PC5b - Numerical Integration

Question 1a

This following R-script can be used:

```
rm(list = ls()) # clear memory
cat("\f")       # clear screen

primitive_1a <- function(x){
  return(1/4*x^4-1/2*x^2+x)
}

library(pracma)
a <- 0          # lower integral limit
b <- 1          # upper integral limit
N <- 2          # number of subintervals
h <- (b-a)/N    # width of the subiterval
x <- seq(a,b,h)
f <- x^3-x+1
area_exact  <- rep(0,N)
area_approx <- rep(0,N)
error       <- rep(0,N)
cat(sprintf('                      Area            Area\n'))
cat(sprintf('Interval:             Exact:          Approximation:     Error:\n'))
cat(sprintf('==============================================================\n'))
for(i in 1:N){
  area_exact[i] <- primitive_1a(x[i+1])-primitive_1a(x[i])
  area_approx[i] <- 1/2*h*(f[i]+f[i+1])
  error[i] <- area_exact[i]-area_approx[i]
  cat(sprintf('(%d): (%5.2f,%.2f)   %12.8f   %12.8f   %12.8f\n',
              i,a+(i-1)*h,a+i*h,area_exact[i],area_approx[i],error[i]))
}
cat(sprintf('==============================================================\n'))
cat(sprintf('Total:              %12.8f   %12.8f   %12.8f\n',sum(area_exact),sum(area_ap
no_obs <- length(x)        # total number of x-values = N+1
w <- rep(1,no_obs)
if(no_obs>2){
  w[2:(no_obs-1)] <- 2
}
area_approx2 <- h/2*sum(w*f)
cat(sprintf('Check: h/2*sum(w*f)=%.8f\n',area_approx2))
```

```
##
```

```
                            Area                Area
## Interval:                 Exact:              Approximation:      Error:
## ==================================================================
## (1): ( 0.00,0.50)         0.39062500          0.40625000     -0.01562500
## (2): ( 0.50,1.00)         0.35937500          0.40625000     -0.04687500
## ==================================================================
## Total:                    0.75000000          0.81250000     -0.06250000
## Check: h/2*sum(w*f)=0.81250000
```

k	N	Error	Factor=error$(k-1)$/error(k)
1	2	-0.0625	*
2	4	-0.015625	4
3	8	-0.00390625	4
4	16	-0.00097656	4

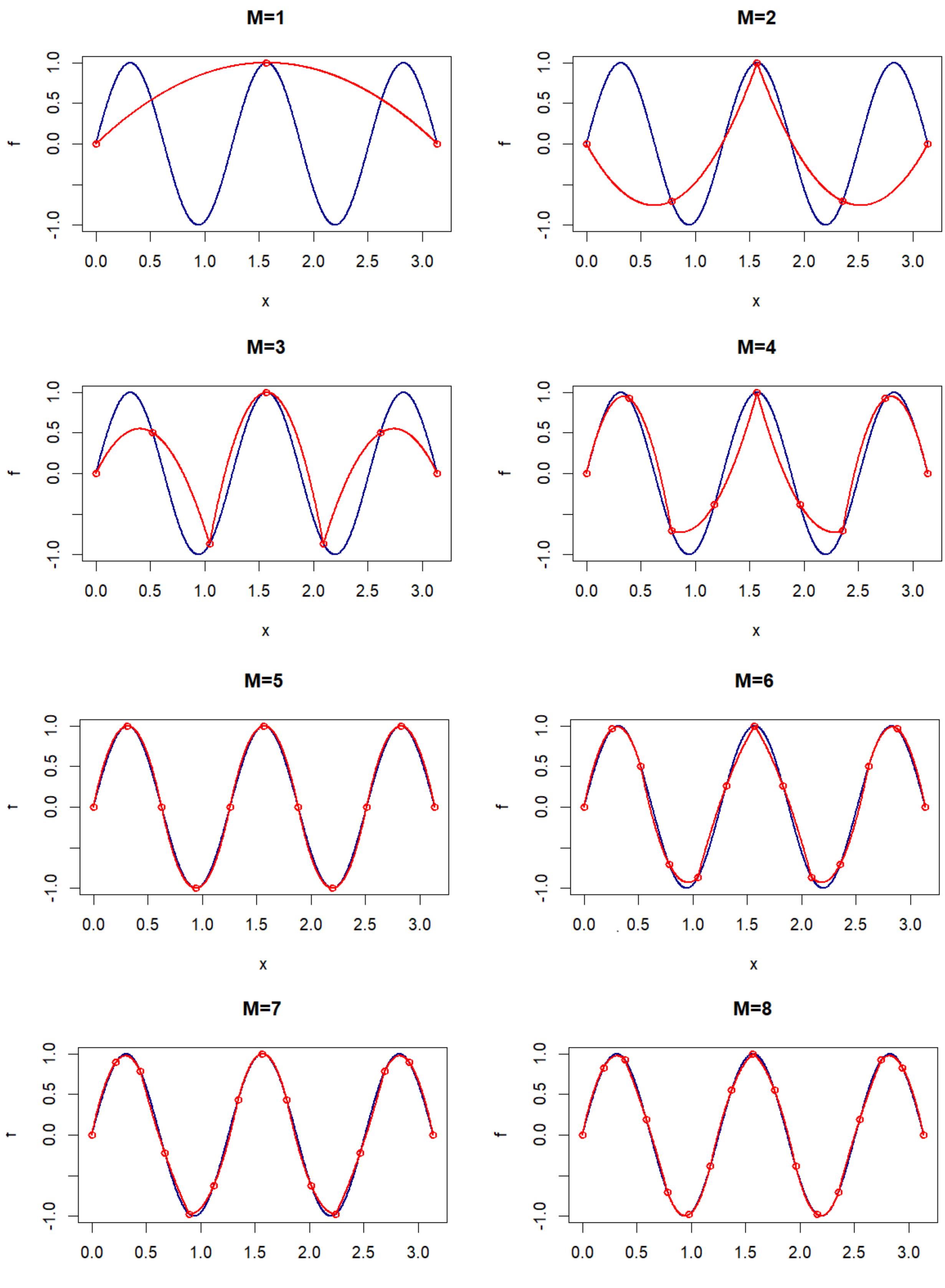
Question 1b

Next, do the same but now for $\int_1^2 e^{-\frac{1}{2}x^2}\mathrm{d}x$.

k	N	Error	Factor=error$(k-1)$/error(k)
1	2	-0.00712910	*
2	4	-0.00175737	4.0567
3	8	-0.00043782	4.0139
4	16	-0.00010936	4.0035

When h halves, the error becomes approximately 4 times smaller.

Question 2a



Question 2b

```
x <- seq(0,pi/3,length=3)
y <- sin(5*x)
polyfit(x,y,2)
```

```
## [1] -3.403222e+00  2.736853e+00 -9.436896e-16
```

So, $-3.403222x^2 + 2.736853x$. The exact area below this quadratic polynomial equals:

```
library(Ryacas)
t <- ysym("t")
# symbolic antiderivative or indefinite integral
integrate(-3.403222*t^2+2.736853*t,t)
```

```
## y: ((-3.403222)*t^3)/3+(2.736853*t^2)/2
```

```
# exact area below function for t=0,..., to pi/3
int.exact <- integrate(-3.403222*t^2+2.736853*t,t,0,pi/3)
eval(as_r(int.exact))
```

```
## [1] 0.1979162
```

This corresponds with:

```
h <- pi/6
h/3*(y[1]+4*y[2]+y[3])
```

```
## [1] 0.1979159
```

Except for rounding, the answers coincide!

Question 2c

```r
rm(list = ls()) # clear memory
cat("\f")       # clear screen

primitive_2 <- function(x){
  return(-cos(5*x)/5)
}

library(pracma)
a <- 0          # lower integral limit
b <- pi         # upper integral limit
M <- 3          # number of subintervals
NoPoints <- 2*M+1
h <- (b-a)/(NoPoints-1)   # width between two x-values (different from width subiterval
x <- seq(a,b,h)
f <- sin(5*x)
area_exact  <- rep(0,M)
area_approx <- rep(0,M)
error       <- rep(0,M)
cat(sprintf('                      Area           Area\n'))
cat(sprintf('Interval:             Exact:         Approximation:     Error:\n'))
cat(sprintf('==============================================================\n'))
for(i in 1:M){
  index <- 2*(i-1)+1
  area_exact[i] <- primitive_2(x[index+2])-primitive_2(x[index])
  area_approx[i] <- 1/3*h*(f[index]+4*f[index+1]+f[index+2])
  error[i] <- area_exact[i]-area_approx[i]
  cat(sprintf('(%d): (%5.2f,%.2f)   %12.8f   %12.8f   %12.8f\n',
              i,a+(i-1)*h,a+i*h,area_exact[i],area_approx[i],error[i]))
}
cat(sprintf('==============================================================\n'))
cat(sprintf('Total:              %12.8f   %12.8f   %12.8f\n',sum(area_exact),sum(area_ap
w <- rep(2,NoPoints)      # default weight of 2
w[1] <- w[NoPoints] <- 1  # first and last weight of 1
w[seq(2,NoPoints,2)] <-4  # 4 add even indices
area_approx2 <- h/3*sum(w*f)
cat(sprintf('Check: h/3*sum(w*f)=%.8f\n',area_approx2))
```

```
##
                      Area           Area
## Interval:             Exact:         Approximation:     Error:
## ==============================================================
## (1): ( 0.00,0.52)     0.10000000     0.19791590    -0.09791590
## (2): ( 0.52,1.05)     0.20000000     0.39583181    -0.19583181
## (3): ( 1.05,1.57)     0.10000000     0.19791590    -0.09791590
```

```
## ==============================================================
## Total:                0.40000000     0.79166361    -0.39166361
## Check: h/3*sum(w*f)=0.79166361
```

These table is given by:

M	Error	Factor = error(M–1)/error(M)
1	–1.6943951	*
2	1.35736220	–1.25
3	–0.39166361	–3.47
4	–0.05829837	6.72
5	–0.01887902	3.09
6	–0.00814552	2.32

The error is indeed getting smaller (although the factor shows a lot of variability).

Question 3

You can find the path of the spuRs package using `find.package("spuRs")`. On my computer, this gives

```
find.package("spuRs")
```

```
## [1] "C:/R/R-4.0.5/library/spuRs"
```

Please, substitute your path in the

- `source("C:/R/R-4.0.5/library/spuRs/resources/scripts/simpson_n.r")`

statement below.

```
rm(list=ls())

library(spuRs)    # for newtonraphson()
source("C:/R/R-4.0.5/library/spuRs/resources/scripts/simpson_n.r")

phi <- function(x) return(exp(-x^2/2)/sqrt(2*pi))

Phi <- function(z) {
  if (z < 0) {
    return(0.5 - simpson_n(phi, z, 0))
  } else {
```

```r
    return(0.5 + simpson_n(phi, 0, z))
  }
}

newtonraphson(function(z) c(Phi(z) - 0.5, phi(z)), 0)
```

```
## Algorithm converged

## [1] 0
```

```r
newtonraphson(function(z) c(Phi(z) - 0.95, phi(z)), 0)
```

```
## At iteration 1 value of x is: 1.127983
## At iteration 2 value of x is: 1.505239
## At iteration 3 value of x is: 1.630773
## At iteration 4 value of x is: 1.644693
## At iteration 5 value of x is: 1.644854
## At iteration 6 value of x is: 1.644854
## Algorithm converged

## [1] 1.644854
```

```r
newtonraphson(function(z) c(Phi(z) - 0.975, phi(z)), 0)
```

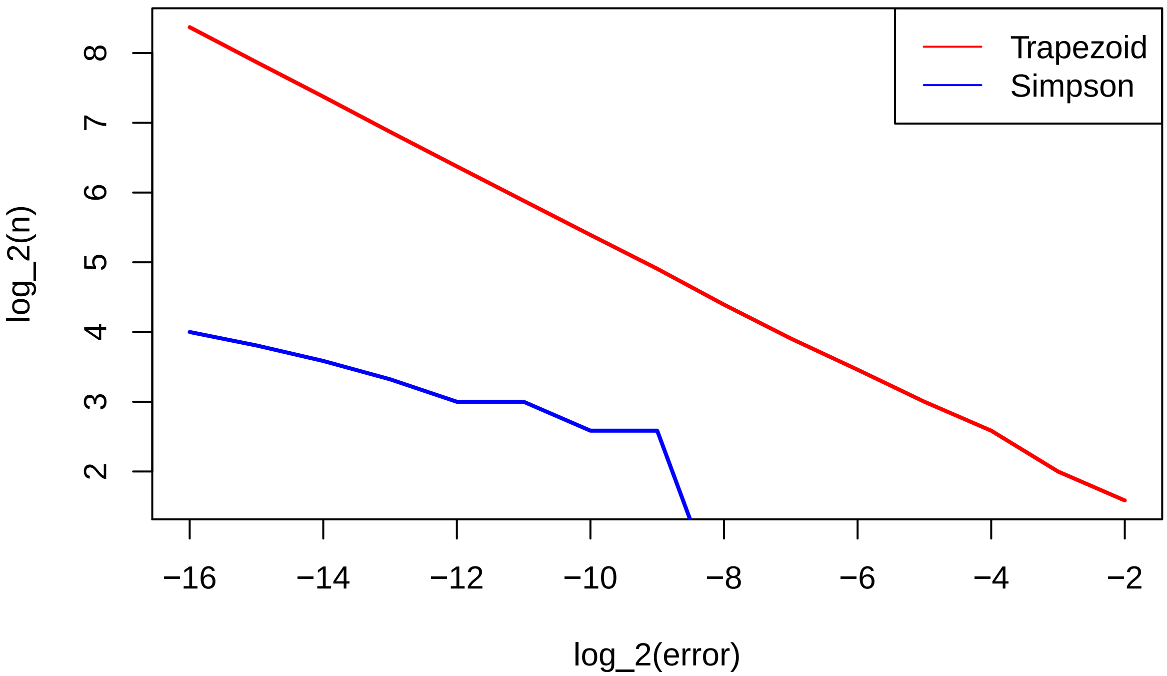
```
## At iteration 1 value of x is: 1.190648
## At iteration 2 value of x is: 1.658624
## At iteration 3 value of x is: 1.892671
## At iteration 4 value of x is: 1.955809
## At iteration 5 value of x is: 1.959947
## At iteration 6 value of x is: 1.959964
## Algorithm converged

## [1] 1.959964
```

```r
newtonraphson(function(z) c(Phi(z) - 0.99, phi(z)), 0)
```

```
## At iteration 1 value of x is: 1.228248
## At iteration 2 value of x is: 1.759464
## At iteration 3 value of x is: 2.104157
## At iteration 4 value of x is: 2.280355
## At iteration 5 value of x is: 2.324003
## At iteration 6 value of x is: 2.326342
## At iteration 7 value of x is: 2.326348
## Algorithm converged

## [1] 2.326348
```

Question 4

```r
rm(list = ls()) # clear the workspace

source("C:/R/R-4.0.5/library/spuRs/resources/scripts/trapezoid.r")
source("C:/R/R-4.0.5/library/spuRs/resources/scripts/simpson_n.r")

# test function
# we require that the integral from 0 to 1 is 1
ftn <- function(x){ 5*x^4 }

Trap <- function(n){ trapezoid(ftn, 0, 1, n) }
Simp <- function(n){ simpson_n(ftn, 0, 1, n) }

e_vec <- 2^(-2:-16)           # errors to achieve
n_T <- rep(0, length(e_vec))  # partition sizes for trapezoid
n_S <- rep(0, length(e_vec))  # partition sizes for Simpson
for (i in 1:length(e_vec)) {
  e <- e_vec[i]
  n <- 1
  while(abs(Trap(n)-1) > e) { n <- n+1 }
  n_T[i] <- n
  n <- 1
  while(abs(Simp(n)-1) > e) { n <- n+1 }
  n_S[i] <- n
}

plot(log(e_vec,2), log(n_T,2), type="l",
     xlab="log_2(error)", ylab="log_2(n)", col="red", lwd=2)
lines(log(e_vec,2), log(n_S,2), col="blue", lwd=2)
legend("topright", c("Trapezoid", "Simpson"), col=c("red", "blue"), lty=1)
```



We clearly see that Simpson 1/3-rule requires much less subintervals than the trapedoidal rule.