SDS353 - Homework 4

Scott Jackson & Weston Engelstad

8.8.2

(a)

$$\begin{aligned}\widehat{\beta}_{RR} &= \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i \cdot \beta)^2 + \lambda\sum_{j=1}^{n}\beta_j^2 \\ &= n^{-1}(y - x\beta)^T(y - x\beta) + \lambda\beta^T\beta\end{aligned}$$

$$(*)\ \ u^T u = \sum_i u_i^2$$

(b)

$$\frac{\partial}{\partial\beta}(\widehat{\beta}_{RR}) = 2(x^T x)\beta - 2x^T y + 2n\lambda\beta$$

Setting this equal to zero and solving for β:

$$\begin{aligned}&2(x^T x)\beta - 2x^T y + 2n\lambda\beta = 0 \\ &(x^T x)\beta + n\lambda\beta = x^T y \\ &\beta = (x^T x + n\lambda\mathbf{I})^{-1}x^T y\end{aligned}$$

(c)

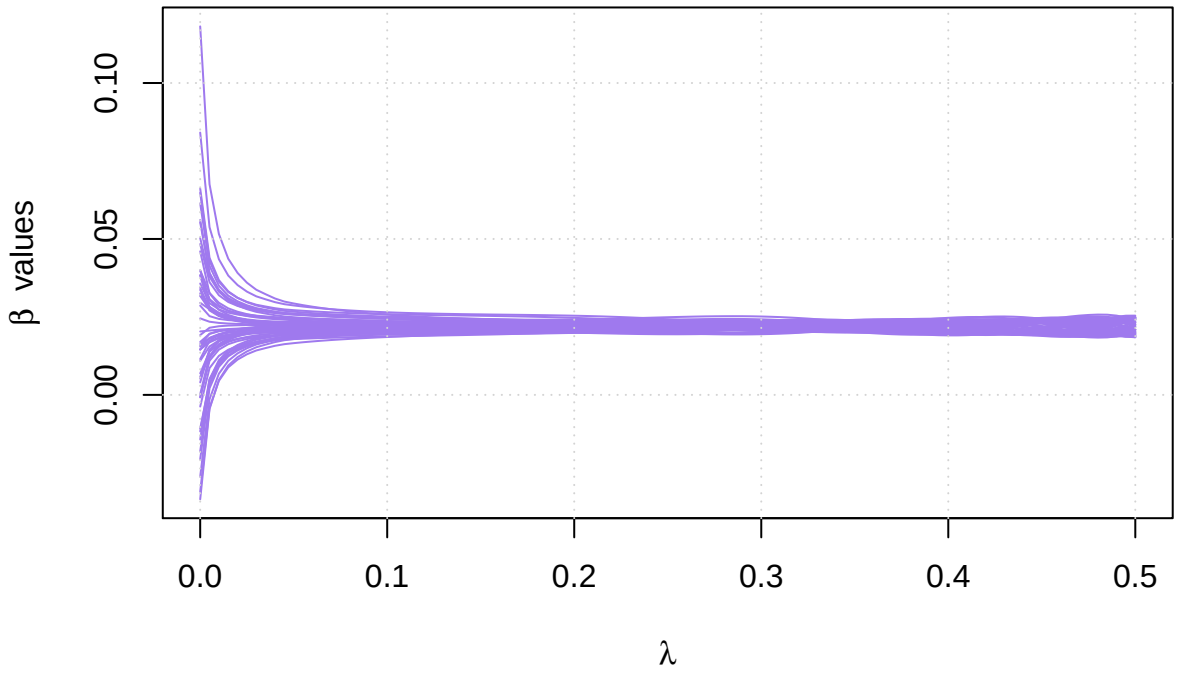
λ is a tuning parameter which controls how much the second term contributes to the cost function. As $\lambda \to 0$ then ridge regression just becomes the linear regression estimate. As $\lambda \to \infty$, the cost function will approach ∞, and so $||\beta||$ must approach 0.

(d)

The plot below demonstrates how the coefficients shrink as λ gets big. This also explains why ridge regression is called a ***shrinkage estimator***, as it is fitting weights under the constraint imposed by the regularization term, the contribution of which is controlled by the hyper-parameter λ.

```
Z <- runif(2000, -1, 1)
epsilon <-  rnorm(2000, 0, 0.05)
Y <- Z + epsilon

X <- matrix(nrow=50, ncol=2000)
for (i in 1:50) {
  X[i,] <- 0.9*Z + rnorm(2000, 0, 0.05)
}

X.train <- X[,1:1000]
```

```
X.test <- X[,1001:2000]
y.train <- Y[1:1000]
y.test <- Y[1001:2000]

grid <- seq(0, .5, by=0.005)
Beta <- matrix(nrow=50, ncol=length(grid))

ridge.mod <- glmnet(X.train %>% t(), y.train, alpha=0, lambda=grid)
plot(1, type="n",
     xlab=expression(lambda), ylab=beta ~ ' values',
     xlim=range(grid), ylim=c(min(coef(ridge.mod)), max(coef(ridge.mod))))
for (i in 2:dim(coef(ridge.mod))[1]) {
  lines(grid, coef(ridge.mod)[i,] %>% rev(), col='mediumpurple2')
}
grid()
```



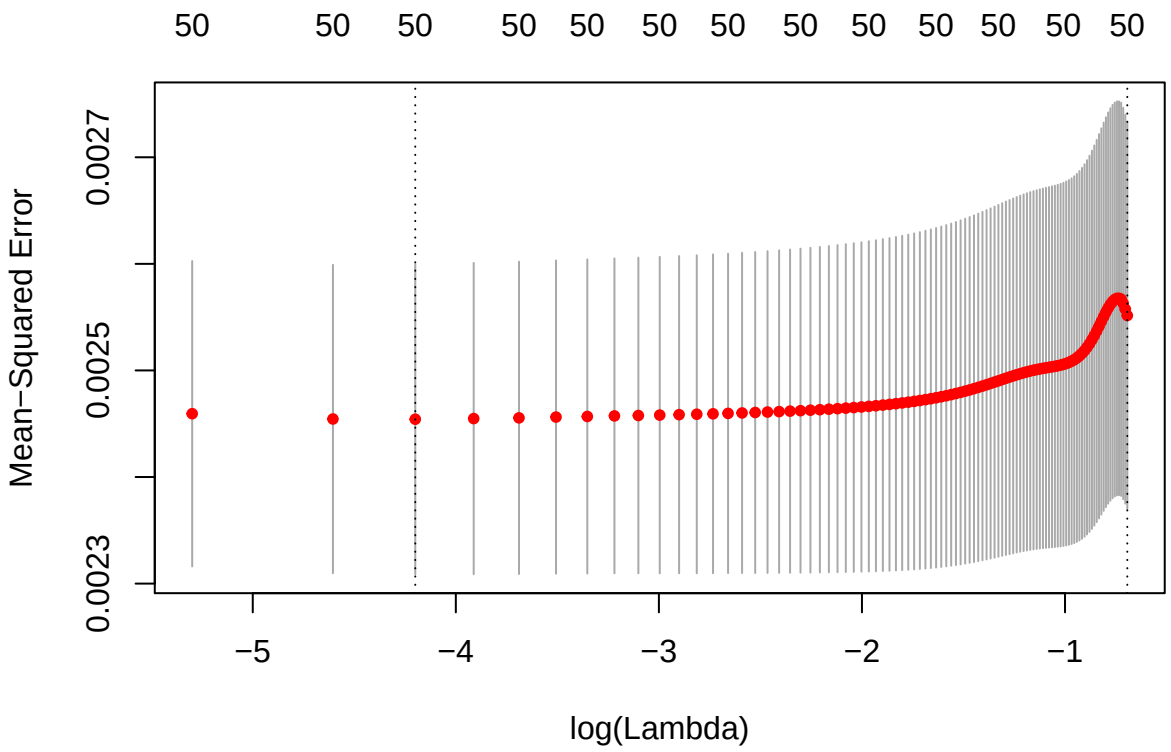
(e)

As we can see, ridge regression preforms much better than just simple linear regression.

```
ridge.mod <- cv.glmnet(X.train %>% t(), y.train, alpha=0, lambda=grid, type.measure='mse')
cat('Optimal Lambda from CV: ', ridge.mod$lambda.min, '\n')
```

```
## Optimal Lambda from CV:  0.015
```

```
lm.mod <- lm(y.train ~ X.train %>% t())
plot(ridge.mod)
```




```
mse.rr <- (y.test - predict(ridge.mod, newx=X.test %>% t(), s='lambda.min'))^2 %>% mean()
mse.lm <- (y.test - predict(lm.mod, newx=X.test %>% t()))^2 %>% mean()

cat('Ridge Regression MSE: ', mse.rr, '\nLinear Regression MSE: ', mse.lm, '\n')
```

```
## Ridge Regression MSE:  0.002570952
## Linear Regression MSE:  0.6674546
```

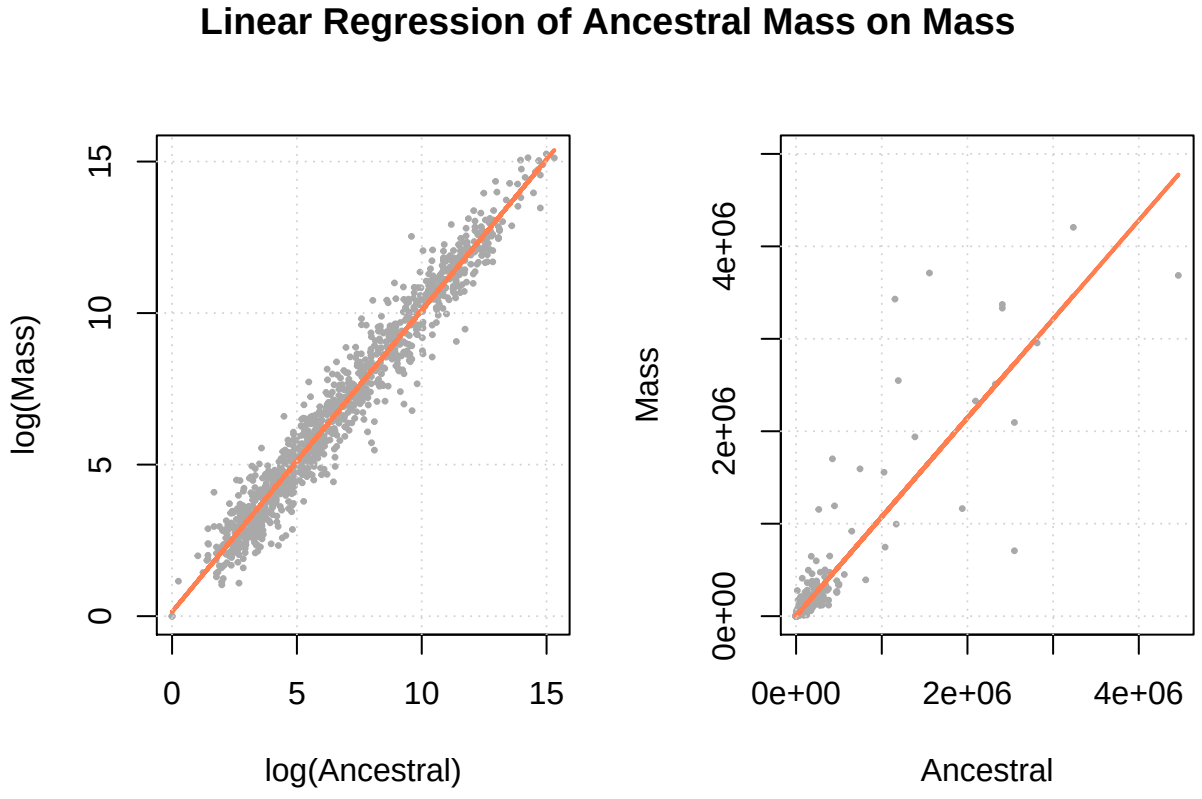
1.

As we can see below, we get a coefficient of 0.141 for β_0, and 0.995 for β_1. This model is fit on a log-log scale, but the results are on a non-log scale. As a result, interpretation becomes slightly more complicated. The intercept denotes that if our slope term is 0, then we will make a prediction of $\exp(\beta_0) = 1.1515931$. The slope tells us that for each unit increase in `ln_old_mass` we see a β_1 increase in `ln_mass`. When we bring this back to the original scale however, each unit increase in `ln_old_mass` results in an exponential increase in `mass`.

```
nampd <- read.csv('data/nampd.csv')
data <- nampd %>%
        dplyr::select(., ln_old_mass, ln_mass) %>%
        na.omit()
data$old_mass <- exp(data$ln_old_mass)
data$mass <- exp(data$ln_mass)

lm.fit <- lm(ln_mass ~ ln_old_mass, data=data)
summary(lm.fit)$coefficients
```

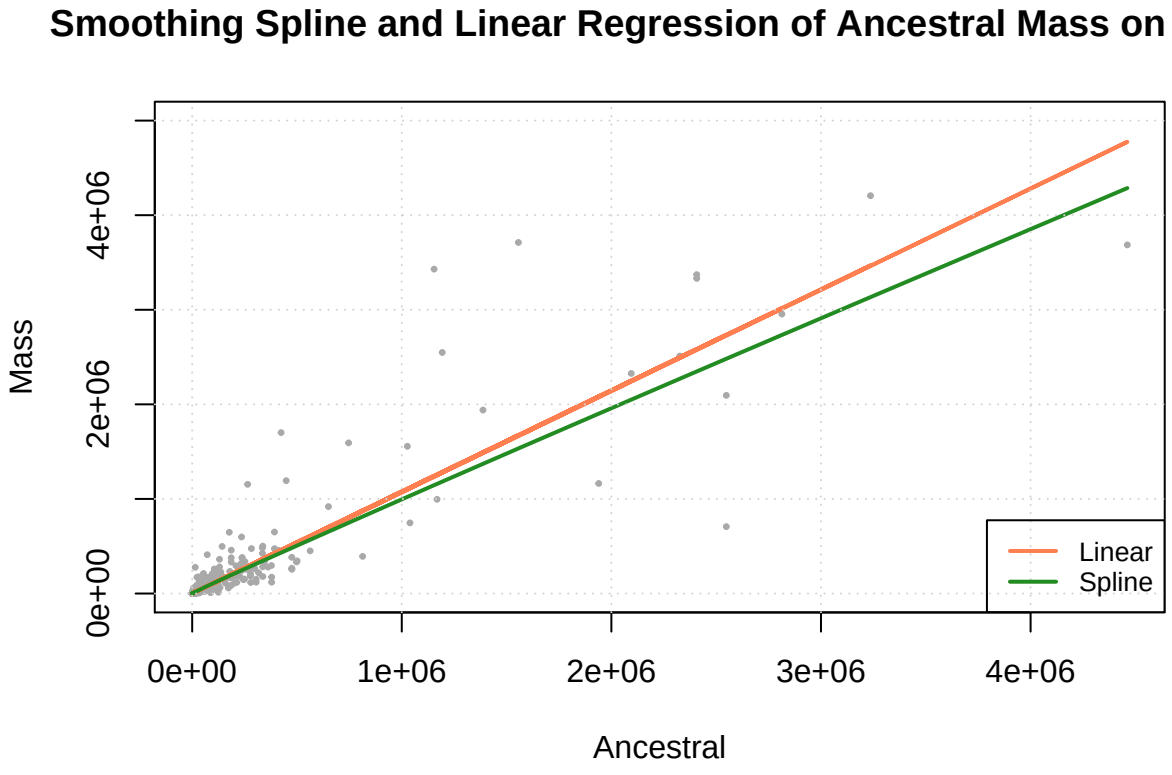
```
##              Estimate  Std. Error    t value    Pr(>|t|)
## (Intercept) 0.1411463 0.049045867   2.877843 0.004080976
## ln_old_mass 0.9952194 0.006510785 152.857054 0.000000000
```

```r
lm.preds <- predict(lm.fit)

par(mfrow=c(1,2), mar=c(4.5,4.5,1,1), oma=c(0,0,4,0))

plot(data$ln_old_mass, data$ln_mass,
     cex=.5,
     col='darkgrey',
     xlab='log(Ancestral)',
     ylab='log(Mass)',
     pch=20)
lines(data$ln_old_mass, lm.preds, lwd=2, col='coral')
grid()

plot(data$old_mass, data$mass,
     cex=.5,
     col='darkgrey',
     xlab='Ancestral',
     ylab='Mass',
     ylim=c(0,5e6),
     pch=20)
lines(data$old_mass, exp(lm.preds), lwd=2, col='coral')
grid()

title('Linear Regression of Ancestral Mass on Mass', outer=T)
```



2.

```
x <- data$ln_old_mass
y <- data$ln_mass

spl.fit <- smooth.spline(x, y, cv=F)

plot(data$old_mass, data$mass,
     cex=.5,
     col='darkgrey',
     xlab='Ancestral',
     ylab='Mass',
     ylim=c(0,5e6),
     pch=20)
lines(data$old_mass, exp(lm.preds), lwd=2, col='coral')
lines(exp(spl.fit$x), exp(spl.fit$y), lwd=2, col='forestgreen')
grid()

title('Smoothing Spline and Linear Regression of Ancestral Mass on Mass')
legend('bottomright',
       legend=c('Linear', 'Spline'),
       col=c('coral', 'forestgreen'),
       lty=1, lwd=2, cex=.8)
```

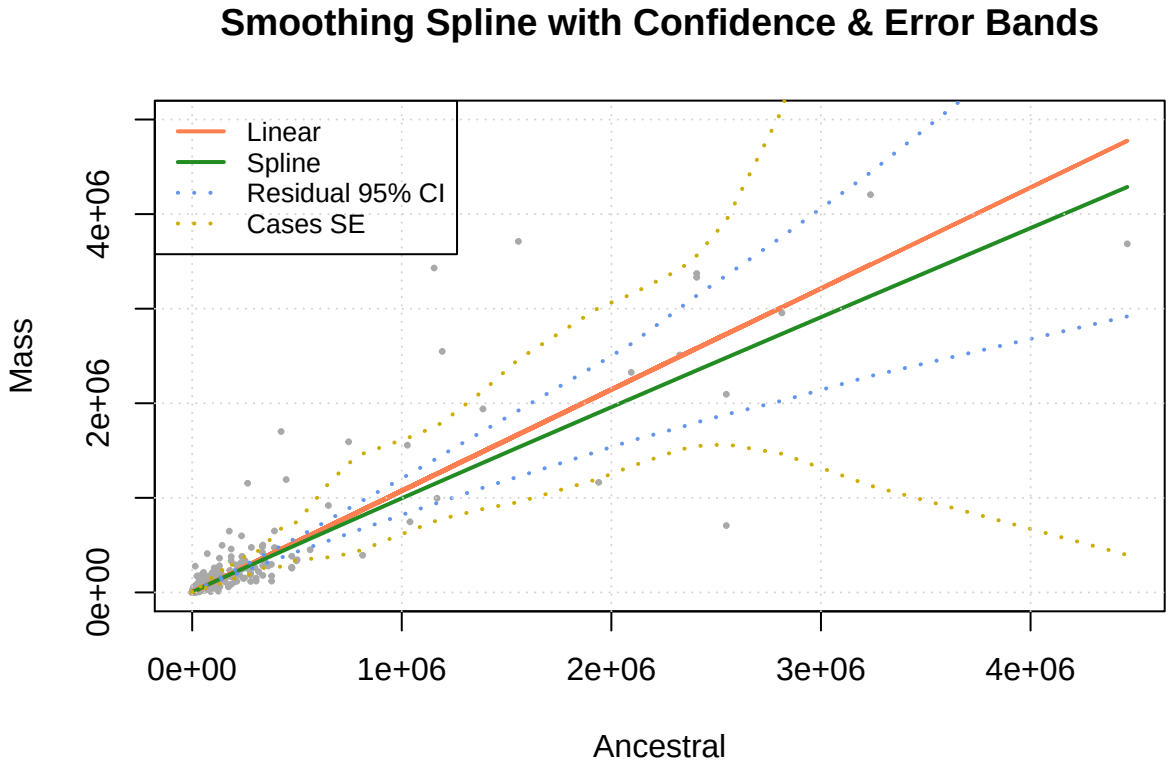


3.

```
resample <- function(x) {
  sample(x, size=length(x), replace=TRUE)
}

boot.ci <- function(B, t.hat, alpha, simulator, statistic) {
  t.boot <- replicate(B, statistic(simulator()))
  ci.lower <- 2*t.hat - apply(t.boot, 1, quantile, probs=1-alpha/2)
  ci.upper <- 2*t.hat - apply(t.boot, 1, quantile, probs=alpha/2)
  cis <- rbind(ci.lower, ci.upper)
  return(cis)
}

boot.sd <- function(B, simulator, statistic) {
  sd.boot <- replicate(B, statistic(simulator()))
  sd.points <- apply(sd.boot, 1, sd)
  return(sd.points)
}

resample.residuals <- function() {
  new.frame <- data
  new.ln_mass <- fitted(spl.fit) + resample(resid(spl.fit))
  new.frame$ln_mass <- new.ln_mass
  return(new.frame)
}

resample.cases <- function() {
  sample.rows <- resample(1:nrow(data))
  return(data[sample.rows,])
}

fit.spl <- function(data) {
  fit <- smooth.spline(data$ln_old_mass, data$ln_mass)
  return(fit)
}

eval.spl <- function(spl) {
  return(predict(spl, x=data$ln_old_mass)$y)
}

spl.statistic <- function(data) {
  return(eval.spl(fit.spl(data)))
}

main.curve <- eval.spl(spl.fit)

spl.resid.ci <- boot.ci(B=1000,
                        t.hat=main.curve,
                        alpha=.05,
                        simulator=resample.residuals,
                        statistic=spl.statistic)
```

```
spl.cases.se <- spl.cases.se <- boot.sd(B=1000,
                        simulator=resample.cases,
                        statistic=spl.statistic)

df <- data.frame('old_mass'=data$old_mass,
                 'ci_lower'=exp(spl.resid.ci[1,]),
                 'ci_upper'=exp(spl.resid.ci[2,]),
                 'se_lower'=exp(main.curve - 2*spl.cases.se),
                 'se_upper'=exp(main.curve + 2*spl.cases.se)) %>%
  .[order(.$old_mass),]
```

```
plot(data$old_mass, data$mass,
     cex=.5,
     col='darkgrey',
     xlab='Ancestral',
     ylab='Mass',
     ylim=c(0,5e6),
     pch=20)
lines(data$old_mass, exp(lm.preds), lwd=2, col='coral')
lines(exp(spl.fit$x), exp(spl.fit$y), lwd=2, col='forestgreen')
matlines(df$old_mass, cbind(df$ci_lower, df$ci_upper), lwd=2, lty=3, col='cornflowerblue')
matlines(df$old_mass, cbind(df$se_lower, df$se_upper), lwd=2, lty=3, col='gold3')
grid()

title('Smoothing Spline with Confidence & Error Bands')
legend('topleft',
       legend=c('Linear', 'Spline', 'Residual 95% CI', 'Cases SE'),
       col=c('coral', 'forestgreen', 'cornflowerblue', 'gold3'),
       lty=c(1,1,3,3), lwd=2, cex=.8)
```



4.

(a)

The code for `rmass` is below. It takes 4 parameters:

- `Xa` - the ancestral mass
- `r` - an estimated spline function
- `sigma.2` - the variance for Z in the model
- `max.retries` - maximum number of times to look for a valid X_D, to avoid infinite looping

```
x.min <- 1.8
x.max <- 1e15

rmass <- function(Xa, r, sigma.2=0.63, max.retries=100000) {
  sigma <- sqrt(sigma.2)
  interp <- predict(r, x=log(Xa))$y
  Xd <- exp(interp + rnorm(1, 0, sigma))

  retries = 0
  while (Xd > x.max || Xd < x.min) {
    Xd <- exp(interp + rnorm(1, 0, sigma))

    retries = retries + 1
    if (retries > max.retries) {
      return(-1)
    }
  }

  return(Xd)
}
```

(b)

The following segment makes sure that the output is always in the range $[x_{min}, x_{max}]$. It is worth noting that for sufficiently large X_A outside the allowed range, along with sufficiently small `sigma.2`, we will not be able to return an X_D.

```
# check both boundaries
for (i in 1:10000) {
  stopifnot(rmass(1.8, spl.fit) >= x.min)
}
print('Lower Bound Test - Done.')
```

```
## [1] "Lower Bound Test - Done."
```

```
for (i in 1:10000) {
  stopifnot(rmass(1e15, spl.fit) <= x.max)
}
print('Upper Bound Test - Done.')
```

```
## [1] "Upper Bound Test - Done."
```

```
# check random numbers in range
for (i in 1:10000) {
  r <- runif(1, x.min, x.max)
```

```
  Xd <- rmass(r, spl.fit)
  stopifnot(Xd >= x.min || Xd <= x.max)
}
print('Random Test - Done.')
```

```
## [1] "Random Test - Done."
```

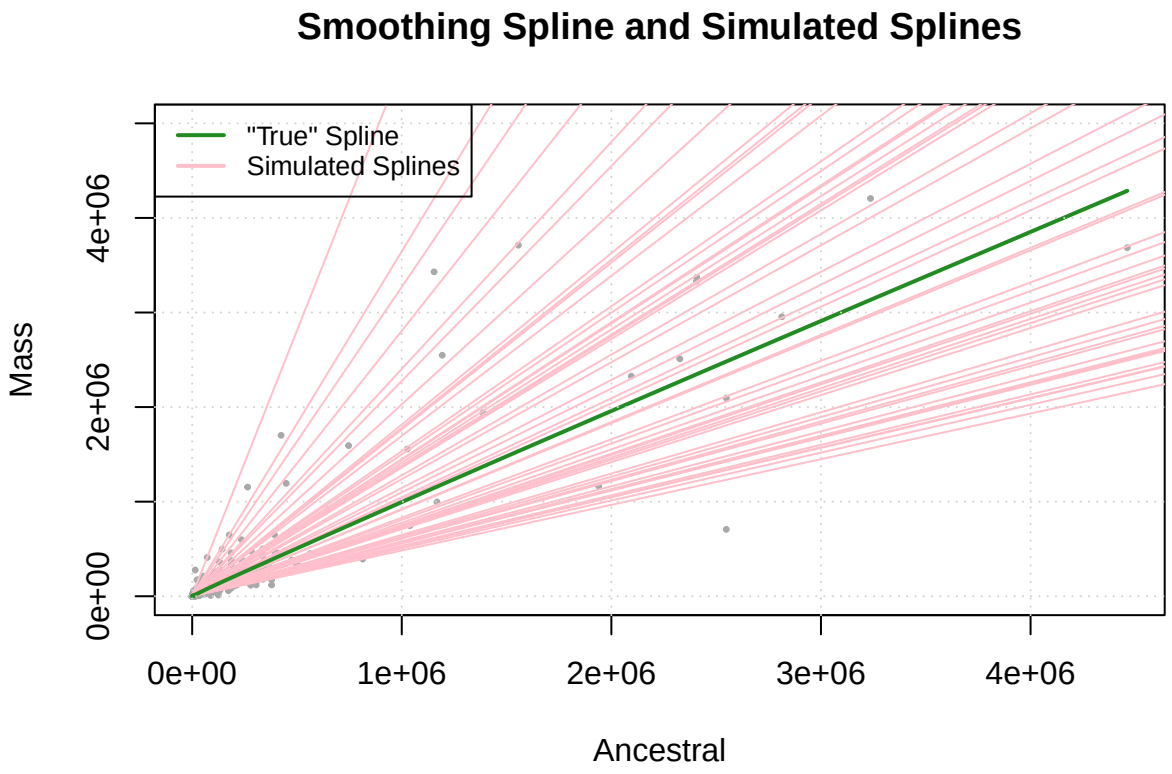
```
# check number out of bounds
stopifnot(rmass(1e18, spl.fit) == -1)
print('Out of Bounds Test - Done.')
```

```
## [1] "Out of Bounds Test - Done."
```

(c)

The below code simulates data using `rmass` and fits smoothing splines at each iteration. We can see that there is quite a bit of variability in the simulated spline fits, due to the Z term in the model.

```
x.grid <- seq(x.min, x.max, length.out=150)

plot(data$old_mass, data$mass,
     cex=.5,
     col='darkgrey',
     xlab='Ancestral',
     ylab='Mass',
     ylim=c(0,5e6),
     pch=20)
for (i in 1:50) {
  new.mass <- rmass(x.grid, spl.fit)
  spl.sim <- smooth.spline(x.grid, new.mass)
  lines(spl.sim$x, spl.sim$y, col='pink', lwd=1)
}
lines(exp(spl.fit$x), exp(spl.fit$y), lwd=2, col='forestgreen')
grid()

title('Smoothing Spline and Simulated Splines')
legend('topleft',
       legend=c('\"True\" Spline', 'Simulated Splines'),
       col=c('forestgreen', 'pink'),
       lwd=2, cex=.8)
```



5.

The code below implements the function `origin`, which simply calls `rmass` twice, replacing one of the values in the vector `Xa`, and appending the second value to the end of the vector, finally returning the modified `Xa`.

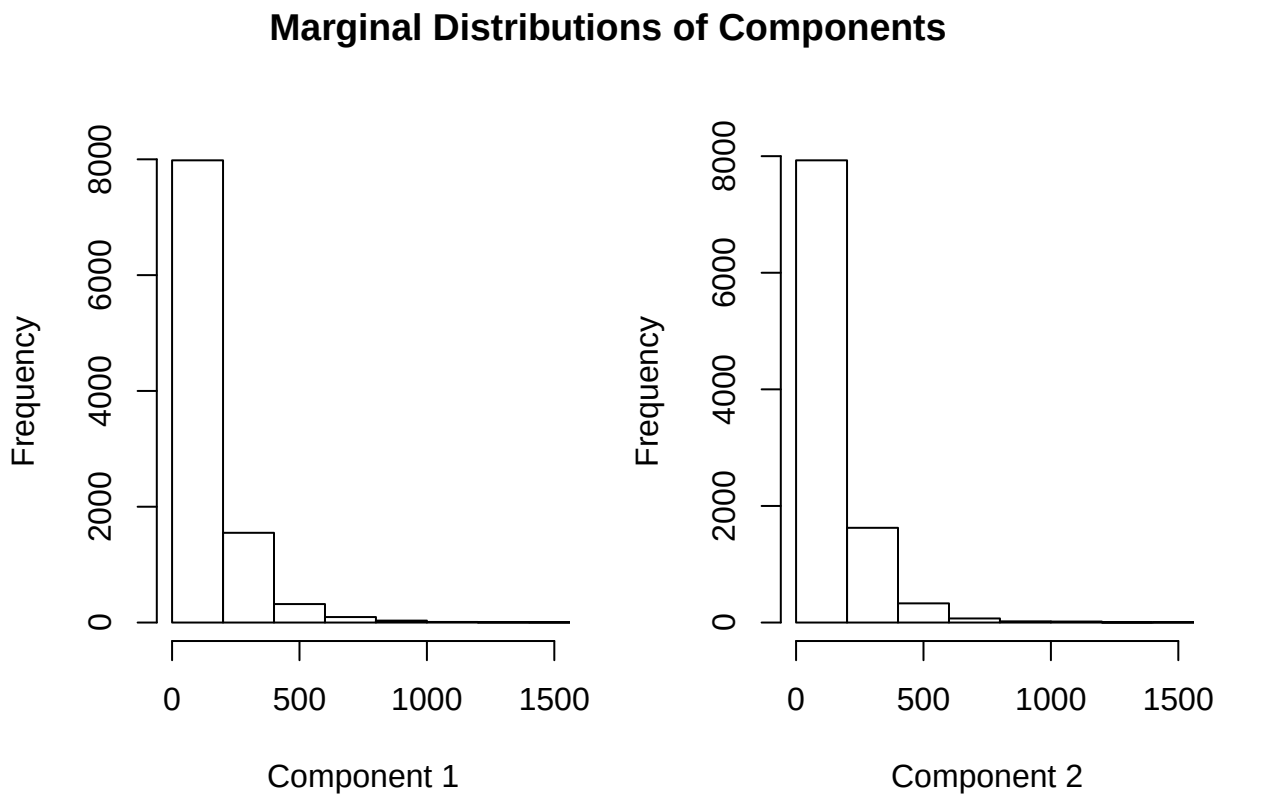
```
origin <- function(Xa, r, sigma.2=0.63, max.retries=100000) {
  idx <- sample(length(Xa), 1)
  Xd1 <- rmass(Xa[idx], r, sigma.2, max.retries)
  Xd2 <- rmass(Xa[idx], r, sigma.2, max.retries)

  Xa[idx] <- Xd1
  Xa <- append(Xa, Xd2)

  return(Xa)
}
```

(a)

We can clearly see that by simulating using a scalar, the marginal distributions of the components are the same, and that they are also uncorrelated with one another.

```
v = 100
M <- matrix(nrow=10000, ncol=2)
for (i in 1:10000) {
  M[i,] <- origin(v, spl.fit)
}

par(mfrow=c(1,2), mar=c(4.5,4.5,1,1), oma=c(0,0,4,0))
```

```r
hist(M[,1], breaks=10, xlim=c(0,1500), main='', xlab='Component 1')
hist(M[,2], breaks=10, xlim=c(0,1500), main='', xlab='Component 2')

title('Marginal Distributions of Components', outer=T)
```




```r
cor(M[,1], M[,2])
```

```
## [1] 0.00504149
```

(b)

The following code checks to make sure that if the input vector to `origin` has length m, then the output vector has length $m+1$ by generating random vectors of lengths in the range $[1, 3000]$, and ensuring for each input length it is indeed 1 greater.

```r
for (i in 1:3000) {
  vec <- rnorm(i, 500, 100)
  stopifnot(length(origin(vec, spl.fit)) == i + 1)
}
print('Done.')
```

```
## [1] "Done."
```

(c)

This code is very similar to that for **(b)**, however at each iteration we simply check that the length of the intersection of the two vectors is indeed $m - 1$.

```
for (i in 1:3000) {
  vec <- rnorm(i, 500, 100)
  stopifnot(intersect(vec, origin(vec, spl.fit)) %>% length() == i - 1)
}
print('Done.')
```

```
## [1] "Done."
```

6.

(a)

The code below implements `extinct.prob` and ensures that it returns the right values.

```
extinct.prob <- function(x, rho=.025, beta=(1/5000)) {
  return(beta * x ^ rho)
}

rho <- 0.5
beta <- 1/200

a <- beta * 100^rho
b <- beta * 1600^rho
c <- beta *  10000^rho

stopifnot(c(a,b,c) == extinct.prob(c(100,1600, 10000), rho, beta))
print('Done.')
```

```
## [1] "Done."
```

(b)

The test below ensures that if $\rho = 0$ then the output of `extinct.prob` is β.

```
for (i in 1:100000) {
  stopifnot(extinct.prob(i, 0) == (1/5000))
}
print('Done.')
```

```
## [1] "Done."
```

(c)

The code below tests to make sure that with an input vector of equivalent value, `extinct.prob` outputs all equivalent probabilities.

```
input <- rep(10, 10)

output <- extinct.prob(input)
stopifnot((output == output[1]) %>% sum() == length(input))
print('T1 - Done.')
```

```
## [1] "T1 - Done."
```

```
output <- extinct.prob(input, .5, (1/400))
stopifnot((output == output[1]) %>% sum() == length(input))
print('T2 - Done.')
```

```
## [1] "T2 - Done."
```

```
output <- extinct.prob(input, .0123, (1/123))
stopifnot((output == output[1]) %>% sum() == length(input))
print('T3 - Done.')
```

```
## [1] "T3 - Done."
```

(d)

```
input <- rnorm(10, 100, 10)
stopifnot(extinct.prob(input) %>% unique() %>% length() == length(input))
print('Done.')
```

```
## [1] "Done."
```