Problem 4

a)

```r
# truncated power base function
# when added to model for a cubic polynomial this
# keeps f continuous for 1st and 2nd derivatives at each knot
# x is predictor
# z is knot

h <- function(x,z) {
   ifelse(x>z,(x-z)^3,0)
}
```

b)

```r
#
# xs is vector of predictors
# h is a matrix of truncated power base functions

hs <- function(xs,z) {
    sapply(xs,h, z=z)
}
```

c)

```r
#
# xs is predictors
# zs is knots

splinebasis <- function(xs,zs) {
   mhs <- matrix(data=0,nrow=length(xs),ncol=length(zs))
   for(j in 1:length(zs)){
       mhs[,j] = sapply( xs, hs, zs[j])
   }
   m <- cbind(xs, xs^2, xs^3, mhs)
}
```

d)

```r
set.seed(2019)
x = runif(100)
y = sin(10*x)
```

e)

```r
knots=c(1/4,2/4,3/4)

out3 = lm(y~I(splinebasis(x,knots)))

print(out3)
```

```
##
```

```
## Call:
## lm(formula = y ~ I(splinebasis(x, knots)))
##
## Coefficients:
##                 (Intercept)  I(splinebasis(x, knots))xs
##                     -0.1197                     16.2295
##    I(splinebasis(x, knots))    I(splinebasis(x, knots))
##                    -68.1331                     56.4607
##    I(splinebasis(x, knots))    I(splinebasis(x, knots))
##                     62.9915                   -284.9395
##    I(splinebasis(x, knots))
##                    259.4466
```

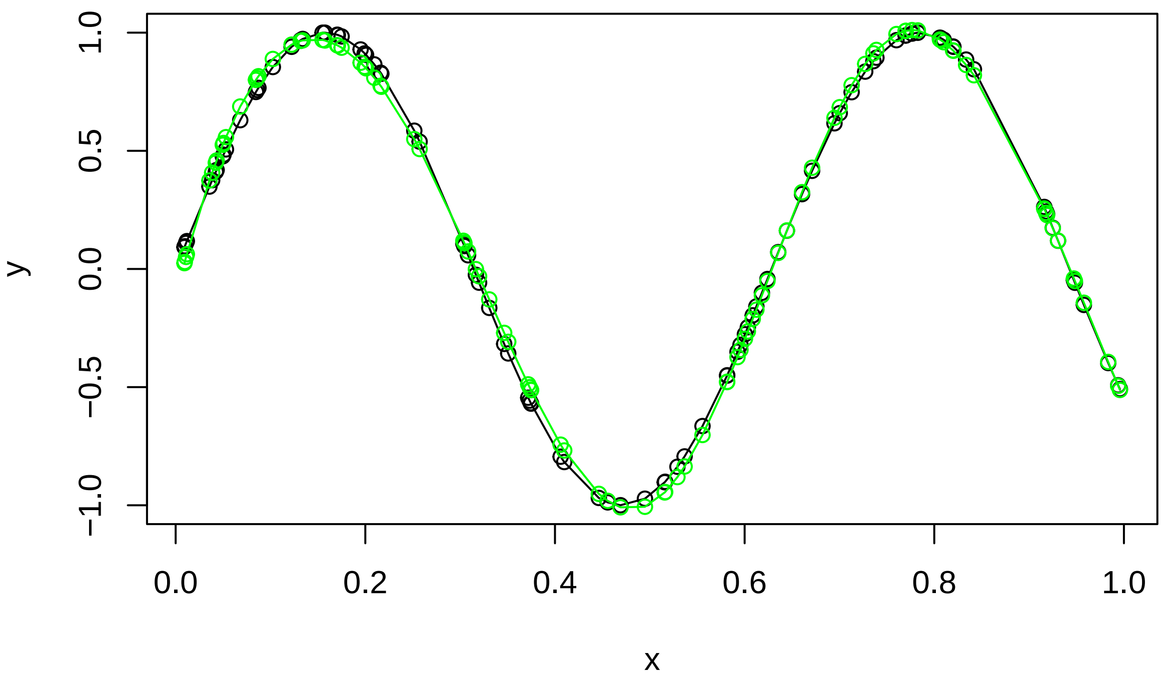
```r
yhat <- predict(out3)

reorder <- order(x)

plot(x,y)

lines(x[reorder],y[reorder])

points(x,yhat, col='green')
lines(x[reorder],yhat[reorder], col='green')
```



The spline (in green) does an OK job of approximating the function. The green line is fairly close to the black line but the spline is a bit low both on the left maximum and the central minimum.

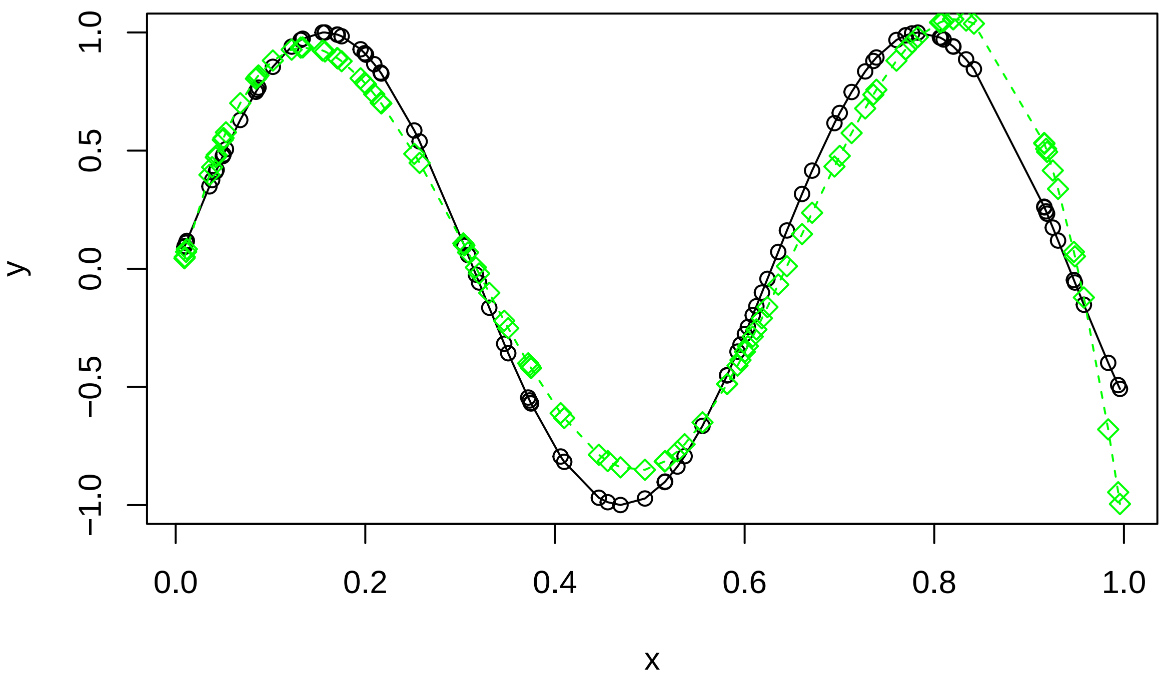
f)

```r
k1=c(1/2)
k2=c(1/3,2/3)
k3=c(1/4,2/4,3/4)
k4=c(1/5,2/5,3/5,4/5)
k5=c(1/6,2/6,3/6,4/6,5/6)
```

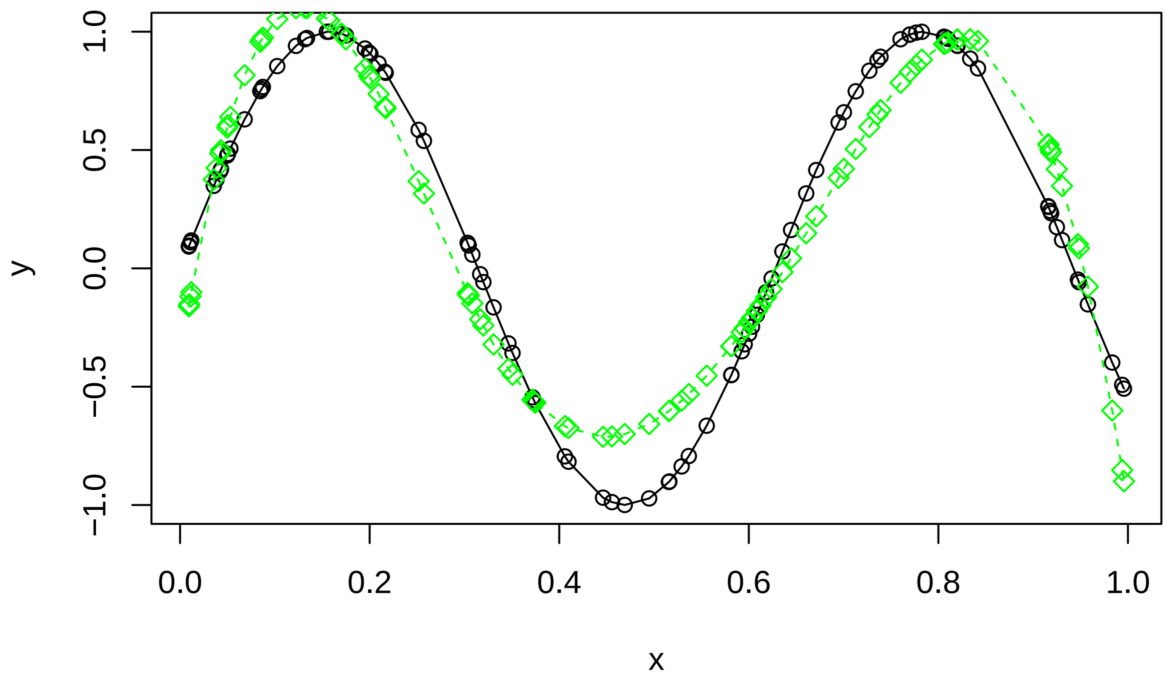
```r
k6=c(1/7,2/7,3/7,4/7,5/7,6/7)
k7=c(1/8,2/8,3/8,4/8,5/8,6/8,7/8)
k8=c(1/9,2/9,3/9,4/9,5/9,6/9,7/9,8/9)
k9=c(1/10,2/10,3/10,4/10,5/10,6/10,7/10,8/10,9/10)

runner <- function (k) {
  m <- lm(y~I(splinebasis(x,k)))
  yhat <- predict(m)
  reorder <- order(x)
  plot(x,y)
  lines(x[reorder],y[reorder])
  points(x,yhat, col='green',pch=5)
  lines(x[reorder],yhat[reorder], col='green',lty=2)
}
```

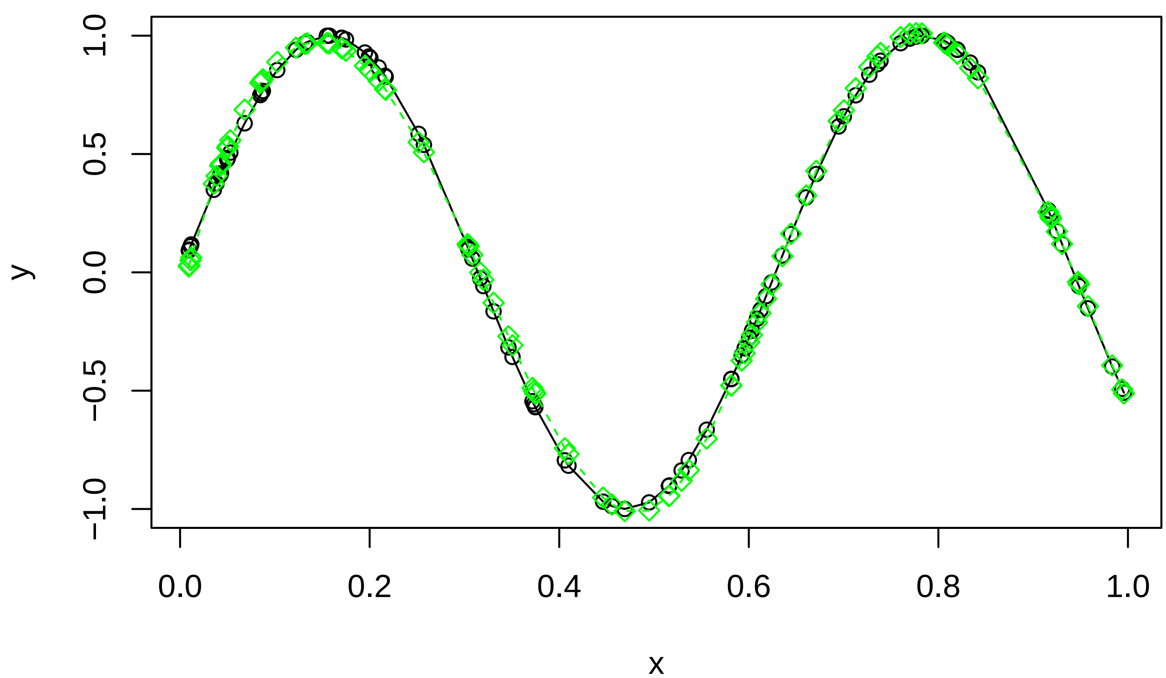
```r
o1 <- runner(k1)
```



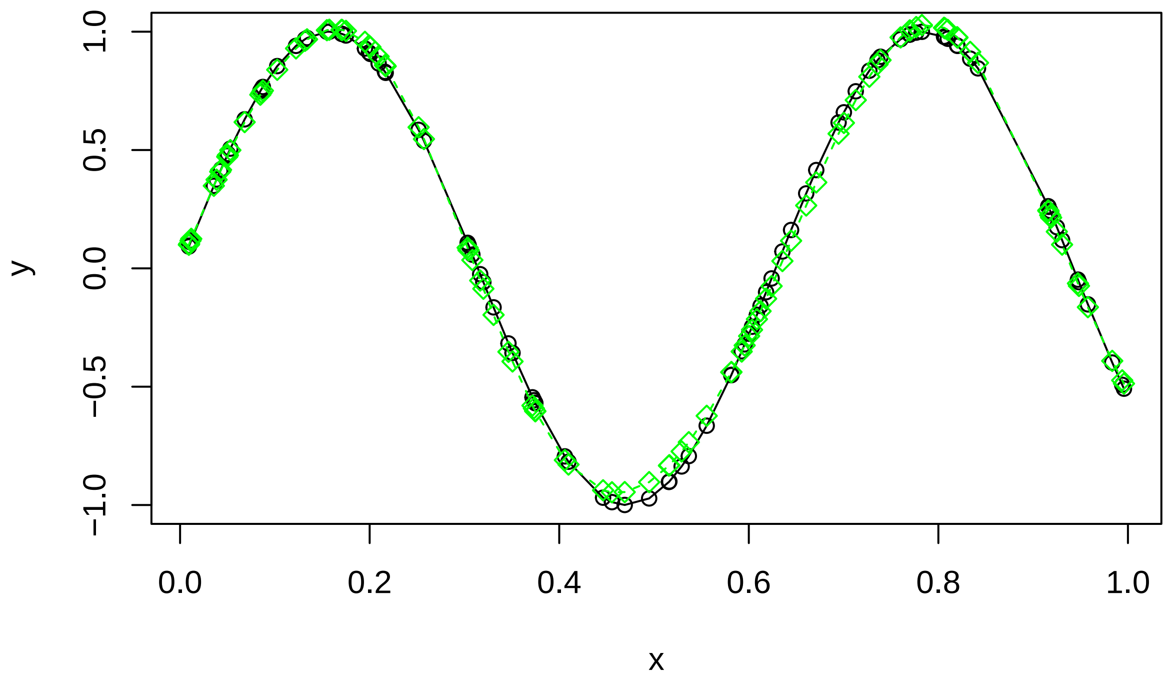

```r
o2 <- runner(k2)
```



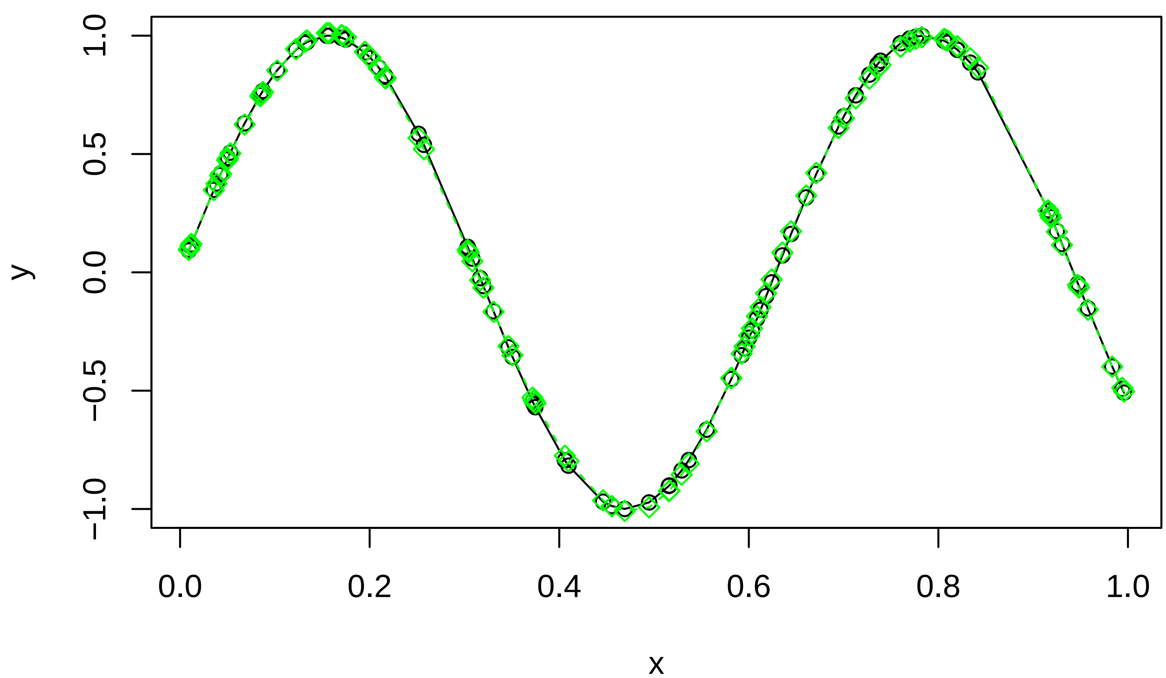

```
o3 <- runner(k3)
```



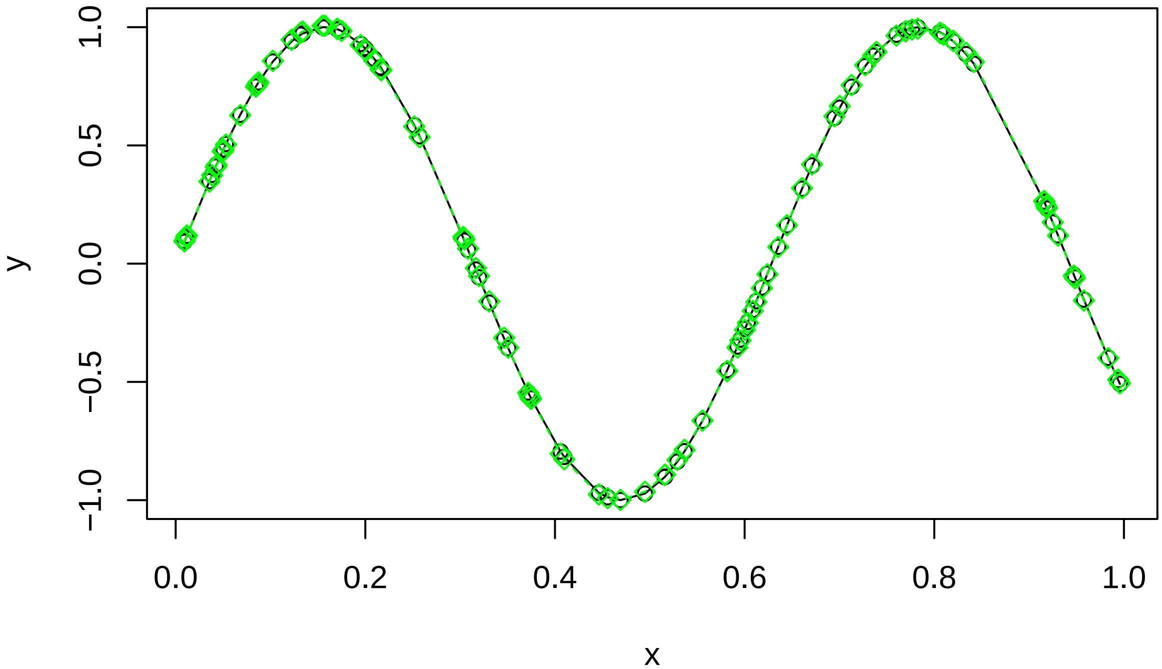

```
o4 <- runner(k4)
```



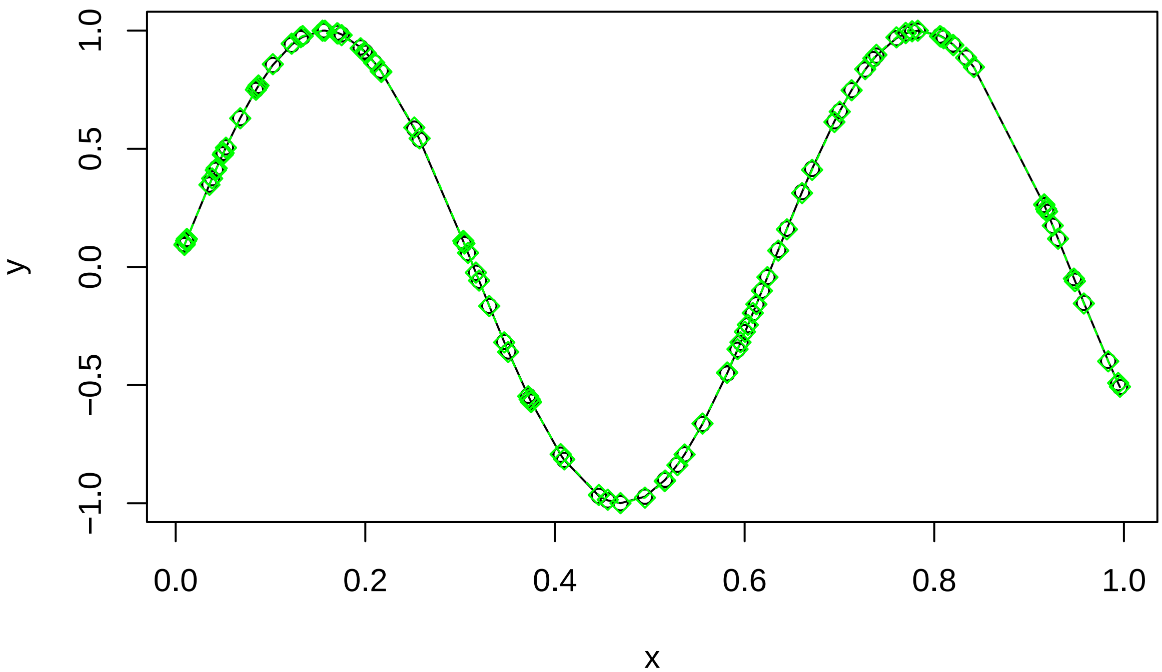

```r
o5 <- runner(k5)
```



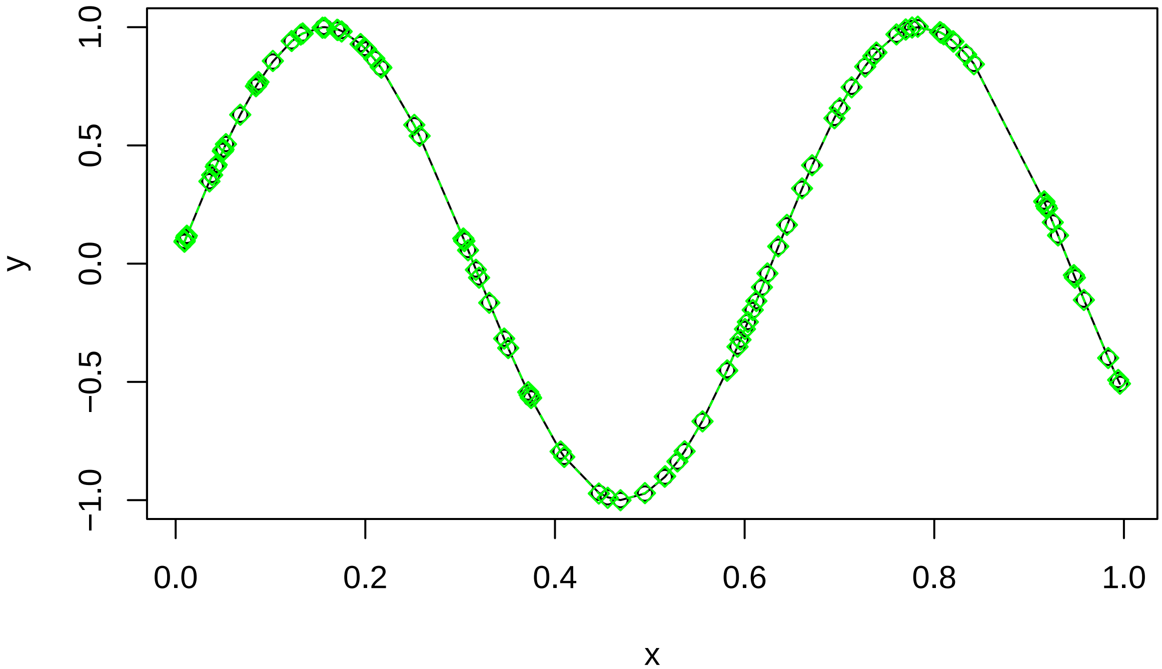

```r
o6 <- runner(k6)
```



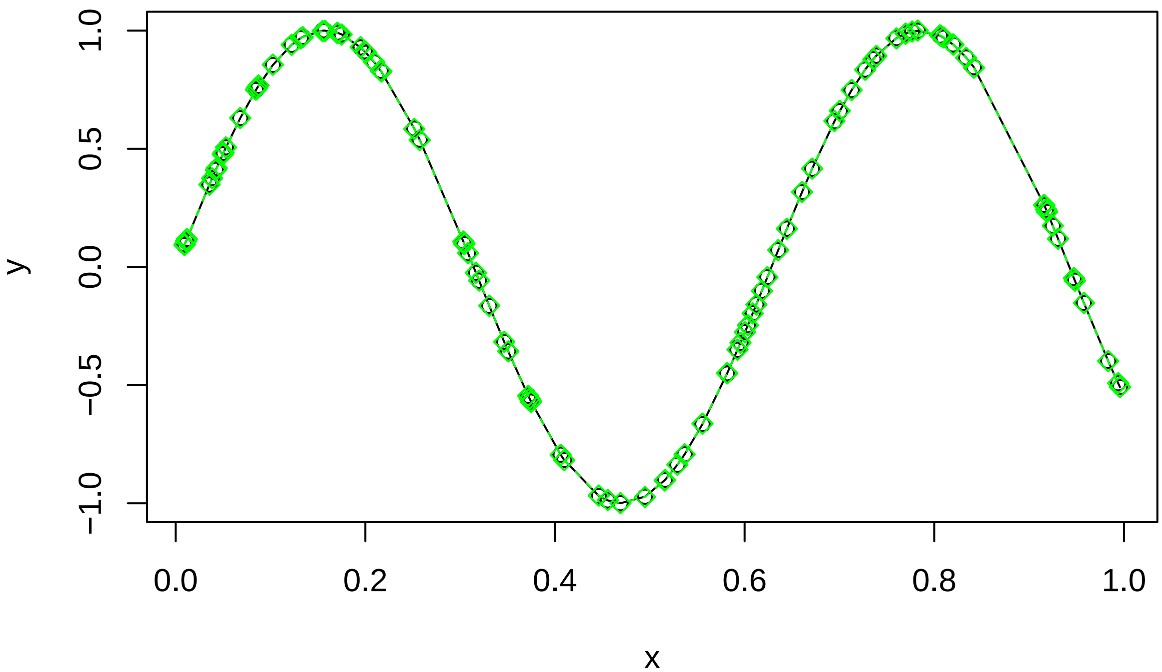

```
o7 <- runner(k7)
```




```
o8 <- runner(k8)
```




```r
o9 <- runner(k9)
```



g)

Yes, as the number of knots increases we expect the curve to become a better approximation of the true curve because it is increasingly flexible and better able to match the true data. However, in the real world when the true data is unknown, increasing the knots would likely lear to overfiting. Picking a good k could be done by using a validation or cross-validation approach.