②

a) $$\min_{\alpha} \sum_{i=1}^{N} |y_i - \sum_{j=1}^{K-4} \alpha_j b_j(x^{(i)})|^2 + \rho\int_{x_{min}}^{x_{max}} \left(\left(\sum_{j=1}^{K-4}\alpha_j b_j(z)\right)''\right)^2 dz$$

In HW 1, I showed $\sum_{i=1}^{N} |y_i - \sum_{j=1}^{D} \alpha_j h_j(x^{(i)})|^2 = \|y - B\alpha\|^2$ where $B \in \mathbb{R}^{N\times D}$. So $\sum_{i=1}^{N} |y_i - \sum_{j=1}^{K-4}\alpha_j b_j(x^{(i)})|^2 = \|y - B\alpha\|^2$ Where $B \in \mathbb{R}^{N\times(K-4)}$. It remains to show $\rho\int_{x_{min}}^{x_{max}}\left(\left(\sum_{j=1}^{K-4}\alpha_j b_j(z)\right)''\right)^2 dz$ $= \rho a^T\Omega\alpha$:

$$\rho\int_{x_{min}}^{x_{max}}\left(\sum_{j=1}^{K-4}\alpha_j b_j(z)\right)''\left(\sum_{j=1}^{K-4}\alpha_j b_j(z)\right)'' dz = \rho\int_{x_{min}}^{x_{max}}\left(\sum_{j=1}^{K-4}\alpha_j b_j''(z)\right)\left(\sum_{i=1}^{K-4}\alpha_i b_i''(z)\right)dz$$

$$= \rho\int_{x_{min}}^{x_{max}}\left(\sum_{j=1}^{K-4}\sum_{i=1}^{K-4}\alpha_j\alpha_i b_j''(z) b_i''(z)\right)dz = \rho\sum_{j=1}^{K-4}\sum_{i=1}^{K-4}\alpha_j\alpha_i\int_{x_{min}}^{x_{max}} b_j''(z) b_i''(z)\,dz$$

Now for each $i,j$, $\int_{x_{min}}^{x_{max}} b_j''(z) b_i''(z)\,dz$ is a number, denote it $\Omega_{ji}$ $\circledast$

Then we have $\rho\sum_{j=1}^{K-4}\sum_{i=1}^{K-4}\alpha_j\alpha_i\Omega_{ji}$. By HW 1, we know this is just $\rho\,\alpha\cdot(\Omega\alpha) = \rho\alpha^T\Omega\alpha$, where $\Omega \in \mathbb{R}^{K-4\times K-4}$

And so putting everything together we get: $\min_{\alpha}\|y - \beta\alpha\|^2 + \rho\alpha^T\Omega\alpha$ where $\beta \in \mathbb{R}^{N\times K-4}$, $\Omega \in \mathbb{R}^{K-4\times K-4}$ and symmetric and each entry in $\Omega$ is $\Omega_{kl} = \int_{x_{min}}^{x_{max}} b_k''(z) b_l''(z)\,dz$ as shown in $\circledast$.

Now lets find the soln to the min problem:

$$\min_{\alpha}\left[\|y - \beta\alpha\|^2 + \rho\alpha^T\Omega\alpha\right] = \min_{\alpha}\left[(y-\beta\alpha)\cdot(y-\beta\alpha) + \rho\alpha^T\Omega\alpha\right]$$

$$= \min_{\alpha}\left[y^Ty - 2y^T\beta\alpha + \alpha^T\beta^T\beta\alpha + \rho\alpha^T\Omega\alpha\right] = \min_{\alpha} f(\alpha)$$

$f(\alpha)$ is quadratic and we know how to take its gradient:

$\nabla f(\alpha) = -2\beta^T y + 2\beta^T\beta\alpha + 2\rho\Omega\alpha$ since both $\Omega$ and $\beta^T\beta$ are symmetric

Solving for the critical point: $\nabla f(\alpha) = 0 \Rightarrow$

$\beta^T\beta\alpha + \rho\Omega\alpha = \beta^T y \Rightarrow \alpha^* = (\beta^T\beta + \rho\Omega)^{-1}\beta^T y$, if $(\beta^T\beta + \rho\Omega)$ is invertible $(\rho > 0)$. If $\rho > 0$, then $\alpha^*$ is a min

# HW12

Ines Pancorbo

4/13/2020

## 2 b)

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

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

```r
# 1000 knots equally spaced between
# the minimum and maximum x values (ages) of data_f
x_min <- range(data_f$age)[1]
x_max <- range(data_f$age)[2]
knots <- seq(x_min, x_max, length.out = 1000)
x_min
```

```
## [1] 9.4
```

```r
x_max
```

```
## [1] 25.55
```

```r
# 100000 equally spaced points in mygrid,
# should be dense enough since range of age is not too big
mygrid <- seq(x_min, x_max, length.out = 100000)

# distance between points
h <- mygrid[2] - mygrid[1]
```

```r
library("splines")
B <- splineDesign(knots = knots,
                  x = data_f$age[order(data_f$age)],
                  outer.ok = T)

Bpp <- splineDesign(knots = knots,
                    x = mygrid,
                    derivs = 2,
                    outer.ok = T)

# checking dim of Bpp, should be 100000 x (1000-4)
dim(Bpp)
```

```
## [1] 100000    996
```

```r
# checking dim of B,
# should be (length(data_f$age)=259) x (1000-4)
dim(B)
```

```
## [1] 259 996
```

The $\Omega$ matrix via numerical integration is:

```r
# Aproximating each entry in Omega matrix using Riemann integration
omega <- (t(Bpp) %*% Bpp)*h

# double-checking dimensions of omega, should be (1000-4) x (1000-4)
dim(omega)
```

```
## [1] 996 996
```

```r
# printing the first 20 entries of first row of omega
omega[1,1:20]
```

```
##  [1]  6.311774e+05 -3.550285e+05 -5.518051e+00  3.944960e+04  0.000000e+00
##  [6]  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
## [11]  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
## [16]  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
```

## 2 c)

For any given $B$, $B^TB$ is symmetric and positive semi-definite. Positive semi-definite because for any given $x$, $x^TB^TBx = (Bx)^T(Bx) = \|Bx\|^2 \geq 0$. Because $B^TB$ is positive semi-definite, we are not guaranteed $B^TB$ is invertible. For my given $B$, lets see if $B^TB$ is invertible or not.

```r
# Lets first show B^TB is not invertible
det(t(B) %*% B)
```

```
## [1] 0
```

The determinant of $B^TB$ is 0 so $B^TB$ is not invertible.

Now, $\Omega$ is symmetric since by definition each entry of $\Omega$ is $\Omega_{kl} = \int_{x_{min}}^{x_{max}} b_k''(z)b_l''(z)dz$ and multiplication is commutative (i.e., $b_k''(z)b_l''(z) = b_l''(z)b_k''(z) => \Omega_{kl} = \Omega_{lk}$). Further, it is positive semi-definite by construction: $x^T\Omega x = \int_{x_{min}}^{x_{max}}((\sum_{j=1}^{996} x_jb_j(z))'')^2dz \geq 0$. Again, we are not guaranteed invertibility but lets see if $\Omega$ is invertible.

```r
# Using the determinant to check invertability is not
# the best approach
# You can easily get an overflow during the numerical computation of the determinant
# For example:
det(omega)
```

```
## [1] Inf
```

```r
# The fact that det = infty might indicate omega is close to singular
# but R is still able to give you an inverse:
solve(omega)[1,1:5]
```

```
## [1] 6.765061e-06 1.169034e-05 1.708152e-05 2.232080e-05 2.757378e-05
```

```r
# Given numerical precision, a better approach is finding kappa(matrix),
# if this value exceeds 10^16, like we saw in class, then the matrix
# is computationally singular
kappa(omega)
```

```
## [1] 49872460901
```

```r
log10(kappa(omega))
```

```
## [1] 10.69786
```

So kappa(omega) is approx $10^{11} < 10^{16}$, and therefore omega is computationally invertible (=> invertible). Omega is consequently positive definite.

Now it is easy to show that for my given $B$ (which is positive semi-definite), for my given $\Omega$ (which is positive definite), and for any given $\rho > 0, B^T B + \rho\Omega$ will be positive definite and thus, invertible:

If $\rho > 0$ then $\rho\Omega$ is still positive definite: By the spectral decomposition theorem, $\Omega = QDQ^T \Rightarrow \rho\Omega = \rho(QDQ^T) \Rightarrow \rho\Omega = Q\rho DQ^T$. In other words $\rho\Omega$ is a symmetric matrix with the same eigenvectors as $\Omega$ and eigenvalues equal to the eigenvalues of $\Omega$ times $\rho$ (consequently, all positive eigenvalues).

Now, since by definition of $\rho\Omega$ being positive definite we have $x^T \rho\Omega x > 0$ and by definition of $B^T B$ being positive semi-definite we have $x^T B^T B x \geq 0$, we have as a result that for any given $x$, $x^T(B^T B + \rho\Omega)x = x^T B^T B x + x^T \rho\Omega x > 0$.

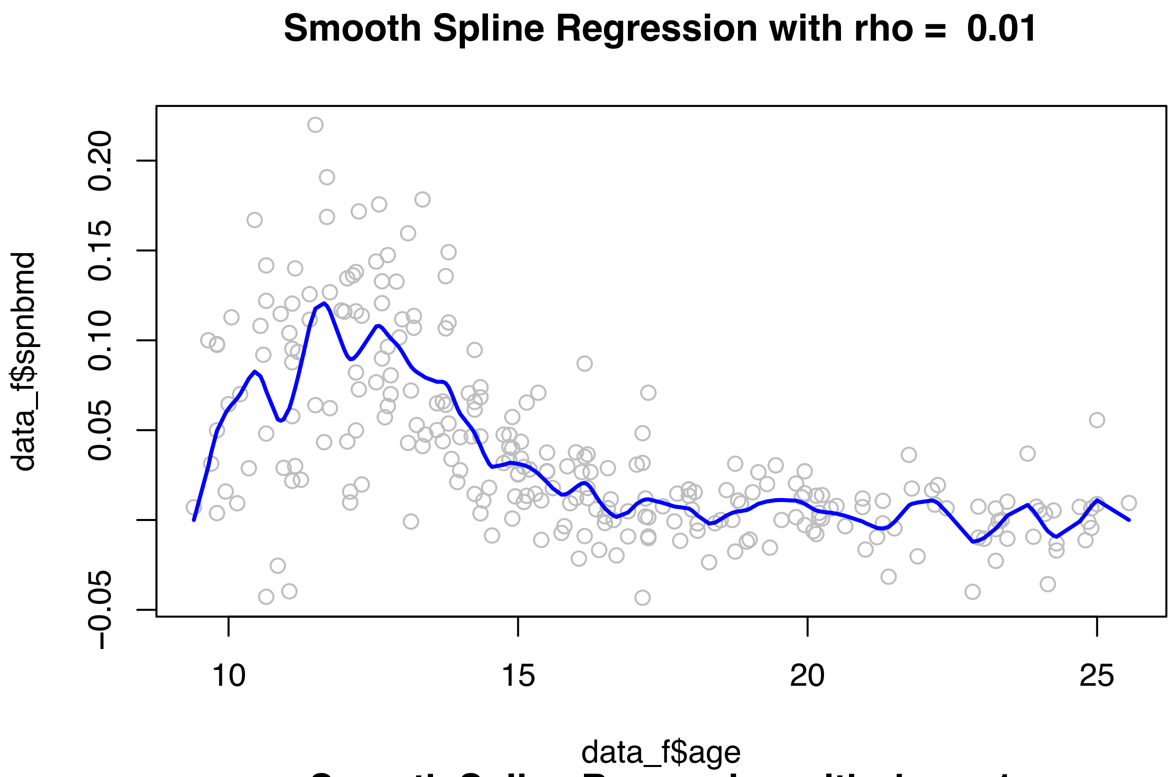
Consequently, we have that for my given $B$, for my given $\Omega$, and for any given $\rho > 0, B^T B + \rho\Omega$ will be positive definite and thus, invertible.

## 2 d)

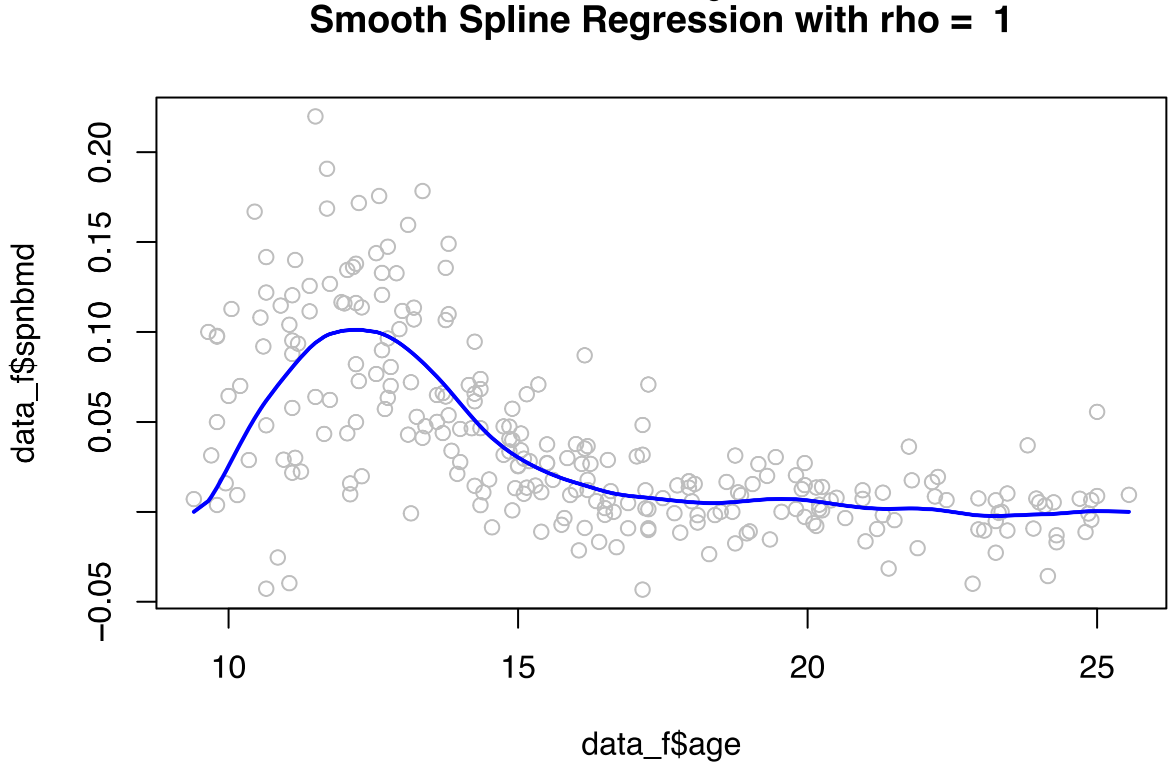
```r
# function
smooth_spline_regression <- function(rho){

  # calculating the coefficients of the spline given work in 2 a)
  alpha <- solve(t(B)%*%B+rho*omega,
                 t(B)%*%(data_f$spnbmd[order(data_f$age)]))

  ###################### spline ##########################
  spline <- B%*%alpha
  ######################################################

  # plot
  plot(data_f$age, data_f$spnbmd, col = "grey")
  points(data_f$age[order(data_f$age)],
         spline,
         col = "blue",
         lwd = 2,
         type = "l")
  title(paste("Smooth Spline Regression with rho = ", rho))
}
```

Now, using the function defined above, calculate alpha using $\rho = 0.01, 1, 100$ and plot resulting spline.

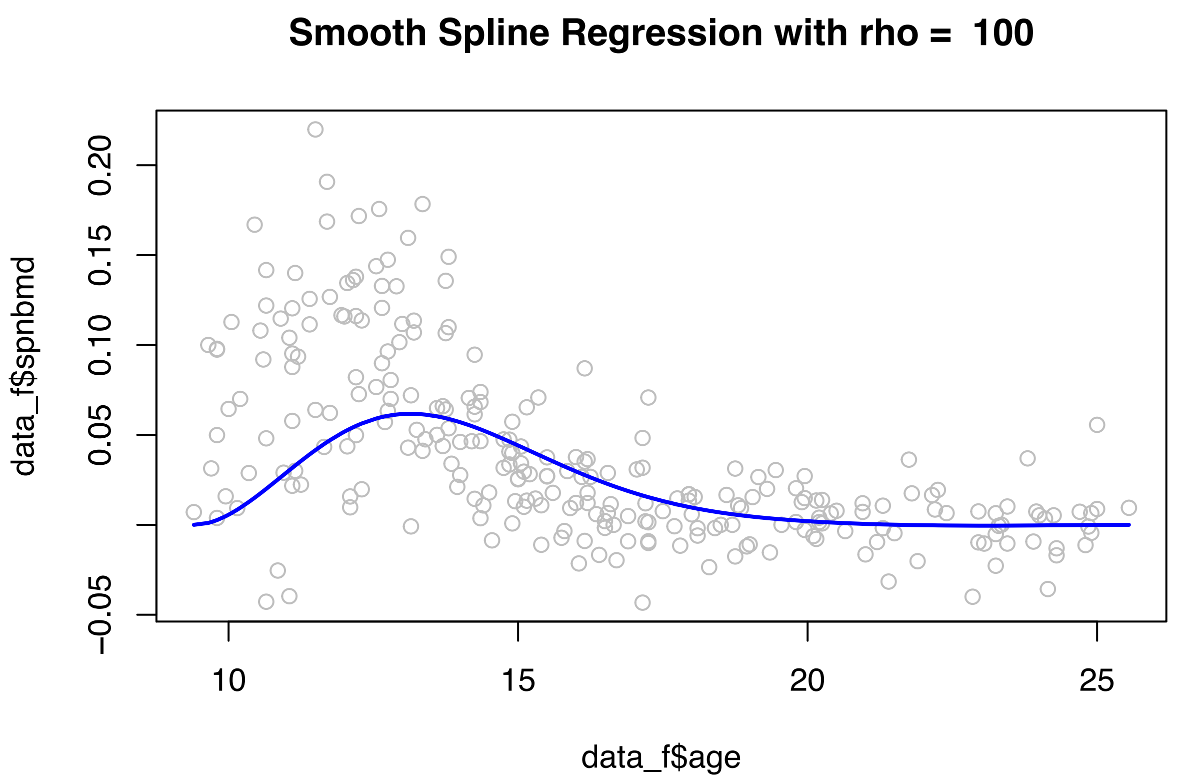
```r
for (rho in c(0.01, 1, 100)){
  smooth_spline_regression(rho)
}
```



Smooth Spline Regression with rho = 0.01
data_f$spnbmd
−0.05
0.05
0.10
0.15
0.20
10
15
20
25
data_f$age



Smooth Spline Regression with rho = 1
data_f$spnbmd
−0.05
0.05
0.10
0.15
0.20
10
15
20
25
data_f$age



**Comment on the fit in each case:**

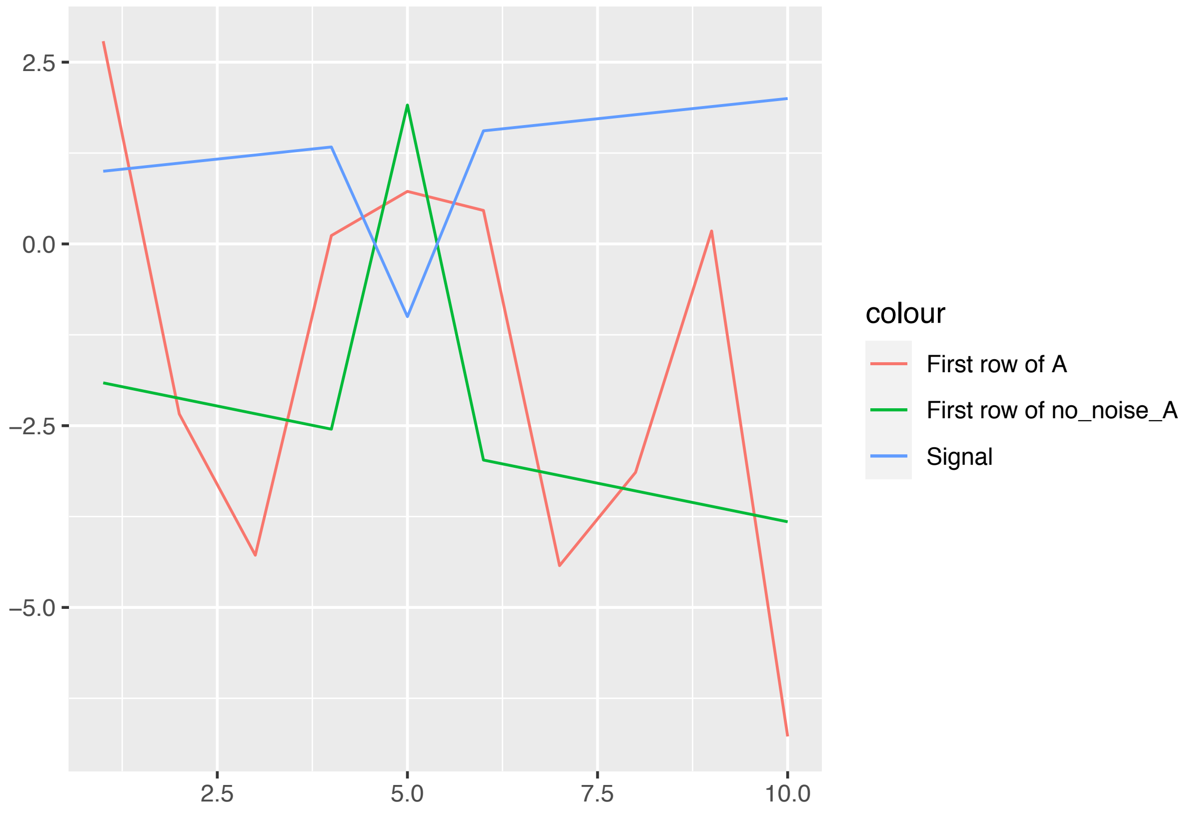
It looks like for $\rho = 0.01$ the spline regression is overfitting our data (the wiggly line, and how this wiggly line passes through some of the data points) and so a penalty term of $\rho = 0.01$ might be too small. For $\rho = 100$, it looks like the spline regression is underfitting our data (it looks almost linear) so a penalty term of $\rho = 100$ might be too big. Lastly, for $\rho = 1$, the spline regression seems like a decent fit (it looks like it is capturing the "shape" of the data, and the spline isn't too "wiggly" or too linear).

## 3 a)

```r
# reading in as matrix A
A <- as.matrix(read.csv("A.txt", header = FALSE, sep=" "))
first_row_A <- A[1,]

# reading in as matrix no_noise_A
no_noise_A <- as.matrix(read.csv("no_noise_A.txt", header = FALSE, sep=" "))
first_row_no_noise_A <- no_noise_A[1,]

# getting signal
sig <- seq(1,2,length.out = 10)
sig[5] <- -1

# getting q
q <- as.matrix(read.csv("q.txt", header = FALSE, sep=" "))
```

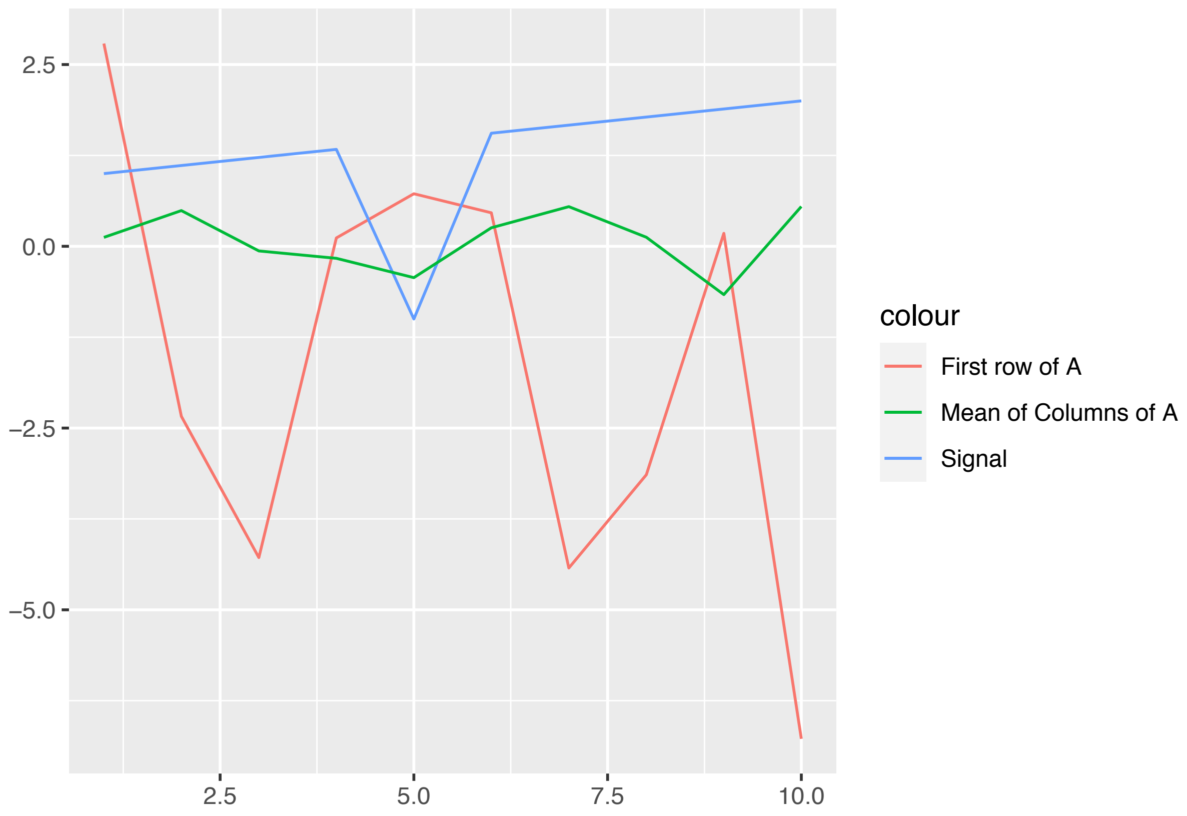
Now, lets plot the values in the first row of A, the first row of no_noise_A, and the underlying signal.

```r
library(ggplot2)
ggplot() +
geom_line(aes(x = seq(1,ncol(A)), y = first_row_A, color = "First row of A")) +
geom_line(aes(x = seq(1,ncol(A)), y = first_row_no_noise_A, color = "First row of no_noise_A")) +
geom_line(aes(x = seq(1,ncol(A)), y = sig, color = "Signal")) +
xlab("") + ylab("")
```



Now, lets plot the values in the first row of $A$, the 10 values corresponding to the means of the 10 columns of $A$, and the signal to see if it is possible to figure out the signal from the first row of $A$ or the colMeans(A).

```r
library(ggplot2)
ggplot() +
  geom_line(aes(x = seq(1,ncol(A)), y = first_row_A, color = "First row of A")) +
  geom_line(aes(x = seq(1,ncol(A)), y = sig, color = "Signal")) +
  geom_line(aes(x = seq(1,ncol(A)), y = colMeans(A), color = "Mean of Columns of A")) +
  xlab("") + ylab("")
```



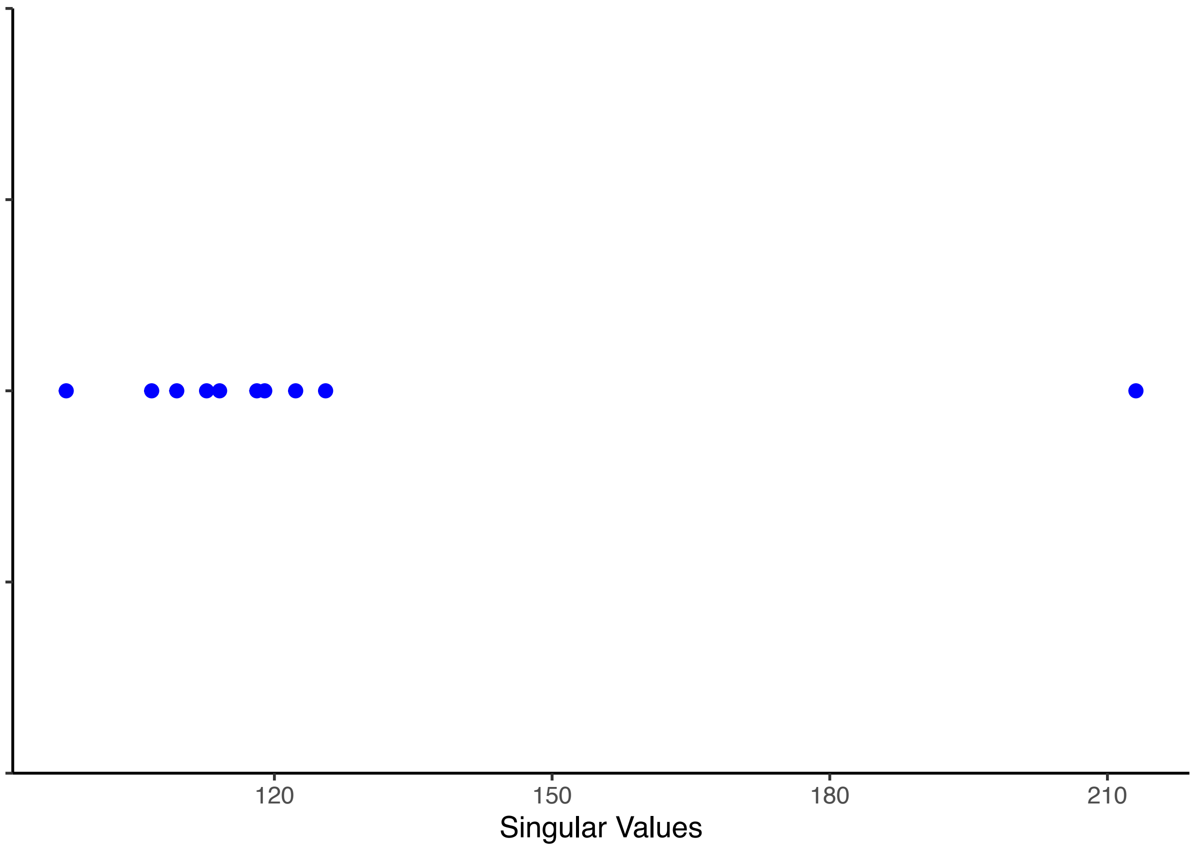
At least from the plot, it doesn't look like there is a relationship/connection between the column means of $A$ and the signal, or the first row of $A$ and the signal. So I would say, no, I cannot tell what the signal is by looking at the means of the 10 columns of $A$, or by looking at the first row of $A$.

## 3 b)

```r
# find svd of A
svd <- svd(A)
# get U and V
U <- svd$u
V <- svd$v

# get singular values of A
singular_values <- svd$d
# should get 10
length(singular_values)
```

```
## [1] 10
```

```r
# plot singular values of A
ggplot() +
  geom_point(aes(x=singular_values, y = rep(0,length(singular_values))), size=2, color='blue') +
  theme_classic() +
  theme(axis.text.y=element_blank()) +
  xlab("Singular Values") + ylab("")
```



**Comment on the singular values:**

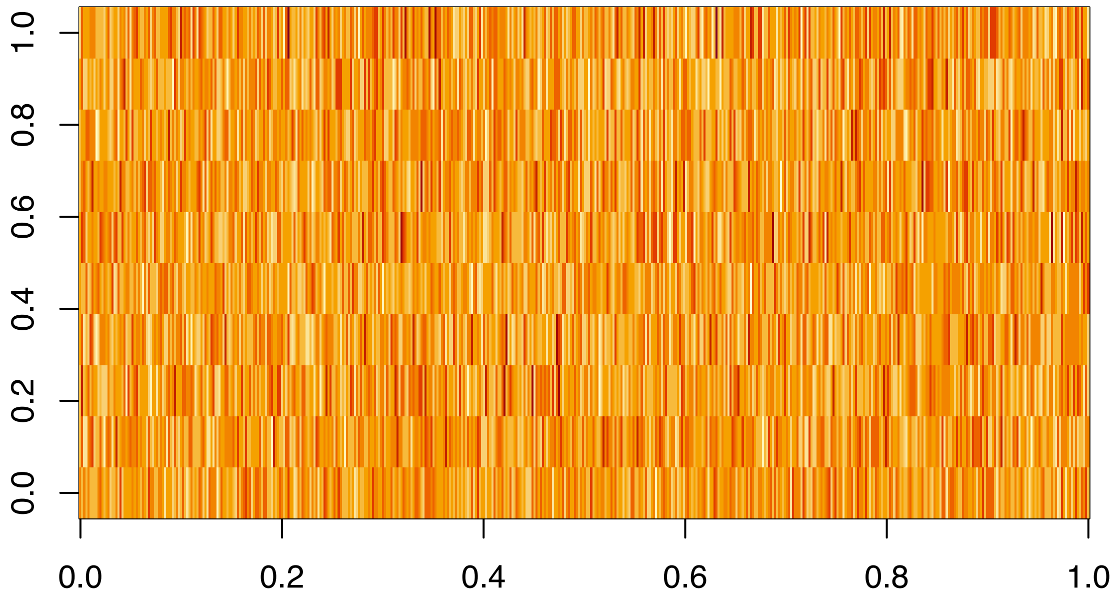
Some things to keep in mind first:

(1) The rank of a matrix can represent the amount of unique information stored by a matrix (higher rank => more unique information) and it is the number of non-zero singular values
(2) One can think of the SVD as a method that decomposes $A$ into a linear combination of rank-1 matrices weighted by the corresponding singular value

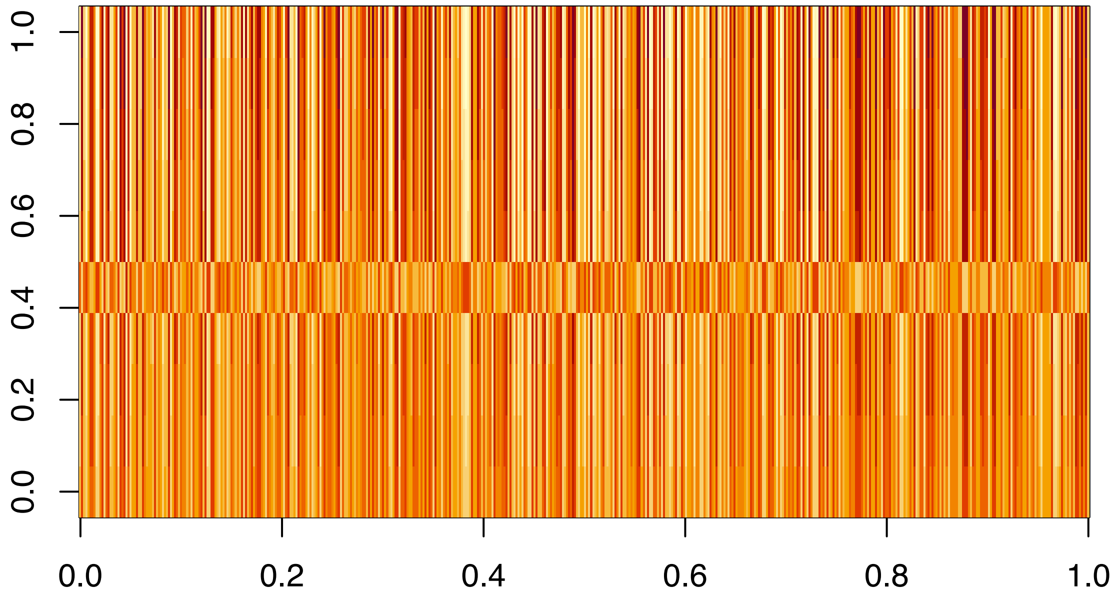
Having mentioned the above: the SVD of matrix $A$ will express matrix $A$ as a sum of 10 rank-1 matrices weighted by the corresponding singular value. From the plot of singular values, you can see that there is one that dominates. Therefore, the plot is essentially telling us that we can approximate $A$ well with the rank-1 matrix corresponding to that dominant singular value (i.e., approximate $A$ with $s_1(u_1 v_1^T)$, where $s_1$ is the dominant singular value, entry $(1, 1)$ of the diagonal matrix, and $u_1, v_1$ are the first column vectors of the matrices $U$ and $V$). In other words, we can think of matrix $A$ (rank 10) as a rank 1 matrix. This makes sense given that matrix $A$ (rank 10) is just a "noisy" version of matrix no_noise_A (rank 1).

Lets now find this rank-1 approximation of $A$ ($s_1(u_1 v_1^T)$) and denote it $A_1$.

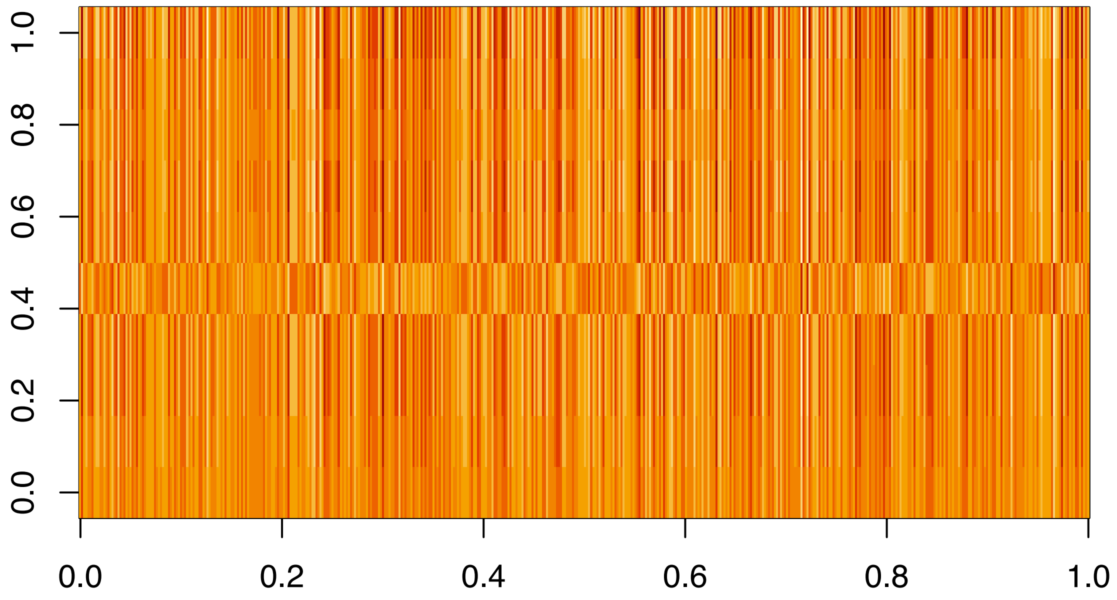
```r
# get matrices U and V
U <- svd$u
V <- svd$v

# find A1
A1 <- singular_values[1]* (U[,1] %*% t(V[,1]))

# Now use image() to visualize the three
# matrices and confirm that A1 removes the noise from A
image(A)
```



```r
image(no_noise_A)
```



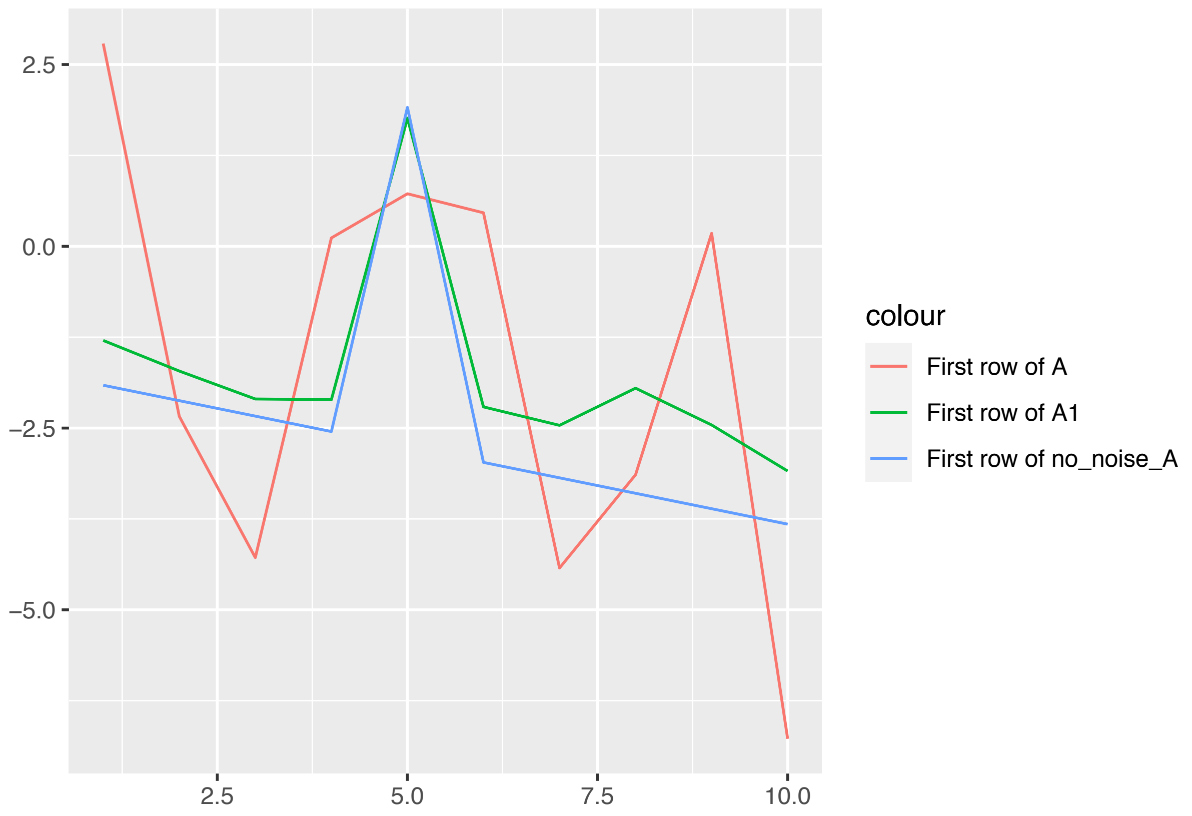
```r
image(A1)
```



From what was explained above, it would be reasonable to expect that this rank-1 approximation $A_1$ of $A$ be similar to no_noise_A. That is, we should expect $A_1$ to throw out much of the "noise" and little of the "underlying signal and vector $q$." From the images, since $A_1$ is very similar to no_noise_A (a little bit blurrier but still dispays similar patterns) it is reasonable to conclude that $A_1$ removes the noise from $A$.

Now lets compare the first rows of $A$, no_noise_A, and $A_1$ (the 1-rank approx of $A$ corresponding to the dominant singular value) by plotting them.

```r
library(ggplot2)
ggplot() +
  geom_line(aes(x = seq(1,ncol(A)), y = first_row_A, color = "First row of A")) +
  geom_line(aes(x = seq(1,ncol(A)), y = A1[1,], color = "First row of A1")) +
  geom_line(aes(x = seq(1,ncol(A)), y = no_noise_A[1,], color = "First row of no_noise_A")) +
  xlab("") + ylab("")
```



From what we saw with image(), it makes sense that the first rows of $A_1$ and no_noise_A are similar. The plot above confirms this. The first row of $A$ is not similar to the first row of $A_1$ or first row of no_noise_A due to the presence of noise.

Answering last question: **Given a row of $A$ in the form $q \cdot sig + noise$, what role do $v_1$, $s_1$ and $u_1$ have in capturing $q$ and $sig$?**

Let $U\Sigma V^T$ be the svd of the matrix $A$. Two things first:

(1) By construction $A = q(sig)^T + noise$, where $q$, $sig$ are column vectors in $R^{500}$ and $R^{10}$ respectively, and noise is a 500 x 10 matrix. So it follows that the ith row of $A$ will be $(q_i)sig + noise[i,]$.

(2) We know that $U$ is a 500 x 500 matrix, and $V$ is a 10 x 10 matrix. Since $A$ has rank 10 we know that the first 10 columns of $U$ are an orthonormal basis for the column space of $A$, and the 10 columns of $V$ are an orthonormal basis for the row space of $A$.

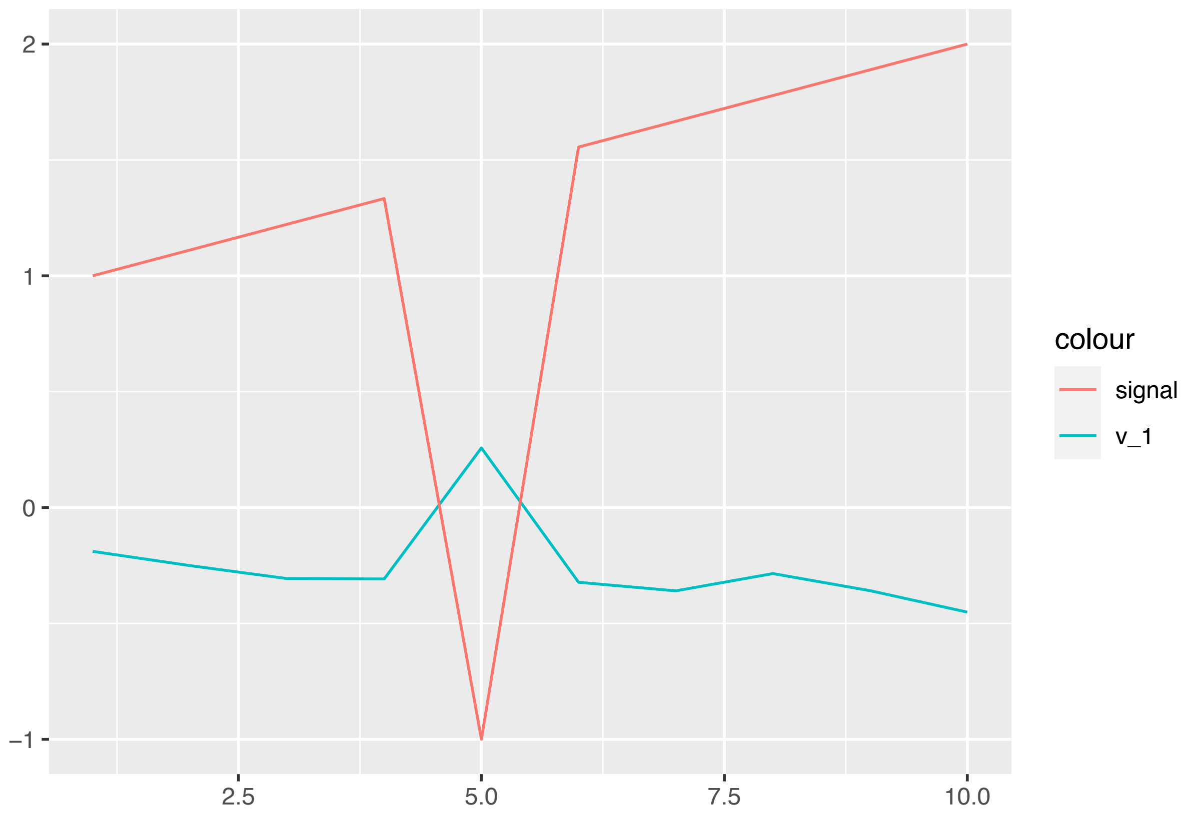
From the points discussed above, any row of $A$ will be in the span of $V$. Lets take the first row of A as an example:

```r
# coefficients for linear combination
solve(V,A[1,])
```

```
##  [1]  6.8511699  1.1537398 -0.5037349  0.3690536 -4.1620982  0.8170252
##  [7]  2.6494192 -4.5955250  1.6988247 -3.1671041
```

So each row of $A$ is connected to each of the 10 columns vectors $v_i$ in $V$ in that sense (each row of $A$ is a linear combination of the $v_i$'s). However, from the work done above, we can denoise $A$ (i.e., $A \approx q(sig)^T$) by approximating $A$ with the 1-rank matrix $s_1(u_1v_1^T)$ (i.e., $A \approx s_1(u_1v_1^T) \approx q(sig)^T$). So the row space of $q(sig)^T$ can be approximated with the span of the vector $v_1$. And so it is clear, that $v_1$ should be able to "capture" the vector $sig$. If we plot $v_1$ and $sig$ we should see similarity.

```r
ggplot() +
  geom_line(aes(x = seq(1,ncol(A)), y = V[,1], color = "v_1")) +
  geom_line(aes(x = seq(1,ncol(A)), y = sig, color = "signal")) +
  xlab("") + ylab("")
```



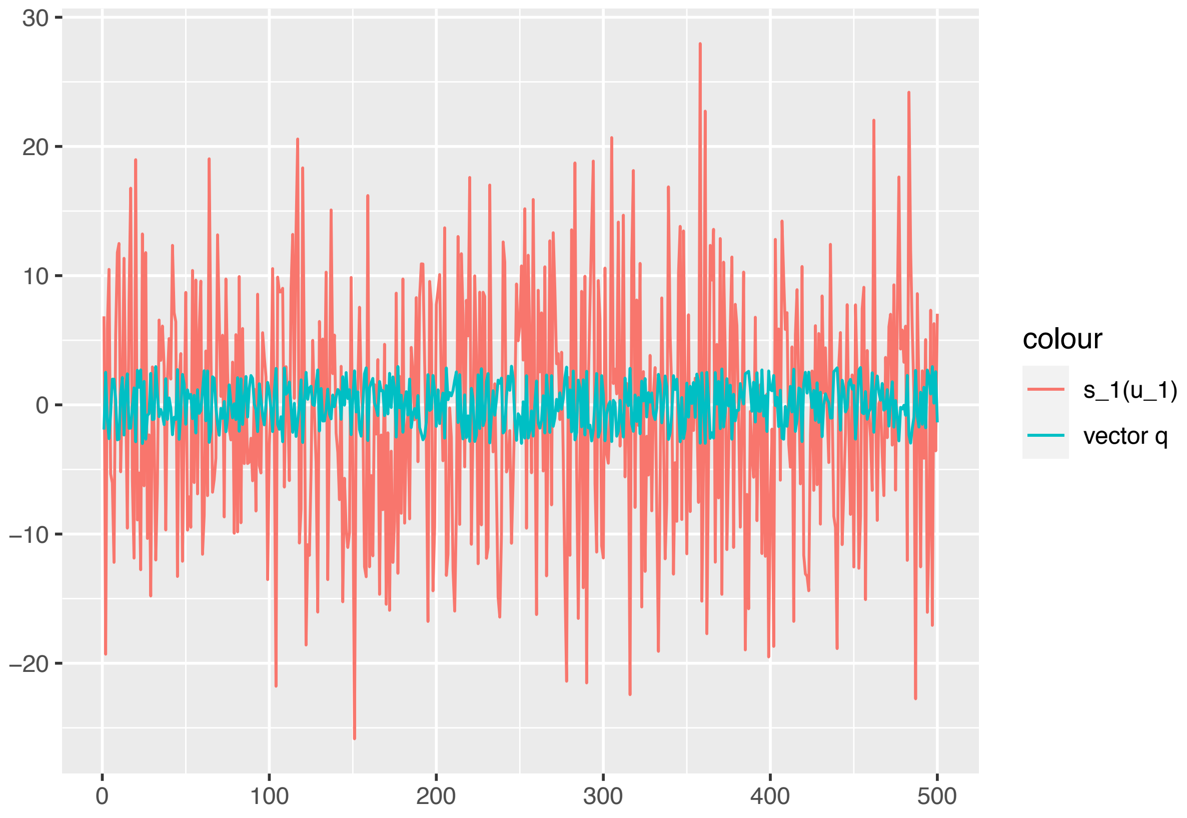
Note: the vectors in $U$ and $V$ are defined up to a sign. That is, the svd is not exactly “unique”: One can flip any right singular vector, as long as the corresponding left singular vector is flipped as well. And so, taking this into account, you can see how $v_1$ (or -$v_1$) captures the signal (both the signal and $v_1$ (or -$v_1$) lines are shaped similarly, even if scale is different).

Now, that we have discussed the role $v_1$ plays in any given row of $A$ (it captures the $sig$ vector), lets turn to $q$.

Since by construction, $A = q(sig)^T + noise$, each column of $A$ is the vector $q$ times a corresponding entry in $sig$ plus some $noise$. So lets think of $q$ in terms of the column space of $A$.

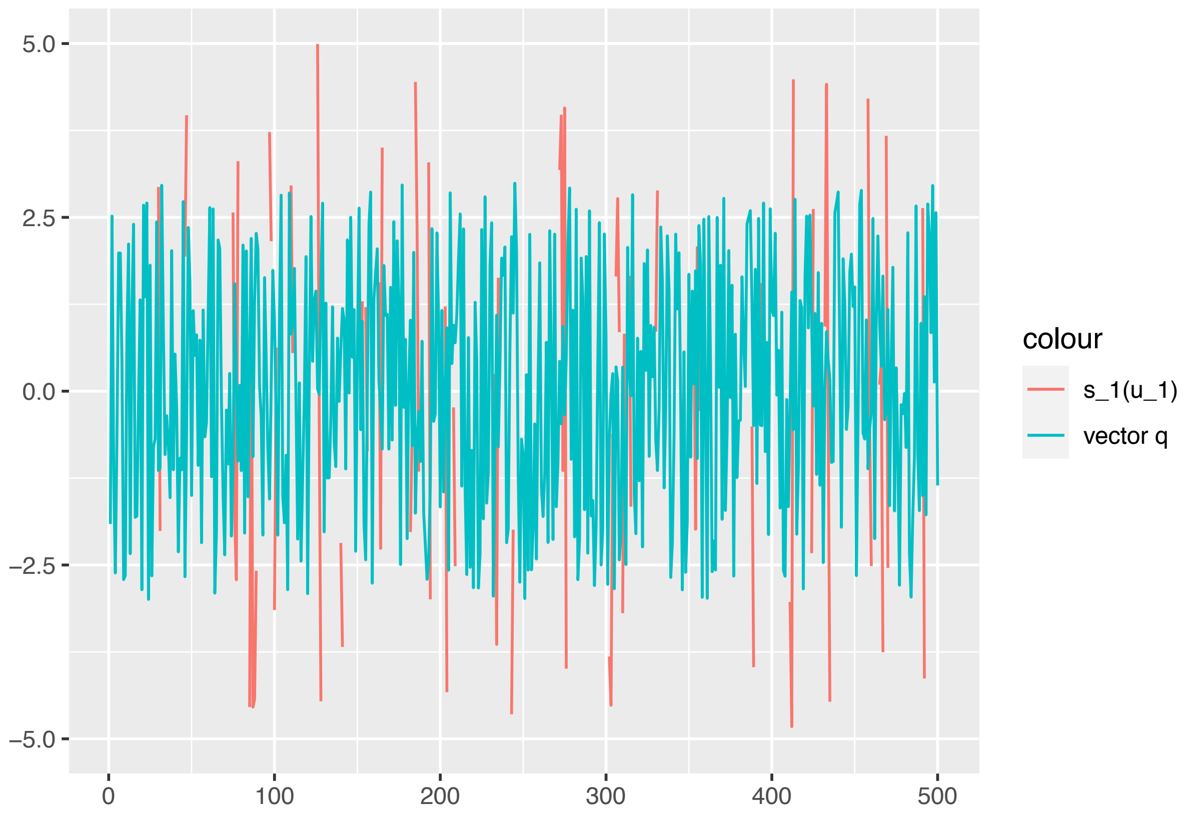
Now, since $A$ is of rank 10, the first 10 column vectors in $U$ are an orthonormal basis for the column space of $A$. However, from work done above, since approximating $A$ with $s_1(u_1v_1^T)$ will give you an approximate denoised version of $A$ (i.e., $s_1(u_1v_1^T) \approx q(sig)^T$), the first vector in $U$, $u_1$, will be an approximate basis for the column space of $q(sig)^T$. So the column space of $q(sig)^T$ can be approximated with the span of the vector $u_1$. So it becomes clear that $u_1$ should be able to “capture” the vector $q$, and in terms of mappings we can think of $(s_1)u_1$ mapping a scalar to $q$. Therefore, we should see a relationship between $(s_1)u_1$ and $q$ if we plot them together.

```r
ggplot() +
  geom_line(aes(x = seq(1,nrow(A)), y = U[,1]*singular_values[1], color = "s_1(u_1)")) +
  geom_line(aes(x = seq(1,nrow(A)), y = q, color = "vector q")) +
  xlab("") + ylab("")
```



As can be seen from the plot $(s_1)u_1$ captures the vector $q$ (even though scale is different, the line corresponding to $q$ and the line corresponding to $(s_1)u_1$ display a similar shape). We can zoom in to see how they intersect in the range they share:

```r
ggplot() +
  geom_line(aes(x = seq(1,nrow(A)), y = U[,1]*singular_values[1], color = "s_1(u_1)")) +
  geom_line(aes(x = seq(1,nrow(A)), y = q, color = "vector q")) +
  xlab("") + ylab("") +
  ylim(-5,5)
```


5.0
2.5
0.0
−2.5
−5.0
0
100
200
300
400
500
colour
s_1(u_1)
vector q