HW4 Addendum

Isaac Kleisle-Murphy

5/23/2022

d.)

(4)

First, we instantiate our `relu_kernel_regressor()` function, which performs the regression via the `relu` kernel.

```
vstack <- function(x){
  # Stacks a vector vertically, with the vector^T
  # repeated over rows
  lapply(1:length(x), function(i){
    matrix(x, nrow=1)
  }) %>%
    do.call("rbind", .)
}

hstack <- function(x){
  # Stacks a vector horizontally, with the vector
  # repeated over columns
  lapply(1:length(x), function(i){
    matrix(x, ncol=1)
  }) %>%
    do.call("cbind", .)
}

relu_kernel_regressor <- function(x, y, lambda=1.0){
  ### create order stats
  xsorted = sort(x)
  ysorted = matrix(y[order(x)], ncol=1)
  N = length(x)

  ### Make Gram matrix
  Phi = cbind(
    1.,
    (hstack(xsorted) - vstack(xsorted))[, 1:(N-1)]
  )
  ### relu
  Phi[Phi < 0] = 0.

  ### make mask
  M = Phi
```

```
  ### zero intercepts
  M[, 1] = 0.
  ### insert 1's
  M[M > 0] = 1.

  ### solve
  theta = (
    solve(
      t(Phi) %*% Phi
      + lambda * t(M) %*% M
    ) %*% (
      t(Phi) %*% ysorted
    )
  )
  ### insample fit
  yhat = Phi %*% theta

  return(
    list(
      theta=theta,
      yhat=yhat
    )
  )
}
```

To convince you that it works, consider the following regression, where data is generated via

$$y_i = sin(2\pi x_i) + \epsilon_i$$

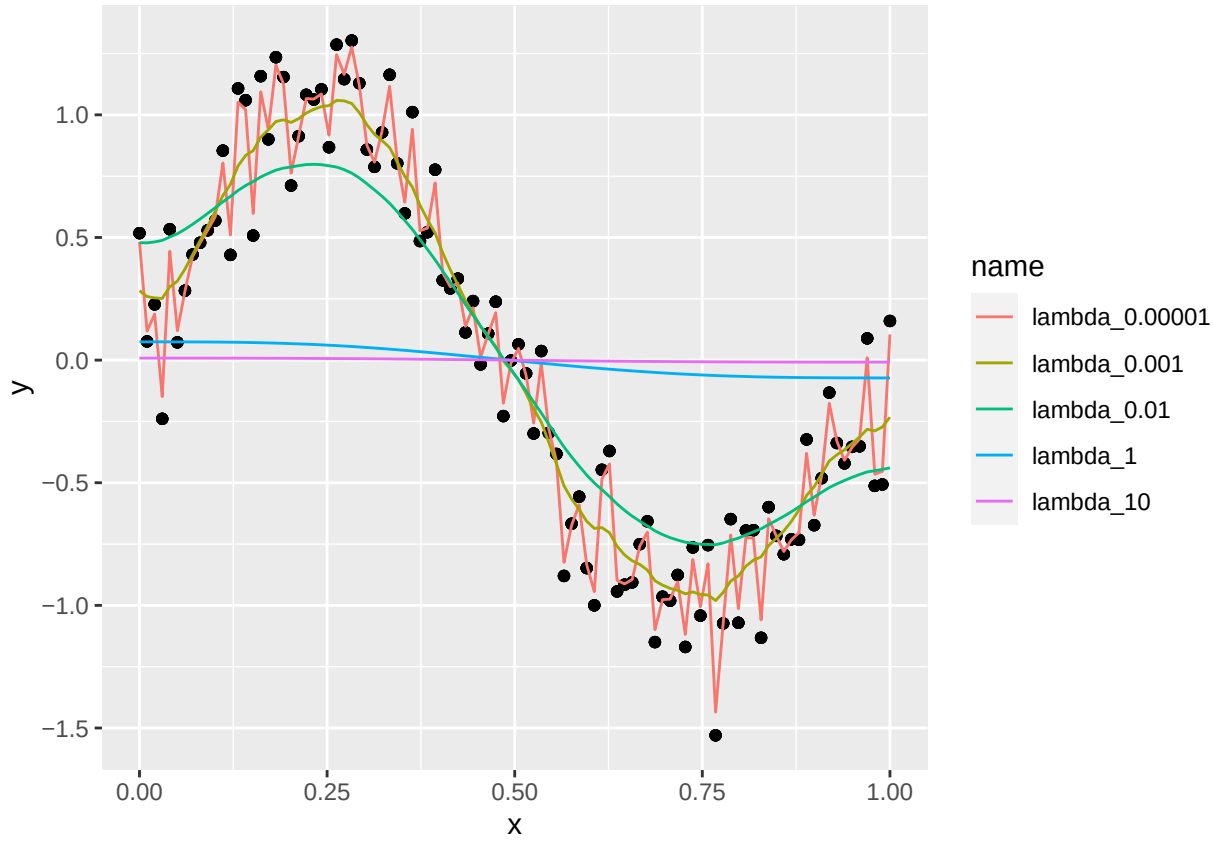
where

$$\epsilon_i \overset{iid}{\sim} N(0, 1)$$

and the x_i are $n = 100$ evenly spaced points in $[0, 1]$. We have, for $\lambda = 1e-5, 1e-3, 1e-2, 1, 10$:

```
set.seed(205)
## make data
x = seq(0, 1, length.out=100)
y = rnorm(length(x), sin(x * (2 * pi)), .25)[order(x)]
x = sort(x)


fit_l0.00001 = relu_kernel_regressor(x, y, 0.00001)
fit_l0.001 = relu_kernel_regressor(x, y, 0.001)
fit_l0.01 = relu_kernel_regressor(x, y, 0.01)
fit_l1 = relu_kernel_regressor(x, y, 1.0)
fit_l10 = relu_kernel_regressor(x, y, 10.0)

plot_df = data.frame(
  x=x,
  y=y,
  lambda_0.00001=fit_l0.00001$yhat,
  lambda_0.001=fit_l0.001$yhat,
```

```
  lambda_0.01=fit_l0.01$yhat,
  lambda_1=fit_l1$yhat,
  lambda_10=fit_l10$yhat
) %>%
  tidyr::pivot_longer(
    .,
    cols=paste0(
      "lambda_",
      c("0.00001", "0.001", "0.01", "1", "10")
    )
)

ggplot(plot_df, aes(x=x, y=y))+
  geom_point() +
  geom_path(
    aes(x=x, y=value, color=name)
  )
```



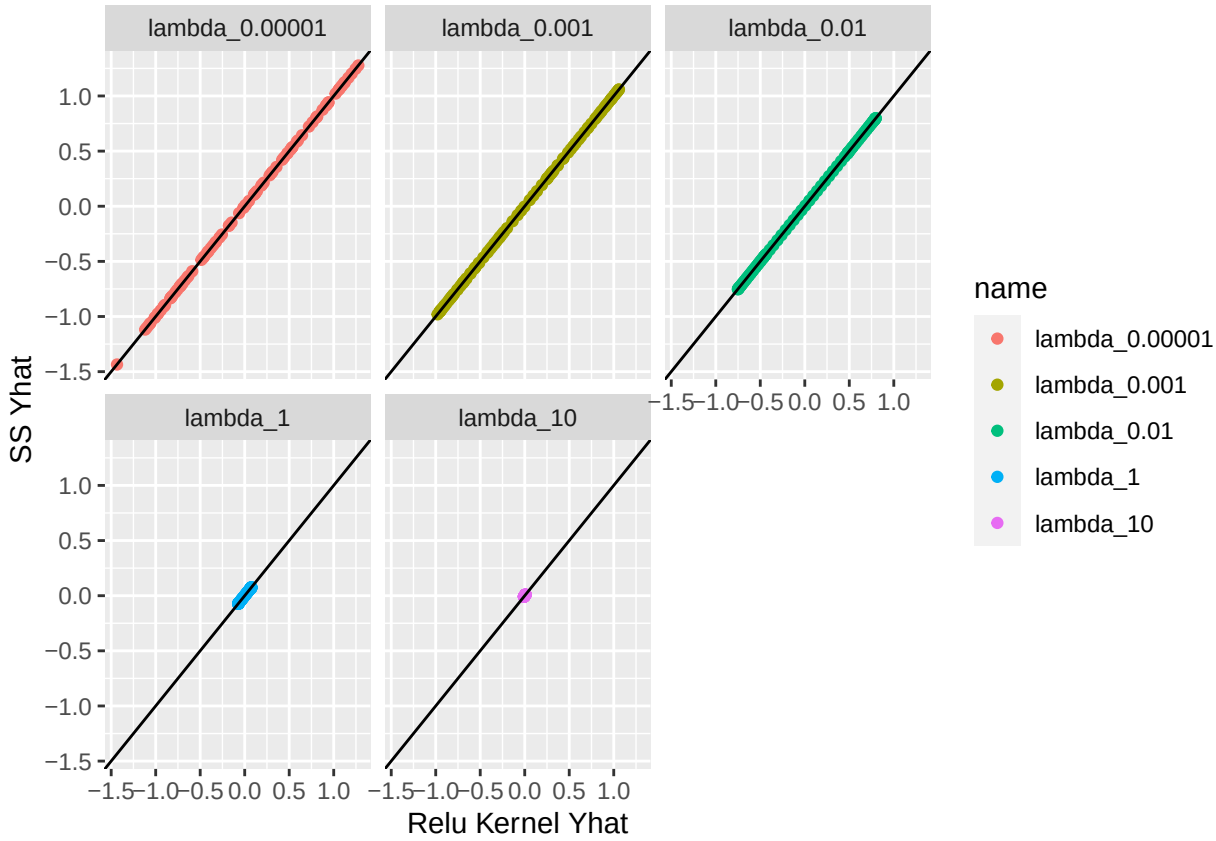
As expected the $\lambda = 1e-5$ induces a near linear interpolation, while the $\lambda = 1, 10$ practically zero out the coefficients and "flatten" the regression. Intuitively, the $||Y - \Phi\theta||_2^2$ wants to linearly interpolate as much as possible and the $||M\theta||_2^2$ term wants to flatten as much as possible. Hence, when λ is tiny, linear interpolation dominates, while when λ is larger, flattening dominates – both of which we see above.

(5)

When `m=1`, the `ss()` function just amounts to a linear interpolation, which as set forth in 1A is the same as our regressor. Hence, when λ is set the same, `ss(m=1)` becomes a linear interpolation (evenly-spaced feature assumption holds, by construction of x) under some level of penalization – which is exactly the penalized relu kernel regressor set forth above. *So the two should be the same.*

As evidence in support of this, consider comparison of in-sample $\hat{y}$ predictions under both of the methods.

```
ss_l0.00001 = npreg::ss(x, y, m=1, all.knots=T, lambda=0.00001)
ss_l0.001 = npreg::ss(x, y, m=1, all.knots=T, lambda=0.001)
ss_l0.01 = npreg::ss(x, y, m=1, all.knots=T, lambda=0.01)
ss_l1 = npreg::ss(x, y, m=1, all.knots=T, lambda=1.0)
ss_l10 = npreg::ss(x, y, m=1, all.knots=T, lambda=10.0)

spline_df = data.frame(
  x=x,
  y=y,
  lambda_0.00001=ss_l0.00001$y,
  lambda_0.001=ss_l0.001$y,
  lambda_0.01=ss_l0.01$y,
  lambda_1=ss_l1$y,
  lambda_10=ss_l10$y
) %>%
  tidyr::pivot_longer(
    .,
    cols=paste0(
      "lambda_",
      c("0.00001", "0.001", "0.01", "1", "10")
    )
) %>%
  inner_join(
    .,
    plot_df, by=c("x", "y", "name"),
    suffix=c("_relu", "_ss")
)

ggplot(spline_df, aes(x=value_relu, y=value_ss, color=name)) +
  geom_point() +
  facet_wrap(~name) +
  labs(x="Relu Kernel Yhat", y="SS Yhat") +
  geom_abline(slope = 1)
```



Indeed, we see equality between the $\hat{y}$'s under the two methods, as expected

2.)

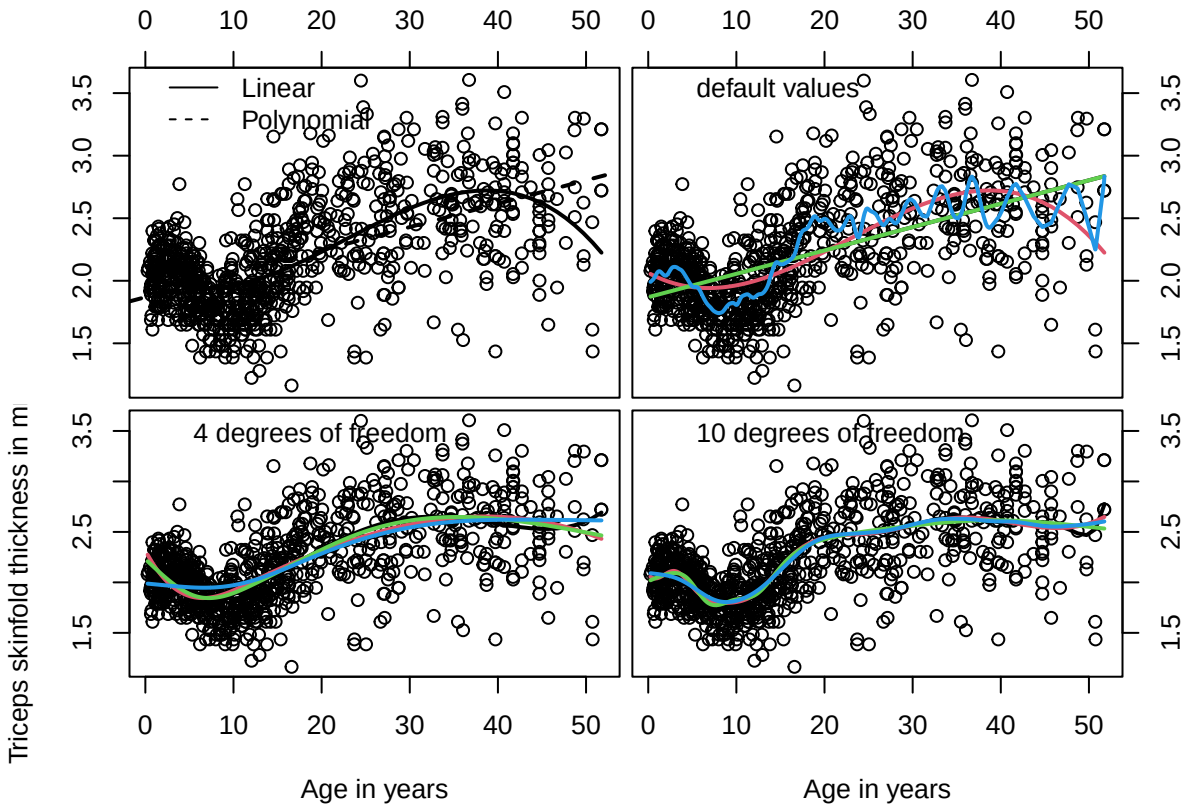
a.)

First, we reproduce the plots, as provided in the paper Appendix.

```
triceps = read.csv("~/Downloads/triceps/triceps.csv")
x = triceps$age;
y = triceps$lntriceps[order(x)]
x = sort(x)

## function to define spline basis
pbase <- function(x, p) {
  u <- (x - min(x)) / (max(x) - min(x))
  u <- 2 * (u - 0.5)
  P <- outer(u, seq(0, p, by = 1), "^")
  P
}

### setup
vdist <- hdist <- 0.2
layout( matrix(1:4, 2, 2, byrow=TRUE), widths=c(10,10), heights=c(10, 10))
par(mar= c(vdist, 4, 3, hdist))
```

```
### linear + labels
plot(x, y, ylab="", xlab="Age in years", axes=FALSE)
axis(2); axis(3); box()
abline(lm(y~x), lwd=2, lty=2)
U <- pbase(x,3)
lines(x, U %*% coef(lm(y~U-1)), lwd=2)
legend(0.05, 3.8, c("Linear", "Polynomial"), col=1, lty=1:2, bty="n")
par(mar= c(vdist, hdist, 3, 4))
plot(x, y, ylab="Triceps skinfold thickness in mm (log scale)", xlab="Age in years", axes=FALSE)
axis(3); axis(4); box()

## fit models
fit.poly <- lm(y ~ poly(x))
fit.bs <- lm(y ~ bs(x) )
fit.ns <- lm(y ~ ns(x) )
fit.sp <- smooth.spline(y ~ x)

## add fit lines to the plot
lines(x, predict(fit.poly, data.frame(x=x)), col=1, lwd=2)
lines(x, predict(fit.bs, data.frame(x=x)), col=2, lwd=2)
lines(x, predict(fit.ns, data.frame(x=x)), col=3, lwd=2)
lines(fit.sp, col=4, lwd=2)
legend(0.05, 3.8, "default values", col=1, bty="n")
par(mar= c(5, 4, vdist, hdist))
plot(x, y, ylab="Triceps skinfold thickness in mm (log)", xlab="Age in years", axes=FALSE)
axis(1); axis(2); box()

## fit models
fit.poly.4 <- lm(y~ poly(x,4))
fit.bs.4 <- lm(y~ bs(x, df=4) )
fit.ns.4 <- lm(y~ ns(x, df=4) )
fit.sp <- smooth.spline(y~ x, df=4)

## add fit lines to the plot
lines(x, predict(fit.poly.4, data.frame(x=x)), col=1, lwd=2)
lines(x, predict(fit.bs.4, data.frame(x=x)), col=2, lwd=2)
lines(x, predict(fit.ns.4, data.frame(x=x)), col=3, lwd=2)
lines(fit.sp, col=4, lwd=2)
legend(0.05, 3.8, "4 degrees of freedom", col=1, bty="n")
par(mar= c(5, hdist, vdist, 4))
plot(x, y, ylab="Triceps skinfold thickness in mm (log)", xlab="Age in years", axes=FALSE)
axis(1); axis(4); box()

## fit models
fit.poly.10 <- lm(y~ poly(x,10))
fit.bs.10 <- lm(y~ bs(x, df=10) )
fit.ns.10 <- lm(y~ ns(x, df=10) )
fit.sp <- smooth.spline(y ~ x, df=10)

## add fit lines to the plot
lines(x, predict(fit.poly.10, data.frame(x=x)), col=1, lwd=2)
lines(x, predict(fit.bs.10, data.frame(x=x)), col=2, lwd=2)
lines(x, predict(fit.ns.10, data.frame(x=x)), col=3, lwd=2)
```

```
lines(fit.sp, col=4,lwd=2)
legend(0.05, 3.8, "10 degrees of freedom", col=1, bty="n")
```




```
layout(matrix(1, 1, 1))
```

b.)

Then, we make functions to tune the various basis regressions. These are configured below:

```
loo.cv.df <- function(x, y, basis.fn, df, fit.fn=lm){
  #' Note that x and y should be sorted so that x is monotonically increasing
  N = length(x)
  # LOO except for endpoints, so we don't have to predict beyond boundary knots
  sapply(2:(N-1), function(i){
    fit.df = data.frame(
      x = x[(1:N) != i],
      y = y[(1:N) != i]
    )
    i
    ### Case 1: bs, ns, ss, poly
    if(basis.fn %in% c("bs", "ns", "poly")){
      args = ifelse(basis.fn == "poly", "degree=", "df=")
      formula.str = paste0(
```

```
        "y ~ ",
        basis.fn,
        "(x, ",
        args,
        df,
        ")"
      )
      fit = fit.fn(
        as.formula(formula.str),
        fit.df
      )

      yhat = (predict(fit, data.frame(x=x[i])))
    }else{
      fit = smooth.spline(fit.df$x, fit.df$y, df=df)
      yhat = predict(fit, data.frame(x=x[i]))$y
      yhat = as.numeric(yhat)
    }

    (y[i] - yhat)^2
  }) %>%
    mean() -> mse.loo
  mse.loo
}

tune.smoother <- function(x, y, basis.fn, df.grid, fit.fn=lm){
  # tune
  sapply(df.grid, function(df){
    loo.cv.df(x, y, basis.fn, df)
  }) -> cv.result
  # print winner
  best.idx = which.min(cv.result)
  df.best = df.grid[best.idx]
  cat("--------------------------------------\n",
      "Tuning results\n  Basis: ", basis.fn,
      "\n  Best DF: ", df.best,
      "\n  LOO MSE: ", cv.result[best.idx], "\n",
      "--------------------------------------\n")

  fit.df = data.frame(x=x, y=y)

  ### Case 1: bs, ns, ss, poly
  if(basis.fn %in% c("bs", "ns", "poly")){
    args = ifelse(basis.fn == "poly", "degree=", "df=")
    formula.str = paste0(
      "y ~ ",
      basis.fn,
      "(x, ",
      args,
      df.best,
      ")"
    )
    fit = fit.fn(
```

```
      as.formula(formula.str),
      fit.df
    )
  }else{
    fit = smooth.spline(fit.df$x, fit.df$y, df=df.best)
  }
  return(fit)
}
```

We then tune each of the functions LOO-style, and plot the results (going back to `ggplot2`, for familiarity, at this point). Hyperparam configurations are printed as part of the script.

```
df_seq = c(4, 6, 8, 10, 12, 20)
fit.poly.tune = tune.smoother(x, y, "poly", df_seq)
```

```
## ------------------------------------
##  Tuning results
##   Basis:  poly
##   Best DF:  10
##   LOO MSE:  0.09977551
##  ------------------------------------
```

```
fit.bs.tune = tune.smoother(x, y, "bs", df_seq)
```

```
## ------------------------------------
##  Tuning results
##   Basis:  bs
##   Best DF:  10
##   LOO MSE:  0.09925858
##  ------------------------------------
```

```
fit.ns.tune = tune.smoother(x, y, "ns", df_seq)
```

```
## ------------------------------------
##  Tuning results
##   Basis:  ns
##   Best DF:  8
##   LOO MSE:  0.0989379
##  ------------------------------------
```

```
fit.sp.tune = tune.smoother(x, y, "smooth.spline", df_seq)
```

```
## ------------------------------------
##  Tuning results
##   Basis:  smooth.spline
##   Best DF:  12
##   LOO MSE:  0.09975969
##  ------------------------------------
```

```
## add fit lines to the plot
plot(x, y, ylab="", xlab="Age in years", axes=T)
lines(x, predict(fit.poly.tune, data.frame(x=x)), lty=1)
lines(x, predict(fit.bs.tune, data.frame(x=x)), col="red", lty=1)
lines(x, predict(fit.ns.tune, data.frame(x=x)), col="green", lty=1)
lines(x, predict(fit.sp.tune, x)$y, col="blue", lty=1)
```