HW3

Isaac Kleisle-Murphy

4/30/2022

5.16.4

```
motorcycle = read.csv("./data/motorcycle.csv") %>%
  mutate(idx=row_number())
motorcycle %>%
  head()
```

```
##   times accel      strata  v idx
## 1   2.4   0.0 1       3.7 NA   1
## 2   2.6  -1.3 1       3.7 NA   2
## 3   3.2  -2.7 1       3.7 NA   3
## 4   3.6   0.0 1       3.7 NA   4
## 5   4.0  -2.7 1       3.7 NA   5
## 6   6.2  -2.7 1       3.7 NA   6
```

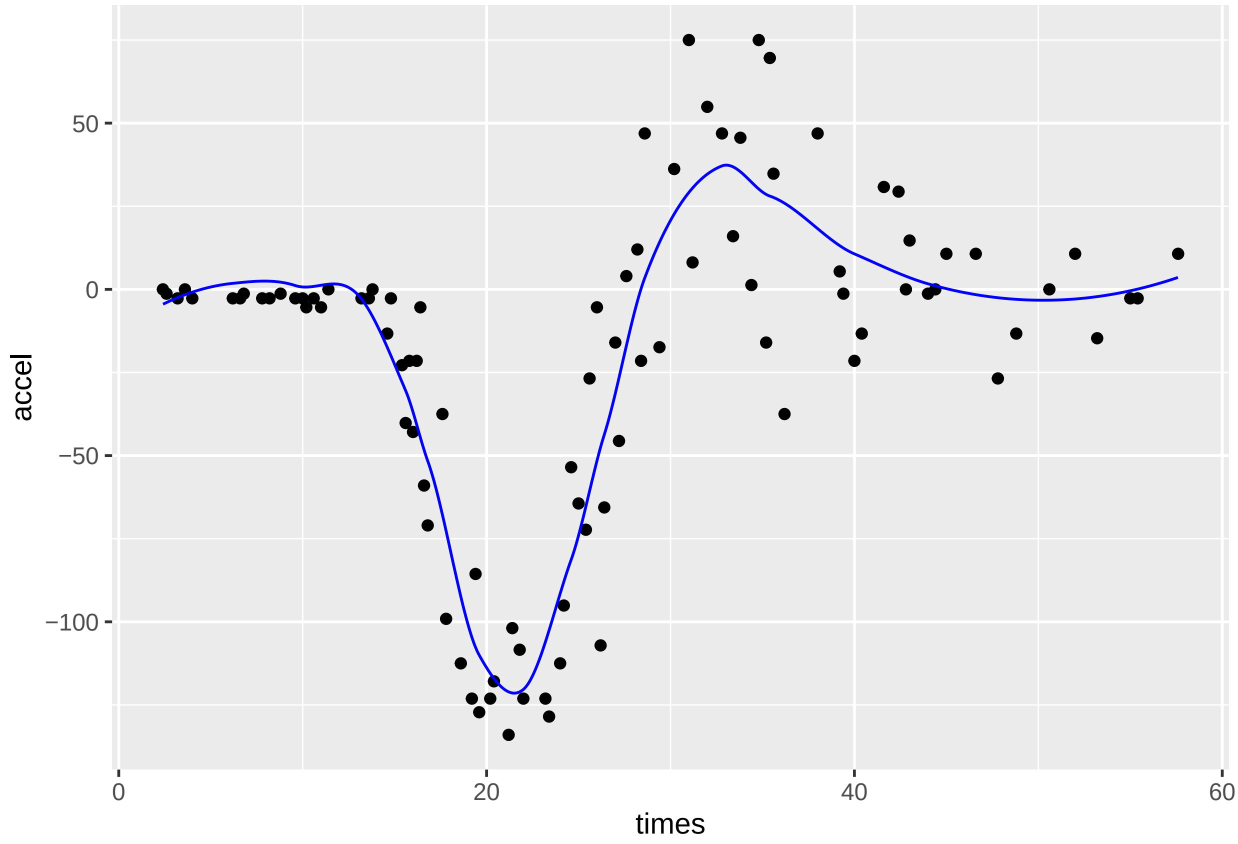
Here, we perform LOO cross validation (`n=94`, so pretty small). According to `stats::loess()`' documentation, the smoothing parameter here is `span`; I assume this is synonymous with α in a tri-cube function? Either way, that's the parameter we'll tune over, with respect to MSE

```
lapply(seq(.1, 2.5, length.out=25), function(span){
  sapply(1:nrow(motorcycle), function(i){
    fit = loess(accel ~ times, motorcycle %>% filter(idx != i), span=span);
    yhat = predict(fit, newdata =motorcycle %>% filter(idx == i))
    (motorcycle$accel[i] - yhat)^2
  }) %>%
    as.numeric() -> mse_loo
  c(span, mean(mse_loo, na.rm=T))
}) %>%
  do.call("rbind", .) %>%
  data.frame() %>%
  `colnames<-`(c("span", "mse_loo")) -> cv_result

cv_result %>% arrange(mse_loo) %>% head()
```

```
##   span  mse_loo
## 1  0.4 537.4832
## 2  0.3 540.5290
## 3  0.5 582.1378
## 4  0.2 583.6715
## 5  0.6 719.3641
## 6  0.1 821.8802
```

We select `span=0.4`, and proceed to fit and visualize the tuned model.

```
fit = loess(accel ~ times, data=motorcycle, span=cv_result$span[which.min(cv_result$mse_loo)])
pred_df = data.frame(times=seq(min(motorcycle$times) + .01, max(motorcycle$times) - .01, length.out=100
pred_df$yhat = predict(fit, newdata=pred_df)

ggplot() +
  geom_point(data=motorcycle, mapping=aes(x=times, y=accel), color="black") +
  geom_path(data=pred_df, mapping=aes(x=times, y=yhat), color="blue")
```



5.16.12

Make the data and CV fn:

```
doppler <- function(z){
  sqrt(z * (1 - z)) * sin((2 * pi)/ (z + .05))
}
set.seed(605400)
N = 1000
xi = (1:N) / N
### instantiate data
y1 = doppler(xi) + .1 * rnorm(N, 0, 1)
y2 = doppler(xi) + 1 * rnorm(N, 0, 1)
y3 = doppler(xi) + 3 * rnorm(N, 0, 1)
```

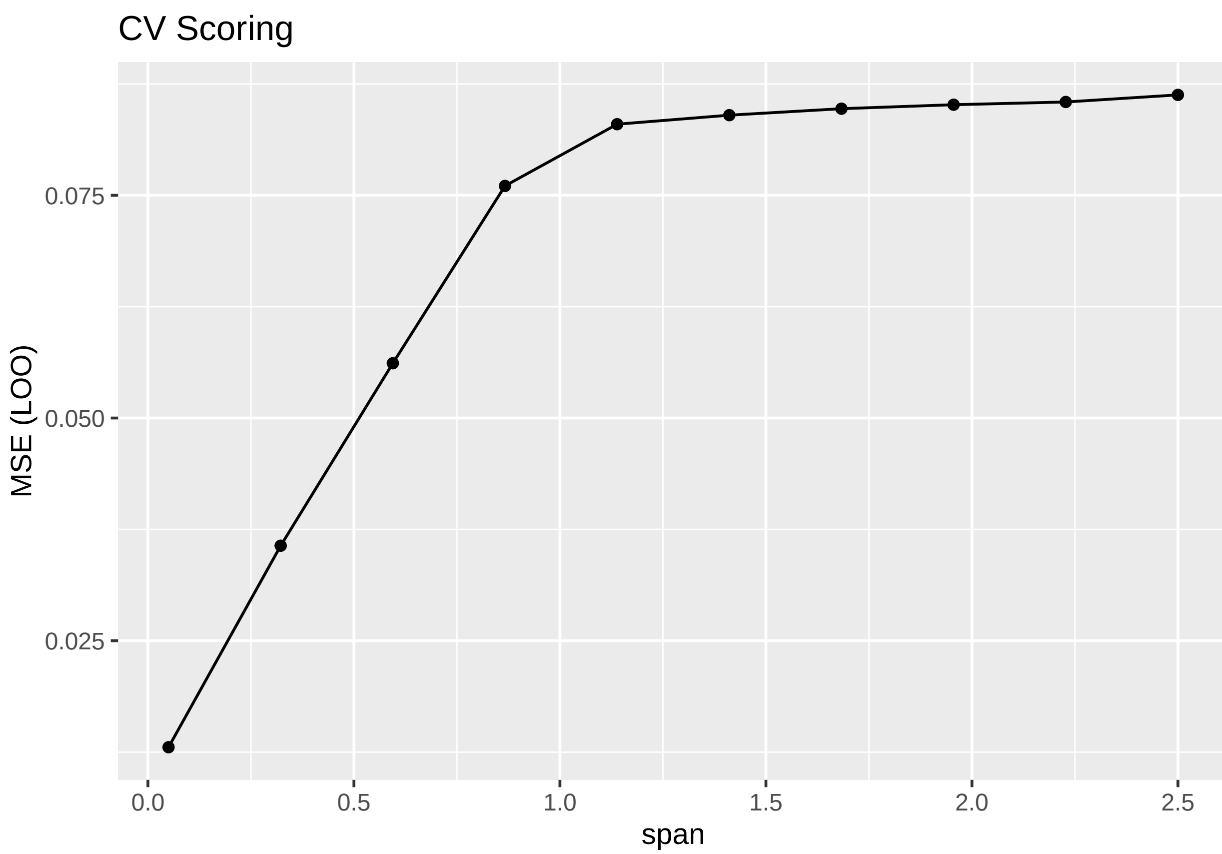
```r
### make a function to do the CV
y = y1; x = xi

cv_loess <- function(x, y, lwr_span=.05, upr_span=2.5, length.out=10){
  lapply(seq(lwr_span, upr_span, length.out=length.out), function(span){
    cat("-------------\n")
    N = length(y)
    sapply(1:N, function(i){
      # grab all data
      data_train = data.frame(
        x=x[(1:N) != i],
        y=y[(1:N) != i]
      )
      data_val = data.frame(
        x=x[i],
        y=y[i]
      )
      fit = loess(y ~ x, data=data_train, span=span);
      yhat = predict(fit, newdata=data_val)
      as.numeric(
        (y[i] - yhat)^2
      )
    })  -> mse_loo
    c(span, mean(mse_loo, na.rm=T))
  }) %>%
    do.call("rbind", .) %>%
    data.frame() %>%
    `colnames<-`(c("span", "mse_loo")) -> cv_result
  cv_result
}
```

CV and for the first dataset:

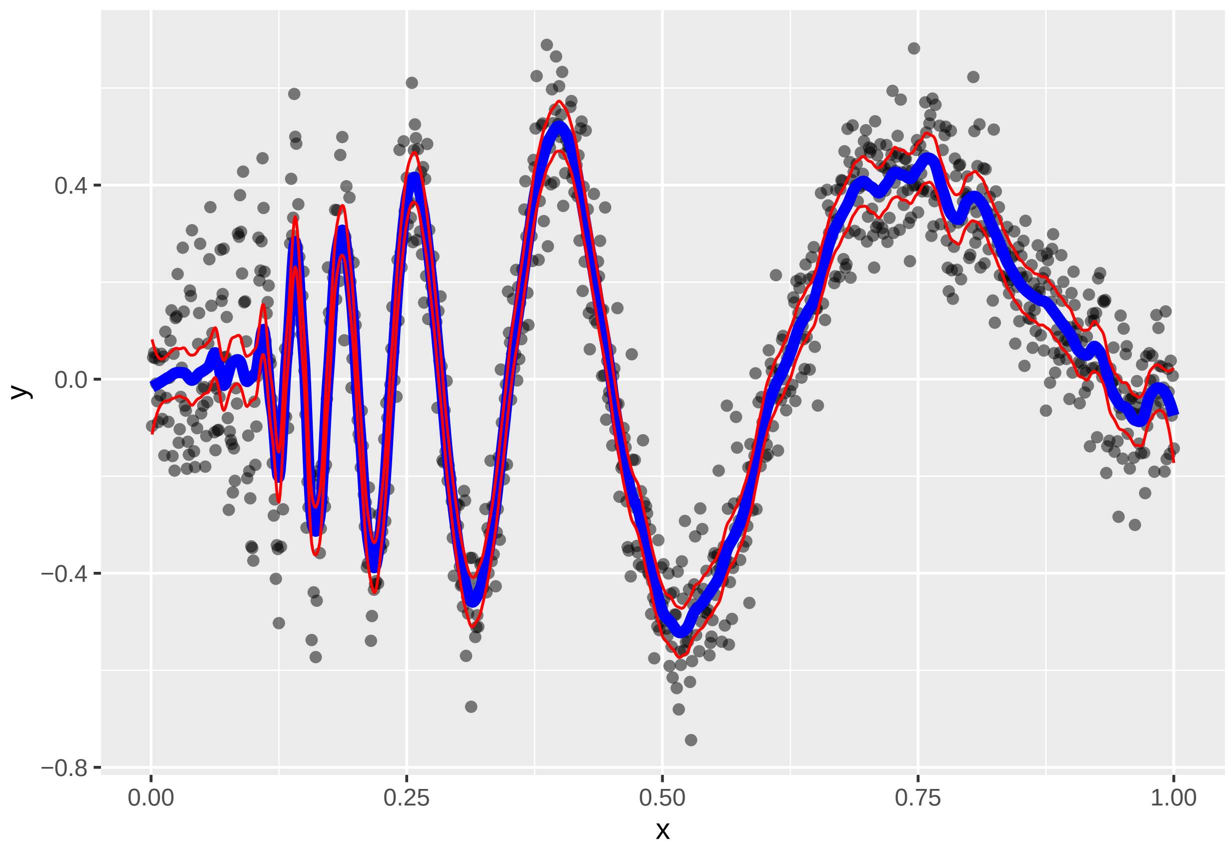
```r
Y = y1
cv_result = cv_loess(x=xi, y=Y)
```

```
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
```

```r
## plot MSE
ggplot(cv_result, aes(x=span, y=mse_loo)) +
  geom_path() +
  geom_point() +
  labs(title="CV Scoring", x="span", y="MSE (LOO)")
```




```
## final fit
fit = loess(Y ~ xi, span=cv_result$span[which.min(cv_result$mse_loo)])

ggplot(
  data.frame(
    x=xi,
    yhat=predict(fit, xi),
    y=Y,
    ci_upr = predict(fit, xi) + 1.96 * predict(fit, xi, se=T)$se.fit,
    ci_lwr = predict(fit, xi) - 1.96 * predict(fit, xi, se=T)$se.fit
  ),
  aes(x=x, y=y)
) +
  geom_point(alpha=.5) +
  geom_path(aes(x=x, y=yhat), color="blue", size=2) +
  geom_path(aes(x=x, y=ci_upr), color="red")  +
  geom_path(aes(x=x, y=ci_lwr), color="red")
```

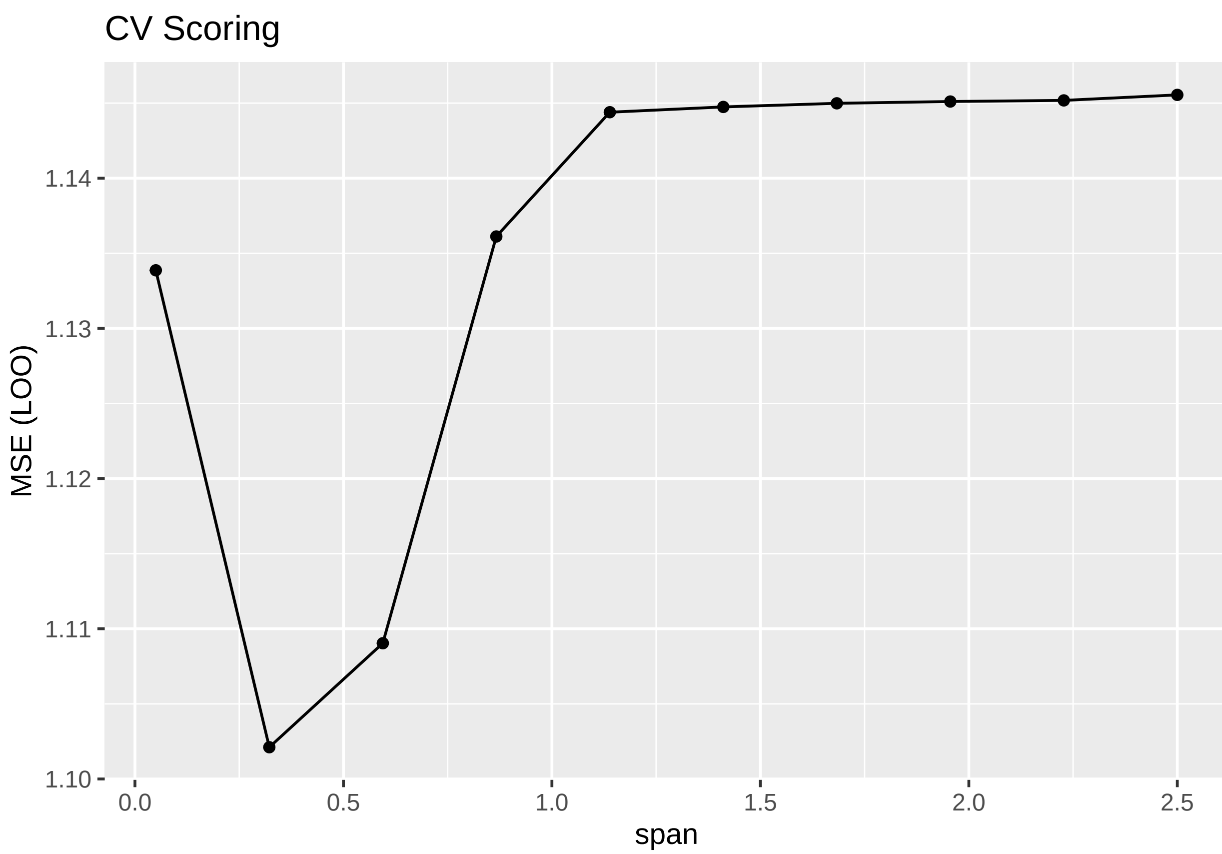


For the second:

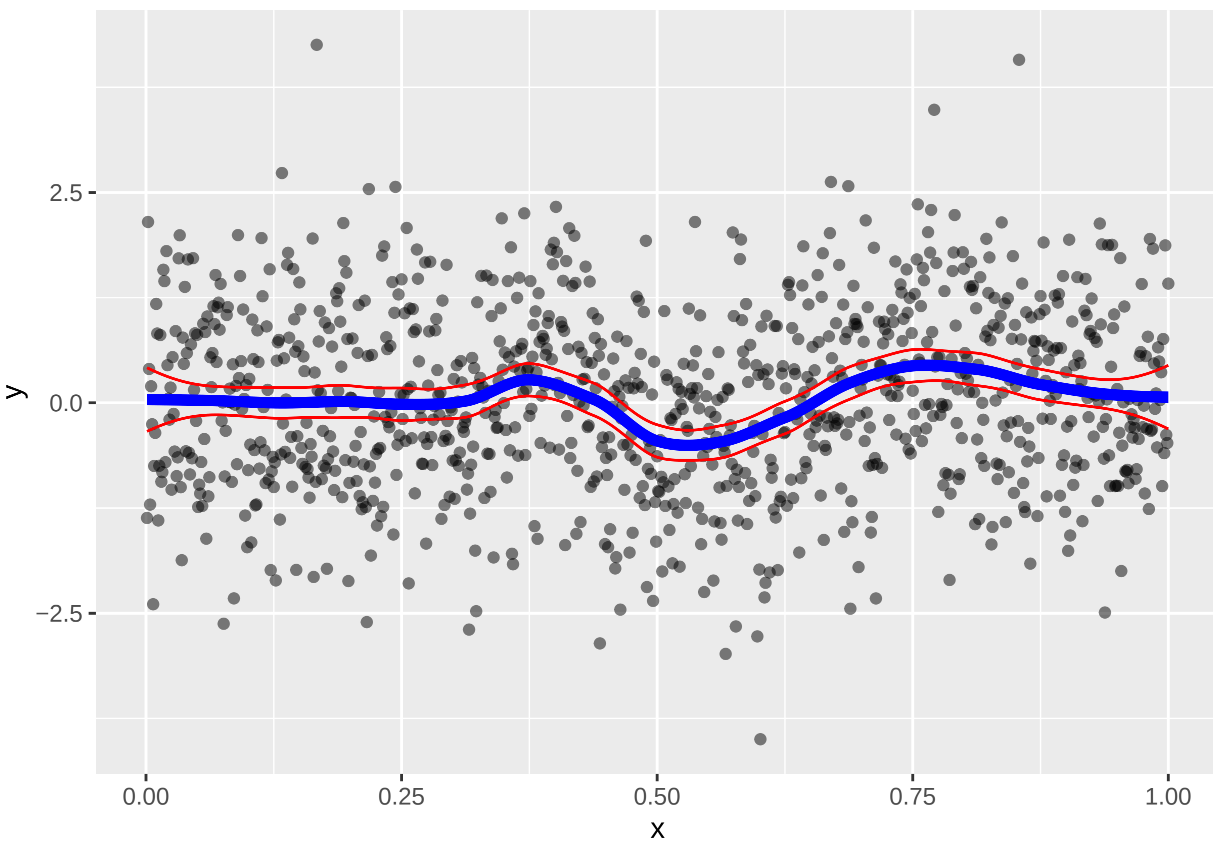
```
Y = y2
cv_result = cv_loess(x=xi, y=Y)
```

```
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
```

```
## plot MSE
ggplot(cv_result, aes(x=span, y=mse_loo)) +
  geom_path() +
  geom_point() +
  labs(title="CV Scoring", x="span", y="MSE (LOO)")
```




```r
## final fit
fit = loess(Y ~ xi, span=cv_result$span[which.min(cv_result$mse_loo)])

ggplot(
  data.frame(
    x=xi,
    yhat=predict(fit, xi),
    y=Y,
    ci_upr = predict(fit, xi) + 1.96 * predict(fit, xi, se=T)$se.fit,
    ci_lwr = predict(fit, xi) - 1.96 * predict(fit, xi, se=T)$se.fit
  ),
  aes(x=x, y=y)
) +
  geom_point(alpha=.5) +
  geom_path(aes(x=x, y=yhat), color="blue", size=2) +
  geom_path(aes(x=x, y=ci_upr), color="red")  +
  geom_path(aes(x=x, y=ci_lwr), color="red")
```




```r
predict(fit, xi)
```

```
##     [1]  3.895791e-02  3.883518e-02  3.871206e-02  3.858847e-02  3.846432e-02
##     [6]  3.833956e-02  3.821409e-02  3.808784e-02  3.796074e-02  3.783272e-02
##    [11]  3.770368e-02  3.757357e-02  3.744229e-02  3.730978e-02  3.717596e-02
##    [16]  3.704076e-02  3.690409e-02  3.676588e-02  3.662606e-02  3.648454e-02
##    [21]  3.634126e-02  3.619613e-02  3.604909e-02  3.590004e-02  3.574893e-02
##    [26]  3.559566e-02  3.544017e-02  3.528238e-02  3.512221e-02  3.495959e-02
##    [31]  3.479443e-02  3.462667e-02  3.445623e-02  3.428303e-02  3.410700e-02
##    [36]  3.392805e-02  3.374612e-02  3.356112e-02  3.337298e-02  3.318163e-02
##    [41]  3.298699e-02  3.278897e-02  3.258752e-02  3.238254e-02  3.217396e-02
##    [46]  3.196172e-02  3.174572e-02  3.152589e-02  3.130217e-02  3.107447e-02
##    [51]  3.084271e-02  3.060682e-02  3.036673e-02  3.012235e-02  2.987361e-02
##    [56]  2.962044e-02  2.936276e-02  2.910048e-02  2.883355e-02  2.856187e-02
##    [61]  2.828538e-02  2.800400e-02  2.771764e-02  2.742066e-02  2.710779e-02
##    [66]  2.677959e-02  2.643662e-02  2.607943e-02  2.570856e-02  2.532458e-02
##    [71]  2.492803e-02  2.451947e-02  2.409944e-02  2.366851e-02  2.322722e-02
##    [76]  2.277613e-02  2.231579e-02  2.184675e-02  2.136956e-02  2.088478e-02
##    [81]  2.039296e-02  1.989465e-02  1.939040e-02  1.888077e-02  1.836631e-02
##    [86]  1.784757e-02  1.732510e-02  1.679946e-02  1.627119e-02  1.574086e-02
##    [91]  1.520901e-02  1.467620e-02  1.414298e-02  1.360989e-02  1.307750e-02
##    [96]  1.254636e-02  1.201701e-02  1.149001e-02  1.096592e-02  1.044528e-02
##   [101]  9.928653e-03  9.416584e-03  8.909627e-03  8.408336e-03  7.913262e-03
##   [106]  7.424959e-03  6.943978e-03  6.470873e-03  6.006195e-03  5.550497e-03
##   [111]  5.104331e-03  4.668251e-03  4.242808e-03  3.828555e-03  3.426044e-03
```

```
## [116]  3.035828e-03  2.658459e-03  2.294490e-03  1.944474e-03  1.608961e-03
## [121]  1.288506e-03  9.836604e-04  6.949768e-04  4.230075e-04  1.683052e-04
## [126] -6.096185e-05 -2.575563e-04 -4.223219e-04 -5.561028e-04 -6.597429e-04
## [131] -7.340860e-04 -7.799762e-04 -7.982575e-04 -7.897736e-04 -7.553687e-04
## [136] -6.958866e-04 -6.121713e-04 -5.050667e-04 -3.754167e-04 -2.240654e-04
## [141] -5.185668e-05  1.403656e-04  3.517573e-04  5.814747e-04  8.286737e-04
## [146]  1.092510e-03  1.372141e-03  1.666721e-03  1.975407e-03  2.297356e-03
## [151]  2.631722e-03  2.977662e-03  3.334332e-03  3.700888e-03  4.076487e-03
## [156]  4.460284e-03  4.851435e-03  5.249096e-03  5.652424e-03  6.060575e-03
## [161]  6.472704e-03  6.887967e-03  7.305521e-03  7.724522e-03  8.144126e-03
## [166]  8.563488e-03  8.981765e-03  9.398113e-03  9.811689e-03  1.022165e-02
## [171]  1.062714e-02  1.102734e-02  1.142138e-02  1.180843e-02  1.218764e-02
## [176]  1.255818e-02  1.291919e-02  1.326983e-02  1.360925e-02  1.393663e-02
## [181]  1.425110e-02  1.455182e-02  1.483796e-02  1.510867e-02  1.536310e-02
## [186]  1.560041e-02  1.581975e-02  1.602029e-02  1.618221e-02  1.628741e-02
## [191]  1.633757e-02  1.633442e-02  1.627964e-02  1.617495e-02  1.602204e-02
## [196]  1.582261e-02  1.557838e-02  1.529103e-02  1.496228e-02  1.459382e-02
## [201]  1.418736e-02  1.374459e-02  1.326723e-02  1.275697e-02  1.221552e-02
## [206]  1.164458e-02  1.104585e-02  1.042103e-02  9.771821e-03  9.099933e-03
## [211]  8.407064e-03  7.694918e-03  6.965195e-03  6.219598e-03  5.459830e-03
## [216]  4.687592e-03  3.904587e-03  3.112518e-03  2.313086e-03  1.507993e-03
## [221]  6.989422e-04 -1.123645e-04 -9.242249e-04 -1.734937e-03 -2.542798e-03
## [226] -3.346106e-03 -4.143158e-03 -4.932254e-03 -5.711689e-03 -6.479762e-03
## [231] -7.234771e-03 -7.975014e-03 -8.698788e-03 -9.404391e-03 -1.009012e-02
## [236] -1.075428e-02 -1.139515e-02 -1.201105e-02 -1.260026e-02 -1.316109e-02
## [241] -1.369184e-02 -1.419079e-02 -1.465625e-02 -1.508652e-02 -1.547990e-02
## [246] -1.583468e-02 -1.614915e-02 -1.642163e-02 -1.665040e-02 -1.683376e-02
## [251] -1.699287e-02 -1.714927e-02 -1.730187e-02 -1.744954e-02 -1.759120e-02
## [256] -1.772572e-02 -1.785200e-02 -1.796895e-02 -1.807544e-02 -1.817037e-02
## [261] -1.825264e-02 -1.832114e-02 -1.837476e-02 -1.841240e-02 -1.843295e-02
## [266] -1.843530e-02 -1.841835e-02 -1.838098e-02 -1.832210e-02 -1.824060e-02
## [271] -1.813536e-02 -1.800529e-02 -1.784928e-02 -1.766621e-02 -1.745499e-02
## [276] -1.721450e-02 -1.694365e-02 -1.664131e-02 -1.630640e-02 -1.593779e-02
## [281] -1.553439e-02 -1.509508e-02 -1.461876e-02 -1.410433e-02 -1.355067e-02
## [286] -1.295669e-02 -1.232126e-02 -1.164329e-02 -1.092168e-02 -1.015530e-02
## [291] -9.343065e-03 -8.483857e-03 -7.576573e-03 -6.620106e-03 -5.613348e-03
## [296] -4.555195e-03 -3.444538e-03 -2.280271e-03 -1.061287e-03  2.135193e-04
## [301]  1.545256e-03  2.935029e-03  4.383945e-03  5.893110e-03  7.463633e-03
## [306]  9.096618e-03  1.079317e-02  1.255440e-02  1.438142e-02  1.627532e-02
## [311]  1.823722e-02  2.026823e-02  2.236944e-02  2.463822e-02  2.716454e-02
## [316]  2.993847e-02  3.295006e-02  3.618934e-02  3.964639e-02  4.331125e-02
## [321]  4.717397e-02  5.122460e-02  5.545320e-02  5.984981e-02  6.440450e-02
## [326]  6.910731e-02  7.394829e-02  7.891750e-02  8.400498e-02  8.920080e-02
## [331]  9.449500e-02  9.987763e-02  1.053387e-01  1.108684e-01  1.164567e-01
## [336]  1.220935e-01  1.277691e-01  1.334735e-01  1.391966e-01  1.449286e-01
## [341]  1.506594e-01  1.563793e-01  1.620781e-01  1.677460e-01  1.733730e-01
## [346]  1.789492e-01  1.844646e-01  1.899092e-01  1.952732e-01  2.005466e-01
## [351]  2.057193e-01  2.107816e-01  2.157233e-01  2.205347e-01  2.252056e-01
## [356]  2.297263e-01  2.340867e-01  2.382769e-01  2.422869e-01  2.461068e-01
## [361]  2.497267e-01  2.531365e-01  2.563264e-01  2.592864e-01  2.620065e-01
## [366]  2.644769e-01  2.666875e-01  2.686284e-01  2.702896e-01  2.716613e-01
## [371]  2.727335e-01  2.734961e-01  2.739393e-01  2.740532e-01  2.738277e-01
## [376]  2.733431e-01  2.726884e-01  2.718669e-01  2.708819e-01  2.697366e-01
## [381]  2.684344e-01  2.669785e-01  2.653722e-01  2.636189e-01  2.617217e-01
```

```
## [386]  2.596839e-01  2.575089e-01  2.552000e-01  2.527603e-01  2.501932e-01
## [391]  2.475021e-01  2.446900e-01  2.417605e-01  2.387166e-01  2.355618e-01
## [396]  2.322992e-01  2.289323e-01  2.254642e-01  2.218982e-01  2.182377e-01
## [401]  2.144858e-01  2.106460e-01  2.067215e-01  2.027155e-01  1.986313e-01
## [406]  1.944723e-01  1.902417e-01  1.859428e-01  1.815788e-01  1.771531e-01
## [411]  1.726690e-01  1.681297e-01  1.635385e-01  1.588986e-01  1.542135e-01
## [416]  1.494863e-01  1.447204e-01  1.399190e-01  1.350854e-01  1.302229e-01
## [421]  1.253347e-01  1.204243e-01  1.154947e-01  1.105494e-01  1.055916e-01
## [426]  1.006246e-01  9.565166e-02  9.067609e-02  8.570115e-02  8.073014e-02
## [431]  7.576634e-02  7.081302e-02  6.587349e-02  6.095101e-02  5.604887e-02
## [436]  5.117035e-02  4.631874e-02  4.149733e-02  3.652232e-02  3.121552e-02
## [441]  2.558860e-02  1.965323e-02  1.342107e-02  6.903804e-03  1.130905e-04
## [446] -6.939400e-03 -1.424200e-02 -2.178304e-02 -2.955085e-02 -3.753377e-02
## [451] -4.572011e-02 -5.409823e-02 -6.265644e-02 -7.138309e-02 -8.026649e-02
## [456] -8.929498e-02 -9.845690e-02 -1.077406e-01 -1.171343e-01 -1.266265e-01
## [461] -1.362054e-01 -1.458594e-01 -1.555769e-01 -1.653460e-01 -1.751553e-01
## [466] -1.849929e-01 -1.948473e-01 -2.047068e-01 -2.145596e-01 -2.243942e-01
## [471] -2.341989e-01 -2.439619e-01 -2.536717e-01 -2.633165e-01 -2.728848e-01
## [476] -2.823647e-01 -2.917447e-01 -3.010131e-01 -3.101582e-01 -3.191683e-01
## [481] -3.280319e-01 -3.367371e-01 -3.452724e-01 -3.536260e-01 -3.617864e-01
## [486] -3.697418e-01 -3.774806e-01 -3.849910e-01 -3.922615e-01 -3.992804e-01
## [491] -4.060359e-01 -4.125165e-01 -4.187105e-01 -4.246061e-01 -4.301917e-01
## [496] -4.354557e-01 -4.403864e-01 -4.449721e-01 -4.492011e-01 -4.530618e-01
## [501] -4.566614e-01 -4.601158e-01 -4.634268e-01 -4.665960e-01 -4.696250e-01
## [506] -4.725156e-01 -4.752693e-01 -4.778878e-01 -4.803728e-01 -4.827259e-01
## [511] -4.849488e-01 -4.870432e-01 -4.890107e-01 -4.908530e-01 -4.925716e-01
## [516] -4.941683e-01 -4.956448e-01 -4.970027e-01 -4.982436e-01 -4.993692e-01
## [521] -5.003811e-01 -5.012811e-01 -5.020707e-01 -5.027517e-01 -5.033257e-01
## [526] -5.037942e-01 -5.041591e-01 -5.044220e-01 -5.045844e-01 -5.046481e-01
## [531] -5.046148e-01 -5.044860e-01 -5.042634e-01 -5.039488e-01 -5.035437e-01
## [536] -5.030498e-01 -5.024687e-01 -5.018022e-01 -5.010519e-01 -5.002193e-01
## [541] -4.993063e-01 -4.983144e-01 -4.972453e-01 -4.961007e-01 -4.948822e-01
## [546] -4.935914e-01 -4.922301e-01 -4.907999e-01 -4.893024e-01 -4.877393e-01
## [551] -4.861122e-01 -4.844229e-01 -4.826729e-01 -4.808640e-01 -4.789977e-01
## [556] -4.770758e-01 -4.750998e-01 -4.730716e-01 -4.709926e-01 -4.688645e-01
## [561] -4.666891e-01 -4.644680e-01 -4.622028e-01 -4.598403e-01 -4.573285e-01
## [566] -4.546714e-01 -4.518725e-01 -4.489355e-01 -4.458644e-01 -4.426627e-01
## [571] -4.393342e-01 -4.358826e-01 -4.323117e-01 -4.286253e-01 -4.248269e-01
## [576] -4.209204e-01 -4.169096e-01 -4.127980e-01 -4.085895e-01 -4.042879e-01
## [581] -3.998967e-01 -3.954198e-01 -3.908610e-01 -3.862238e-01 -3.815121e-01
## [586] -3.767296e-01 -3.718801e-01 -3.669672e-01 -3.619947e-01 -3.569663e-01
## [591] -3.518858e-01 -3.467569e-01 -3.415833e-01 -3.363688e-01 -3.311171e-01
## [596] -3.258319e-01 -3.205169e-01 -3.151760e-01 -3.098128e-01 -3.044310e-01
## [601] -2.990344e-01 -2.936268e-01 -2.882118e-01 -2.827932e-01 -2.773747e-01
## [606] -2.719601e-01 -2.665531e-01 -2.611574e-01 -2.557767e-01 -2.504148e-01
## [611] -2.450755e-01 -2.397624e-01 -2.344793e-01 -2.292299e-01 -2.240180e-01
## [616] -2.188472e-01 -2.137214e-01 -2.086442e-01 -2.036194e-01 -1.986508e-01
## [621] -1.937419e-01 -1.888967e-01 -1.841188e-01 -1.794119e-01 -1.747798e-01
## [626] -1.701003e-01 -1.652525e-01 -1.602418e-01 -1.550736e-01 -1.497532e-01
## [631] -1.442862e-01 -1.386780e-01 -1.329339e-01 -1.270594e-01 -1.210598e-01
## [636] -1.149408e-01 -1.087075e-01 -1.023656e-01 -9.592027e-02 -8.937709e-02
## [641] -8.274143e-02 -7.601871e-02 -6.921436e-02 -6.233379e-02 -5.538242e-02
## [646] -4.836567e-02 -4.128896e-02 -3.415772e-02 -2.697735e-02 -1.975329e-02
## [651] -1.249094e-02 -5.195738e-03  2.126908e-03  9.471575e-03  1.683284e-02
```

```
## [656] 2.420529e-02 3.158351e-02 3.896206e-02 4.633554e-02 5.369852e-02
## [661] 6.104558e-02 6.837131e-02 7.567028e-02 8.293707e-02 9.016628e-02
## [666] 9.735246e-02 1.044902e-01 1.115741e-01 1.185987e-01 1.255587e-01
## [671] 1.324485e-01 1.392628e-01 1.459961e-01 1.526430e-01 1.591982e-01
## [676] 1.656562e-01 1.720115e-01 1.782588e-01 1.843926e-01 1.904075e-01
## [681] 1.962981e-01 2.020590e-01 2.076847e-01 2.131698e-01 2.185090e-01
## [686] 2.236968e-01 2.287277e-01 2.335964e-01 2.383698e-01 2.431180e-01
## [691] 2.478398e-01 2.525341e-01 2.571997e-01 2.618356e-01 2.664406e-01
## [696] 2.710136e-01 2.755534e-01 2.800591e-01 2.845295e-01 2.889633e-01
## [701] 2.933596e-01 2.977173e-01 3.020351e-01 3.063120e-01 3.105468e-01
## [706] 3.147385e-01 3.188859e-01 3.229880e-01 3.270435e-01 3.310514e-01
## [711] 3.350106e-01 3.389199e-01 3.427783e-01 3.465846e-01 3.503376e-01
## [716] 3.540364e-01 3.576797e-01 3.612664e-01 3.647955e-01 3.682658e-01
## [721] 3.716762e-01 3.750256e-01 3.783128e-01 3.815368e-01 3.846965e-01
## [726] 3.877906e-01 3.908181e-01 3.937779e-01 3.966689e-01 3.994900e-01
## [731] 4.022399e-01 4.049177e-01 4.075222e-01 4.100522e-01 4.125068e-01
## [736] 4.148846e-01 4.171848e-01 4.194060e-01 4.215472e-01 4.236073e-01
## [741] 4.255852e-01 4.274797e-01 4.292898e-01 4.310143e-01 4.326521e-01
## [746] 4.342020e-01 4.356631e-01 4.370340e-01 4.383139e-01 4.395014e-01
## [751] 4.405962e-01 4.415999e-01 4.425140e-01 4.433402e-01 4.440803e-01
## [756] 4.447360e-01 4.453088e-01 4.458006e-01 4.462129e-01 4.465476e-01
## [761] 4.468062e-01 4.469904e-01 4.471020e-01 4.471426e-01 4.471140e-01
## [766] 4.470177e-01 4.468556e-01 4.466292e-01 4.463403e-01 4.459906e-01
## [771] 4.455817e-01 4.451153e-01 4.445932e-01 4.440169e-01 4.433883e-01
## [776] 4.427089e-01 4.419805e-01 4.412048e-01 4.403834e-01 4.395180e-01
## [781] 4.386104e-01 4.376621e-01 4.366750e-01 4.356506e-01 4.345907e-01
## [786] 4.334970e-01 4.323711e-01 4.312147e-01 4.300296e-01 4.288174e-01
## [791] 4.275797e-01 4.263184e-01 4.250350e-01 4.237312e-01 4.224088e-01
## [796] 4.210695e-01 4.197149e-01 4.183466e-01 4.169665e-01 4.155761e-01
## [801] 4.141773e-01 4.127715e-01 4.113607e-01 4.099463e-01 4.085302e-01
## [806] 4.071140e-01 4.056994e-01 4.042880e-01 4.028816e-01 4.014819e-01
## [811] 4.000906e-01 3.987092e-01 3.973396e-01 3.959141e-01 3.943663e-01
## [816] 3.926999e-01 3.909187e-01 3.890263e-01 3.870264e-01 3.849228e-01
## [821] 3.827191e-01 3.804190e-01 3.780262e-01 3.755444e-01 3.729773e-01
## [826] 3.703287e-01 3.676021e-01 3.648014e-01 3.619301e-01 3.589920e-01
## [831] 3.559908e-01 3.529302e-01 3.498138e-01 3.466454e-01 3.434287e-01
## [836] 3.401674e-01 3.368651e-01 3.335256e-01 3.301525e-01 3.267495e-01
## [841] 3.233205e-01 3.198689e-01 3.163986e-01 3.129132e-01 3.094165e-01
## [846] 3.059121e-01 3.024036e-01 2.988950e-01 2.953897e-01 2.918915e-01
## [851] 2.884042e-01 2.849313e-01 2.814767e-01 2.780439e-01 2.746367e-01
## [856] 2.712588e-01 2.679139e-01 2.646057e-01 2.613378e-01 2.581141e-01
## [861] 2.549380e-01 2.518135e-01 2.487441e-01 2.457336e-01 2.427856e-01
## [866] 2.399038e-01 2.370920e-01 2.343539e-01 2.316930e-01 2.291132e-01
## [871] 2.266182e-01 2.242115e-01 2.218970e-01 2.196783e-01 2.175591e-01
## [876] 2.154896e-01 2.134177e-01 2.113441e-01 2.092692e-01 2.071938e-01
## [881] 2.051182e-01 2.030431e-01 2.009691e-01 1.988966e-01 1.968263e-01
## [886] 1.947587e-01 1.926944e-01 1.906339e-01 1.885778e-01 1.865266e-01
## [891] 1.844809e-01 1.824414e-01 1.804084e-01 1.783826e-01 1.763646e-01
## [896] 1.743548e-01 1.723539e-01 1.703625e-01 1.683810e-01 1.664100e-01
## [901] 1.644502e-01 1.625020e-01 1.605660e-01 1.586427e-01 1.567329e-01
## [906] 1.548369e-01 1.529553e-01 1.510888e-01 1.492378e-01 1.474029e-01
## [911] 1.455848e-01 1.437839e-01 1.420007e-01 1.402360e-01 1.384902e-01
## [916] 1.367638e-01 1.350575e-01 1.333718e-01 1.317072e-01 1.300644e-01
## [921] 1.284438e-01 1.268460e-01 1.252717e-01 1.237213e-01 1.221954e-01
```

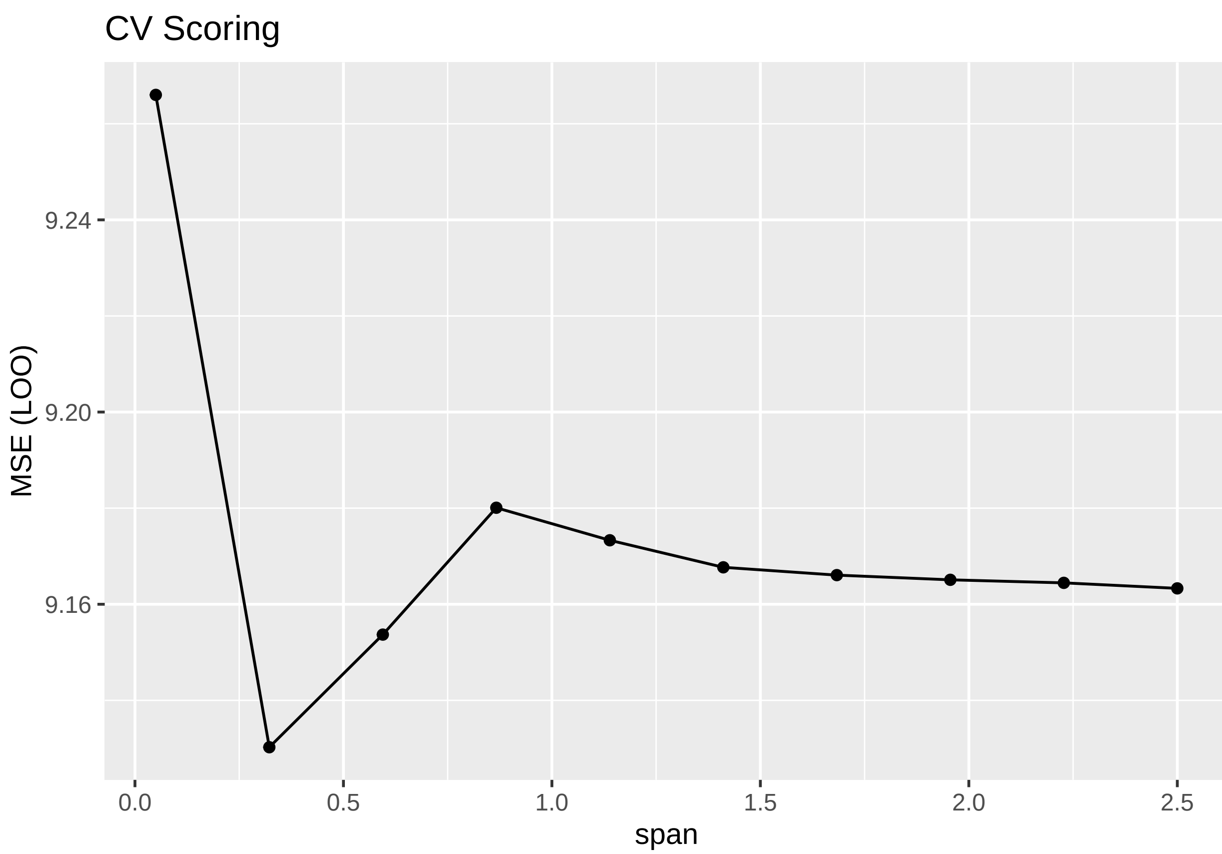
```
## [926] 1.206946e-01 1.192194e-01 1.177704e-01 1.163481e-01 1.149532e-01
## [931] 1.135861e-01 1.122474e-01 1.109377e-01 1.096576e-01 1.084076e-01
## [936] 1.071882e-01 1.060000e-01 1.048436e-01 1.037128e-01 1.026010e-01
## [941] 1.015080e-01 1.004339e-01 9.937861e-02 9.834198e-02 9.732398e-02
## [946] 9.632454e-02 9.534358e-02 9.438105e-02 9.343686e-02 9.251096e-02
## [951] 9.160328e-02 9.071374e-02 8.984227e-02 8.898881e-02 8.815328e-02
## [956] 8.733563e-02 8.653577e-02 8.575364e-02 8.498918e-02 8.424230e-02
## [961] 8.351295e-02 8.280105e-02 8.210653e-02 8.142933e-02 8.076938e-02
## [966] 8.012660e-02 7.950093e-02 7.889230e-02 7.830064e-02 7.772588e-02
## [971] 7.716795e-02 7.662678e-02 7.610231e-02 7.559446e-02 7.510316e-02
## [976] 7.462835e-02 7.416996e-02 7.372792e-02 7.330215e-02 7.289260e-02
## [981] 7.249918e-02 7.212184e-02 7.176049e-02 7.141509e-02 7.108554e-02
## [986] 7.077179e-02 7.047377e-02 7.019140e-02 6.992462e-02 6.967336e-02
## [991] 6.943755e-02 6.921712e-02 6.901200e-02 6.882212e-02 6.864741e-02
## [996] 6.848781e-02 6.834325e-02 6.821365e-02 6.809894e-02 6.799907e-02
```

And for the third:

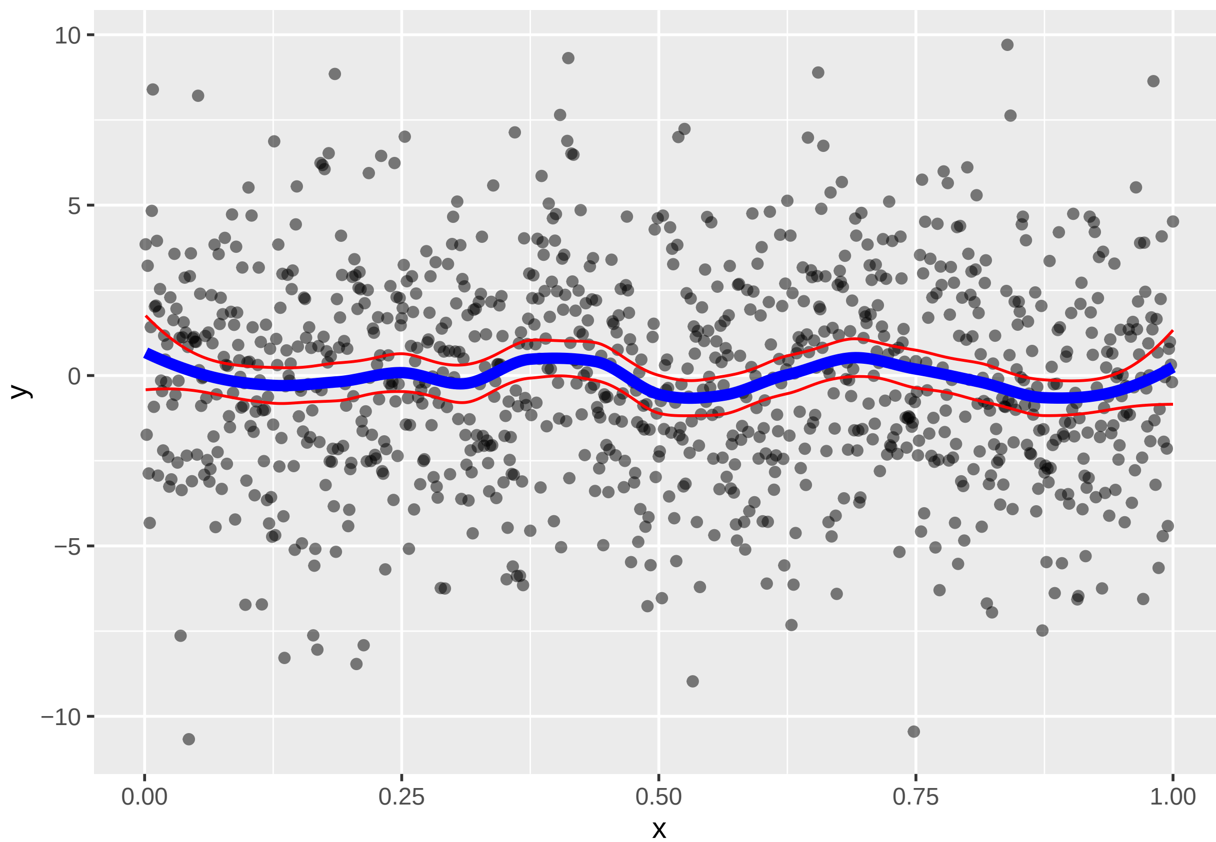
```r
Y = y3
cv_result = cv_loess(x=xi, y=Y)
```

```
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
## -------------
```

```r
## plot MSE
ggplot(cv_result, aes(x=span, y=mse_loo)) +
  geom_path() +
  geom_point() +
  labs(title="CV Scoring", x="span", y="MSE (LOO)")
```




```r
## final fit
fit = loess(Y ~ xi, span=cv_result$span[which.min(cv_result$mse_loo)])

ggplot(
  data.frame(
    x=xi,
    yhat=predict(fit, xi),
    y=Y,
    ci_upr = predict(fit, xi) + 1.96 * predict(fit, xi, se=T)$se.fit,
    ci_lwr = predict(fit, xi) - 1.96 * predict(fit, xi, se=T)$se.fit
  ),
  aes(x=x, y=y)
) +
  geom_point(alpha=.5) +
  geom_path(aes(x=x, y=yhat), color="blue", size=2) +
  geom_path(aes(x=x, y=ci_upr), color="red")  +
  geom_path(aes(x=x, y=ci_lwr), color="red")
```



5.16.15

For notational convenience, let's change the problem to minimizing $\sum_i^N (Y_i - \hat{\theta}_i) + \ldots$, i.e. using $\hat{\mu}_i := \hat{\theta}_i$. We can then treat each row of data as an indicator for being in that row, with $\Theta \in \mathbb{R}^N$ as an OLS-style coefficient, and we will recover the above loss function, i.e.

$$X = I_N$$

and

$$\Theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_N \end{bmatrix}$$

We would then seek to minimize

$$||Y - X\Theta|_2^2 = ||Y - I_N\Theta||_2^2$$

subject to whatever penalty.

a.)

Now, we are minimizing

$$||Y - I_N\Theta||_2^2 + \lambda||\Theta||_2^2,$$

which is a ridge regression. Thus, we have

$$\hat{\Theta} = (I^T I + \lambda I)^{-1} I^T Y = \frac{1}{1+\lambda}(Y).$$

b.)

Now, we minimize

$$||Y - I_N\Theta||_2^2 + \lambda||\Theta||_1,$$

which we immediately recognize as a Lasso regression. Hence, the coordinate-descent/fixed-point solution for a lasso regression of Y on I subject to λ L1 regularization will produce our estimator.

c.)

In a general OLS setting, recall that in any OLS setting with coefficients $\theta \in mathbbR^D$, we have that $P(\theta_d = 0) = 0$; hence the penalization scheme is trivial, with no penalty incurred, and so the minimization will simply amount to

$$\arg\min_{\Theta} ||Y - X\Theta||_2^2 + \lambda \sum_d \not\Vdash(\theta_d = 0) = \arg\min_{\Theta} ||Y - X\Theta||_2^2 \implies \hat{\Theta} = (X^T X)^{-1} X^T Y.$$

Hence, in this setting, we will ostensibly have

$$\arg\min_{\Theta} ||Y - I_N\Theta||_2^2 \implies \hat{\Theta} = Y.$$

However, there is one catch: if one of the Y_i's equals zero, then a $\hat{\theta}_d$ will necessarily equal zero, and we will incur some sort of penalty; meanwhile, every other RSS for which $Y_i = \theta_i \neq 0$ will be minimized. As a practical solution, we'll do the following: for all $i : Y_i = 0$, we will assign $\theta_i = \epsilon$ for some very tiny $\epsilon > 0$. This will have the effect of adding minimal RSS penalty on the rows where $Y_i = 0$, while allowing all other $Y_i \neq 0$ sums-of-squares to remain minimized. Hence, we construct $\hat{\Theta}$ such that $\hat{\theta}_i = Y_i + \epsilon\not\Vdash(Y_i = 0)$ for some sufficiently small (ideally, as small as possible) $\epsilon > 0$.

5.16.22

Using default configurations for all models ### i.) The MLR fit is:

```
airquality_wr = airquality %>%
  filter(!is.na(Ozone), !is.na(Solar.R), !is.na(Wind), !is.na(Temp))

fit_mlr = lm(Ozone ~ Solar.R + Wind + Temp, data=airquality_wr)
```

ii.)

The tree-based fit (using default settings):

```
fit_tree = rpart::rpart(Ozone ~ Solar.R + Wind + Temp, data=airquality_wr)
```

iii.)

The MARS fit is:

```
fit_mars = earth::earth(Ozone ~ Solar.R + Wind + Temp, data=airquality_wr)
```

iv.)

The GAM fit is:

```
fit_gam = mgcv::gam(Ozone ~ Solar.R + Wind + Temp, data=airquality_wr)
```

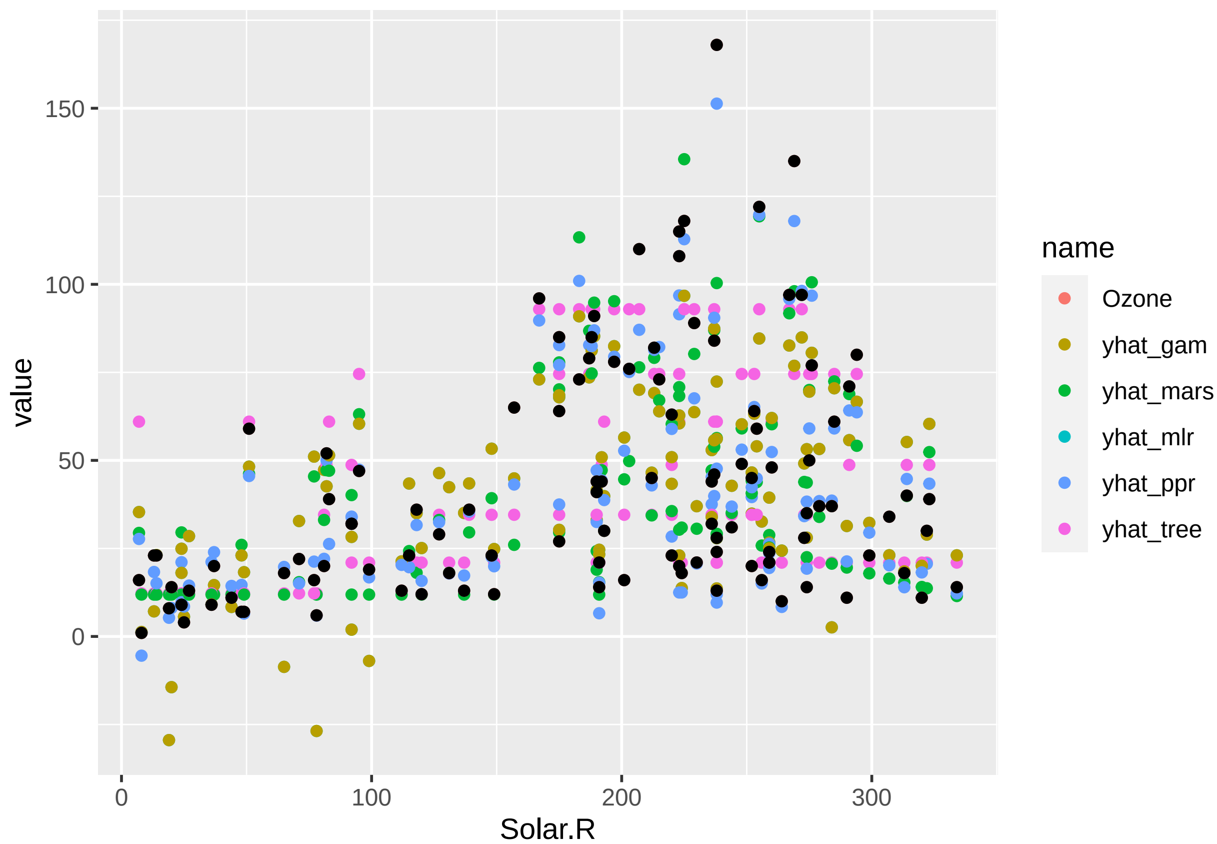
v.)

The PPR fit is

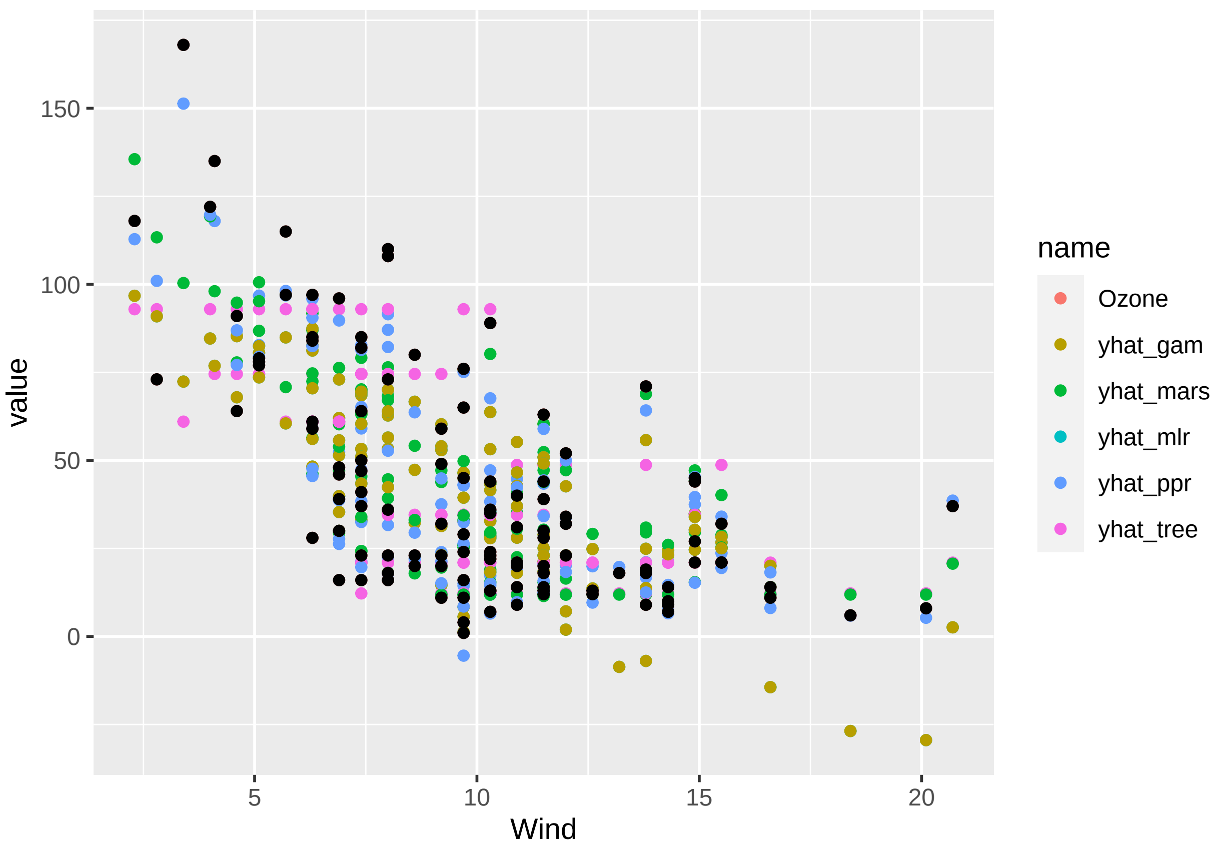
```
fit_ppr = ppr(
  x=as.matrix(airquality_wr[, c("Solar.R", "Wind", "Temp")]),
  y=airquality_wr$Ozone,
  nterms=5
)
```

Then, we re-predict on the training set

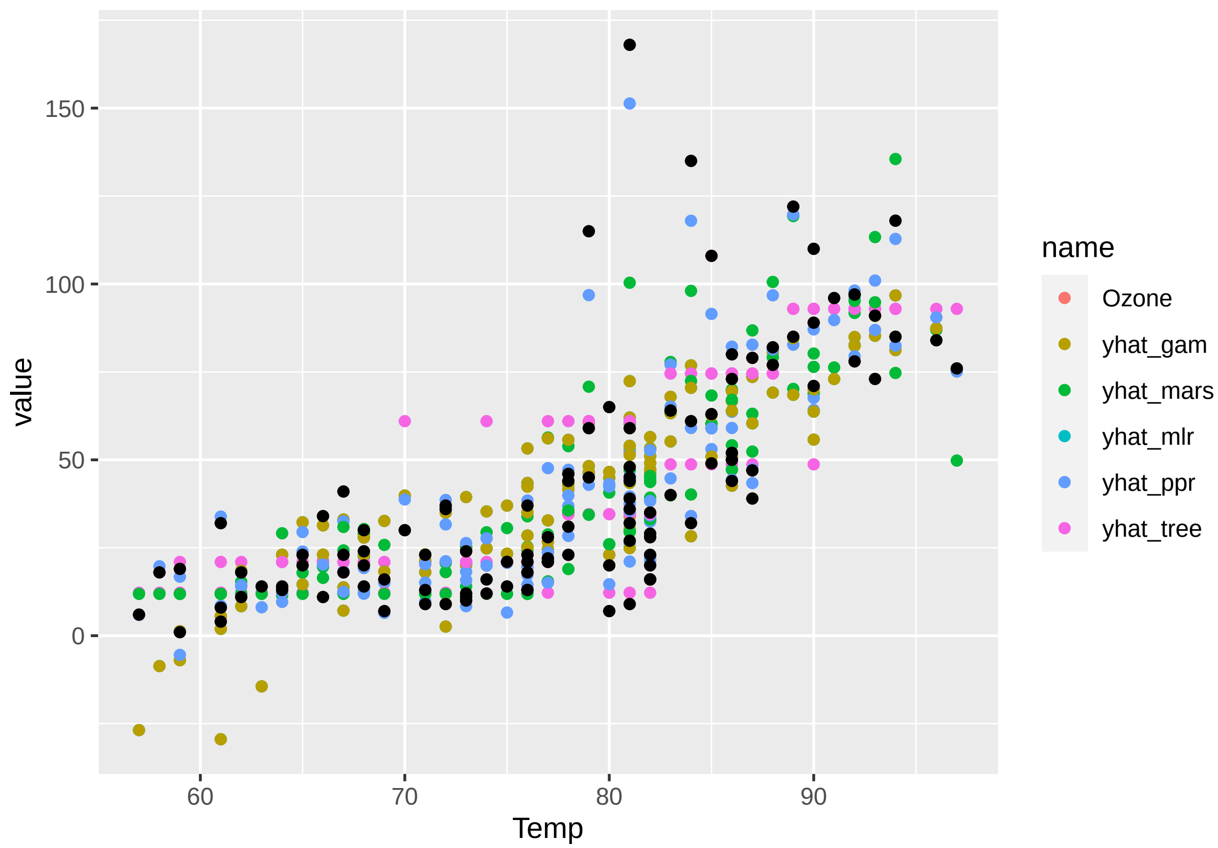
```
pred_df= airquality_wr # copy
pred_df$yhat_mlr = predict(fit_mlr, newdata=pred_df)
pred_df$yhat_tree = predict(fit_tree, newdata=pred_df)
pred_df$yhat_mars = predict(fit_mars, newdata=pred_df)
pred_df$yhat_gam = predict(fit_gam, newdata=pred_df)
pred_df$yhat_ppr = predict(fit_ppr, as.matrix(airquality_wr[, c("Solar.R", "Wind", "Temp")]))

pred_df = pred_df %>%
  tidyr::pivot_longer(., cols=c("yhat_mlr", "yhat_tree", "yhat_mars", "yhat_gam", "yhat_ppr", "Ozone"))

ggplot(pred_df, aes(x=Solar.R, y=value, color=name)) +
  geom_point() +
  geom_point(
    data=pred_df %>% filter(name == "Ozone"),
    mapping=aes(x=Solar.R, y=value),
    color="black"
  )
```




```
ggplot(pred_df, aes(x=Wind, y=value, color=name)) +
  geom_point() +
  geom_point(
    data=pred_df %>% filter(name == "Ozone"),
    mapping=aes(x=Wind, y=value),
    color="black"
  )
```




```
ggplot(pred_df, aes(x=Temp, y=value, color=name)) +
  geom_point() +
  geom_point(
    data=pred_df %>% filter(name == "Ozone"),
    mapping=aes(x=Temp, y=value),
    color="black"
  )
```



In general, we see that each of the predictions is roughly the same – they're all generally in the same neighborhood for a given covariate, and generally match the shape/spread/layout of the observed data (in the scatterplot sense). However, there are two things of note – the PPR is most willing to make "aggressive" predictions, particularly high predictions for ozone. We see a handful of these in the `75 <= Temp <= 90` range / `200 <= Solar.R <= 300` range. Perhaps this is overfit due to a lack of regularization; alternatively, it maybe an inherent preference towards higher variance. Second, we see the GAM underestimate for high wind / low temp datapoints (plots 2/3 in gold). Again, it's hard to make generalizations here, but the behavior does stand out.

6.9.6

i.) Densities

First, we perform LOO CV over kernels and bandwidths. Following from 6.4, we compute the CV estimator of risk. Notably, I cannot get `integrate()` to work over these densities, so I'll just do the first part of J by just taking an area under the density (AUC). I've left in the relevant code however.

```
library(zoo)
```

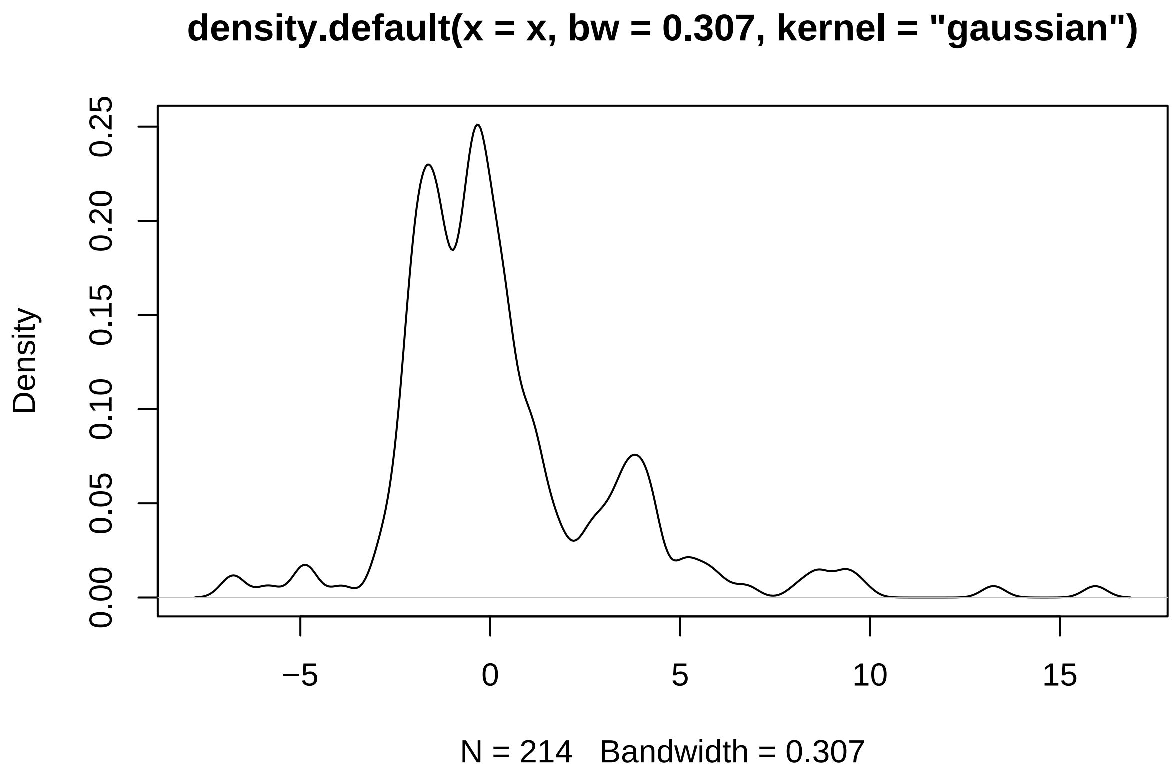
```
##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
x = read.csv("./data/forensic.csv")[, 2] # columns are adjused one over
N = length(x)
lapply(seq(1e-3, 5, length.out=50), function(w){
  lapply(c( # iterate over possibilities
    "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine", "optcosine"
  ), function(ker){
    #' Compute the holdout loss
    sapply(1:N, function(i){
      kde_ = density(x[(1:N) != i], bw=w, kernel=ker);
      fout = approx(kde_$x, kde_$y, xout=x[i])$y
    }) %>%
      mean(., na.rm=T) * 2 -> cv_holdout_loss
    # compute the full fit
    kde_full = density(x, bw=w, kernel=ker)

    ### This won't work ###
    # squared fhat function to integrate over
    # fhat_n <- function(z){
    #   result = (approx(kde_full$x, kde_full$y, xout=z)$y) ^ 2
    #   # hacky solution
    #   result = ifelse(is.na(result), 0, result)
    #   result
    # }
    # full_dens = integrate(fhat_n, lower=-10, 20)
    ###  ###

    id <- order(x)
    x_ = (kde_full$x) ^ 2; y_ = (kde_full$y) ^ 2
    id <- order(x_)
    AUC <- sum(diff(x_[id]) * rollmean(y_[id],2))
    data.frame(kernel=ker, bw=w, risk=AUC - cv_holdout_loss)
  }) %>%
    do.call("rbind", .)
}) %>%
  do.call("rbind", .) -> cv_results

cv_results  %>%
  arrange(risk) %>%
  head(5)
```

```
##       kernel        bw       risk
## 1   gaussian 0.3070612 -0.1217934
## 2     cosine 0.3070612 -0.1215729
## 3 triangular 0.3070612 -0.1215602
## 4   gaussian 0.5111020 -0.1215452
## 5   biweight 0.3070612 -0.1214726
```

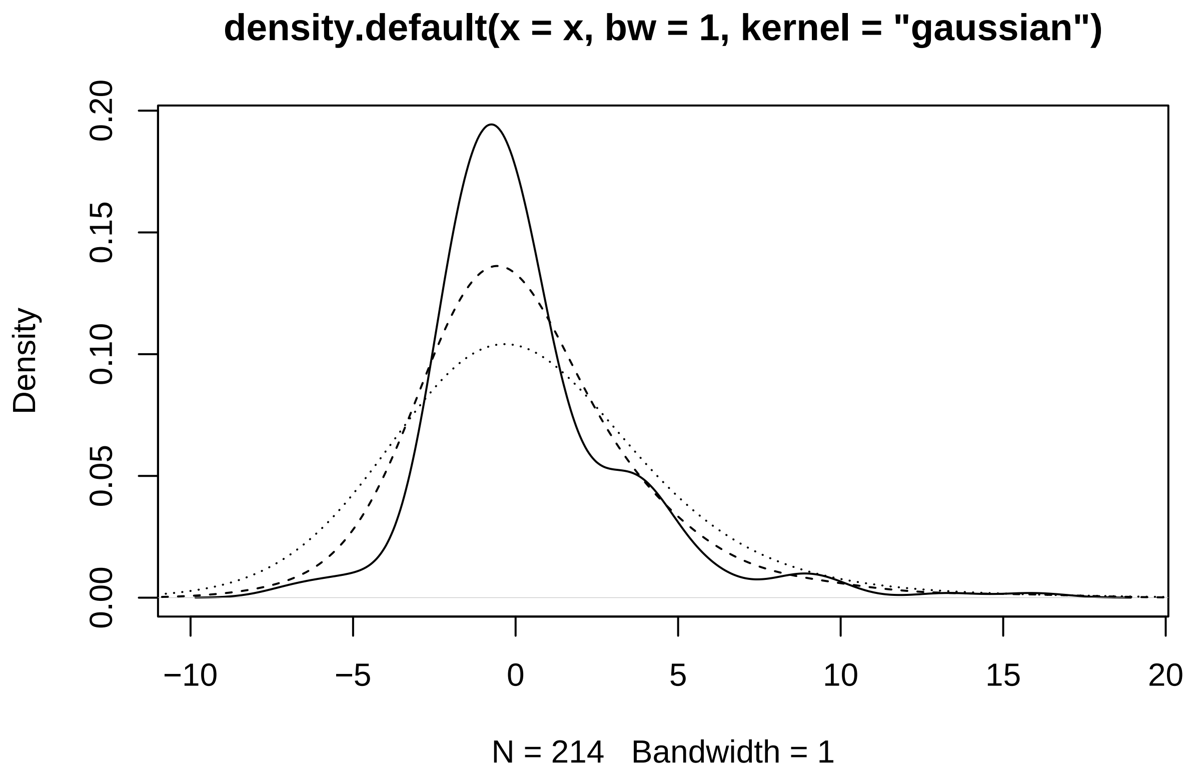
Hence, we choose a `bw=.307` and a Gaussian kernel; we proceed to refit

```r
kdefit = density(x, bw=.307, kernel="gaussian")

plot(kdefit)
```



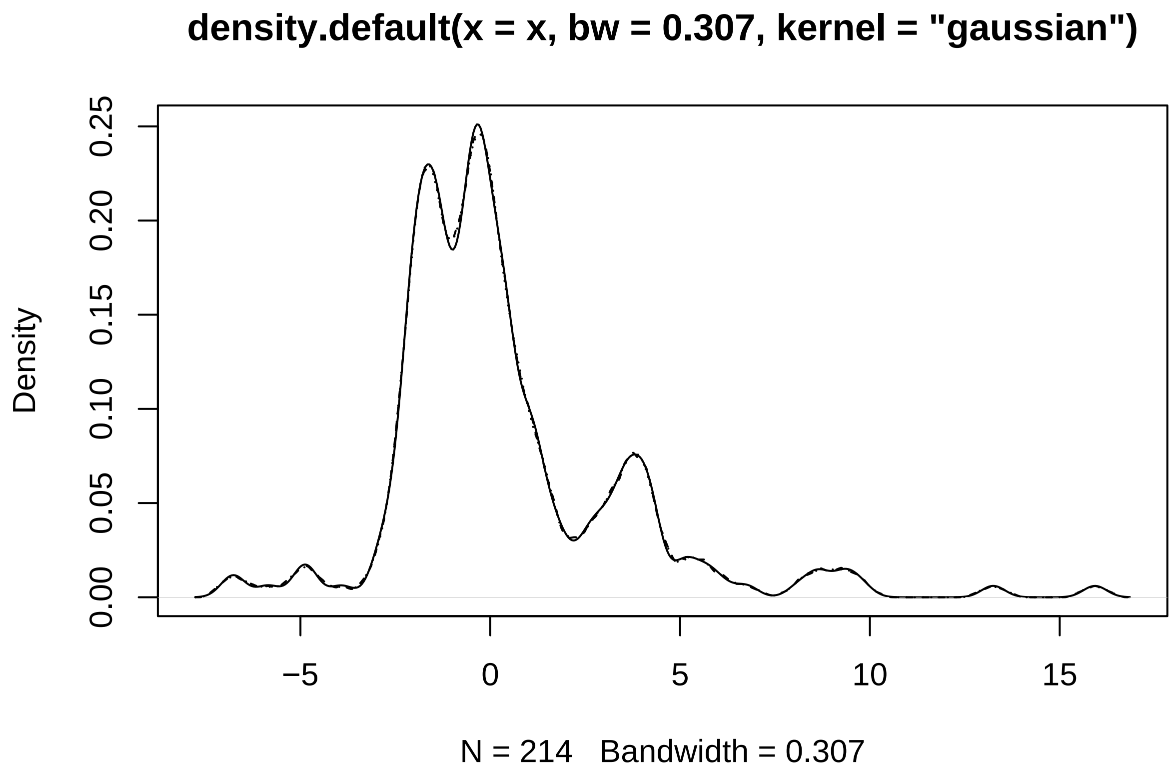
We see some modality and flexibility in this fit; if we had instead chosen `bw=1` or `bw=2` (dashed) or `bw=3` (dotted) with this Gaussian KDE, we'd have seen something progressively more smooth and unimodal

```
plot(density(x, bw=1, kernel="gaussian"))
lines(density(x, bw=2, kernel="gaussian"), lty="dashed")
lines(density(x, bw=3, kernel="gaussian"), lty="dotted")
```



We can also plot versus rectangular (black), triangular (dashed), and cosine kernels for comparison (all reverting to `bw=.307` here):

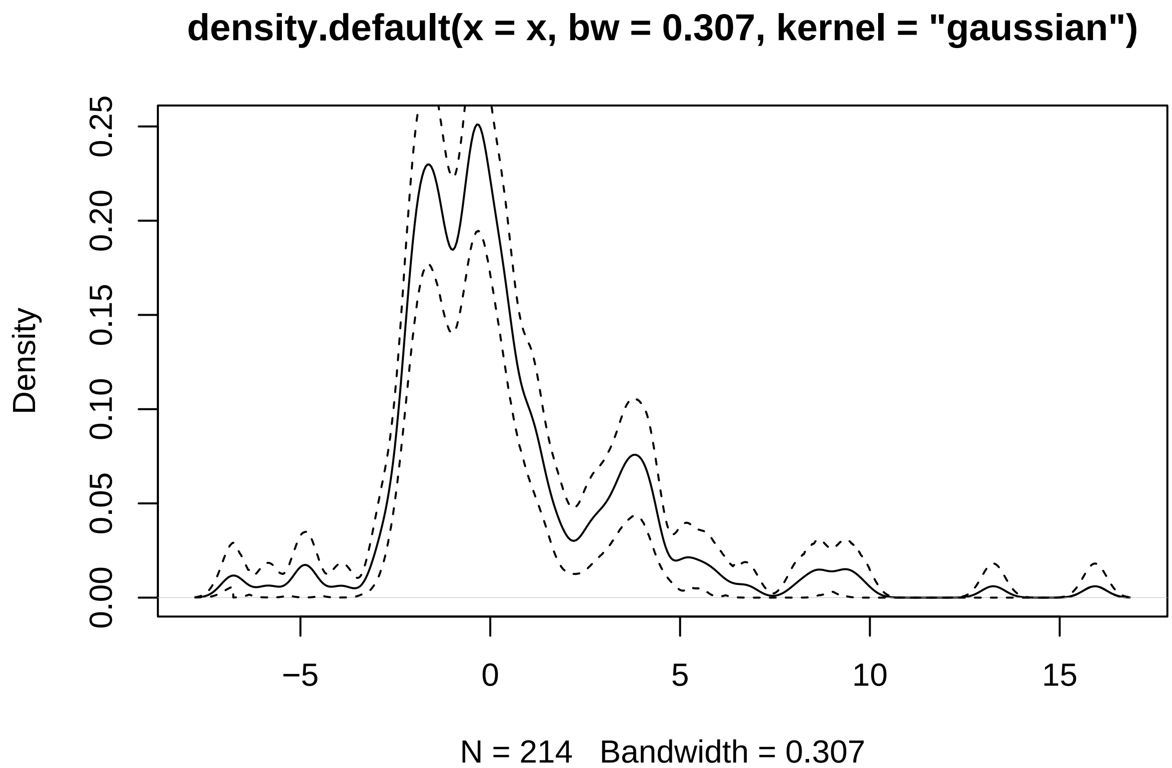
```
plot(density(x, bw=.307, kernel="gaussian"))
lines(density(x, bw=.307, kernel="triangular"), lty="dashed")
lines(density(x, bw=.307, kernel="cosine"), lty="dotted")
```



Impressively, all come out the same. upon inspection, it looks like an issue with the `kernel` argument in `density()`. For whatever reason, the `kernel` argument doesn't seem to be leveraged within the `stats::density()` function.

Then, we bootstrap (`S=10000`) under the original settings to get a 95% band:

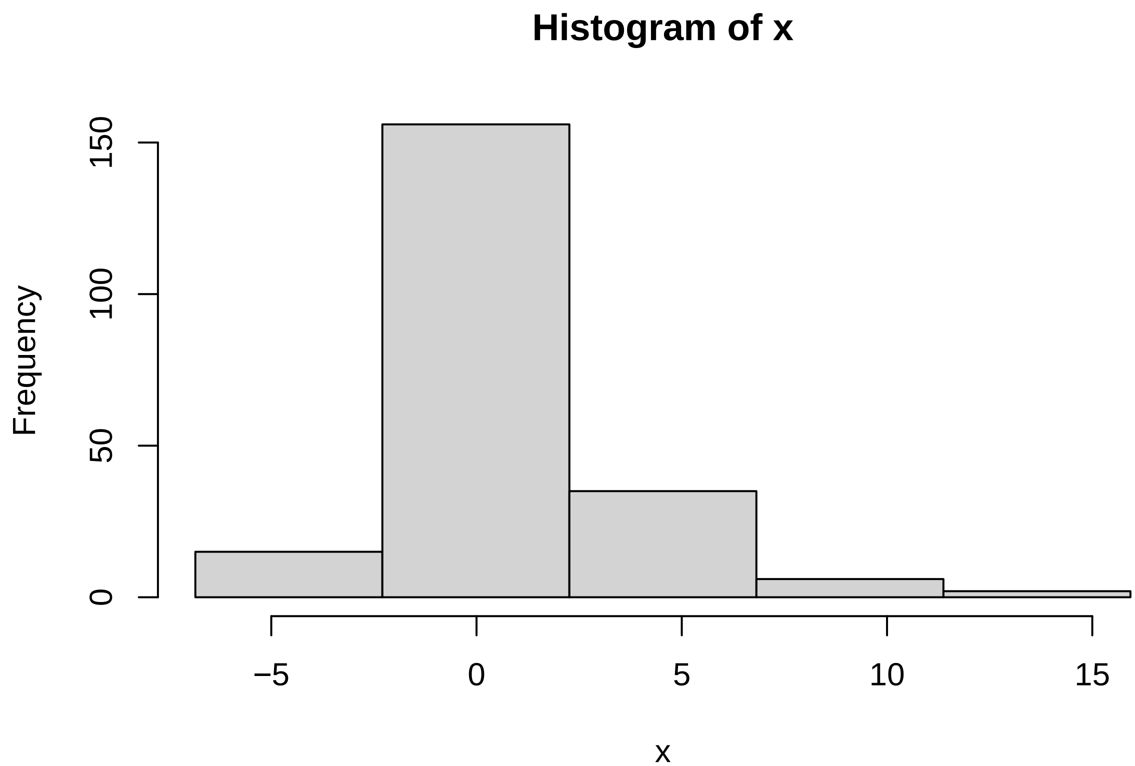
```
xhat = seq(-8, 17, length.out=5000)
lapply(1:20000, function(i){
  xi = sample(x, length(x), replace=T)
  kd = density(xi, bw=.307, kernel="gaussian")
  yhat = approx(kd$x, kd$y, xout=xhat)$y
  }
) %>%
  do.call("rbind", .) -> boot_result

conf_band = apply(boot_result, 2, quantile, c(.025, .95), na.rm=T) %>% t()
plot(kdefit)
lines(xhat, conf_band[, 1], lty="dashed")
lines(xhat, conf_band[, 2], lty="dashed")
```

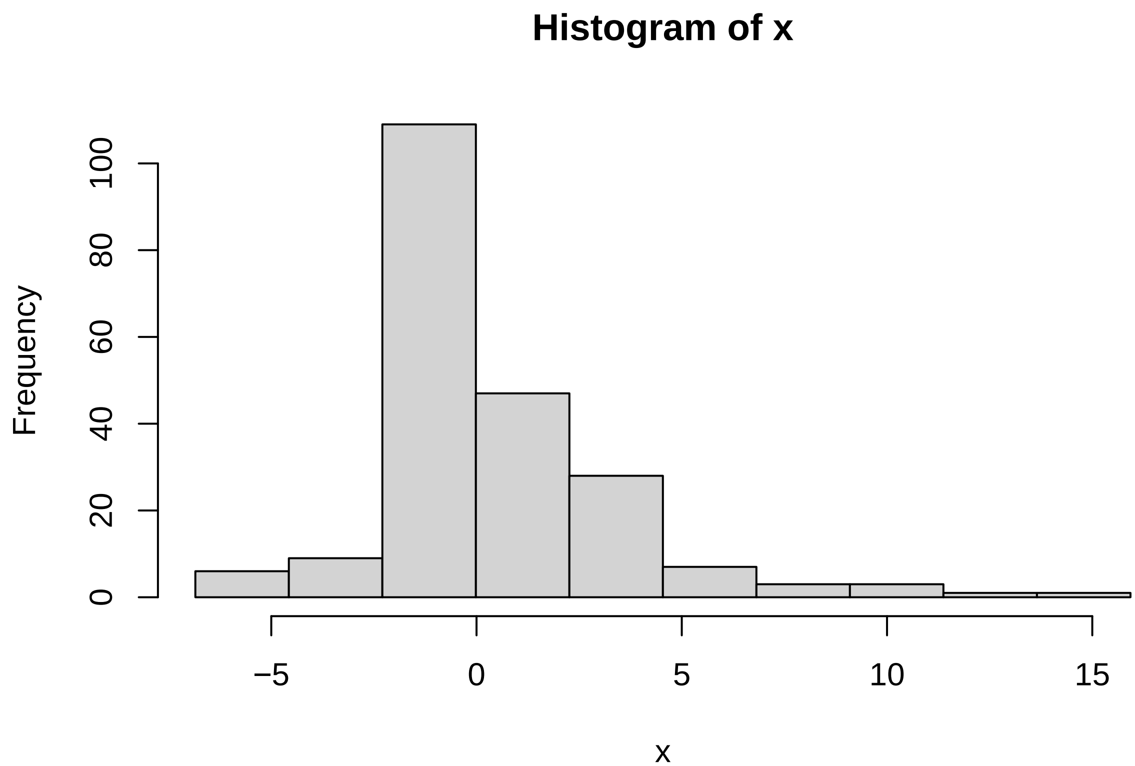


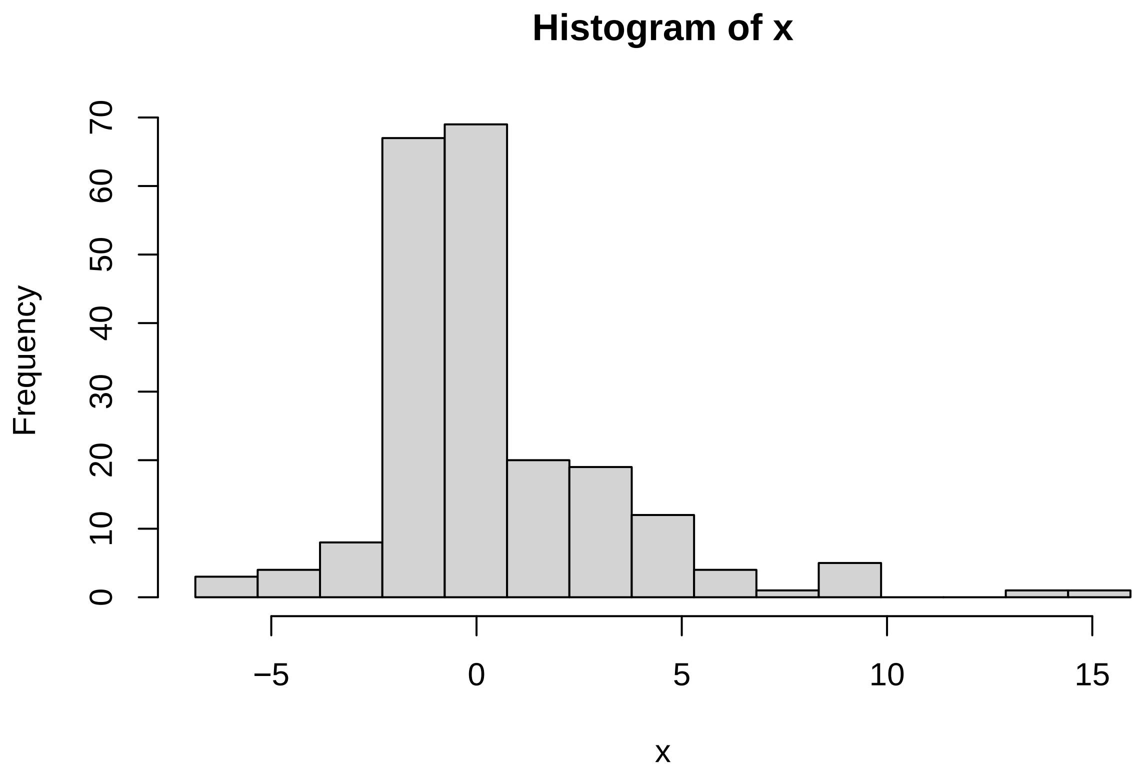
ii.) Histograms

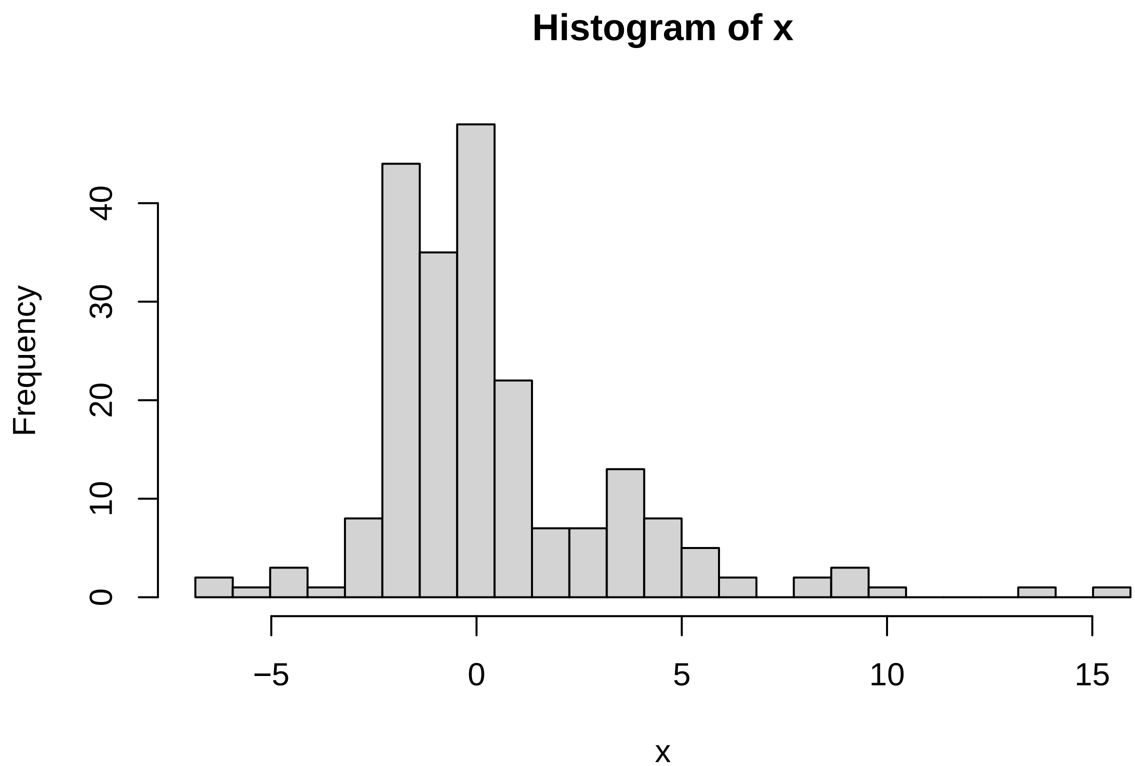
Below, we see that the behavior of the histograms (unsurprisingly) mirrors that of the KDE: that is, more buckets/smaller binwidths are like smaller bandwidths, in that they create more multi-model/“erratic” histograms. In contrast, fewer buckets/larger binwidths are like larger bandwidths, insofar as they lump more of the data/neighbors together and create a smoother fit. To see this, consider the histograms for `bin_width=[5, 10, 15, 25, 50]`:

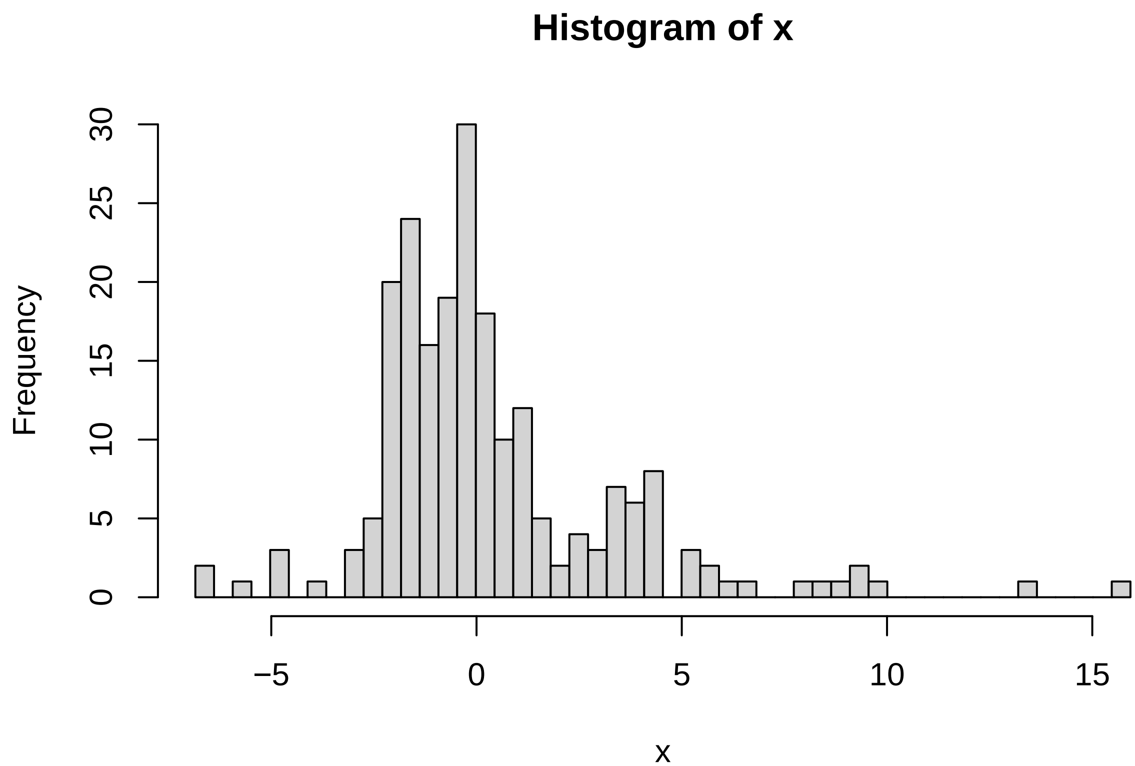
```
for (bw in c(5, 10, 15, 25, 50)){
  hist(
    x,
    breaks=seq(min(x), max(x), length.out=bw + 1)
  )
}
```


Histogram of x
Frequency
0
50
100
150
−5
0
5
10
15
x


Histogram of x
Frequency
0
20
40
60
80
100
−5
0
5
10
15
x


Histogram of x
Frequency
0
10
20
30
40
50
60
70
−5
0
5
10
15
x


Histogram of x
Frequency
0
10
20
30
40
−5
0
5
10
15
x



6.9.10

Function to generate density described in 6.10:

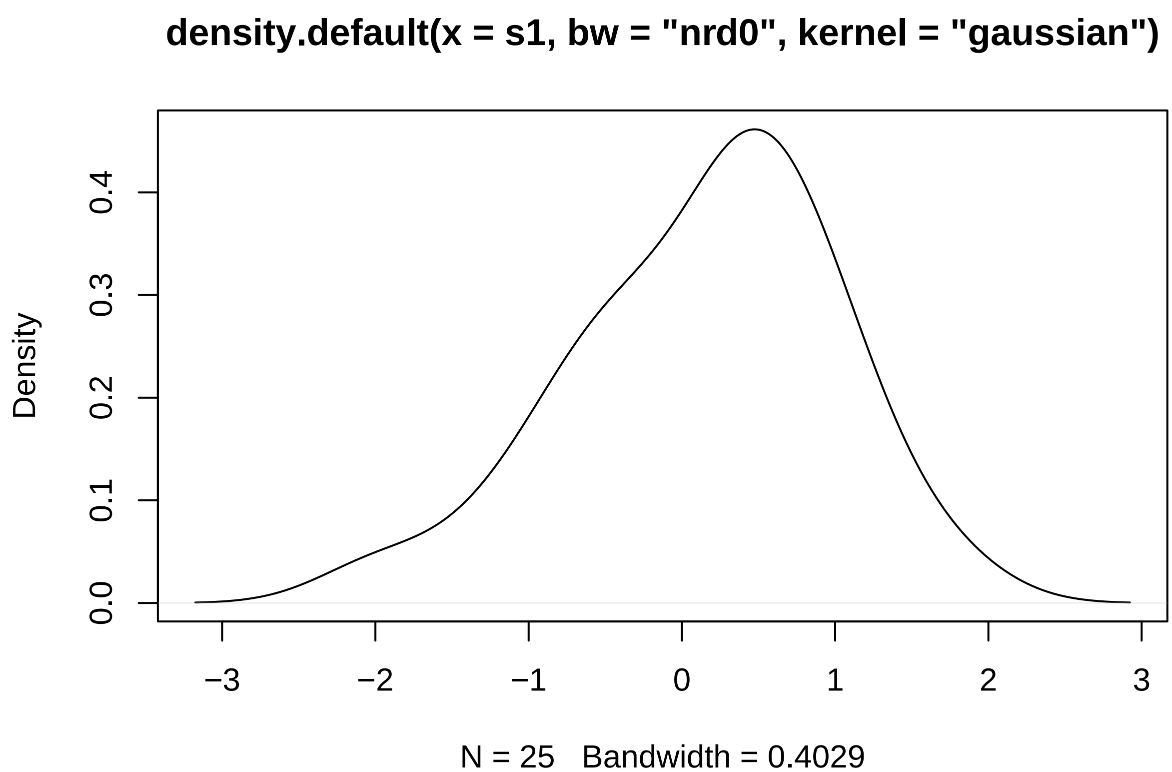
```
generate_data <- function(n, random_state=123){
  #' misread the problem
  set.seed(random_state)
  draws = cbind(
    rnorm(n, 0, 1),
    rnorm(n, 0/2 - 1, 1/10),
    rnorm(n, 1/2 - 1, 1/10),
    rnorm(n, 2/2 - 1, 1/10),
    rnorm(n, 3/2 - 1, 1/10),
    rnorm(n, 4/2 - 1, 1/10)
  )
  z = sample(1:6, n, replace=T, prob=c(1/2, rep(1/10, 5)))
  sapply(1:n, function(i) draws[i, z[i]])
}
```

We then draw the requisite $n = 25, 50, 100, 1,000$ samples:

```
s1 = generate_data(25); kde1 = density(s1, bw="nrd0", kernel="gaussian")
s2 = generate_data(50); kde2 = density(s2, bw="nrd0", kernel="gaussian")
s3 = generate_data(100); kde3 = density(s3, bw="nrd0", kernel="gaussian")
s4 = generate_data(1000); kde4 = density(s4, bw="nrd0", kernel="gaussian")
```

First, for `n=25`, we have:

```
plot(kde1)
```

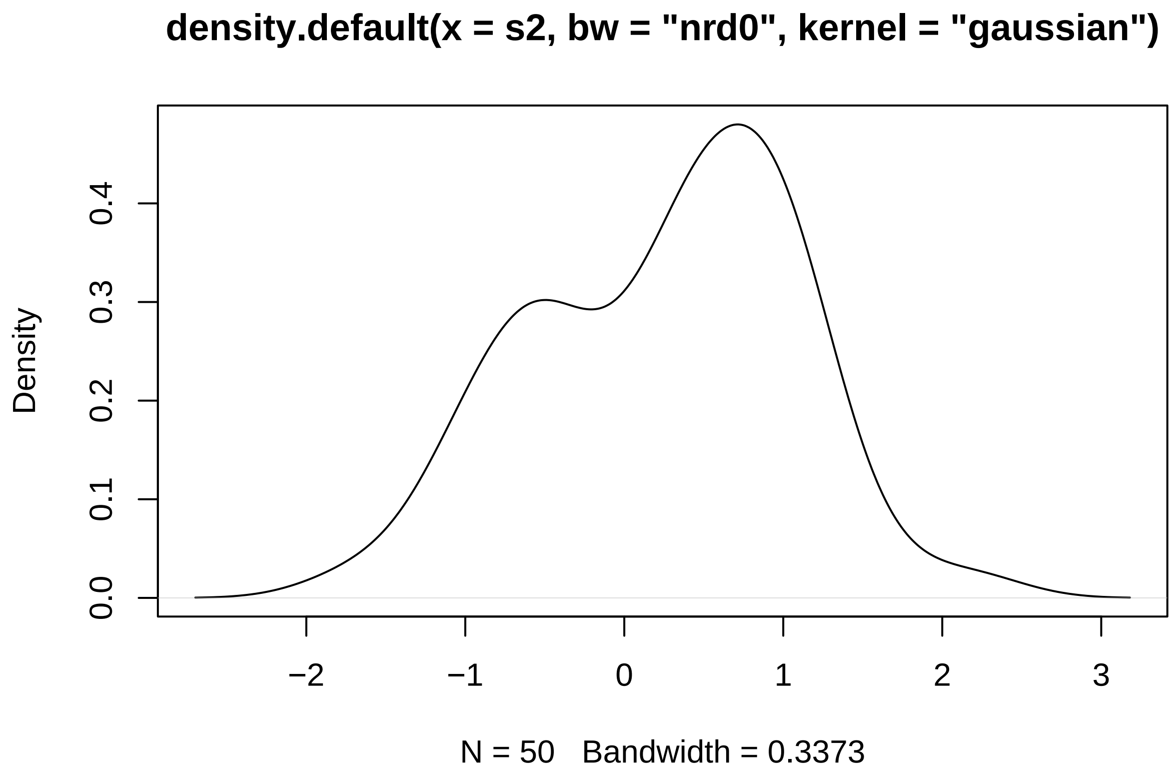



```
kde1
```

```
##
## Call:
##  density.default(x = s1, bw = "nrd0", kernel = "gaussian")
##
## Data: s1 (25 obs.);  Bandwidth 'bw' = 0.4029
##
##        x                 y
##  Min.   :-3.1755   Min.   :0.0004461
##  1st Qu.:-1.6506   1st Qu.:0.0237991
##  Median :-0.1258   Median :0.1020541
##  Mean   :-0.1258   Mean   :0.1637743
##  3rd Qu.: 1.3991   3rd Qu.:0.3004078
##  Max.   : 2.9239   Max.   :0.4613882
```

Second, for `n=50`, we have:

```
plot(kde2)
```

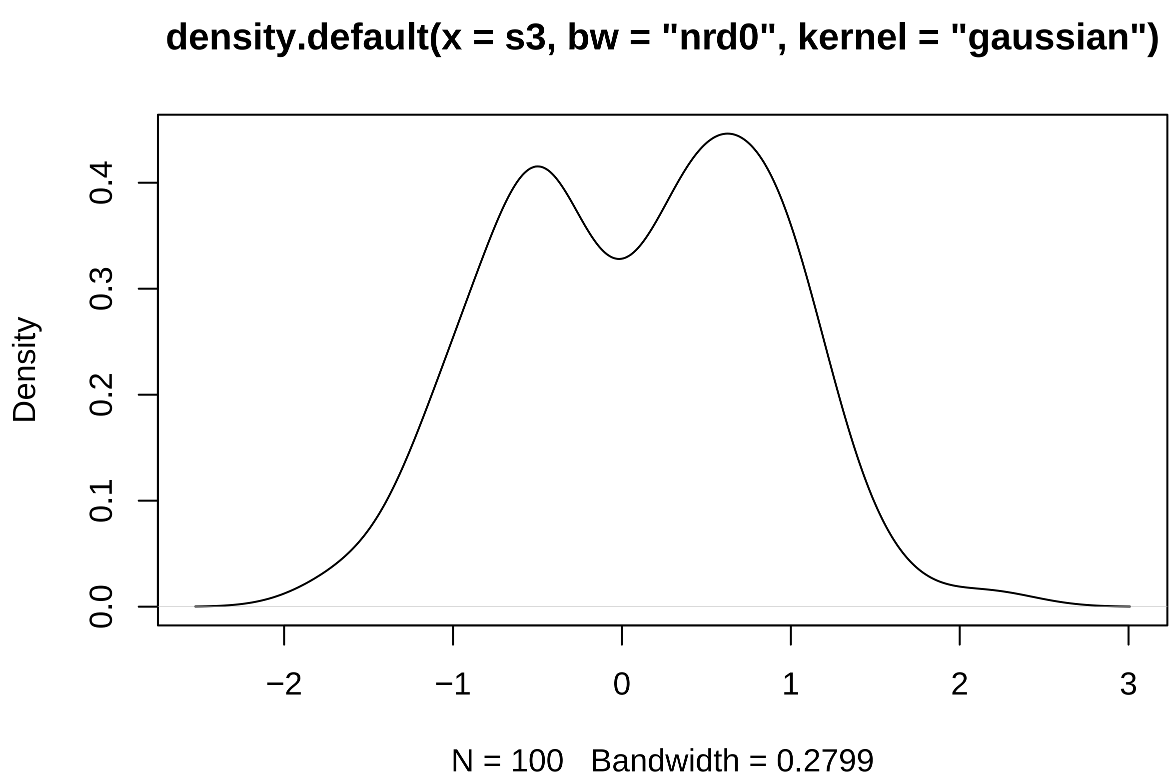



```
kde2
```

```
##
## Call:
##  density.default(x = s2, bw = "nrd0", kernel = "gaussian")
##
## Data: s2 (50 obs.);  Bandwidth 'bw' = 0.3373
##
##        x                 y
##  Min.   :-2.6985   Min.   :0.0002666
##  1st Qu.:-1.2287   1st Qu.:0.0185649
##  Median : 0.2411   Median :0.1081126
##  Mean   : 0.2411   Mean   :0.1699148
##  3rd Qu.: 1.7109   3rd Qu.:0.3000611
##  Max.   : 3.1808   Max.   :0.4800403
```

Third, for `n=100`, we have:

```
plot(kde3)
```

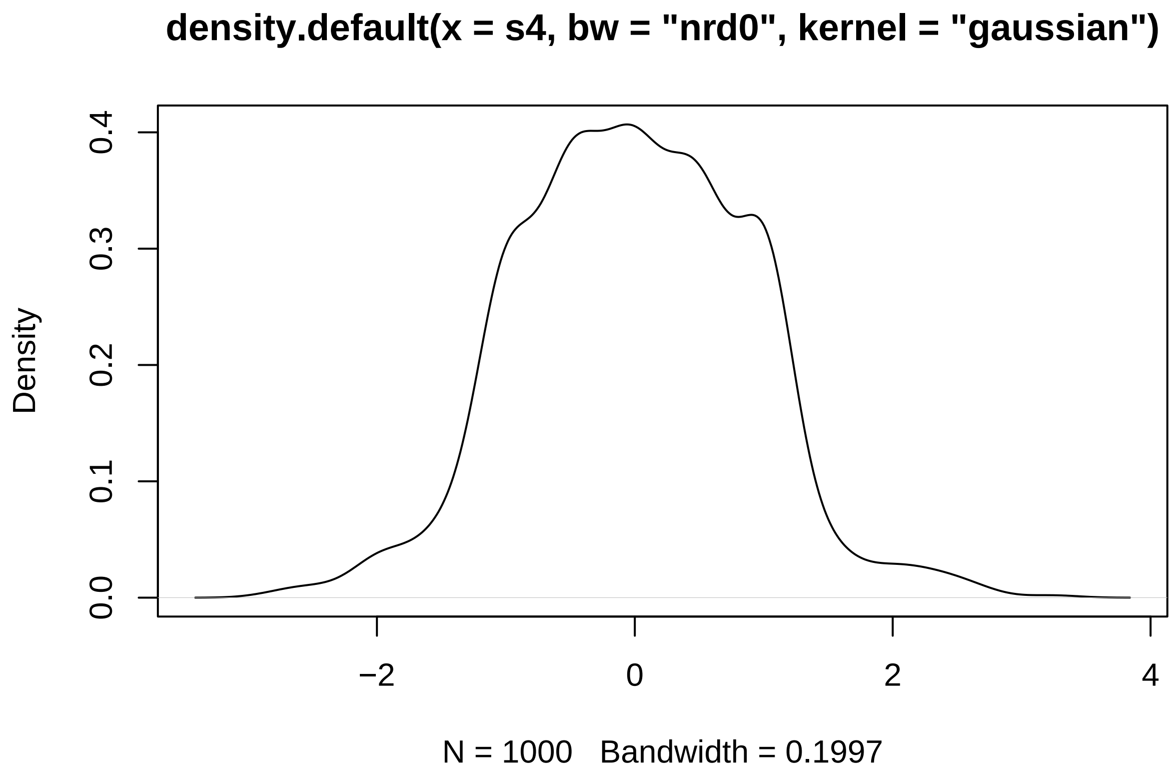



```
kde3
```

```
##
## Call:
##  density.default(x = s3, bw = "nrd0", kernel = "gaussian")
##
## Data: s3 (100 obs.); Bandwidth 'bw' = 0.2799
##
##        x                 y
##  Min.   :-2.5263   Min.   :0.0001605
##  1st Qu.:-1.1426   1st Qu.:0.0155587
##  Median : 0.2411   Median :0.1200554
##  Mean   : 0.2411   Mean   :0.1804883
##  3rd Qu.: 1.6249   3rd Qu.:0.3556999
##  Max.   : 3.0086   Max.   :0.4463318
```

Fourth, for `n=1000`, we have:

```
plot(kde4)
```

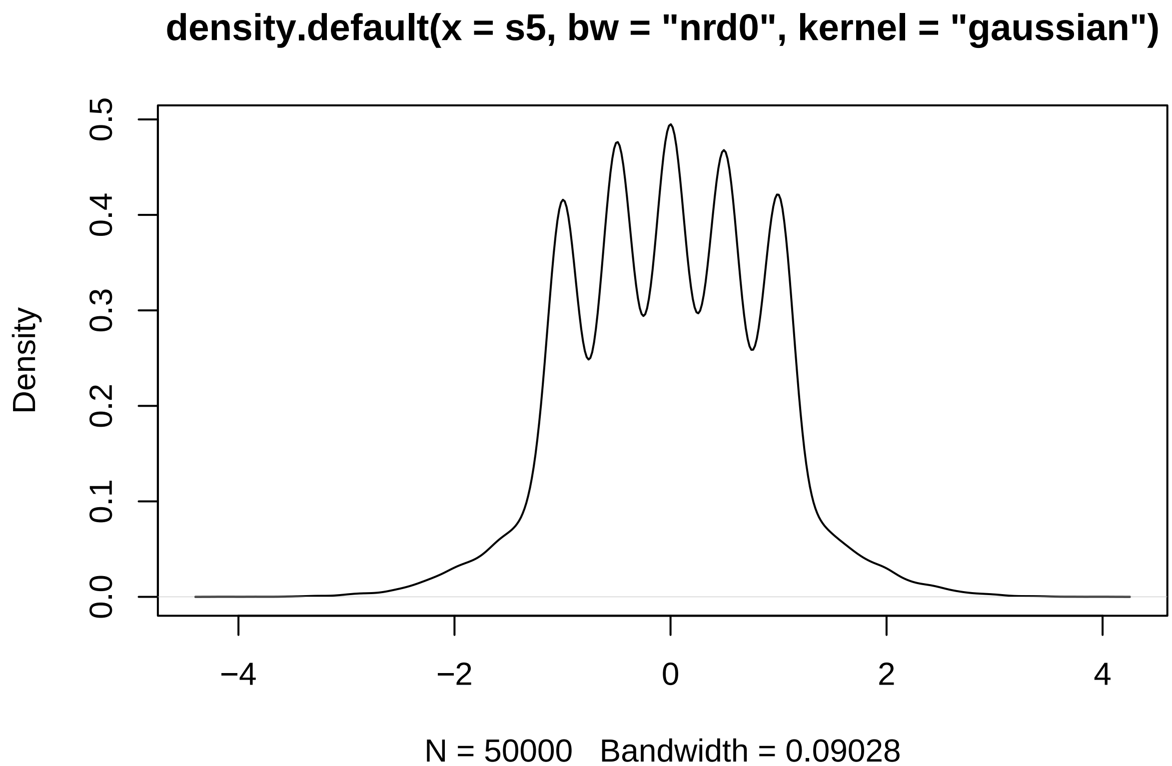



```
kde4
```

```
##
## Call:
##  density.default(x = s4, bw = "nrd0", kernel = "gaussian")
##
## Data: s4 (1000 obs.);    Bandwidth 'bw' = 0.1997
##
##        x                 y
##  Min.   :-3.4090   Min.   :0.0000226
##  1st Qu.:-1.5967   1st Qu.:0.0084124
##  Median : 0.2156   Median :0.0403480
##  Mean   : 0.2156   Mean   :0.1378110
##  3rd Qu.: 2.0279   3rd Qu.:0.3245994
##  Max.   : 3.8402   Max.   :0.4067794
```

In general, we see that thethe KDE method (under `bw = nrd0`) tends towards something much smoother, from something much more bell-shaped and fat-tailed. Notably, as `N` increases, we see that density (in geologic terms) begins to take a butte shape (i.e. the smoothed claw) and remains unimodal, suggesting the KDE (with `bw=nrd0`) is not quite ready to divvy up the five spikes. However, when we run with `N=50000`, then that sample size is sufficient for the KDE to reflect the the true multimodality:

```
s5 = generate_data(50000); kde5 = density(s5, bw="nrd0", kernel="gaussian")
plot(kde5)
```




```
kde5
```

```
##
## Call:
##  density.default(x = s5, bw = "nrd0", kernel = "gaussian")
##
## Data: s5 (50000 obs.);   Bandwidth 'bw' = 0.09028
##
##        x                  y
##  Min.   :-4.39997   Min.   :0.0000011
##  1st Qu.:-2.23658   1st Qu.:0.0009637
##  Median :-0.07318   Median :0.0203796
##  Mean   :-0.07318   Mean   :0.1154461
##  3rd Qu.: 2.09022   3rd Qu.:0.2625725
##  Max.   : 4.25362   Max.   :0.4949208
```

In other words, it appears that the KDE under these settings requires a much higher threshold before going multimodal, in contrast to the regression method.