Math 504 HW11

Jeff Gould

4/1/2020

2

a

$$S(x) = \sum_{j=1}^{D} \alpha_j h_j(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \cdots + \alpha_D h_D(x)$$

which create a linear function space F, We have previously shown that

$$\min_{f \in F} \sum_{i=1}^{N} (y_i - f(x^{(i)}))^2 = \min_{\alpha \in \mathbb{R}^k} ||y - B\alpha||^2$$

where B is an $N \times K$ matrix of functions of $b_i(x)$, where entry $B_{kl} = b_l(x_k)$ in a linear function space F, and α is the coefficients. So with $S(x)$ being the $f(x)$, $h_j(x)$ is analogous to $b_j(x)$, and the α's fill the same roll. Then

$$\min_{S \in F} \sum_{i=1}^{N} (y_i - S(x_i))^2 = \min_{\alpha \in \mathbb{R}^k} ||y - B\alpha||^2 \to \alpha = (B^T B)^{-1} B^T y$$

where as mentioned α is the vector of α_j's, and each $B_{kl} = h_l(x_k)$

b

$$S(x) = \begin{cases} S_0(x) & \text{if } x < \xi_1 \\ S_1(x) & \text{if } x \in [\xi_1, \xi_2) \\ S_2(x) & \text{if } x \geq \xi_2, \end{cases} = \alpha_0 + \alpha_1 x + \alpha_2 x^2 +_3 x^3$$

c

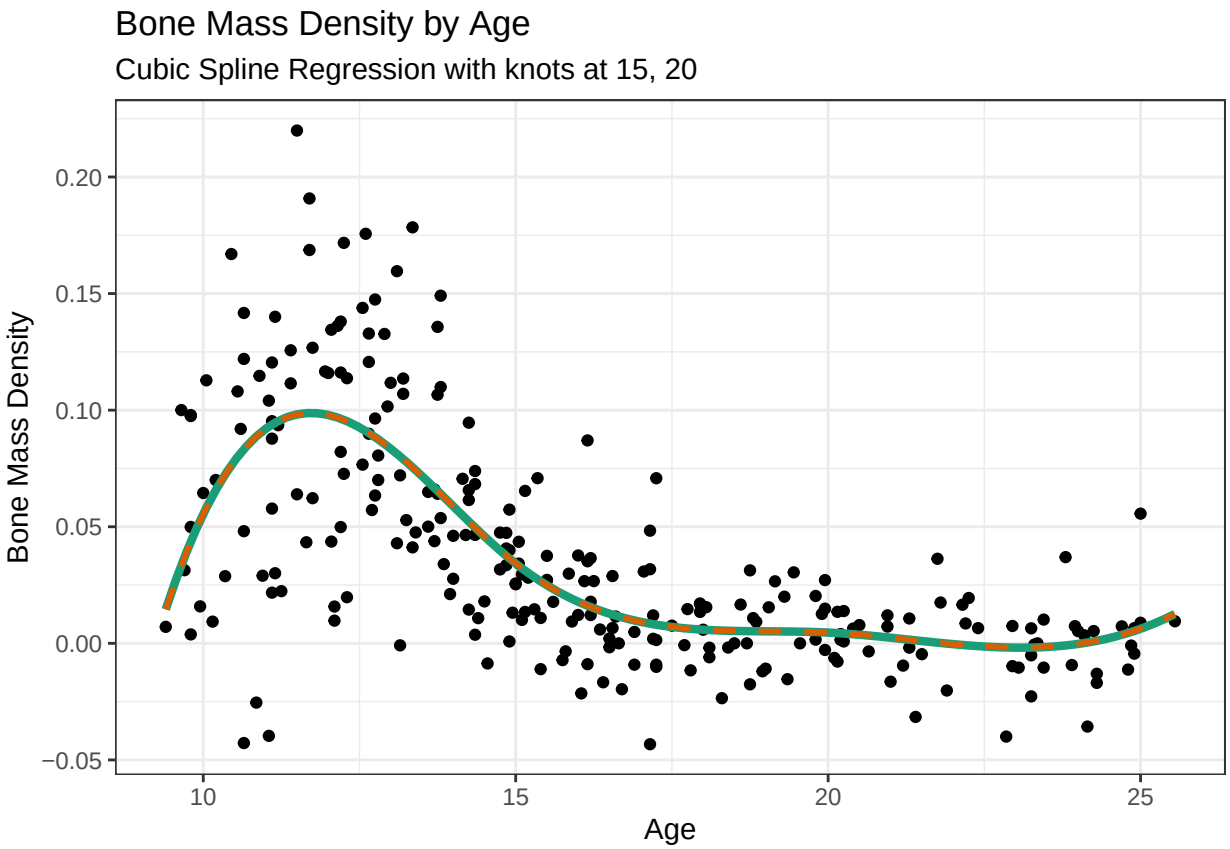
d

```
bone_mass <- read_delim("BoneMassData.txt", delim = " ")

females <- bone_mass %>% filter(gender == "female")


y <- females$spnbmd
x <- females$age
```

```r
h_5 <- function(x, zeta_1 = 15){ifelse((x - zeta_1) > 0, (x - zeta_1)^3, 0)}
h_6 <- function(x, zeta_2 = 20){ifelse((x - zeta_2) > 0, (x - zeta_2)^3, 0)}

B <- matrix(
  c(rep(1, length(x)),
  x,
  x^2,
  x^3,
  h_5(x),
  h_6(x)),
  ncol = 6
)

alpha <- solve(t(B) %*% B) %*% t(B) %*% y
```

So $S(x)$ =-4.2488263 + 0.9762416x + -0.0716499x^2 + 0.0017069x^3 + -0.0021305$[x-15]_+^3$ + 7.2804957 $\times$ $10^{-4}[x-20]_+^3$

```r
S_x <- function(x, zeta_1 = 15, zeta_2 = 20, A = alpha){
  H <- matrix(
    c(rep(1, length(x)),
    x,
    x^2,
    x^3,
    h_5(x),
    h_6(x)),
    ncol = 6
  )
  Sx <- apply(H,1,function(x){sum(A*x)})
  return(Sx)
}

ggplot(data = females, aes(x = age, y = spnbmd)) +
  geom_point() +
  theme_bw() +
  stat_function(fun = S_x, color = "#1B9E77", size = 1.5) +
  geom_smooth(method = "lm", formula = y ~ bs(x, knots=c(15,20)),
              alpha = 0, color = "#D95F02", linetype = 2, size = 1) +
  labs(x = "Age", y = "Bone Mass Density",
       title = "Bone Mass Density by Age",
       subtitle = "Cubic Spline Regression with knots at 15, 20")
```



Here the solid teal line is our $S(x)$ plotted over the data, and the dashed orange line is the spline regression formed using the `bs` function to compute the spline with knots at 15 and 20. As you can see, the regressions match up perfectly.

3

$f(x) = e^x \to f'(x) = e^x \to f'(0) = 1$

```
options(digits = 16)
i <- seq(-20,0,1)
h <- 10^i

finite_difference_1 <- function(x,h){
  fx <- (exp(x + h) - exp(x)) / h

  return(fx)
}

finite_difference_2 <- function(x,h){
  fx <- (exp(x + h) - exp(x - h)) / (2 * h)

  return(fx)
}

differences_1 <- sapply(h, finite_difference_1, x = 0)
differences_2 <- sapply(h, finite_difference_2, x = 0)
```

```
results <- data.frame(
  h = format(10^i, scientific = T, digits = 2),
  fin_diff_1 = differences_1,
  fin_diff_2 = differences_2
)

print(xtable::xtable(results, digits = 16), comment = F)
```

	h	fin_diff_1	fin_diff_2
1	1e-20	0.0000000000000000	0.0000000000000000
2	1e-19	0.0000000000000000	0.0000000000000000
3	1e-18	0.0000000000000000	0.0000000000000000
4	1e-17	0.0000000000000000	0.0000000000000000
5	1e-16	0.0000000000000000	0.5551115123125783
6	1e-15	1.1102230246251565	1.0547118733938987
7	1e-14	0.9992007221626409	0.9992007221626409
8	1e-13	0.9992007221626409	0.9997558336749535
9	1e-12	1.0000889005823410	1.0000333894311098
10	1e-11	1.0000000827403710	1.0000000827403710
11	1e-10	1.0000000827403710	1.0000000827403710
12	1e-09	1.0000000827403710	1.0000000272292198
13	1e-08	0.9999999939225290	0.9999999939225290
14	1e-07	1.0000000494336803	0.9999999994736442
15	1e-06	1.0000004999621837	0.9999999999732445
16	1e-05	1.0000050000069649	1.0000000000121023
17	1e-04	1.0000500016671410	1.0000000016668897
18	1e-03	1.0005001667083846	1.0000001666666813
19	1e-02	1.0050167084167949	1.0000166667499921
20	1e-01	1.0517091807564771	1.0016675001984410
21	1e+00	1.7182818284590451	1.1752011936438014

```
result_errors <- data.frame(h = results$h,
                            error_1 = results$fin_diff_1 - 1,
                            error_2 = results$fin_diff_2 - 1)

print(xtable::xtable(result_errors, digits = 16), comment = F)
```

So we see that our highest accuracy was achieved at `h=1e-06` using the second finite difference method, where we were accurate out to 10 digits. Using the first finite difference method, we were able to achieve eight digits of accuracy at `h=1e-08`.

Reasons for error:

Magnitude of error is given by $|F_h(x) - f(x)|$. While the numerical formula for $F_h^{(1)}(x) = \frac{f(x+h)-f(x)}{h}$, because of computing restrictions the actual formula we are computing is $F_h^{(1)}(x) = \frac{f(x+h)(1+\epsilon_1)-f(x)(1+\epsilon_2)}{h} = \frac{f(x+h)-f(x)}{h} + \frac{f(x+h)\epsilon_1-f(x)\epsilon_2}{h}$, $|\epsilon_i| < 10^{-16}$

Here the first term is our Taylor Series error, and the second term is our round off error due to machine limitations.

Now we can re-write $F_h(x) = \frac{[f(x)+f'(x)h+\frac{1}{2}f''(x)h^2]-f(x)}{h} + \frac{C\epsilon_m}{h} = f'(x)+\frac{1}{2}f''(x)h+\frac{C\epsilon_m}{h}$, where $\epsilon_m = 10^{-16}$, ie machine epsilon.

	h	error_1	error_2
1	1e-20	-1.0000000000000000	-1.0000000000000000
2	1e-19	-1.0000000000000000	-1.0000000000000000
3	1e-18	-1.0000000000000000	-1.0000000000000000
4	1e-17	-1.0000000000000000	-1.0000000000000000
5	1e-16	-1.0000000000000000	-0.4448884876874217
6	1e-15	0.1102230246251565	0.0547118733938987
7	1e-14	-0.0007992778373591	-0.0007992778373591
8	1e-13	-0.0007992778373591	-0.0002441663250465
9	1e-12	0.0000889005823410	0.0000333894311098
10	1e-11	0.0000000827403710	0.0000000827403710
11	1e-10	0.0000000827403710	0.0000000827403710
12	1e-09	0.0000000827403710	0.0000000272292198
13	1e-08	-0.0000000060774710	-0.0000000060774710
14	1e-07	0.0000000494336803	-0.0000000005263558
15	1e-06	0.0000004999621837	-0.0000000000267555
16	1e-05	0.0000050000069649	0.0000000000121023
17	1e-04	0.0000500016671410	0.0000000016668897
18	1e-03	0.0005001667083846	0.0000001666666813
19	1e-02	0.0050167084167949	0.0000166667499921
20	1e-01	0.0517091807564771	0.0016675001984410
21	1e+00	0.7182818284590451	0.1752011936438014

So our error becomes $|F_h(x) - f'(x)| = |\frac{1}{2}f''(x)h| + \frac{C\epsilon_m}{h}$, take the derivative with respect to h and set equal to 0 to solve for the optimal h to minimize error: $|\frac{1}{2}f''(x)| - \frac{C\epsilon_m}{h^2} = 0 \rightarrow h^2 = \frac{C\epsilon_m}{|1/2f''(x)|} \rightarrow h \approx \sqrt{\epsilon_m} = 10^-8$.

So this explains why `h=1e-08` produced the smallest error for our first finite difference equation. To explain some of the output a little further: the reason our output is exactly 0 for `h=1e-16` and smaller is because when we subtract that from $f(x) = 1$, the computer is only able to store 16 significant figures, so the extra $1e - 16$ can't be stored, and the computer calculates $\frac{1-1}{10^{-16}} = 0$.

Looking at our second finite difference equation, we follow the same process but see we are able to reach slightly more accurate results, and with a larger h.

4

In general, the cdf $F(x)$ of a distribution is $P(X \leq x) \rightarrow P(-\infty < X \leq x)$. However here we have defined $F(X)$ with a lower bound of 0. Since we know the standard normal distribution is centered at 0, with $P(X \leq 0) = 0.5$ and $P(X > 0) = 0.5$, then

$$F(x) = \int_0^x dz \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \rightarrow F(\infty) = 0.5$$

Using `R`'s built in integrate function, we see that an upper limit of 7.5 gives a value of 0.4999999999999883, accurate to 12 decimal places, so we set the upper limit for our subdivisions at 7.5. Increasing the upperbound beyond this will give us minimal added area and increase our error. However, we make this bound adjustable to the function's user.

```
norm_cdf <- function(z){1 / (sqrt(2 * pi)) * exp(-z^2 / 2)}
integrate(norm_cdf, 0, 7.5, subdivisions = 100)
```

0.4999999999999883 with absolute error < 3.2e-07

```r
Fapprox <- function(n, method = "reimann", upperlimit = 7.5){
  `%notin%` = Negate(`%in%`)
  if(method %notin% c("reimann", "trapezoid", "useR")){
    return("Please enter a valid integration method")
  }

  h <- upperlimit / n

  rectangle <- function(z){
    height <- 1 / sqrt(2 * pi) * exp(-(z^2) / 2)
    width <- h
    area <- height * width
    return(area)
  }

  trapezoid <- function(z){
    height_1 <- 1 / sqrt(2 * pi) * exp(-(z^2) / 2)
    height_2 <- 1 / sqrt(2 * pi) * exp(-((z + h)^2) / 2)
    area <- (height_1 + height_2) / 2 * h

  }

  if(method == "reimann"){
    cuts <- seq(0, upperlimit, h)
    norm_cdf <- sum(sapply(cuts[-n], rectangle))
  }else if(method == "trapezoid"){
    cuts <- seq(0, upperlimit, h)
    norm_cdf <- sum(sapply(cuts[-n], trapezoid))
  }else{
    norm_cdf_func <- function(z){1 / (sqrt(2 * pi)) * exp(-z^2 / 2)}
    norm_cdf <- integrate(norm_cdf_func, 0,
                          upperlimit,
                          subdivisions = n)$value
  }
  return(norm_cdf)

}

n_s <- c(10,100,1000,10000)
evaluations <- data.frame(
  n = format(n_s, digits = 0),
  Riemann = sapply(n_s, Fapprox),
  Trapezoid = sapply(n_s, Fapprox, method = "trapezoid"),
  useR = sapply(n_s, Fapprox, method = "useR")
)

print(xtable::xtable(evaluations, digits = 16), comment = F)
```

Looking at the table of our results, we see that the `rectangle` method consistently overestimates the probability by a good amount, only reaching 3 digits of accuracy after we bump `n` to 10000. This is what we would expect as the normal curve is a decreasing function so we would consistently be adding additional area at each step. The trapezoid method is much more accurate, achieving 10 digits of accuracy at nust `n=10`, and almost reaching the same accuracy as `R` at `n=10000`. The accuracy of `R`'s built in integral function did

	n	Riemann	Trapezoid	useR
1	10	0.6496033551123254	0.4999999999808031	0.4999999999999883
2	100	0.5149603355149982	0.4999999999999565	0.4999999999999883
3	1000	0.5014960335514724	0.4999999999999680	0.4999999999999883
4	10000	0.5001496033551186	0.4999999999999681	0.4999999999999883

not change with varying values of `n`.