty) \end{cases}}$$

$\square$

(b) Solve the problem in (a) numerically by **solveinteq**, using even $n := 2, 4, 6, \ldots, 16$ and odd integers $m = 1, 3, 5, \ldots, 9$. Tabulate the errors at integration points

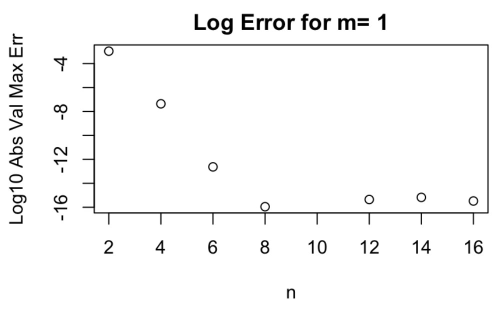
$$E_n := \max_{1 \leq 1 \leq n} |u_i - y_m(t_i)|$$

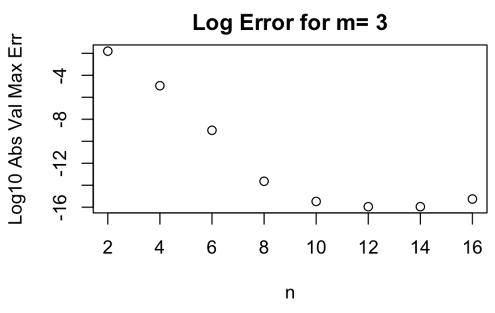
versus m and n.

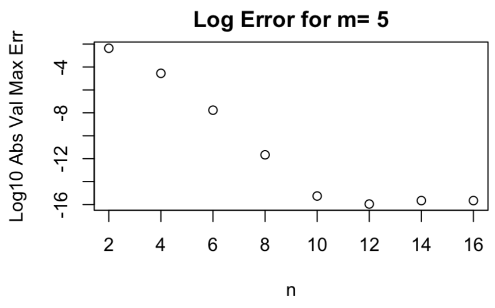
Solution. Firstly, we were instructed to use a `gaussint` from a previous homework, which has many dependencies on prior homeworks (i.e. `bisection`, `pleg`, `ectr`). Instead, we use `library('statmod')` in R for the `gauss.quad` function, which should work exactly like the `gaussint` function we developed. Tabulating errors at integration points, we have:

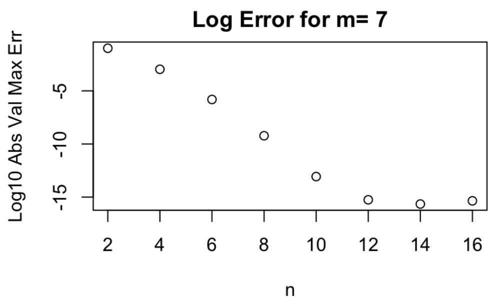
```
> errTable.5b
> # each row is m = 1,3,5,7,9
> # going down a column (fixed m) gives n = 2,4,6,8,10,12,14,16
                 [,1]              [,2]               [,3]               [,4]               [,5]
[1,] 1.08252012687e-03 1.55007669495e-02 -4.32074948065e-03 -1.03078391278e-01 -9.84097473300e-02
[2,] 4.35395013376e-08 1.11368748568e-05  2.80552542121e-05 -1.07699687069e-03 -5.24719093355e-03
[3,] 2.35034214313e-13 9.86015047388e-10  1.71740159871e-08 -1.55168634197e-06 -2.25845809637e-05
[4,] 1.11022302463e-16 2.28705943073e-14  2.17870166352e-12 -5.96305671507e-10 -2.30101551146e-08
[5,] 0.00000000000e+00 3.33066907388e-16  5.55111512313e-16 -8.52651282912e-14 -8.33000335376e-12
[6,] 4.44089209850e-16 1.11022302463e-16  1.11022302463e-16  5.55111512313e-16 -1.11022302463e-15
[7,] 6.66133814775e-16 1.11022302463e-16  2.22044604925e-16  2.22044604925e-16  3.33066907388e-16
[8,] 3.33066907388e-16 5.55111512313e-16  2.22044604925e-16  4.44089209850e-16  4.44089209850e-16
```

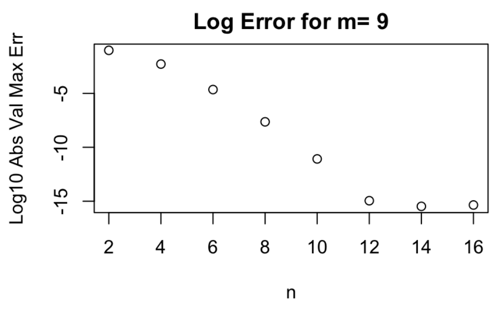
The above matrix is neat as is, but we plot for the visual appeal. Keeping m constant and varying n, we have:



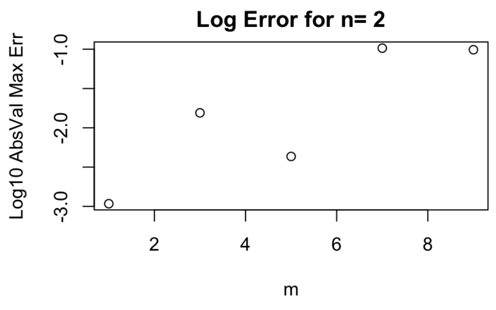


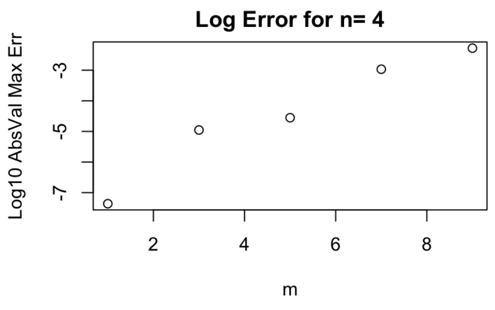


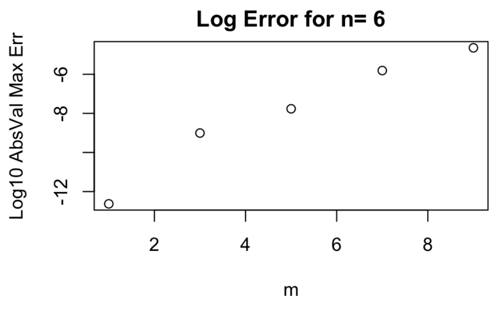


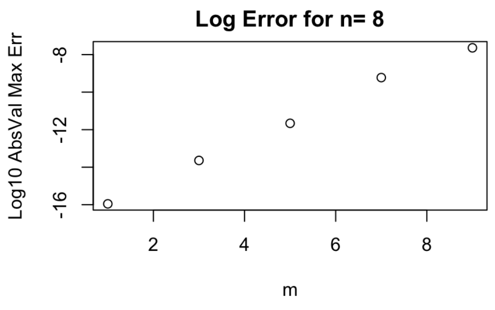


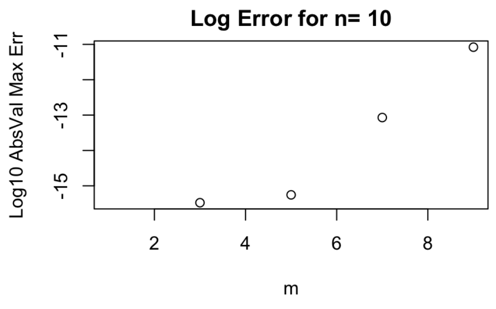
Now holding n constant and varying m, we have:

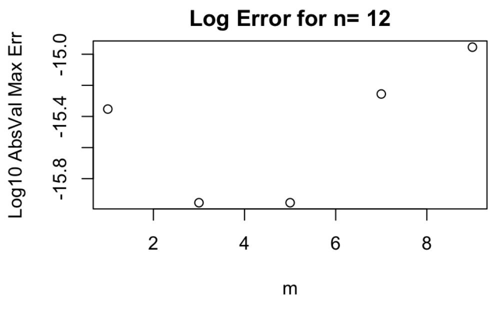


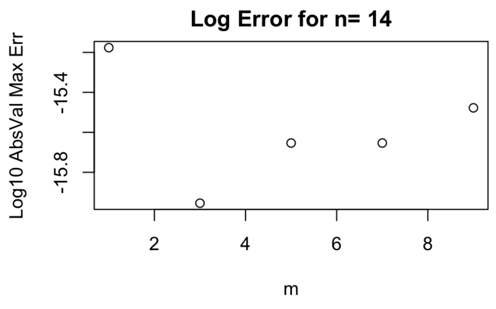


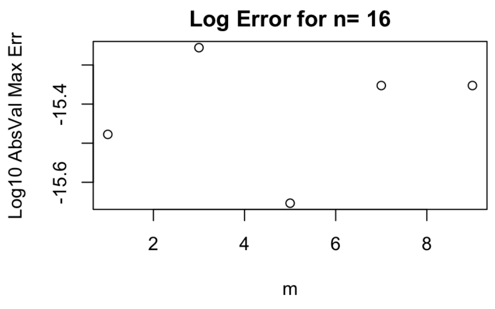












To tabulate errors and generate plots:

```
  kernel.5b <- function(t,s) {
    return(cos(t) * sin(s))
  }
  rhs.5b <- function(t,p) {
    ifelse( p == 1,
      (1 + (1 - cos(2))/4) * cos(t), # m = 1
      cos(p*t) + cos(t)* ( (p-1)*sin(1)*sin(p) + cos(p-1) - 1 )/ (p**2 - 1)
    )
  }

  getErr.5b <- function(n,p) {
    # n : number of gauss points
    # p : parameter, constant m \in \R
    tu <- solveinteq(0, 1, kernel.5b, rhs.5b, p, n)
    t <- tu[[1]] # eval points
    u <- tu[[2]] # computed soln

    true.sol <- matrix(nrow = n, ncol = 1)
    for (i in 1:n) {
      true.sol[i] <- cos( p * t[i] ) # y_m(t) := cos(mt) as dictated in part (a)
    }
    return(max( u - true.sol )) # tabulate max errors E_n
  }

  maxerr.5b <- c()
  for (n in seq(2,16,by=2)) {
    for (m in seq(1,9,by=2)) {
      maxerr.5b <- c(maxerr.5b, getErr.5b(n,m))
    }
  }

  errTable.5b <- matrix(data = maxerr.5b, nrow= 8, byrow = TRUE)

 ## plot errTable.5b
for (m in 1:5) {
  plot( seq(2,16,by=2), log(abs(errTable.5b[,m]) , base=10),
        main=paste('Log Error for m=',2*m-1), xlab='n', ylab='Log10 Abs Val Max Err' )
}

for (n in 1:8) {
  plot( seq(1,9,by=2), log(abs(errTable.5b[n,]), base=10),
        main=paste('Log Error for n=',2*n), xlab='m', ylab='Log10 AbsVal Max Err')
}
```

And our `solveinteq`:

```r
solveinteq <- function( a, b, kernel, rhs, p, n ) {
  # return ~
  #   t : evaluation points in [a,b]
  #   u : solution values at eval pts
  # a, b : endpoints of interval
  # kernel : function handle for kernel
  #   K = kernel(t,s) of integral equation
  # rhs : function handle for RHS
  #   f = rhs(t,p) of integral equation
  # p : parameters for rhs
  # n : number of quadrature points and weights

  library('statmod') # use for gauss.quad function

  t <- matrix(nrow = n, ncol = 1)
  w <- matrix(nrow = n, ncol = 1)
  B <- matrix(nrow = n, ncol = 1)
  A <- matrix(nrow = n, ncol = n)

  h <- (b-a) / 2
  m <- (a+b) / 2

  wt <- gauss.quad(n) # built-in ; not gaussint
  gauss.w <- wt$weights
  gauss.t <- wt$nodes

  for (i in 1:n) {
    w[i] <- h * gauss.w[i]
    t[i] <- m + h * gauss.t[i]
  }
  for (i in 1:n) {
    B[i] <- rhs( t[i], p )
    for (j in 1:n) {
      A[i,j] <- w[j] * kernel( t[i], t[j] )
    }
  }
  return(list(t, solve(A + diag(n),B))) # t, u
}

```

□

(c) For an arbitrary right-hand side f and the specific kernel $K(t,s) = \cos(t)\sin(s)$ in part (a) above, find a formula for the exact solution u of the linear system of equations (4).

Solution. Just as we programmed in part (b), above, here we (analytically) set up the $n \times n$ linear system:

$$u_i + \sum_{j=1}^{n} K(t_i, t_j) w_j u_j = f(t_i, p),$$

now for some arbitrary $f(t_i, p)$ and only given the kernel $K(t_i, t_j) := \cos(t_i)\sin(t_j)$ and solve for exact solution u. Recall in Lecture on Thursday, Strain gives the following approach to solving a rank-1 perturbation problem:

Let A be a rank-1 matrix, and express A as a row vector left-multiplied by a column vector. That is,

$$A = ab^T = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \begin{bmatrix} b_1 & \cdots & b_n \end{bmatrix} = \begin{bmatrix} a_1 b_1 & \cdots & a_1 b_n \\ \vdots & & \vdots \\ a_n b_1 & \cdots & a_n b_n \end{bmatrix}$$

Then let $(I + A)x = v$, so that:

$$\boxed{v = x + a\underbrace{(b^T x)}_{\text{scalar}}} \implies x = v - \underbrace{(b^T x)}\, a$$

We have an idea: let's take the dot product across the equation with b.

$$\begin{aligned} b^T x &= b^T v - (b^T x)(b^T a) \\ (1 + b^T a) b^T x &= b^T v \end{aligned}$$

and hence if $b^T a \neq -1$,

$$\boxed{b^T x = \frac{b^T v}{1 + b^T a}}.$$

To set this up (actually it turns out we reach the same conclusion independently), we set:

$$\begin{aligned} \underbrace{f(t_i, p)}_{=:v} &= u_i + \sum_{j=1}^{n} K(t_i, t_j) w_j u_j \\ &= u_i + \sum_{j=1}^{n} \left[\cos(t_i)\sin(t_j)\right] w_j u_j \\ &= \underbrace{u_i}_{=:x} + \underbrace{\left[\sum_{j=1}^{n} \sin(t_j) w_j u_j\right]}_{=:b^T x,\ \text{scalar}} \underbrace{\cos(t_i)}_{=:a}, \end{aligned}$$

where this summation gives a constant (scalar) once we know u_j. To use the above, we need the appropriate assignments. Namely, we need some b such that

$$\sum_{j=1}^{n} \sin(t_j) w_j u_j = b^T x = b^T f(t_i, p)$$

To do this, we recall that GEPP (Gaussian Elimination with Partial Pivoting) gives entries of the matrix on the main diagonal, namely $i = j$. So we write, from the above:

$$x := u_i = v - b^T x a = f(t_i, p) - \left[\sum_{j=1}^{n} \sin(t_j) w_j \underbrace{u_j}\right] \cos(t_i), \tag{5}$$

and accordingly, for $i = j$,

$$u_j = f(t_j, p) - \left[\sum_{j=1}^{n} \sin(t_j) w_j u_j\right] \cos(t_j).$$

Now, bringing this into the expression for $b^T x = \sum_{j=1}^{n} \sin(t_j) w_j u_j$, we get:

$$\begin{aligned}
b^T x &= \sum_{j=1}^{n} \sin(t_j) w_j \left[f(t_j, p) - \left(b^T x\right) \cos(t_j)\right] \\
&= \sum_{j=1}^{n} \sin(t_j) w_j f(t_j, p) - (b^T x) \sum_{j=1}^{n} \sin(t_j)\cos(t_j) w_j \\
(b^T x)\left[1 + \sum_{j=1}^{n} \frac{w_j}{2} \sin(2t_j)\right] &= \sum_{j=1}^{n} \sin(t_j) w_j f(t_j, p) \\
b^T x &= \boxed{\frac{\sum_{j=1}^{n} \sin(t_j) w_j f(t_j, p)}{1 + \frac{1}{2}\sum_{j=1}^{n} w_j \sin(2t_j)}},
\end{aligned}$$

where our trigonometric identity $\sin(2x) = 2\sin(x)\cos(x)$ actually obscured the fact that we reached the same conclusion as Strain rather than use his result. Consider:

$$b^T x = \frac{\sum_{j=1}^{n} \sin(t_j) w_j f(t_j, p)}{1 + \frac{1}{2}\sum_{j=1}^{n} w_j \sin(2t_j)} = \frac{\left[\sum_{j=1}^{n} \sin(t_j) w_j\right]^T f(t_j, p)}{1 + \left[\sum_{j=1}^{n} \sin(t_j) w_j\right]^T \cos(t_j)} = \frac{b^T v}{1 + b^T a}.$$

Now with twice the confidence $(2 \cdot 0 = 0)$, we conclude:

$$\boxed{u_i = f(t_i, p) - \frac{\sum_{j=1}^{n} \sin(t_j) w_j f(t_j, p)}{1 + \frac{1}{2}\sum_{j=1}^{n} w_j \sin(2t_j)} \cos(t_i)}$$

$\square$

(d) Use the error formula for Hermite interpolation to show that the local truncation error in (a)

$$\tau_i = y(t_i) + \sum_{j=1}^{n} [w_j K(t_i, t_j) y(t_j)] - f(t_i)$$

is bounded by

$$|\cos(t_i)| \left(\int_0^1 \prod_{i=1}^{n} (t - t_i)^2 \, dt \right) \left| \frac{v^{(2n)(\xi)}}{(2n)!} \right|$$

as $n \to \infty$, where $v(s) := \sin(s) y(s)$.

Solution. At the start of the problem, we are given:

$$\begin{aligned} y(t) + \int_a^b K(t,s) y(s) \, ds &= f(t,p), && \text{(we use this here)} \\ u_i + \sum_{j=1}^{n} K(t_i, t_j) w_j u_j &= f(t_i, p) && \text{(we use this in part (e))} \end{aligned}$$

and in part (a), we found:

$$f(t,p) = \begin{cases} \left[1 + \frac{1-\cos(2)}{4}\right] \cos(t), & m = 1 \\ \cos(mt) + \left[\frac{(m-1)\sin(1)\sin(m) + \cos(m-1) - 1}{m^2 - 1}\right] \cos(t), & m \in (0,1) \cup (1, \infty) \end{cases}$$

Now we consider for $[a,b] := [0,1]$, with $t = t_i, s = t_j$ for discretization, the local truncation error in (a) is given as:

$$\begin{aligned} \tau_i &= y(t_i) + \sum_{j=1}^{n} [w_j K(t_i, t_j) y(t_j)] - \underbrace{f(t_i, p)} \\ &= \cancel{y(t_i)} + \sum_{j=1}^{n} [w_j K(t_i, t_j) y(t_j)] - \left[\cancel{y(t_i)} + \int_a^b K(t_i, s) y(s) \, ds \right] \\ &= \sum_{j=1}^{n} [w_j \cos(t_i) \sin(t_j) y(t_j)] - \int_0^1 \cos(t_i) \sin(s) y(s) \, ds \\ &= \cos(t_i) \left[\sum_{j=1}^{n} [w_j \sin(t_j) y(t_j)] - \int_0^1 \sin(s) y(s) \, ds \right], \end{aligned}$$

where our chosen interpolation points are gaussian integration points. Recall from the Gaussian Integration handout posted online by Strain (`gauss.pdf`), 'We use Hermite interpolation to build integration rules with degree of precision $2n+1$ and points t_j'. So the bracketed expression is the interpolation error via our familiar Hermite interpolation error expression, which from our Homework 4, we proved precisely that each of the three factors in the following error estimate is inevitable:

$$\int_a^b f(x) \, dx - \sum_{j=1}^{5} u_j f(y_j) = \frac{1}{(2 \cdot 5)!} f^{(10)}(\xi) \int_a^b \prod_{i=1}^{5} (y - y_i)^2 \, dy.$$

Because we showed one example in class and now one more in a previous homework, by Strain induction, we have a theorem for all $n \in \mathbb{N}$:

Theorem.

$$\int_a^b f(x) \, dx - \sum_{j=1}^{n} u_j f(y_j) = \frac{f^{(2n)(\xi)}}{(2n)!} \int_a^b \prod_{i=1}^{n} (y - y_i)^2 \, dy$$

Using this theorem on our previous equations for local truncation error, setting $v(s) := \sin(s)y(s)$ as instructed, we have:

$$\begin{aligned}
|\tau_i| &= \left|\cos(t_i)\left(\sum_{j=1}^{n}[w_j \sin(t_j)y(t_j)] - \int_0^1 \sin(s)y(s)\ ds\right)\right| \\
&= |\cos(t_i)| \left|\sum_{j=1}^{n}[w_j v(t_j)] - \int_0^1 v(s)\ ds\right| \\
&= \boxed{|\cos(t_i)| \left|\frac{v^{(2n)}(\xi)}{(2n)!}\right| \left|\int_0^1 \prod_{i=1}^{n}(t-t_i)^2\ dt\right|},
\end{aligned}$$

which was to be shown.

□

(e) Assume that all the derivatives of the exact solution y in (a) are bounded by

$$|y^{(n)}(t)| \leq m^n$$

for some fixed $m > 0$. Use parts (d) and (c) to prove that

$$E_n \leq 2 \max_i |\tau_i| \to 0$$

as $n \to \infty$.

Solution. Let $i \in \texttt{1 : n}$. Suppose that E_n is as we defined in lecture to be a relaxed error bound:

$$E_n := \max_i |y(t_i) - u_i| .$$

Now instead of comparing against the integral, we use (as given in the beginning of the question):

$$u_i + \sum_{j=1}^{n} K(t_i, t_j) w_j u_j = f(t_i, p).$$

Expecting to not cancel $y(t_i)$ but instead get an error term $e_i := y(t_i) - u_i$, we use the given expression for local truncation error from (d) to write:

$$\begin{aligned}
\tau_i &= y(t_i) + \sum_{j=1}^{n} [w_j K(t_i, t_j) y(t_j)] - f(t_i) \\
&= y(t_i) + \sum_{j=1}^{n} w_j K(t_i, t_j) y(t_j) - \left(u_i + \sum_{j=1}^{n} w_j K(t_i, t_j) u_j \right) \\
&= y(t_i) - u_i + \sum_{j=1}^{n} w_j K(t_i, t_j)[y(t_j) - u_j] \\
&= e_i + \sum_{j=1}^{n} w_j K(t_i, t_j) e_j \qquad \Longrightarrow \boxed{e_i = y(t_i) - u_i = \tau_i - \sum_{j=1}^{n} w_j K(t_i, t_j) e_j}
\end{aligned}$$

From part (c), we showed:

$$\begin{aligned}
u_i &= f(t_i, p) - \frac{\sum_{j=1}^{n} \sin(t_j) w_j f(t_j, p)}{1 + \frac{1}{2} \sum_{j=1}^{n} w_j \sin(2t_j)} \cos(t_i) \\
&= f(t_i, p) - \frac{\sum_{j=1}^{n} \cos(t_i) \sin(t_j) w_j f(t_j, p)}{1 + \frac{1}{2} \sum_{j=1}^{n} w_j \sin(2t_j)} \\
&= f(t_i, p) - \frac{\sum_{j=1}^{n} w_j K(t_i, t_j) f(t_j, p)}{1 + \frac{1}{2} \sum_{j=1}^{n} w_j \sin(2t_j)}
\end{aligned}$$

so bringing τ_j into our boxed expression above as $f(t_i, p)$ gives:

$$e_i = \tau_i - \frac{w_j K(t_i, t_j) \tau_j}{1 + \frac{1}{2} \sum_{j=1}^{n} w_j \sin(2t_j)}$$

Now because the factors in the fraction are nonnegative (to see this, consider the weights are nonnegative, and $\forall_{t_j \in [0,1]}, \sin(t_j), \sin(2t_j) \geq 0$), we conclude that a relaxed (but valid) error bound is to take

$$|e_i| \leq |\tau_i| \leq \max_i |\tau_i|, \qquad \text{and} \qquad \left| \frac{w_j K(t_i, t_j) \tau_j}{1 + \frac{1}{2} \sum_{j=1}^{n} w_j \sin(2t_j)} \right| \leq \max_i |\tau_i|$$

Recall that we defined $E_n := \max_i |y(t_i) - u_i|$ at the beginning of the problem, so we have:

$$\begin{aligned}
E_n = \max_i |y(t_i) - u_i| &= \max_i |e_i| \\
&= \max_i \left| \tau_i - \frac{w_j K(t_i, t_j)\tau_j}{1 + \frac{1}{2}\sum_{j=1}^n w_j \sin(2t_j)} \right| \\
&\leq \max_i |\tau_i| + \max_i \left| -\frac{w_j K(t_i, t_j)\tau_j}{1 + \frac{1}{2}\sum_{j=1}^n w_j \sin(2t_j)} \right| \\
&\leq \max_i |\tau_i| + \max |\tau_i| \\
&= \boxed{2 \cdot \max_i |\tau_i|},
\end{aligned}$$

as desired.

□