②

a) Let $\mathcal{F} = \text{span}\{h_1(x), h_2(x), \dots, h_D(x)\}$. By assumption if $S(x) \in \mathcal{F}$

then $S(x) = \sum_{j=1}^{D} \alpha_j h_j(x)$ ⊛

We want $\min\limits_{S(x)\in\mathcal{F}} \sum_{i=1}^{N} (y_i - S(x^{(i)}))^2$, for $(y_i, x^{(i)})$, $y_i \in \mathbb{R}$, $x^{(i)} \in \mathbb{R}^n$

Because of ⊛ $\min\limits_{S(x)\in\mathcal{F}} \sum_{i=1}^{N} (y_i - S(x^{(i)}))^2 = \min\limits_{\vec{\alpha}\in\mathbb{R}^D} \sum_{i=1}^{N} (y_i - \sum_{j=1}^{D} \alpha_j h_j(x^{(i)}))^2 =$

Let $r_i := y_i - \sum_{j=1}^{D} \alpha_j h_j(x)$.

Then:

$$\vec{r} := \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix} = \begin{bmatrix} y_1 - \alpha_1 h_1(x^{(1)}) - \dots - \alpha_D h_D(x^{(1)}) \\ y_2 - \alpha_1 h_1(x^{(2)}) - \dots - \alpha_D h_D(x^{(2)}) \\ \vdots \\ y_N - \alpha_1 h_1(x^{(N)}) - \dots - \alpha_D h_D(x^{(N)}) \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{y} - \underbrace{\begin{bmatrix} h_1(x^{(1)}) & \dots & h_D(x^{(1)}) \\ h_1(x^{(2)}) & \dots & h_D(x^{(2)}) \\ \vdots & & \\ h_1(x^{(N)}) & \dots & h_D(x^{(N)}) \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_D \end{bmatrix}}_{\vec{\alpha}}$$

And so $\min\limits_{S(x)\in\mathcal{F}} \sum_{i=1}^{N} (y_i - S(x^{(i)}))^2 = \min\limits_{\vec{\alpha}\in\mathbb{R}^D} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{D} \alpha_j h_j(x^{(i)})\right)^2 =$

$$\min_{\vec{\alpha}\in\mathbb{R}^D} \sum_{i=1}^{N} r_i^2 = \min_{\vec{\alpha}\in\mathbb{R}^D} \|r\|^2 = \min_{\vec{\alpha}\in\mathbb{R}^D} \|y - B\alpha\|^2 = \min_{\alpha\in\mathbb{R}^D} (y - B\alpha)(y - B\alpha)$$

$$= \min_{\alpha\in\mathbb{R}^D} y^T y - 2y^T B\alpha + \alpha^T B^T B \alpha = \min_{\alpha\in\mathbb{R}^D} f(\alpha)$$

$f(\alpha)$ is a quadratic and from lecture 1 we have a closed form solution for its critical point: $\alpha^* = (\beta^T\beta)^{-1}\beta^T y$

Since $x^T \beta^T \beta x = (\beta x)^T \beta x = \|\beta x\|^2 \geq 0 \Rightarrow$ $\beta^T\beta$ is positive semi-definite, α^* is a minimum (if you can invert $\beta^T\beta$, only if positive definite). So the soln to the minimization problem is $\alpha^* = (\beta^T\beta)^{-1}\beta^T y$

b) $S(x)$ will be defined as: S_1

$$S(x) = \begin{cases} a_0 + b_0 x + c_0 x^2 + d_0 x^3 \; (S_1) & \text{if } x < 15 \\ a_1 + b_1 x + c_1 x^2 + d_1 x^3 \; (S_2) & \text{if } 15 \leq x < 20 \\ a_2 + b_2 x + c_2 x^2 + d_2 x^3 \; (S_3) & \text{if } x \geq 20 \end{cases}$$

with

$$S_1(15) = S_2(15) \qquad \text{and} \qquad S_2(20) = S_3(20)$$

$$S_1'(15) = S_2'(15) \qquad S_2'(20) = S_3'(20)$$

$$S_1''(15) = S_2''(15) \qquad S_2''(20) = S_3''(20)$$

D is the # of h_i fonctions, or since the h_i functions are the basis functions for the linear function space $\mathcal{F}$, $D = \dim(\mathcal{F})$.

$\mathcal{F}$ consists of those splines $S(x)$ for which its parameters satisfy:

$$\underbrace{\begin{bmatrix} 1 & 15 & 15^2 & 15^3 & 1 & 15 & 15^2 & 15^3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2\cdot 15 & 3\cdot 15^3 & 0 & 1 & 2\cdot 15 & 3\cdot 15^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6\cdot 15 & 0 & 0 & 2 & 6\cdot 15 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 20 & 20^2 & 20^3 & 1 & 20 & 20^2 & 20^3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2\cdot 20 & 3\cdot 20^3 & 0 & 1 & 2\cdot 20 & 3\cdot 20^3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 6\cdot 20 & 0 & 0 & 2 & 6\cdot 20 \end{bmatrix}}_{M} \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ d_0 \\ a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

i.e the parameter vector belongs to the kernel of M

Since there is a one-to-one mapping b/w $\mathrm{Ker}(M)$ and the $\mathcal{F}$ and $\dim(\mathrm{Ker}(M)) + \min(\mathrm{row}(M), \mathrm{col}(M)) = \max(\mathrm{row}(M), \mathrm{col}(M))$

we have that $\dim(\mathcal{F}) = \dim(\mathrm{Ker}(M)) = \max(6, 12) - \min(6, 12)$

$= 12 - 6 = 6$ $\boxtimes$

c) Lets first show (i):

A spline for two knots will be of the form ($\xi_1 = 15$, $\xi_2 = 20$)

$$s(x) = \begin{cases} a_0 + b_0 x + c_0 x^2 + d_0 x^3 & \text{if } x < \xi_1 \\ a_1 + b_1 x + c_1 x^2 + d_1 x^3 & \text{if } \xi_1 \le x < \xi_2 \\ a_2 + b_2 x + c_2 x^2 + d_2 x^3 & \text{if } \xi_2 \le x \end{cases}$$

- So $h_1(x)$ is of the form $s(x)$ since we can rewrite it as $a_0 = a_1 = a_2 = 1$ the rest of parameters are 0.
- $h_2(x)$ is of the form $s(x)$ since $b_0 = b_1 = b_2 = 1$ and the rest of parameters are 0
- $h_3(x)$ is of the form $s(x)$ since $c_0 = c_1 = c_2 = 1$ and the rest of parameters are 0
- $h_4(x)$ is of the form $s(x)$ since $d_0 = d_1 = d_2 = 1$ and the ref of parameters are 0
- $h_5(x)$ is of the form $s(x)$ since we can rewrite it as: $a_0 = b_0 = c_0 = d_0 = 0$, $a_1 = b_1 = c_1 = 0$ and $d_1 = 1$ with $x \mapsto x - \xi_1$ $a_2 = b_2 = c_2 = 0$ and $d_2 = 1$, with $x \mapsto x - \xi_1$
- $h_6(x)$ is of the form $s(x)$ since: $a_0 = b_0 = c_0 = d_0 = 0$, $a_1 = b_1 = c_1 = d_1 = 0$, $a_2 = b_2 = c_2 = 0$ and $d_2 = 1$ with $x \mapsto x - \xi_2$

So each $h_i(x)$ is a spline for two knots (in this case $\xi_1 = 15$, $\xi_2 = 20$).

Now lets show (ii):

consider $p(x) = c_1 h_1(x) + c_2 h_2(x) + c_3 h_3(x) + c_4 h_4(x) + c_5 h_5(x) + c_6 h_6(x)$

$$= c_1 + c_2 x + c_3 x^2 + c_4 x^3 + c_5 (x - \xi_1)_+^3 + c_6 (x - \xi_2)_+^3$$

15 20

To show linear independence: If $p(x)$ is the 'zero' function (i.e $p(x) = 0 \ \forall x$) then $c_i = 0$ for $i = 1, 2, \ldots, 6$

Sps $p(x) = 0$

Consider $x < 15$, then $p(x) = c_1 + c_2 x + c_3 x^2 + c_4 x^3 = 0$.

Now suppose $c_i \neq 0$ for some i, then $p(x)$ has a root on each $x \in (-\infty, 15)$. Not possible since $p(x)$ has at most 3 roots

$\Rightarrow c_1 = c_2 = c_3 = c_4 = 0$

consider $15 \le x < 20$, then $p(x) = c_1 + c_2 x + c_3 x^2 + c_4 x^3 + c_5 (x - 15)^3$

$p(x)$ is again a cubic polynomial on $[15, 20)$ and so the argument given for the case $x \in (-\infty, 15)$ applies here as well and so $c_1 = c_2 = c_3 = c_4 = c_5 = 0$

Now consider the last case $x \in [20, +\infty)$ then

$P(x) = c_1 + c_2 x + c_3 x^2 + c_4 x^3 + c_5 (x-15)^3 + c_6 (x-20)^3 = 0$

Again $p(x)$ is a cubic polynomial and the same argument applies: $c_1 = c_2 = c_3 = c_4 = c_5 = c_6 = 0$ o/w $p(x)$ has infinitey many roots (and we know it has up to 3)

So, the $h_i(x)$ are linearly independent. They can only be combined to form the 0 function if the coefficients are all 0.

Now lets show (iii):

From b) we know the spline space for 2 knots has dimension 6. In (ii) I showed a set of 6 function that are linearly indep and in (i) I showed that these six functions are splines of two knots. Therefore $\{1, x, x^2, x^3, (x-15)^3_+, (x-20)^3_+\}$ form a basis for the spline space for 2 knots.

In other words, any spline for two knots is a linear combination of $\{1, x, x^2, x^3, (x-15)^3_+, (x-20)^3_+\}$

$\blacksquare$

HW11

Ines Pancorbo

3/31/2020

2) d.

```r
data <- read.csv("BoneMassData.txt", header = TRUE, sep = ' ', stringsAsFactors = TRUE)

# only working with female
data_f <- data[which(data$gender == "female"),]
```

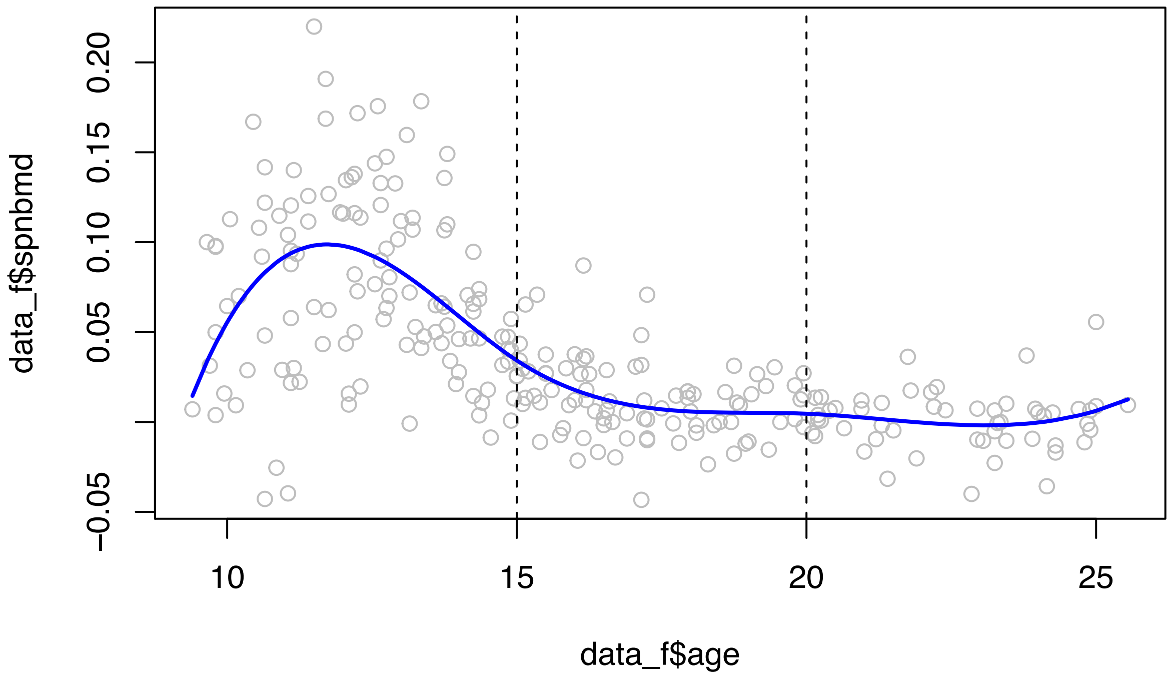
First use a library:

```r
library(splines)

sr <- lm(spnbmd ~ bs(age, knots = c(15, 20)), data=data_f)
summary(sr)
```

```
##
## Call:
## lm(formula = spnbmd ~ bs(age, knots = c(15, 20)), data = data_f)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.132645 -0.017303 -0.000668  0.016140  0.121739
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 0.0145503  0.0132424   1.099   0.2729
## bs(age, knots = c(15, 20))1  0.1524651  0.0236308   6.452 5.58e-10 ***
## bs(age, knots = c(15, 20))2 -0.0242530  0.0146719  -1.653   0.0996 .
## bs(age, knots = c(15, 20))3  0.0008764  0.0229421   0.038   0.9696
## bs(age, knots = c(15, 20))4 -0.0270585  0.0209693  -1.290   0.1981
## bs(age, knots = c(15, 20))5 -0.0019826  0.0238397  -0.083   0.9338
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03531 on 253 degrees of freedom
## Multiple R-squared:  0.5205, Adjusted R-squared:  0.5111
## F-statistic: 54.94 on 5 and 253 DF,  p-value: < 2.2e-16
```

```r
# plotting data and spline
plot(data_f$age, data_f$spnbmd,
     col = "grey")

x <- data_f$age[order(data_f$age)]
points(x,
```

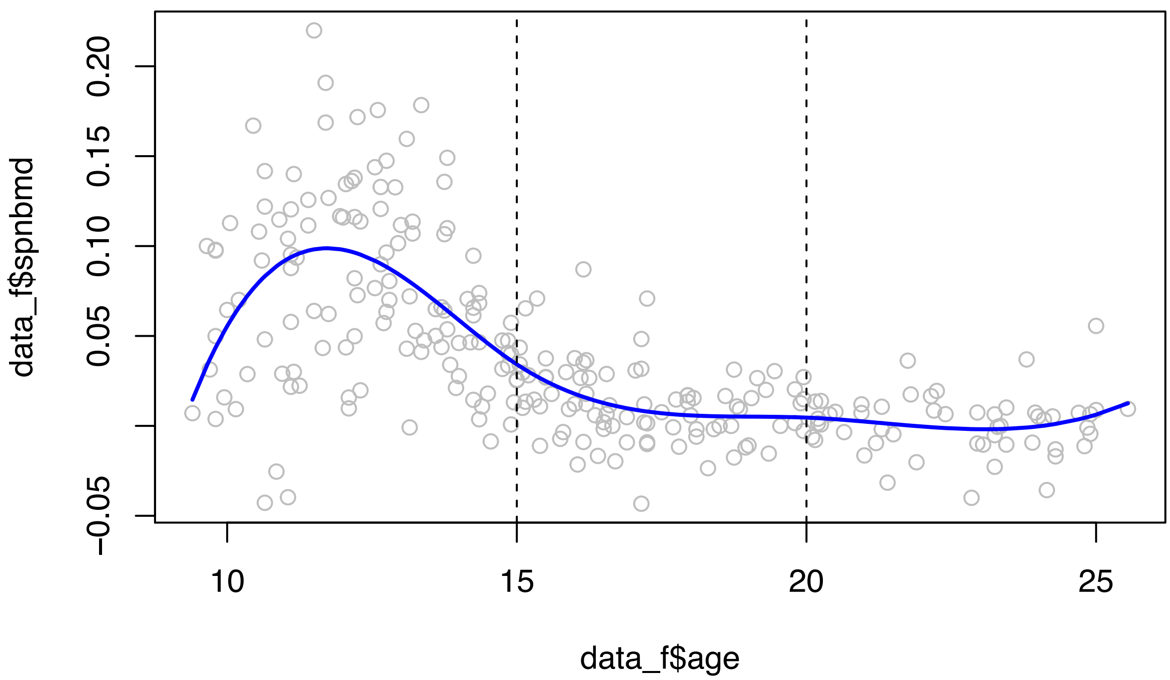
```
        predict(sr, newdata=data.frame(age=x)),
        col = "blue",
        lwd = 2,
        type = "l")

# showing the knots
abline(v = c(15,20),
       lty = 2,
       col = "black")
```



Now do from scratch:

```
# Lets order the column data_f$age
age <- data_f$age[order(data_f$age)]

# From (c) we know what the basis functions are

h0 <- rep(1,length(age))
h1 <- age
h2 <- age^2
h3 <- age^3
h4 <- ifelse((age-15)^3>0, (age-15)^3, 0)
h5 <- ifelse((age-20)^3>0, (age-20)^3, 0)

B <- cbind(h0, h1, h2, h3, h4, h5)

# solving to get min (coefficients)
min <- solve(t(B) %*% B, t(B) %*% data_f$spnbmd[order(data_f$age)])

# Lets get spline y-values evaluated at our age values
# Just linear combination of basis functions with coefficients = min
y <- B %*% min
```

```r
# Lets plot
plot(data_f$age, data_f$spnbmd,
     col = "grey")

points(age,
       y,
       col = "blue",
       lwd = 2,
       type = "l")

# showing the knots
abline(v = c(15,20),
       lty = 2,
       col = "black")
```



3)

```r
# difference functions

one_sided_diff <- function(h){
  return((exp(h) - 1)/h)
}

two_sided_diff <- function(h){
  return((exp(h) - exp(-h))/(2*h))
}
```

```r
options(digits=16)

i <- seq(-20,0)
h <- 10^i
```

```
# getting the differences
one_diff <- sapply(h, one_sided_diff)
two_diff <- sapply(h, two_sided_diff)

# calculating error
error_one <- abs(one_diff-1)
error_two <- abs(two_diff-1)

# use -log in base 10 to solve for the abs of the exponent, n,
# of the above error, which has approx form 10^(-n)
same_digits_one <- ceiling(-log10(error_one))
same_digits_two <- ceiling(-log10(error_two))
```

```
# printing table that displays for each h value, its one-sided difference,
# the error when compared to the actual derivative,
# and how many digits in the one-sided difference are correct
cbind(h, one_diff, error_one, same_digits_one)
```

```
##                          h           one_diff            error_one
##  [1,] 9.999999999999999e-21 0.0000000000000000 1.000000000000000e+00
##  [2,] 1.000000000000000e-19 0.0000000000000000 1.000000000000000e+00
##  [3,] 1.000000000000000e-18 0.0000000000000000 1.000000000000000e+00
##  [4,] 1.000000000000000e-17 0.0000000000000000 1.000000000000000e+00
##  [5,] 1.000000000000000e-16 0.0000000000000000 1.000000000000000e+00
##  [6,] 1.000000000000000e-15 1.1102230246251565 1.102230246251565e-01
##  [7,] 1.000000000000000e-14 0.9992007221626409 7.992778373591136e-04
##  [8,] 1.000000000000000e-13 0.9992007221626409 7.992778373591136e-04
##  [9,] 1.000000000000000e-12 1.0000889005823410 8.890058234101161e-05
## [10,] 9.999999999999999e-12 1.0000000827403710 8.274037099909037e-08
## [11,] 1.000000000000000e-10 1.0000000827403710 8.274037099909037e-08
## [12,] 1.000000000000000e-09 1.0000000827403710 8.274037099909037e-08
## [13,] 1.000000000000000e-08 0.9999999939225290 6.077470970922150e-09
## [14,] 1.000000000000000e-07 1.0000000494336803 4.943368026033568e-08
## [15,] 1.000000000000000e-06 1.0000004999621837 4.999621836532242e-07
## [16,] 1.000000000000000e-05 1.0000050000069649 5.000006964905879e-06
## [17,] 1.000000000000000e-04 1.0000500016671410 5.000166714097531e-05
## [18,] 1.000000000000000e-03 1.0005001667083846 5.001667083845973e-04
## [19,] 1.000000000000000e-02 1.0050167084167949 5.016708416794913e-03
## [20,] 1.000000000000000e-01 1.0517091807564771 5.170918075647712e-02
## [21,] 1.000000000000000e+00 1.7182818284590451 7.182818284590451e-01
##       same_digits_one
##  [1,]               0
##  [2,]               0
##  [3,]               0
##  [4,]               0
##  [5,]               0
##  [6,]               1
##  [7,]               4
##  [8,]               4
##  [9,]               5
## [10,]               8
## [11,]               8
## [12,]               8
```

```
## [13,]                 9
## [14,]                 8
## [15,]                 7
## [16,]                 6
## [17,]                 5
## [18,]                 4
## [19,]                 3
## [20,]                 2
## [21,]                 1
```

In terms of floating point error for one-sided differences: From the lecture video, in order to minimize the error you need to pick an h equal to machine epsilon ˆ 1/2. That is you need to pick an h = 10^{-8}. This will give you the smallest possible error, which is of the order h.

If you look at what was printed above: at $h = 10^{-8}$ the error is of the order 10^{-9}, which is around what it should be (10^{-8}). So the relationship between machine epsilon and the minimum error of one-sided differences holds for this example.

Now: for the two-sided differences.

```r
# printing table that displays for each h value, its two-sided difference,
# the error when compared to the actual derivative,
# and how many digits in the two-sided difference are correct
cbind(h, two_diff, error_two, same_digits_two)
```

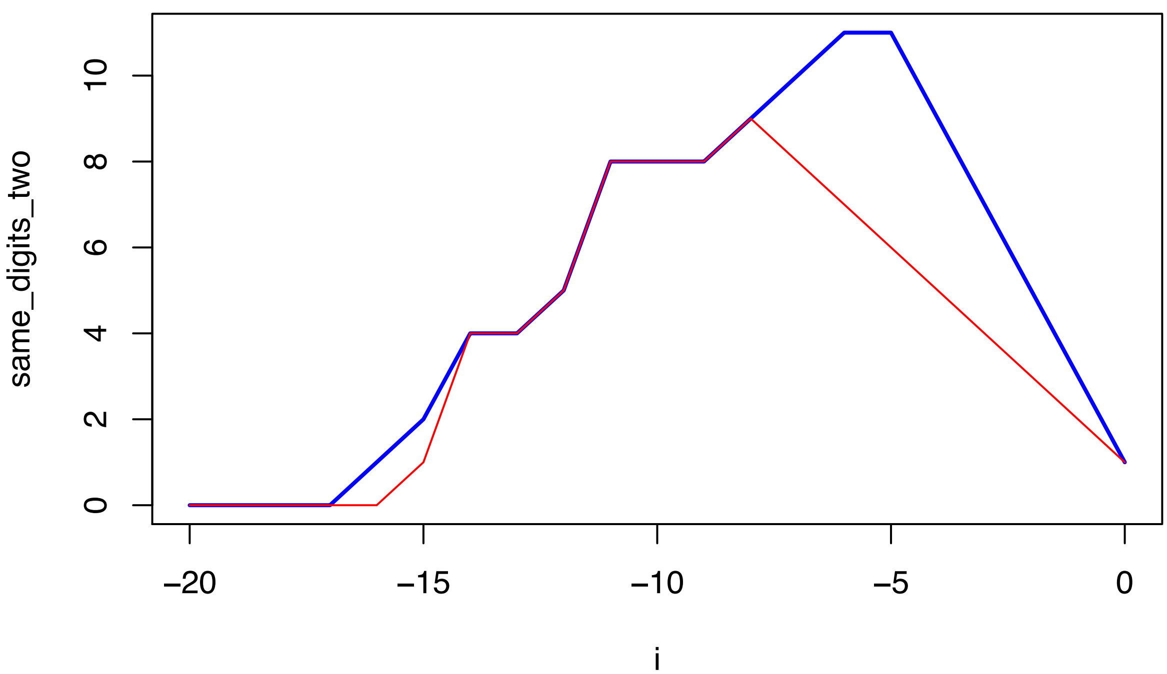
```
##                           h          two_diff             error_two
##   [1,] 9.999999999999999e-21 0.0000000000000000 1.000000000000000e+00
##   [2,] 1.000000000000000e-19 0.0000000000000000 1.000000000000000e+00
##   [3,] 1.000000000000000e-18 0.0000000000000000 1.000000000000000e+00
##   [4,] 1.000000000000000e-17 0.0000000000000000 1.000000000000000e+00
##   [5,] 1.000000000000000e-16 0.5551115123125783 4.448884876874217e-01
##   [6,] 1.000000000000000e-15 1.0547118733938987 5.471187339389871e-02
##   [7,] 1.000000000000000e-14 0.9992007221626409 7.992778373591136e-04
##   [8,] 1.000000000000000e-13 0.9997558336749535 2.441663250465353e-04
##   [9,] 1.000000000000000e-12 1.0000333894311098 3.338943110975379e-05
##  [10,] 9.999999999999999e-12 1.0000000827403710 8.274037099909037e-08
##  [11,] 1.000000000000000e-10 1.0000000827403710 8.274037099909037e-08
##  [12,] 1.000000000000000e-09 1.0000000272292198 2.722921976783255e-08
##  [13,] 1.000000000000000e-08 0.9999999939225290 6.077470970922150e-09
##  [14,] 1.000000000000000e-07 0.9999999994736442 5.263558477963670e-10
##  [15,] 1.000000000000000e-06 0.9999999999732445 2.675548671504657e-11
##  [16,] 1.000000000000000e-05 1.0000000000121023 1.210231914683391e-11
##  [17,] 1.000000000000000e-04 1.0000000016668897 1.666889737350630e-09
##  [18,] 1.000000000000000e-03 1.0000001666666813 1.666666813449069e-07
##  [19,] 1.000000000000000e-02 1.0000166667499921 1.666674999212248e-05
##  [20,] 1.000000000000000e-01 1.0016675001984410 1.667500198440974e-03
##  [21,] 1.000000000000000e+00 1.1752011936438014 1.752011936438014e-01
##        same_digits_two
##   [1,]               0
##   [2,]               0
##   [3,]               0
##   [4,]               0
##   [5,]               1
##   [6,]               2
##   [7,]               4
##   [8,]               4
```

```
##  [9,]                5
## [10,]                8
## [11,]                8
## [12,]                8
## [13,]                9
## [14,]               10
## [15,]               11
## [16,]               11
## [17,]                9
## [18,]                7
## [19,]                5
## [20,]                3
## [21,]                1
```

In terms of floating point error for two-sided differences: From the lecture video, in order to minimize the error you need to pick an h equal to machine epsilon ˆ 1/3. That is you need to pick an h $= 10^{-16/3}$ which is approx 10^{-5}. This will give you the smallest possible error, which is of the order h^2.

If you look at what was printed above: at $h = 10^{-5}$ the error is of the order 10^{-11}, which is approx $h^2 = 10^{-5 \cdot 2}$. So the relationship between machine epsilon and the minimum error of two-sided differences holds for this example.

```r
plot(i,same_digits_two, type="l", col='blue', lwd=2)
lines(i, same_digits_one, col='red')
```



Lastly, in terms of one-sided and two-sided differences: from the plots you can the two-sided difference does a better job at approximating the derivative of e^x at 0, since the error is lower (=> greater number of correct digits in the difference estimate). This makes sense given what was explained in the video, and holds for this example.

4)

```
# choosing reasonable x: from the 68-95-99.7 rule,
# we could integrate until 3 (instead of infty) and we
# would get approx 99.7% of 1/2. To see what x would get us closer to 1/2:
x <- 1
while(pnorm(x)/2 < 1/2){
  cat("number =", x, "prob =", pnorm(x)/2, "\n")
  x <- x + 1
}
```

```
## number = 1 prob = 0.4206723730342715
## number = 2 prob = 0.4886249340259104
## number = 3 prob = 0.4993250509841849
## number = 4 prob = 0.4999841643790834
## number = 5 prob = 0.499999856674214
## number = 6 prob = 0.4999999995067061
## number = 7 prob = 0.4999999999993601
## number = 8 prob = 0.4999999999999997
```

```
cat("number =", x, "prob =", pnorm(x)/2, "\n")
```

```
## number = 9 prob = 0.5
```

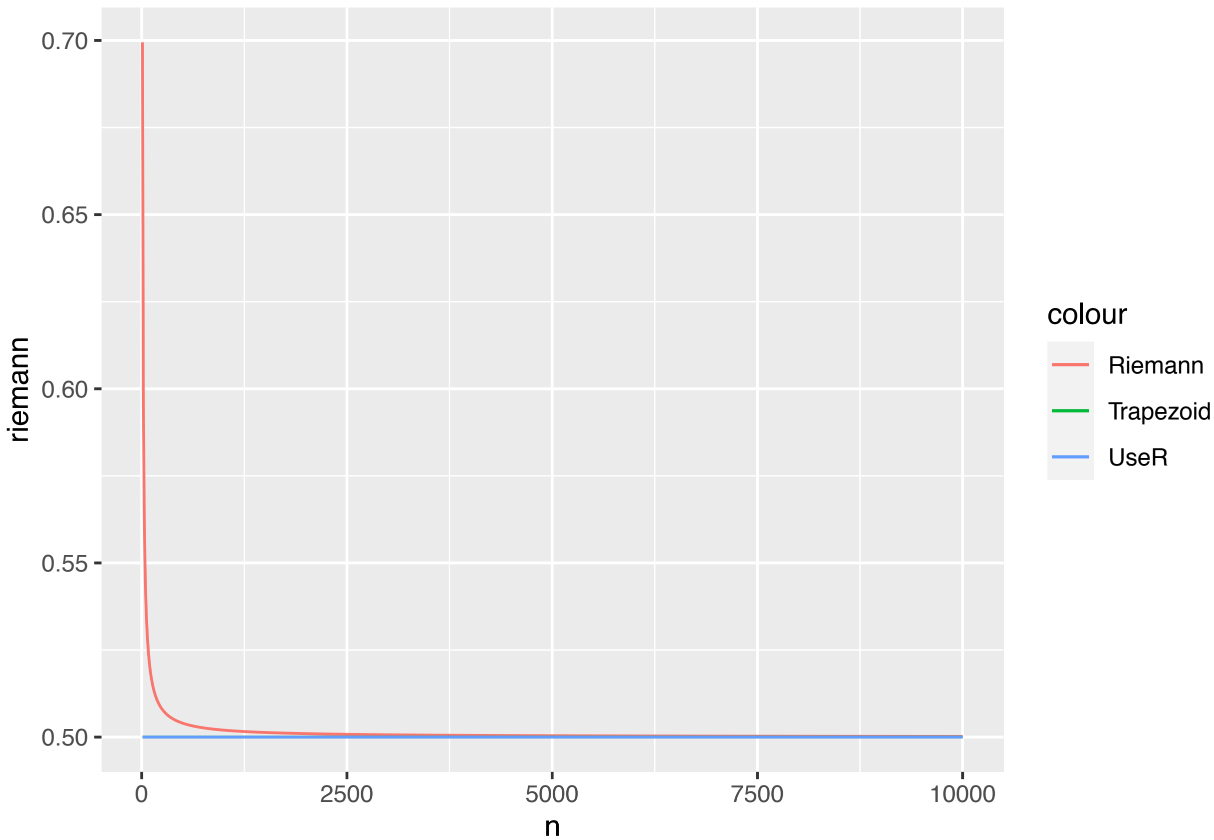
So 9 or above is a reasonable x choice. I'll round to 10 because grid is factors of 10.

```
# Fapprox function
Fapprox <- function(n, method){
  if (method=="riemann"){
    h <- 10/n
    partition <- seq(0,10,h)
    f <- sapply(partition, dnorm)
    return(sum(f*h))
  }
  if (method=="trapezoid"){
    h <- 10/n
    partition <- seq(0,10,h)
    f <- (sapply(partition[2:length(partition)], dnorm)+
      sapply(partition[1:length(partition)-1], dnorm))/2
    return(sum(f*h))
  }
  if (method=="useR"){
    return(integrate(f=dnorm,
                     lower=0,
                     upper=10,
                     subdivisions=n,
                     stop.on.error=FALSE)$value)
  }
  else {return}
}
```

```
# plotting
x <- seq(10, 10000, 10)
df <- data.frame(n=x,
            riemann=sapply(x, Fapprox, method="riemann"),
            trapezoid=sapply(x, Fapprox, method="trapezoid"),
```

```
            useR=sapply(x, Fapprox, method="useR"))
```

```
library(ggplot2)
ggplot(df, aes(x = n)) +
  geom_line(aes(y = riemann, color = "Riemann")) +
  geom_line(aes(y = trapezoid, color = "Trapezoid")) +
  geom_line(aes(y = useR, color = "UseR"))
```



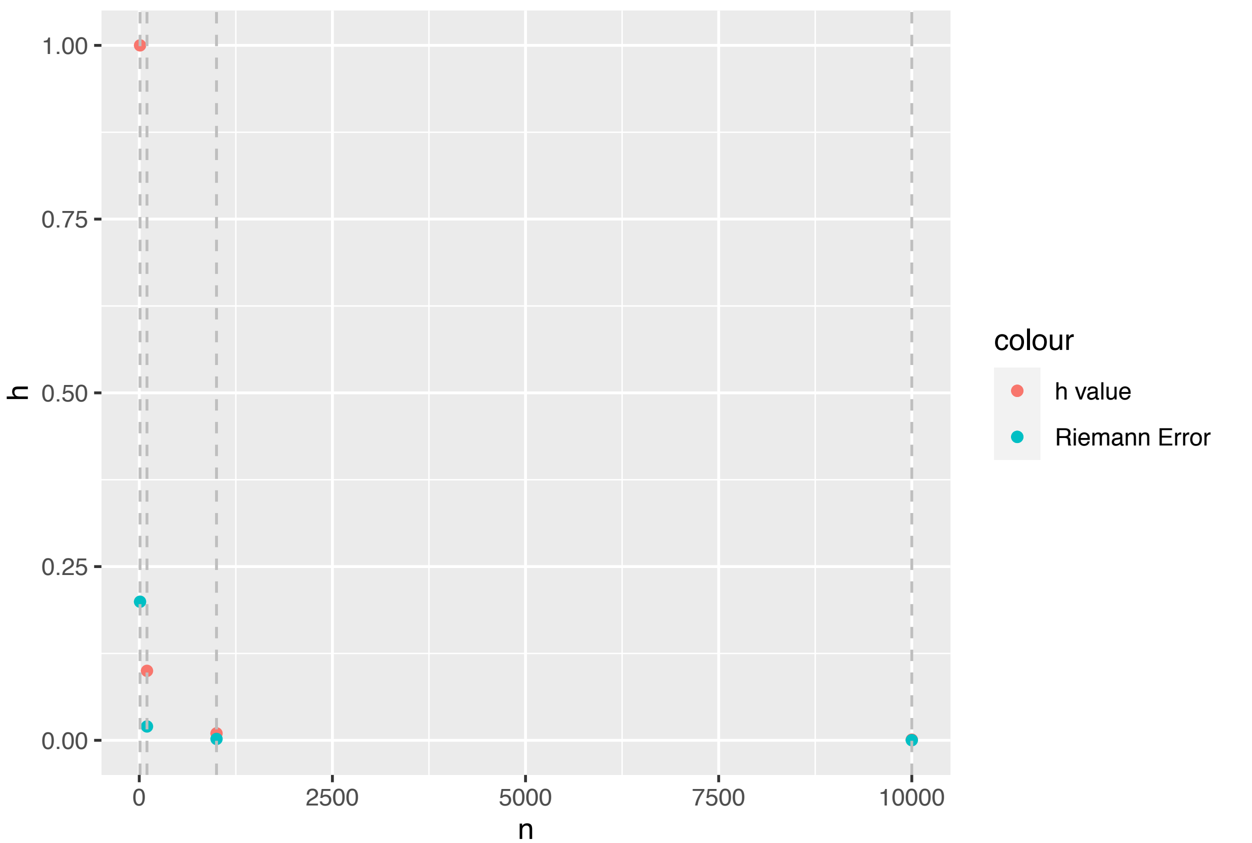
In terms of accuracy: Riemann integration is the least accurate (you can see this from the graph). It ends up converging to 0.5 but converges slower than the Trapezoid method and R's integration function. Also, in terms of Riemann and Trapezoid, this makes sense given that Riemann integrations' error is of order h, whereas the Trapezoid's method error is of order h^2 (=> the error gets small faster for Trapezoid). We can double-check this by looking at the h, h^2 values and Riemann/Trapezoid errors for the given grid of [10, 100, 1000, 10000]:

```
n <- c(10, 100, 1000, 10000)
riemann <- sapply(n, Fapprox, method="riemann")
error_riemann <- abs(0.5-riemann)
trapezoid <- sapply(n, Fapprox, method="trapezoid")
error_trapezoid <- abs(0.5-trapezoid)
useR=sapply(n, Fapprox, method="useR")

df <- data.frame(n=n,
            riemann=riemann,
            h=10/n,
            error_riemann=error_riemann,
            trapezoid=trapezoid,
            h_squared=(10/n)^2,
```

```
            error_trapezoid=error_trapezoid,
            useR=useR
)

df
```

```
##       n             riemann     h        error_riemann         trapezoid
## 1    10 0.6994711428760044 1.000 0.199471142876004426 0.500000002675288
## 2   100 0.5199471140200717 0.100 0.019947114020071655 0.500000000000000
## 3  1000 0.5019947114020072 0.010 0.001994711402007243 0.500000000000000
## 4 10000 0.5001994711402007 0.001 0.000199471140200691 0.500000000000000
##   h_squared       error_trapezoid              useR
## 1     1e+00 2.675287991138475e-09 0.4999999999962937
## 2     1e-02 0.000000000000000e+00 0.4999999999962937
## 3     1e-04 0.000000000000000e+00 0.4999999999962937
## 4     1e-06 0.000000000000000e+00 0.4999999999962937
```

```
ggplot(df) +
  geom_point(aes(x=n, y = h, color = "h value")) +
  geom_point(aes(x=n, y = error_riemann, color = "Riemann Error")) +
  geom_vline(xintercept = 10, linetype="dashed", color="grey") +
  geom_vline(xintercept = 100, linetype="dashed", color="grey") +
  geom_vline(xintercept = 1000, linetype="dashed", color="grey") +
  geom_vline(xintercept = 10000, linetype="dashed", color="grey")
```



We know Riemann's error is of order h (or less) and this is shown in the graph above for each grid point n (grey dashed lines represent a grid point, red points are the h values, which are above or on the blue points, the riemann error). Similarly, the Trapezoid method's error is of order h^2 (or less) and this is shown in the graph below (for each grid point n).

```
ggplot(df) +
  geom_point(aes(x=n, y = h_squared, color = "h value")) +
  geom_point(aes(x=n, y = error_trapezoid, color = "Trapezoid Error")) +
  geom_vline(xintercept = 10, linetype="dashed", color="grey") +
  geom_vline(xintercept = 100, linetype="dashed", color="grey") +
  geom_vline(xintercept = 1000, linetype="dashed", color="grey") +
  geom_vline(xintercept = 10000, linetype="dashed", color="grey")
```