MATH 423/533 Regression and Analysis of Variance

September 28, 2017

Least Squares and Matrix Decompositions

Contents

1 Introduction

1.1 Notation

Let $\mathbb{R}^{n\times p}$ denote the set of real valued n by p matrices. When $M \in \mathbb{R}^{n\times p}$, $A \in \mathbb{R}^{p\times p}$, let M' be the transpose of M and let $|A|$ be the determinant of A. Define the set of symmetric p by p matrices by $\mathbb{S}^p = \{M \in \mathbb{R}^{p\times p} : M = M'\}$. For $A \in \mathbb{S}^p$, let $\varphi_j(A)$, $\varphi_{\max}(A)$, and $\varphi_{\min}(A)$, be the j-th largest eigenvalue, maximum eigenvalue, and minimum eigenvalue of A respectively. Let $\mathbb{S}^p_0 = \{M \in \mathbb{S}^p : \varphi_{\min}(M) \geq 0\}$ be the set of symmetric and non-negative definite (positive semi-definite) matrices and let $\mathbb{S}^p_+ = \{M \in \mathbb{S}^p : \varphi_{\min}(M) > 0\}$ be the set of symmetric and positive definite matrices.

Orthogonal matrix an orthogonal matrix is a square matrix with real entries whose columns and rows are orthogonal unit vectors

$$Q'Q = QQ' = I$$

where I is the identity matrix.

1.2 Introductory example: the Normal linear regression cases model

Let $X \in \mathbb{R}^{n\times p}$ be the nonrandom design matrix, where all entries in its first column equal 1 and let $y = (y_1, \dots, y_n)' \in \mathbb{R}$ be the response vector, where y_i is the value of the response for the i-th case. The model assumes that y is a realization of the random vector

$$Y \sim N_n(X\beta_*, \sigma_*^2 I_n).$$

The density for $N_n(\mu, \Sigma)$ and its logarithm evaluated at $x \in \mathbb{R}^n$ are

$$\phi(x; \mu, \Sigma) = (2\pi)^{-n/2}|\Sigma^{-1}|^{1/2} \exp\left\{-\frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu)\right\},$$

$$\log \phi(x; \mu, \Sigma) = -\frac{n}{2}\log(2\pi) + \frac{1}{2}\log|\Sigma^{-1}| - \frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu).$$

The random log-likelihood function $\ell(\cdot,\cdot;Y) : \mathbb{R}^p \times \mathbb{R}_+ \to \mathbb{R}$ is defined by

$$\begin{aligned}
\ell(\beta,\sigma^2;Y,X) &= \log \phi(Y; X\beta, \sigma^2 I_n) \\
&= -\frac{n}{2}\log(2\pi) + \frac{1}{2}\log|\sigma^{-2}I_n| - \frac{1}{2}(Y-X\beta)'\sigma^{-2}I_n(Y-X\beta) \\
&= -\frac{n}{2}\log(2\pi) + \frac{n}{2}\log\sigma^{-2} - \frac{1}{2}\sigma^{-2}(Y-X\beta)'(Y-X\beta).
\end{aligned}$$

The maximum likelihood estimator is

$$(\hat\beta, \hat\sigma^2) = \underset{(\beta,\sigma^2)\in\mathbb{R}^p\times\mathbb{R}_+}{\arg\min} \; -\ell(\beta,\sigma^2;Y,X).$$

We call $-\ell$ the *objective function*, (β,σ^2) the *optimization variable*, and $\mathbb{R}^p \times \mathbb{R}_+$ *the feasible set*. For reasons we will study soon, $(\hat\beta,\hat\sigma^2)$ solve the two equations $\nabla_\beta - \ell(\beta,\sigma^2;Y,X) = 0$ and $\nabla_{\sigma^2} - \ell(\beta,\sigma^2;Y,X) = 0$.

We have that

$$(Y-X\beta)'(Y-X\beta) = Y'Y - 2\beta'X'Y + \beta'X'X\beta,$$

so

$$\begin{aligned}
\nabla_\beta - \ell(\beta,\sigma^2;Y,X) &= \frac{1}{2}\sigma^{-2}\nabla_\beta\{(Y-X\beta)'(Y-X\beta)\} \\
&= \frac{1}{2}\sigma^{-2}\nabla_\beta\{Y'Y - 2\beta'X'Y + \beta'X'X\beta\} \\
&= \frac{1}{2}\sigma^{-2}(-2X'Y + 2X'X\beta) \\
&= \sigma^{-2}(-X'Y + X'X\beta).
\end{aligned}$$

Then $\nabla_\beta - \ell(\beta,\sigma^2;Y,X) = 0$ is equivalent to $-X'Y + X'X\beta = 0$, so $\hat\beta$ solves

$$X'X\hat\beta = X'Y.$$

At this point, if $(X'X)^{-1}$ exists, then $\hat\beta = (X'X)^{-1}X'Y$, i.e. the maximum likelihood estimator of β_* is the same as the ordinary least squares estimator of β_* when they both exist. Next,

$$\nabla_{\sigma^2} - \ell(\beta,\sigma^2;Y,X) = \frac{n}{2\sigma^2} - \frac{1}{2\sigma^4}(Y-X\beta)'(Y-X\beta).$$

Solving $\nabla_{\sigma^2} - \ell(\beta,\sigma^2;Y,X) = 0$, we have that

$$\frac{n}{2\hat\sigma^2} = \frac{1}{2\hat\sigma^4}(Y-X\beta)'(Y-X\beta).$$

So

$$\hat{\sigma}^2 = \frac{1}{n}(Y - X\beta)'(Y - X\beta).$$

1.3 The linear regression cases model and ordinary least squares

Recall that $y = (y_1, \ldots, y_n)'$ is vector of measured responses for the n cases and $X \in \mathbb{R}^{n \times p}$ is the nonrandom design matrix, where $x_{i1} = 1$ for $i = 1, \ldots, n$. The linear regression cases model assumes that y is realization of

$$Y = X\beta_* + \epsilon,$$

where $\beta_* = (\beta_{*1}, \ldots, \beta_{*p})' \in \mathbb{R}^p$ is the unknown regression coefficient vector and $\epsilon = (\epsilon_1, \ldots, \epsilon_n)'$ has $\epsilon_1, \ldots, \epsilon_n$ iid with an unspecified distribution having mean zero and unknown variance σ_*^2. The normal linear regression cases model is a special case with $\epsilon \sim N_n(0, \sigma_*^2 I_n)$.

The ordinary least squares estimator of β_* is defined by

$$\hat{\beta} = \operatorname*{arg\,min}_{\beta \in \mathbb{R}^p} \|Y - X\beta\|^2, \tag{1}$$

where here the arg min is a set of global minimizers: this set has exactly one point if $X'X$ is invertible. Let $f : \mathbb{R}^p \to \mathbb{R}$ be the objective function in the optimization of (1): $f(\beta) = \|Y - X\beta\|^2$. Then

$$f(\beta) = Y'Y - 2\beta'X'Y + \beta'X'X\beta,$$

and

$$\nabla f(\beta) = -2X'Y + 2X'X\beta.$$

For reasons we will study soon, a minimizer $\hat{\beta}$ solves $\nabla f(\beta) = 0$, so

$$-2X'Y + 2X'X\beta = 0.$$

At this point, if $X'X$ is invertible, then $\hat{\beta} = (X'X)^{-1}X'Y$. This is the same formula that we saw for the maximum likelihood estimator derived in section 1.2. It is less computationally expensive in practice to solve the linear system of equations:

$$X'X\hat{\beta} = X'Y$$

for $\hat{\beta}$ instead of computing $(X'X)^{-1}$ in the closed-form solution.

2 Solving $X'X\hat{\beta} = X'Y$

2.1 Generating data from the Normal linear regression cases model

In order to illustrate methods to solve $X'X\hat{\beta} = X'Y$, we will generate data from the Normal linear regression cases model described in section 1.2. To do this, we will create/pick a design matrix X, an “unknown” regression coefficient vector β_*, and an “unknown” error variance σ_*^2. Let the design matrix be $X = [1_n, v_2, Z]$, where $1_n \in \mathbb{R}^n$ is the vector of ones, $v_2 = (1'_{n/2}, 0, \ldots, 0)' \in \mathbb{R}^n$, and Z is a realization of an n by $p-2$ random matrix with iid $N(0,1)$ entries. Let the regression coefficient vector β_* be a realization of a random vector with iid $N(0, 1/p)$ entries. Set $n = 10$, $p = 5$, and $\sigma_* = 1$.

```
set.seed(680)
n=10; p=5; sigma.star=1

## create the design matrix
Z=matrix( rnorm(n*(p-2)), nrow=n, ncol=(p-2))
v2=c( rep(1, n/2), rep(0, n/2) )
X=cbind(1, v2, Z)

## create the regression coefficient vector
beta.star=p^(-0.5) * rnorm(p)

## generate the responses
y=X%*%beta.star + sigma.star * rnorm(n)
```

The R objects X and y are our design matrix and response vector respectively. The slowest way to compute the realization of $\hat{\beta}$ uses the closed-form solution directly:

```
qr.solve(t(X)%*%X) %*% t(X)%*%y

##           [,1]
##      1.0343738
## v2  -0.7203269
##     -0.7991497
##     -0.1613902
##      0.3673047
```

It can be slightly faster to use the `crossprod` function:

```
qr.solve(crossprod(X)) %*% crossprod(X,y)
```

```
##          [,1]
##     1.0343738
## v2 -0.7203269
##    -0.7991497
##    -0.1613902
##     0.3673047
```

It is faster to bypass the computation of $(X'X)^{-1}$ and solve the linear system of equations $X'X\beta = X'Y$ directly:

```
qr.solve(crossprod(X), crossprod(X,y))
```

```
##          [,1]
##     1.0343738
## v2 -0.7203269
##    -0.7991497
##    -0.1613902
##     0.3673047
```

In the next subsection, we will show faster ways to compute the realization of $\hat{\beta}$ using the QR decomposition of X.

2.2 Using the reduced QR decomposition

Suppose that $X \in \mathbb{R}^{n\times p}$, $n > p$, has rank p. Then we can write $X = QR$, where $Q \in \mathbb{R}^{n\times p}$ with $Q'Q = I_p$ and $R \in \mathbb{R}^{p\times p}$ is upper-triangular; $r_{ij} = 0$ when $i > j$, with non-zero diagonal entries. This is called the QR decomposition of X. Since R is upper-triangular, $|R| = \prod_{j=1}^{p} r_{jj} \neq 0$ meaning that R is invertible.

In R, we compute the reduced QR decomposition of X (created in section 2.1) with

```
out=qr(x=X)
```

We can see the matrix Q and R with

```
qr.Q(out)
```

```
##             [,1]       [,2]        [,3]       [,4]        [,5]
##  [1,] -0.3162278 -0.3162278 -0.35301254  0.2597321  0.39514473
##  [2,] -0.3162278 -0.3162278  0.45145378  0.2552218  0.29707365
##  [3,] -0.3162278 -0.3162278 -0.37589846 -0.4883609 -0.08600028
##  [4,] -0.3162278 -0.3162278 -0.01853430 -0.2834573 -0.11816381
##  [5,] -0.3162278 -0.3162278  0.29599153  0.2568643 -0.48805428
##  [6,] -0.3162278  0.3162278  0.18798743 -0.4969000  0.08101222
##  [7,] -0.3162278  0.3162278 -0.09970480  0.3996677  0.09072316
##  [8,] -0.3162278  0.3162278 -0.07453771  0.1460618 -0.61107576
##  [9,] -0.3162278  0.3162278  0.43550002 -0.1878271  0.29495280
## [10,] -0.3162278  0.3162278 -0.44924493  0.1389977  0.14438757
```

```
qr.R(out)
```

```
##                          v2
## [1,] -3.162278 -1.581139 -1.093070 -0.1305421  0.06342956
## [2,]  0.000000 -1.581139  1.712076 -0.7650413  0.55538693
## [3,]  0.000000  0.000000  3.205225 -0.9106723 -0.27851286
## [4,]  0.000000  0.000000  0.000000 -3.0436444  1.13863549
## [5,]  0.000000  0.000000  0.000000  0.0000000  4.14882954
```

To solve $X'X\hat{\beta} = X'Y$ with $X = QR$:

$$R'Q'QR\hat{\beta} = R'Q'Y$$

and since R is invertible (because R is upper-triangular and $|R| = \prod_{j=1}^{p} r_{jj} \neq 0$), we solve

$$R\hat{\beta} = Q'Y,$$

by backward substitution. This costs $O(np^2)$. This is done in R with

```
beta.hat.qr=backsolve(qr.R(out), crossprod(qr.Q(out), y))
beta.hat.qr
```

```
##             [,1]
## [1,]  1.0343738
## [2,] -0.7203269
## [3,] -0.7991497
## [4,] -0.1613902
## [5,]  0.3673047
```

We could compute the realization of $\hat{\beta}$ through the QR decomposition of X with fewer commands (and faster) by typing

```
qr.coef(qr(x=X), y=y)

##            [,1]
##       1.0343738
## v2   -0.7203269
##      -0.7991497
##      -0.1613902
##       0.3673047

# or equivalently

lm.fit(x=X,y=y)$coefficients

##                     v2
##   1.0343738 -0.7203269 -0.7991497 -0.1613902  0.3673047
```

2.3 Using the Cholesky decomposition

Suppose that $A \in \mathbb{S}^p_+$. Then A can be factored as $A = LL'$, where $L \in \mathbb{R}^{p\times p}$ is lower-triangular: $l_{ij} = 0$ when $i < j$, with positive diagonal entries. This is called the Cholesky decomposition of A. Since L is lower-triangular, $|L| = \prod_{j=1}^p l_{jj} > 0$ so L is invertible.

When $X \in \mathbb{R}^{n\times p}$ with rank p,one can show that $X'X \in \mathbb{S}^p_+$. To see this, for any $u \in \mathbb{R}^p$ for which $u \neq 0$, we have that $u'X'Xu = (Xu)'(Xu) = \|Xu\|^2 > 0$ because X has rank p.

To solve $X'X\beta = X'Y$, we decompose $X'X = LL'$ and solve $X'X\beta = X'Y$, we decompose $X'X = LL'$ and solve

$$Lz = X'Y,$$

by forward substitution. Then we solve $L'\beta = z$ by backward substitution.

We carry this out in `R` on X and y (created in section 2.1). To decompose $X'X = LL'$, we use the `chol` function, which returns $U = L'$:

```
U=chol(crossprod(X))
U

##                    v2
##     3.162278 1.581139  1.093070  0.1305421 -0.06342956
## v2 0.000000 1.581139 -1.712076  0.7650413 -0.55538693
##    0.000000 0.000000  3.205225 -0.9106723 -0.27851286
##    0.000000 0.000000  0.000000  3.0436444 -1.13863549
##    0.000000 0.000000  0.000000  0.0000000  4.14882954
```

Then we solve

$$U'z = X'Y,$$

by forward substitution:

```
z=forwardsolve(t(U), crossprod(X,y))
```

Then we solve $U\hat{\beta} = z$ by backward substitution:

```
beta.hat.chol=backsolve(U, z)
beta.hat.chol

##             [,1]
## [1,]  1.0343738
## [2,] -0.7203269
## [3,] -0.7991497
## [4,] -0.1613902
## [5,]  0.3673047
```

2.4 Illustration of the speed improvement

We illustrate the speed improvement by comparing the computing time of the slow closed-form solution formula to the faster methods. We will use a large n and p for emphasis.

```
set.seed(680)
n=1000
p=950
sigma.star=1/2
beta.star=rnorm(p)
X=cbind(1, matrix(rnorm(n*(p-1)), nrow=n, ncol=(p-1)))
y=X%*%beta.star + sigma.star*rnorm(n)

## compute betahat the fast way (using X=QR) with lm.fit
system.time(expr=(bhat1=lm.fit(x=X,y=y)$coefficients))
```

```
##    user  system elapsed
##   0.508   0.003   0.512

## compute betahat the fast way (using X=QR) with qr.coef
system.time(expr=(bhat2=qr.coef(qr(x=X), y=y)))

##    user  system elapsed
##   0.496   0.002   0.500

## compute betahat with the cholesky decomposition
chol.solve=function(X,y)
{
  U=chol(crossprod(X))
  z=forwardsolve(t(U), crossprod(X,y))
  return( backsolve(U, z) )
}
system.time(expr=(bhat3=chol.solve(X=X, y=y)))

##    user  system elapsed
##   0.627   0.000   0.627

## compute betahat the slow way (using the closed-form expression)
system.time(expr=(bhat4=(qr.solve(crossprod(X))%*%crossprod(X,y))))

##    user  system elapsed
##   2.018   0.034   2.053

## compute betahat solving the linear system by QR:
system.time(expr=(bhat5=(qr.solve(crossprod(X), crossprod(X,y)))))

##    user  system elapsed
##    0.91    0.00    0.91
```

Let's try this again with larger n:

```
set.seed(680)
n=5000
p=1000
sigma.star=1/2
beta.star=rnorm(p)
```

```
X=cbind(1, matrix(rnorm(n*(p-1)), nrow=n, ncol=(p-1)))
y=X%*%beta.star + sigma.star*rnorm(n)

## compute betahat the fast way (using X=QR) with lm.fit
system.time(expr=(bhat1=lm.fit(x=X,y=y)$coefficients))

##    user  system elapsed
##   3.982   0.016   4.007

## compute betahat the fast way (using X=QR) with qr.coef
system.time(expr=(bhat2=qr.coef(qr(x=X), y=y)))

##    user  system elapsed
##   3.966   0.041   4.010

## compute betahat with the cholesky decomposition
system.time(expr=(bhat3=chol.solve(X=X, y=y)))

##    user  system elapsed
##   2.883   0.018   2.902

## compute betahat the slow way (using the closed-form expression)
system.time(expr=(bhat4=(qr.solve(crossprod(X))%*%crossprod(X,y))))

##    user  system elapsed
##   4.351   0.046   4.408

## compute betahat solving the linear system by QR:
system.time(expr=(bhat5=(qr.solve(crossprod(X), crossprod(X,y)))))

##    user  system elapsed
##   3.201   0.016   3.217
```

3 Drawing from the multivariate normal distribution

3.1 Matrices and random matrices

- Let Y denote a $n \times p$ random matrix. Let $B \in \mathbb{R}^{m\times n}$, $C \in \mathbb{R}^{p\times q}$, and $D \in \mathbb{R}^{m\times q}$ be non-random. Then
$$E(BYC + D) = BE(Y)C + D.$$

- Let X denote a p-variate random vector and define $Y = BX + b$, where $B \in \mathbb{R}^{q\times p}$ and $b \in \mathbb{R}^q$ are non-random. Then
$$E(Y) = BE(X) + b,$$
$$var(Y) = Bvar(X)B',$$
where $var(Y)$ is the $q \times q$ covariance matrix of the random vector Y.

- Let $N_p(\mu, \Sigma)$ denote the p-variate normal distribution with mean vector $\mu \in \mathbb{R}^p$ and $p \times p$ symmetric and non-negative definite covariance matrix Σ. Suppose that $W \sim N_p(\mu, \Sigma)$, $B \in \mathbb{R}^{k\times p}$ is non-random, and $b \in \mathbb{R}^k$ is non-random. Then
$$Y = BW + b \sim N_k(B\mu + b, B\Sigma B').$$

3.2 Eigen-decomposition

For $A \in \mathbb{S}^p$ we can write $A = Q\Lambda Q'$ where $Q \in \mathbb{R}^{p\times p}$ is orthogonal matrix (hence $Q'Q = I_p$) and $\Lambda \in \mathbb{R}^{p\times p}$ is diagonal. The columns of Q are called the eigenvectors of A and the diagonal elements of Λ are called the eigenvalues of A and are ordered $\lambda_{11} \geq \lambda_{22} \geq \cdots \lambda_{pp}$.

The eigen-decomposition allows for much easier computation of power series of matrices. Suppose that $A \in \mathbb{S}_0^p$ with eigen-decomposition $A = Q\Lambda Q'$, then $\lambda_{11}, \dots, \lambda_{pp} \geq 0$. We have

$$A^x = Q\text{diag}(\lambda_{11}^x, \dots, \lambda_{pp}^x)Q',$$

where $\text{diag}(\lambda_{11}^x, \dots, \lambda_{pp}^x)$ is a diagonal matrix with (j, j)-th entry λ_{jj}^x and x is any real number.

For example, $A^{1/2} = Q\text{diag}(\lambda_{11}^{1/2}, \dots, \lambda_{pp}^{1/2})Q'$ so

$$\begin{aligned} A^{1/2}A^{1/2} &= Q\text{diag}(\lambda_{11}^{1/2}, \dots, \lambda_{pp}^{1/2})Q'Q\text{diag}(\lambda_{11}^{1/2}, \dots, \lambda_{pp}^{1/2})Q' \\ &= Q\text{diag}(\lambda_{11}^{1/2}, \dots, \lambda_{pp}^{1/2})\text{diag}(\lambda_{11}^{1/2}, \dots, \lambda_{pp}^{1/2})Q' \\ &= Q\text{diag}(\lambda_{11}, \dots, \lambda_{pp})Q' \\ &= A. \end{aligned}$$

If matrix A can be eigen-decomposed and if none of its eigenvalues are zero, then A is non-singular and its inverse is given by

$$A^{-1} = Q\Lambda^{-1}Q'.$$

3.3 Generating an iid sample of size n from $N_p(\mu, \Sigma)$

- Suppose that $\Sigma \in \mathbb{S}_0^p$ and z is an $n \times p$ random matrix, with all entries i.i.d $N(0,1)$. Then $X = 1_n\mu' + Z\Sigma^{1/2}$ has rows that are i.i.d $N_p(\mu,\Sigma)$.
  This is because of the following. The ith row vector of X is $X_i = \mu + \Sigma^{1/2}Z_i$. So $E(X_i) = \mu + \Sigma^{1/2}E(Z_i) = \mu$ and $var(X_i) = \Sigma^{1/2}var(Z_i)\Sigma^{1/2} = \Sigma^{1/2}I_p\Sigma^{1/2} = \Sigma$.

- Suppose that $\Sigma \in \mathbb{S}_+^p$ with Cholesky decomposition $\Sigma = LL'$, and Z is an $n \times p$ random matrix, with all entries i.i.d $N(0,1)$. Then $X = 1_n\mu' + ZL'$ has rows that are i.i.d $N_p(\mu,\Sigma)$. Following the same argument as before, the ith row vector of X is $X_i = \mu + LZ_i$ and $E(X_i) = \mu + LE(Z_i) = \mu$ and $var(X_i) = Lvar(Z_i)L' = LI_pL' = \Sigma$.

The following R code generates a realization of an $n \times p$ random matrix with its n rows i.i.d $N_p(\mu,\Sigma)$. We will pick Σ to have (j,k)-th entry $1(j=k) + 0.5 \cdot 1(j \neq k)$ and we will set $\mu = (1,\ldots,p)'$.

```r
set.seed(680)
n=1000; p=5

Sigma = matrix(0.5, nrow=p, ncol=p)
diag(Sigma)=1
mu=1:p

## generate a realization of the random matrix
## using the eigendecomposition method
eo=eigen(Sigma, symmetric=TRUE)
Sigma.sqrt=eo$vec %*% diag(eo$val^0.5)%*%t(eo$vec)
Z=matrix(rnorm(n*p), nrow=n, ncol=p)
X=rep(1,n)%*%t(mu) + Z%*%Sigma.sqrt

## estimate mu with the sample mean realization
xbar=apply(X, 2, mean)
xbar

## [1] 0.9803332 1.9726989 2.9381484 3.9600465 4.9664519

## estimate Sigma with the sample covariance matrix
## realization
S=cov(X)
S

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0460403 0.4852127 0.5222168 0.5101459 0.4508469
## [2,] 0.4852127 0.9392722 0.4970253 0.4849900 0.4783074
## [3,] 0.5222168 0.4970253 1.0426955 0.5582633 0.4623945
## [4,] 0.5101459 0.4849900 0.5582633 0.9802681 0.4844920
## [5,] 0.4508469 0.4783074 0.4623945 0.4844920 0.9183008

## generate a realization of the random matrix
## using the Cholesky decomposition method
U=chol(Sigma)
Xnew=rep(1,n)%*%t(mu) + Z%*%U

## estimate mu with the sample mean realization
xbar.new=apply(Xnew, 2, mean)
xbar.new

## [1] 1.002703 1.994342 2.952510 3.966844 4.970167

## estimate Sigma with the sample covariance matrix
## realization
S.new=cov(Xnew)
S.new

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0794735 0.5297842 0.5492376 0.5450073 0.4951460
## [2,] 0.5297842 0.9554028 0.4986473 0.4898781 0.4899431
## [3,] 0.5492376 0.4986473 1.0484492 0.5799561 0.4856854
## [4,] 0.5450073 0.4898781 0.5799561 1.0088662 0.5044167
## [5,] 0.4951460 0.4899431 0.4856854 0.5044167 0.9279081
```

4 Centered least squares

In our original setup $X \in \mathbb{R}^{n\times p}$ where all entries in its first column equal one to form an intercept. The/a least squares estimator is

$$\hat\beta = \operatorname*{arg\,min}_{\beta\in\mathbb{R}^p} \|Y - X\beta\|^2, \tag{2}$$

Define $X_{-1} \in \mathbb{R}^{n\times(p-1)}$ as the matrix X with its first column removed. Let $\bar Y = n^{-1}\sum_{i=1}^n Y_i$ and $\bar x' = n^{-1}1_n' X_{-1} = (n^{-1}\sum_{i=1}^n x_{i2}, \dots, n^{-1}\sum_{i=1}^n x_{ip})$. Let $\tilde Y = (Y_1 - \bar Y, \dots, Y_n - \bar Y)'$ and $\tilde X = X_{-1} - 1_n \bar x'$.

Proposition 1. *Suppose that*

$$\hat\beta_{-1} = \operatorname*{arg\,min}_{\beta\in\mathbb{R}^p} \|\tilde Y - \tilde X\beta\|^2,$$

$$\hat\beta_1 = \bar Y - \bar x'\hat\beta_{-1}.$$

Then $(\hat\beta_1, \hat\beta_{-1}')'$ is a global minimizer of (2).

You will prove Proposition 1 in homework 2 using optimality conditions introduced in the next chapter.

Here is an example in R.

```r
set.seed(680)
n=10;p=4
X=cbind(1, matrix(rnorm(n*(p-1)), nrow=n, ncol=(p-1)))
beta.star=rep(1,p)
y=X%*%beta.star + 1*rnorm(n)

## compute the ols estimate the old way
beta.hat=lm.fit(x=X, y=y)$coeff

## compute the ols estimate the new way
xbar=apply(X[,-1], 2, mean); ybar=mean(y)
Xtilde=scale(X[,-1], center=xbar, scale=FALSE); ytilde=y-ybar
beta.hat.minus1=lm.fit(x=Xtilde, y=ytilde)$coeff
beta.hat.1=ybar - sum(xbar * beta.hat.minus1)
beta.hat.new=c(beta.hat.1, beta.hat.minus1)

## compare the old and new solutions
cbind(beta.hat, beta.hat.new)
```

```
##     beta.hat beta.hat.new
## x1 1.1587941    1.1587941
## x2 1.0143784    1.0143784
## x3 0.3233243    0.3233243
## x4 0.7346096    0.7346096
```

4.1 The (reduced) singular value decomposition

The singular value decomposition (SVD) allows us to transform a matrix $X \in \mathbb{R}^{n\times p}$ to diagonal form using matrices with orthonormal columns, i.e.,

$$X = UDV'.$$

here $U \in \mathbb{R}^{n\times q}$ and $V \in \mathbb{R}^{p\times q}$ has orthonormal columns, $D \in \mathbb{R}^{q\times q}$ is diagonal square matrix. We call the columns of U the *left singular vectors* and the columns of V the *right singular vectors*. If U and V^* are real valued, they each are an orthogonal matrix. The diagonal **non-negative** elements of $D = \text{diag}(d_{11}, \dots, d_{qq})$ are called the singular values of D. This is the practical version of the SVD also known as the *reduced SVD*.

Note: If some diagonal entries are not non-negative, you can do the following: SVD factors X as orthogonal U times diagonal D times orthogonal V. Multiplying a row or a column of an orthogonal matrix by -1 still gives an orthogonal matrix, and you can do that and change a corresponding negative diagonal entry in D to a positive entry. In that way, the two orthogonal matrices can be chosen so that the diagonal entries in D matrix are all non-negative.

Calculating the SVD consists of finding the eigenvalues and eigenvectors of XX' and $X'X$. The eigenvectors of $X'X$ make up the columns of V. This is because if $X = UDV'$, $X'X = VDU'UDV' = VD^2V'$. The eigenvectors of XX' make up the columns of U. Also, the singular values are the diagonal entries of the D matrix, i.e. $d_{11}, \dots, d_{qq}$ in D are square roots of eigenvalues from XX' or $X'X$. They and are arranged in descending order. The singular values are always real numbers. If the matrix X is a real matrix, then U and V are also real.

We usually choose $q = \min(n, p)$, since we have the following results

Proposition 2. *Assume $X \in \mathbb{R}^{n\times p}$, $q = \min(n, p)$, and $r \leq q$ denotes the number of positive singular values of X. Then* $\text{rank}(X) = r$.

Therefore if $\text{rank}(X) = \min(n, p)$, choosing $q = \min(n, p)$ will ensure all the diagonal elements of D, i.e. $d_{11}, \dots, d_{qq}$ are positive.

Here is an example in R.

```
set.seed(680)
## compute the case when n>p
n=6; p=4
X=matrix(rnorm(n*p), nrow=n, ncol=p)
q=min(c(n,p))
out=svd(X, nu=q, nv=q)
D=diag(out$d)
D

##          [,1]     [,2]     [,3]     [,4]
## [1,] 4.220568 0.000000 0.000000 0.000000
## [2,] 0.000000 3.038866 0.000000 0.000000
## [3,] 0.000000 0.000000 1.693155 0.000000
## [4,] 0.000000 0.000000 0.000000 1.420011

out$u%*%D%*%t(out$v)

##            [,1]       [,2]       [,3]        [,4]
## [1,] -1.3272313  0.5674886  2.1119256 -0.02556505
## [2,]  1.2512640  0.6481548  1.1628304 -0.21459096
## [3,] -1.4005858  2.2829403 -0.7681466  1.83775992
## [4,] -0.2551534 -0.5528660  1.1405458  1.20168988
## [5,]  0.7529725 -0.1858451 -1.3262941 -1.00385996
## [6,]  1.4896069 -0.9047226 -0.5773271 -1.00352103

X

##            [,1]       [,2]       [,3]        [,4]
## [1,] -1.3272313  0.5674886  2.1119256 -0.02556505
## [2,]  1.2512640  0.6481548  1.1628304 -0.21459096
## [3,] -1.4005858  2.2829403 -0.7681466  1.83775992
## [4,] -0.2551534 -0.5528660  1.1405458  1.20168988
## [5,]  0.7529725 -0.1858451 -1.3262941 -1.00385996
## [6,]  1.4896069 -0.9047226 -0.5773271 -1.00352103

## compute the case when n<p
set.seed(680)
n=4; p=6
```

```
X=matrix(rnorm(n*p), nrow=n, ncol=p)
q=min(c(n,p))
out=svd(X, nu=q, nv=q)
D=diag(out$d)
D

##          [,1]     [,2]     [,3]     [,4]
## [1,] 4.326682 0.000000 0.000000 0.000000
## [2,] 0.000000 2.736586 0.000000 0.000000
## [3,] 0.000000 0.000000 2.139141 0.000000
## [4,] 0.000000 0.000000 0.000000 1.070558

out$u%*%D%*%t(out$v)

##            [,1]      [,2]       [,3]       [,4]        [,5]      [,6]
## [1,] -1.3272313 0.7529725  2.2829403  2.1119256 -1.32629411  1.837760
## [2,]  1.2512640 1.4896069 -0.5528660  1.1628304 -0.57732708  1.201690
## [3,] -1.4005858 0.5674886 -0.1858451 -0.7681466 -0.02556505 -1.003860
## [4,] -0.2551534 0.6481548 -0.9047226  1.1405458 -0.21459096 -1.003521

X

##            [,1]      [,2]       [,3]       [,4]        [,5]      [,6]
## [1,] -1.3272313 0.7529725  2.2829403  2.1119256 -1.32629411  1.837760
## [2,]  1.2512640 1.4896069 -0.5528660  1.1628304 -0.57732708  1.201690
## [3,] -1.4005858 0.5674886 -0.1858451 -0.7681466 -0.02556505 -1.003860
## [4,] -0.2551534 0.6481548 -0.9047226  1.1405458 -0.21459096 -1.003521
```

4.2 The full singular value decomposition

For any matrix $X \in \mathbb{R}^{n\times p}$, we can write $X = \hat{U}\hat{D}V'$ where $\hat{U} = \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{p\times p}$ are orthogonal and $\hat{D} = \mathbb{R}^{n\times p}$ is diagonal with non-negative entries. The number of positive diagonal entries in $\hat{D}$ is the rank of X. This is called the *full singular value decomposition*.

For example, when $\text{rank}(X) = q = \min(n, p)$. The full singular value decomposition extends U to an orthonormal basis of $\mathbb{R}^{n\times n}$ by adding appropriate orthogonal (but otherwise arbitrary) columns and call this new matrix $\hat{U}$. This will also force $D \in \mathbb{R}^{q\times q}$ to be extended to an $\mathbb{R}^{n\times p}$

matrix $\hat{D}$, i.e. if $n \geq p$, we can write

$$X = \hat{U}\hat{D}V' = \left[U, \tilde{U}\right]_{n\times n} \begin{bmatrix} D \\ 0 \end{bmatrix}_{n\times p} V',$$

and if $n < p$, we can write

$$X = \hat{U}\hat{D}V' = \left[U, \tilde{U}\right]_{n\times n} \begin{bmatrix} D & 0 \end{bmatrix}_{n\times p} V',$$

where $\hat{U} = \left[U, \tilde{U}\right] \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{p\times p}$ are orthonormal and D is diagonal with non-negative entries $d_{11}, \ldots, d_{qq}$.

Here is an example in R.

```
## compute the case when n>p
set.seed(680)
n=6; p=4
X=matrix(rnorm(n*p), nrow=n, ncol=p)
out=svd(X, nu=n, nv=p)
D=matrix(0, nrow=n, ncol=p)
diag(D)=out$d
D

##          [,1]     [,2]     [,3]     [,4]
## [1,] 4.220568 0.000000 0.000000 0.000000
## [2,] 0.000000 3.038866 0.000000 0.000000
## [3,] 0.000000 0.000000 1.693155 0.000000
## [4,] 0.000000 0.000000 0.000000 1.420011
## [5,] 0.000000 0.000000 0.000000 0.000000
## [6,] 0.000000 0.000000 0.000000 0.000000

out$u%*%D%*%t(out$v)

##            [,1]       [,2]       [,3]        [,4]
## [1,] -1.3272313  0.5674886  2.1119256 -0.02556505
## [2,]  1.2512640  0.6481548  1.1628304 -0.21459096
## [3,] -1.4005858  2.2829403 -0.7681466  1.83775992
## [4,] -0.2551534 -0.5528660  1.1405458  1.20168988
## [5,]  0.7529725 -0.1858451 -1.3262941 -1.00385996
## [6,]  1.4896069 -0.9047226 -0.5773271 -1.00352103
```

```
X

##             [,1]       [,2]       [,3]        [,4]
## [1,] -1.3272313  0.5674886  2.1119256 -0.02556505
## [2,]  1.2512640  0.6481548  1.1628304 -0.21459096
## [3,] -1.4005858  2.2829403 -0.7681466  1.83775992
## [4,] -0.2551534 -0.5528660  1.1405458  1.20168988
## [5,]  0.7529725 -0.1858451 -1.3262941 -1.00385996
## [6,]  1.4896069 -0.9047226 -0.5773271 -1.00352103

## compute the case when n<p
set.seed(680)
n=4; p=6
X=matrix(rnorm(n*p), nrow=n, ncol=p)
out=svd(X, nu=n, nv=p)
D=matrix(0, nrow=n, ncol=p)
diag(D)=out$d
D

##           [,1]     [,2]     [,3]     [,4] [,5] [,6]
## [1,] 4.326682 0.000000 0.000000 0.000000    0    0
## [2,] 0.000000 2.736586 0.000000 0.000000    0    0
## [3,] 0.000000 0.000000 2.139141 0.000000    0    0
## [4,] 0.000000 0.000000 0.000000 1.070558    0    0

out$u%*%D%*%t(out$v)

##             [,1]      [,2]       [,3]       [,4]        [,5]      [,6]
## [1,] -1.3272313 0.7529725  2.2829403  2.1119256 -1.32629411  1.837760
## [2,]  1.2512640 1.4896069 -0.5528660  1.1628304 -0.57732708  1.201690
## [3,] -1.4005858 0.5674886 -0.1858451 -0.7681466 -0.02556505 -1.003860
## [4,] -0.2551534 0.6481548 -0.9047226  1.1405458 -0.21459096 -1.003521

X

##             [,1]      [,2]       [,3]       [,4]        [,5]      [,6]
## [1,] -1.3272313 0.7529725  2.2829403  2.1119256 -1.32629411  1.837760
## [2,]  1.2512640 1.4896069 -0.5528660  1.1628304 -0.57732708  1.201690
## [3,] -1.4005858 0.5674886 -0.1858451 -0.7681466 -0.02556505 -1.003860
## [4,] -0.2551534 0.6481548 -0.9047226  1.1405458 -0.21459096 -1.003521
```

4.3 Relationship between SVD and PCA

Let the data matrix $X \in \mathbb{R}^{n \times p}$, where n is the number of samples and p is the number of variables. Let us assume that it is centered, i.e. column means have been subtracted and are now equal to zero. Then the $p \times p$ covariance matrix is $C = X'X/n$. It is a symmetric matrix and so it can be diagonalized:

$$C = VLV',$$

where V is a matrix of eigenvectors (each column is an eigenvector) and L is a diagonal matrix with eigenvalues λ_{jj} in the decreasing order on the diagonal.

- The eigenvectors V are called *principal directions* of the data.
- Projections of the data on the principal directions are called *principal components*, also known as PC scores, which can be seen as new, transformed, variables. The j-th principal component is given by j-th column of XV. The coordinates of the i-th data point in the new PC space are given by the i-th row of XV.

If we now perform singular value decomposition of X, we obtain a decomposition

$$X = UDV',$$

where C is the diagonal matrix of singular values d_{jj}. From here one can easily see that

$$C = VDU'UDV'/n = V\frac{D^2}{n}V'$$

meaning that right singular vectors V are principal directions and that singular values are related to the eigenvalues of covariance matrix via $\lambda_{jj} = d_{jj}^2/n$. Principal components are given by $XV = UDV'V = UD$.

4.4 Centered ordinary least squares when $X'X$ is not invertible

The centered ordinary least squares solves

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \|Y - X\beta\|^2.$$

Equivalently, we solve $X'X\beta = X'Y$. When $p > n$, for example. $\text{rank}(X) = n$ and $X'X$ is not invertible, in which case there are multiple solutions.

One solution uses a Moore–Penrose generalized inverse. A Moore–Penrose generalized inverse of a matrix $M \in \mathbb{R}^{n \times p}$ with rank q with reduced singular value decomposition $M = UDV' \in \mathbb{R}^{n \times p}$ is

$M^- = VD^{-1}U' \in \mathbb{R}^{p\times n}$, where $U \in \mathbb{R}^{n\times q}$, $V \in \mathbb{R}^{p\times q}$ and $D \in \mathbb{R}^{q\times q}$. If M is square and invertible, then $M^- = M^{-1}$. If M has full column rank, then $M^- = (M'M)^{-1}M'$.

Come back to our problem, we know that $q = \text{rank}(X) = \min(n-1, p)$. (note that it is not $\min(n,p)$ since X is centered column-wise). To solve $X'X\beta = X'Y$, decompose $X = UDV'$ using the reduced SVD, so

$$VD^2V'\beta = VDU'Y.$$

Choosing $\beta = VD^{-1}U'Y = X^-Y$ solves this equation. Note that $(X'X)^-X' = X^-$.

Here is an example in R.

```r
set.seed(680)
n=5;p=7
X=cbind(1, matrix(rnorm(n*p), nrow=n, ncol=p))
beta.star=rep(1,p+1)
y=X%*%beta.star + 1*rnorm(n)
xbar=apply(X[,-1], 2, mean); ybar=mean(y)
Xtilde=scale(X[,-1], center=xbar, scale=FALSE); ytilde=y-ybar
q=min(c(n-1, p))
out=svd(Xtilde, nu=q, nv=q)
beta.hat.minus1=out$v%*%diag(out$d[1:q]^(-1))%*%t(out$u)%*%ytilde
## confirm that the objective function value is zero at this
## global minimizer:
sum((Xtilde%*%beta.hat.minus1 - ytilde)^2)

## [1] 9.047249e-30
```

5 Case study: ridge regression

5.1 A ridge regression example: polynomial curve fitting

(A part of the text in this section is from the book "Pattern Recognition and Machine Learning" by Christopher M. Bishop.)

In this section, we are given $n = 30$ observations of training samples $\{y_i, x_i\}_{i=1}^n$, where $x_i \sim \text{Unif}(-2, 1)$ and $y_i = g(x_i) + \epsilon_i$ and $g(x) = x + x^2$.

We fit the data using a polynomial function of the form

$$f(x,\beta) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_M x^M = \beta_0 + \sum_{j=1}^{M} \beta_j x^j.$$

where M is the *order* of the polynomial, and x_j denotes x raised to the power of j. The polynomial coefficients $\beta_0, \ldots, \beta_M$ are collectively denoted by the vector β. The values of the coefficients will be determined by fitting the polynomial to the training data. This can be done by minimizing the sum of the squares that measures the misfit between y_i the function $f(x_i, \beta)$:

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} \{y_i - f(x_i, \beta)\}^2. \tag{3}$$

Let $z_1 = x$, ..., $z_M = x^M$, then (3) becomes the OLS with $z_1, \ldots, z_M$ as the predictors:

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} \{y_i - \beta_0 - \sum_{j=1}^{M} \beta_j z_j\}^2.$$

In Figure 1, we show four examples of the results of fitting polynomials having orders $M = 1$, 2, 9, and 17 to the data set $\{y_i, x_i\}_{i=1}^n$.

We notice that the constant first order ($M = 1$) polynomials give rather poor fits to the data and consequently rather poor representations of the function $g(x) = x + x^2$. The second order ($M = 2$) polynomial seems to give the best fit to the function $g(x)$. When we go to a much higher order polynomial ($M = 9$ and 17), we obtain an almost excellent fit to the training data. The polynomial passes through many data points. However, the fitted curve oscillates wildly and gives a very poor representation of the function $g(x)$. This latter behavior is known as *over-fitting.*

We see that, as M increases, the magnitude of the coefficients typically gets larger. In particular for the $M = 17$ polynomial, the coefficients have become finely tuned to the data by developing large positive and negative values so that the corresponding polynomial function matches each of the data points exactly, but between data points (particularly near the ends of the range) the function exhibits the large oscillations observed in Figure 1. Intuitively, what is happening is that the more flexible polynomials with larger values of M are becoming increasingly tuned to the random noise on the target values.

Ridge regression is often used to control the over-fitting phenomenon in such cases, by adding a penalty term to the error function (3) in order to discourage the coefficients from reaching large values:

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \sum_{i=1}^{n} \{y_i - f(x_i, \beta)\}^2 + \lambda \|\beta\|^2, \tag{4}$$

and the coefficient λ governs the relative importance of the regularization term compared with the sum-of-squares error term.

Figure 2 shows the results of fitting the polynomial of order $M = 17$ to the same data set as before but now using the ridge regularized error function given by (4) for $\lambda \in \{100, 5, 10^{-3}, 0\}$. We see that, for a value of $\lambda = 10^{-3}$, the over-fitting has been suppressed and we now obtain a much closer representation of the underlying function $g(x) = x + x^2$. If, however, we use too large a value for λ then we again obtain a poor fit, as shown in Figure 2 for $\lambda = 100$.

```
# example of ridge regression
n = 30
p = 3
x=sort(runif(n,-2,1))
sigma.star=0.3
y=x + x^2 + sigma.star * rnorm(n)

## fit model1: OLS
m1 <- lm(y~x)
summary(m1)

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1247 -0.5518 -0.1267  0.3604  1.6796
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.62851    0.14845   4.234 0.000224 ***
## x            0.09097    0.15099   0.603 0.551679
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7336 on 28 degrees of freedom
## Multiple R-squared:  0.0128,Adjusted R-squared:  -0.02246
## F-statistic: 0.363 on 1 and 28 DF,  p-value: 0.5517

## fit model2: order 2 polynomials
m2 <- lm(y~x+I(x^2))
summary(m2)

##
## Call:
```

```
## lm(formula = y ~ x + I(x^2))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.66888 -0.18998  0.03102  0.12575  0.58689
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.05538    0.07295   0.759    0.454
## x            0.87084    0.08468  10.283 7.77e-11 ***
## I(x^2)       0.93500    0.07368  12.690 6.83e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2831 on 27 degrees of freedom
## Multiple R-squared:  0.8582,Adjusted R-squared:  0.8477
## F-statistic: 81.74 on 2 and 27 DF,  p-value: 3.513e-12
```

```
## fit model3: order 17 polynomials
m3 <- lm(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)
                        +I(x^6)+I(x^7)+I(x^8)+I(x^9))
summary(m3)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) +
##     I(x^7) + I(x^8) + I(x^9))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46358 -0.11233 -0.02761  0.15191  0.43359
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.1108     0.2084   0.532   0.6009
## x             1.8236     0.6602   2.762   0.0120 *
## I(x^2)        0.1784     1.6741   0.107   0.9162
## I(x^3)       -6.1314     3.6112  -1.698   0.1050
## I(x^4)        0.5638     4.6291   0.122   0.9043
## I(x^5)       10.4095     5.8318   1.785   0.0894 .
## I(x^6)        3.0264     6.0054   0.504   0.6198
## I(x^7)       -4.9414     2.5618  -1.929   0.0681 .
## I(x^8)       -2.9969     3.1822  -0.942   0.3575
## I(x^9)       -0.4301     0.8425  -0.511   0.6152
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2578 on 20 degrees of freedom
## Multiple R-squared:  0.9129,Adjusted R-squared:  0.8737
## F-statistic: 23.29 on 9 and 20 DF,  p-value: 1.031e-08
```

```
## fit model2: order 17 polynomials
m4 <- lm(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)
                        +I(x^6)+I(x^7)+I(x^8)+I(x^9)
                        +I(x^10)+I(x^11)+I(x^12)+I(x^13)
                        +I(x^14)+I(x^15)+I(x^16)+I(x^17))
summary(m4)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) +
##     I(x^7) + I(x^8) + I(x^9) + I(x^10) + I(x^11) + I(x^12) +
##     I(x^13) + I(x^14) + I(x^15) + I(x^16) + I(x^17))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35358 -0.05849  0.00010  0.12990  0.39694
##
## Coefficients: (1 not defined because of singularities)
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)     2.160      1.989   1.086   0.2972
## x               4.358      2.271   1.919   0.0772 .
## I(x^2)        -50.768     46.599  -1.089   0.2957
## I(x^3)        -90.575     55.130  -1.643   0.1244
## I(x^4)        370.536    326.976   1.133   0.2776
## I(x^5)        743.625    442.924   1.679   0.1170
## I(x^6)      -1032.240    963.163  -1.072   0.3033
## I(x^7)      -2703.019   1612.130  -1.677   0.1175
## I(x^8)        753.996   1149.288   0.656   0.5232
## I(x^9)       4649.588   2832.740   1.641   0.1247
## I(x^10)      1432.449    799.484   1.792   0.0965 .
## I(x^11)     -3427.246   2197.285  -1.560   0.1428
## I(x^12)     -2620.695   1389.473  -1.886   0.0818 .
## I(x^13)       393.766    385.247   1.022   0.3254
## I(x^14)      1094.689    628.467   1.742   0.1051
## I(x^15)       436.475    234.391   1.862   0.0853 .
## I(x^16)        57.469     29.779   1.930   0.0757 .
## I(x^17)            NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2556 on 13 degrees of freedom
## Multiple R-squared:  0.9443,Adjusted R-squared:  0.8758
## F-statistic: 13.79 on 16 and 13 DF,  p-value: 1.216e-05
```

```
## fit ridge regression: order M=17 polynomials
## using different lambda values.

library(MASS)
rm1=lm.ridge(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)
                        +I(x^6)+I(x^7)+I(x^8)+I(x^9)
                        +I(x^10)+I(x^11)+I(x^12)+I(x^13)
                        +I(x^14)+I(x^15)+I(x^16)+I(x^17), lambda=5)

rm2=lm.ridge(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)
                        +I(x^6)+I(x^7)+I(x^8)+I(x^9)
                        +I(x^10)+I(x^11)+I(x^12)+I(x^13)
                        +I(x^14)+I(x^15)+I(x^16)+I(x^17), lambda=1e-3)

rm3=lm.ridge(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)
                        +I(x^6)+I(x^7)+I(x^8)+I(x^9)
                        +I(x^10)+I(x^11)+I(x^12)+I(x^13)
```

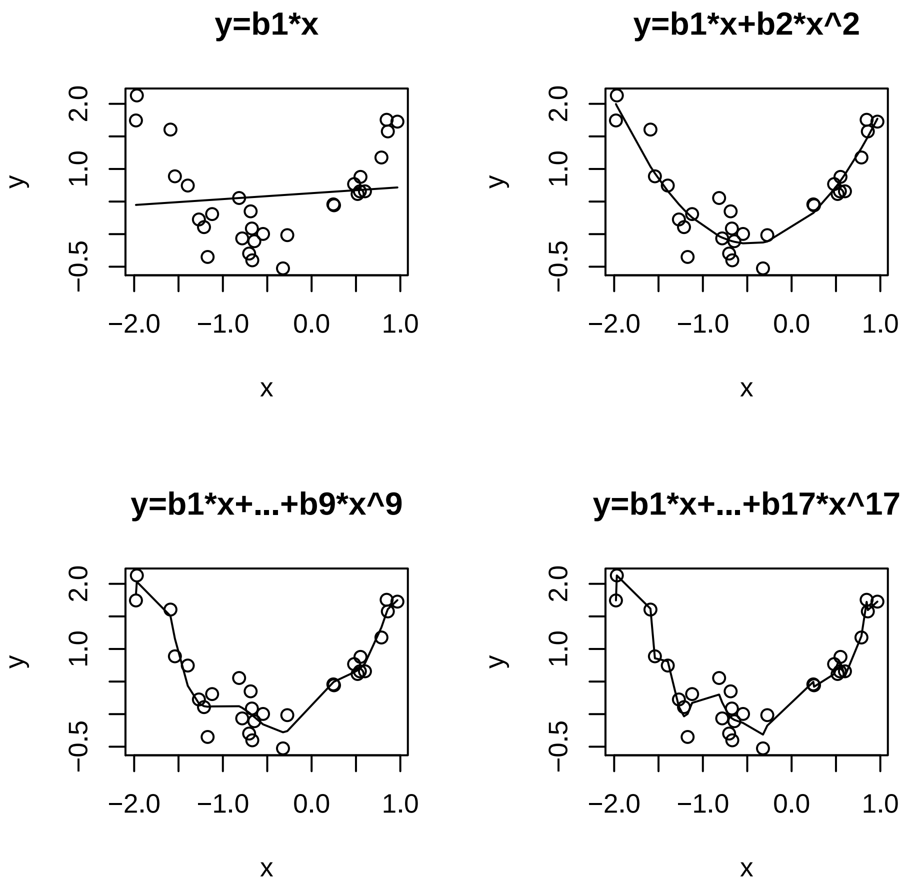

Figure 1: Plots of polynomials having various orders $M = 1$, 2, 9, and 17, shown as solid curves.

```
                           +I(x^14)+I(x^15)+I(x^16)+I(x^17), lambda=1e-10)

rm4=lm.ridge(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)
                           +I(x^6)+I(x^7)+I(x^8)+I(x^9)
                           +I(x^10)+I(x^11)+I(x^12)+I(x^13)
                           +I(x^14)+I(x^15)+I(x^16)+I(x^17), lambda=0)
```

Note: For the following sections, we use notation X to represent $\tilde{X}$, use Y to represent $\tilde{Y}$ for simplicity. Therefore X and Y have been centered: each x_{ij} gets replaced by $x_{ij} - \bar{x}_j$ and y_i gets replaced by $y_i - \bar{y}$. And we consider the case when there are n observations and p non-intercept predictors. Hence $X \in \mathbb{R}^{n\times p}$ and $Y \in \mathbb{R}^n$.

5.2 Computing the ridge penalized least squares estimator with the singular value decomposition

In this section, we will use the linear regression cases model defined in section 1.3. The ridge penalized least squares estimator of $\hat{\beta}^{(\lambda)}$ is defined by

$$\hat{\beta}^{(\lambda)} = \arg\min_{\beta\in\mathbb{R}^p} \left\{ \|Y - X\beta\|^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} = \arg\min_{\beta\in\mathbb{R}^p} \|Y - X\beta\|^2 + \lambda\|\beta\|_2^2, \tag{5}$$

where $\lambda \geq 0$ is a tuning parameter controlling the shrinkage of β. Let f be the objective function of (5). Since f is convex and differentiable, we solve $\nabla f(\beta^{(\lambda)}) = 0$ for $\beta^{(\lambda)}$, which is

$$\begin{aligned} -2X'Y + 2X'X\beta^{(\lambda)} + 2\lambda\beta^{(\lambda)} &= 0 \\ (X'X + \lambda I_p)\beta^{(\lambda)} &= X'Y \qquad (6) \\ \beta^{(\lambda)} &= (X'X + \lambda I_p)^{-1}X'Y, \qquad (7) \end{aligned}$$

The ridge regression solution is again a linear function of Y. The solution adds a positive constant to the diagonal of $X'X$ before inversion. This makes the problem non-singular, even if $X'X$ is not of full rank, and was the main motivation for ridge regression when it was first introduced in statistics.

Computationally, we could either solve (6) with our favorite linear system solver, or we could use the less efficient closed-form solution in (7). But both methods have to be recomputed for a different value of λ, this is computationally expensive when p is large and inefficient if we wish to compute $\beta^{(\lambda)}$ for multiple values of λ. Since we will want to select the best value for λ, it is more

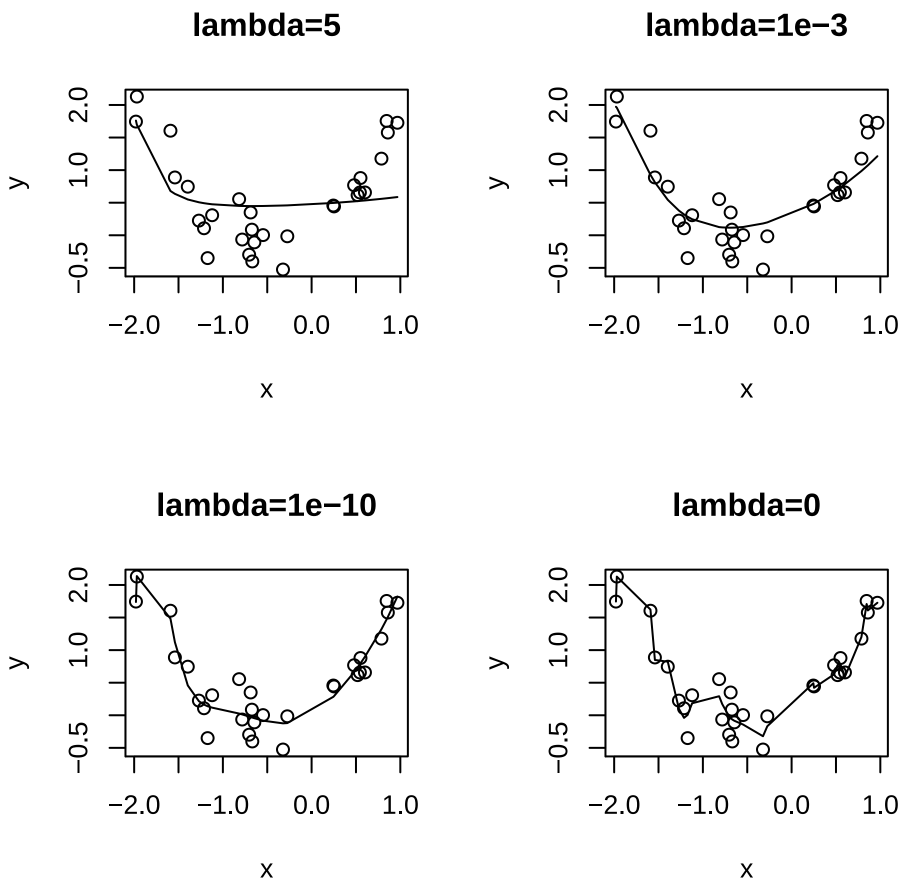

Figure 2: Plots of polynomials ($M = 17$) with different ridge penalty $\lambda \in \{100, 5, 10^{-3}, 0\}$, shown as solid curves.

efficient to take advantage of the singular value decomposition. We continue by deriving a fast way to compute $\beta^{(\lambda)}$.

Suppose that X has rank q (typically $q = \min(n-1, p)$). Using the full singular value decomposition of $X = UDV'$, where $U = \mathbb{R}^{n\times n}$, $V \in \mathbb{R}^{p\times p}$ and $D = \mathbb{R}^{n\times p}$. We have

$$X'X = VD'U'UDV' = VD'DV'$$

and

$$X' = VD'U'.$$

So write (6) as

$$(VD'DV' + \lambda I_p)\beta^{(\lambda)} = VD'U'Y$$

and replacing I_p with VV' gives

$$V(D'D + \lambda I_p)V'\beta^{(\lambda)} = VD'U'Y.$$

We find that $V(D'D + \lambda I_p)V'$ is the eigen decomposition of $X'X + \lambda I_p$, which is in $\mathbb{S}^p_+$ if $\lambda > 0$ or $X'X \in \mathbb{S}^p_+$ (so it's invertible).

Assuming that $\lambda > 0$,

$$\begin{aligned}\beta^{(\lambda)} &= V(D'D + \lambda I_p)^{-1}V'VD'U'Y \\ &= V(D'D + \lambda I_p)^{-1}D'U'Y \\ &= VMU'Y\end{aligned}$$

where the matrix $M = (D'D + \lambda I_p)^{-1}D' \in \mathbb{R}^{p\times n}$ is diagonal where $m_{jj} = d_{jj}/(d_{jj}^2 + \lambda)$ for $j = 1, \ldots, \min(n-1, p)$.

We can avoid multiplication by zero using the reduced SVD. Suppose that X has rank q (typically $q = \min(n-1, p)$) and decompose $X = UDV'$, where $U \in \mathbb{R}^{n\times q}$, $V \in \mathbb{R}^{p\times q}$ and $D \in \mathbb{R}^{q\times q}$. One can prove that

$$\beta^{(\lambda)} = VHU'Y$$

where $H \in \mathbb{R}^{q\times q}$ is diagonal with $h_{jj} = d_{jj}/(d_{jj}^2 + \lambda)$, for $j = 1, \ldots, q$. We see that

$$\lim_{\lambda\to 0^+} \beta^{(\lambda)} = VD^{-1}U'Y = X^{-}Y = \hat{\beta}^{OLS}.$$

So the Moore-Penrose generalized inverse solution is the edge-case of ridge penalized least squares.

Here is an example in R where we generate some data and compute the corresponding realization of $\hat{\beta}^{(\lambda)}$ for $\lambda \in \{10^{-8}, 10^{-7.5}, \ldots, 10^{7.5}, 10^{8}\}$.

```
## generate and center the data
set.seed(680)
n=5; p=10
X=cbind(1, matrix(rnorm(n*p), nrow=n, ncol=p))
beta.star=rep(1,p+1)
y=X%*%beta.star + 1*rnorm(n)
xbar=apply(X[,-1], 2, mean); ybar=mean(y)
Xtilde=scale(X[,-1], center=xbar, scale=FALSE); ytilde=y-ybar
q=min(c(n-1, p))

## compute ridge using full SVD

out=svd(Xtilde, nu=n, nv=p)

lam.vec=10^seq(from=-8, to=8, by=0.5)
beta.hat.matrix1=matrix(NA, nrow=p+1, ncol=length(lam.vec))
for(j in 1:length(lam.vec))
{
  djj=out$d[1:q]
  D=matrix(0,n,p)
  diag(D)[1:q]=djj
  H=matrix(0,p,p)
  diag(H)=1/diag(t(D)%*%D + diag(lam.vec[j],p))
  M=H%*%t(D)
  bhatm1=drop(out$v%*%M%*%t(out$u)%*%ytilde)
  bhat1=ybar - sum(xbar*bhatm1)
  beta.hat.matrix1[,j]=c(bhat1, bhatm1)
}

## compute ridge using reduced SVD

out=svd(Xtilde, nu=q, nv=q)

lam.vec=10^seq(from=-8, to=8, by=0.5)
beta.hat.matrix2=matrix(NA, nrow=p+1, ncol=length(lam.vec))
for(j in 1:length(lam.vec))
{
  H=diag(out$d[1:q]/(out$d[1:q]^2 + lam.vec[j]))
  bhatm1=drop(out$v%*%H%*%t(out$u)%*%ytilde)
  bhat1=ybar - sum(xbar*bhatm1)
  beta.hat.matrix2[,j]=c(bhat1, bhatm1)
}

## look at all of the estimates
display=round(beta.hat.matrix2, 4)
colnames(display)=log10(lam.vec)
display

##            -8    -7.5      -7    -6.5      -6    -5.5      -5    -4.5
##  [1,]  1.4420  1.4420  1.4420  1.4420  1.4420  1.4420  1.4420  1.4420
##  [2,]  0.6800  0.6800  0.6800  0.6800  0.6800  0.6800  0.6800  0.6800
##  [3,]  1.0754  1.0754  1.0754  1.0754  1.0754  1.0754  1.0754  1.0754
##  [4,] -0.2565 -0.2565 -0.2565 -0.2565 -0.2565 -0.2565 -0.2565 -0.2565
##  [5,]  0.0944  0.0944  0.0944  0.0944  0.0944  0.0944  0.0944  0.0944
##  [6,] -0.1089 -0.1089 -0.1089 -0.1089 -0.1089 -0.1089 -0.1089 -0.1089
##  [7,]  1.7489  1.7489  1.7489  1.7489  1.7489  1.7489  1.7489  1.7489
##  [8,]  0.4204  0.4204  0.4204  0.4204  0.4204  0.4204  0.4204  0.4204
##  [9,]  0.2177  0.2177  0.2177  0.2177  0.2177  0.2177  0.2177  0.2177
## [10,] -0.0625 -0.0625 -0.0625 -0.0625 -0.0625 -0.0625 -0.0625 -0.0625
## [11,]  0.2408  0.2408  0.2408  0.2408  0.2408  0.2408  0.2408  0.2408
##            -4    -3.5      -3    -2.5      -2    -1.5      -1    -0.5
##  [1,]  1.4420  1.4420  1.4422  1.4427  1.4443  1.4493  1.4650  1.5116
##  [2,]  0.6799  0.6799  0.6799  0.6797  0.6793  0.6778  0.6733  0.6598
##  [3,]  1.0753  1.0753  1.0751  1.0746  1.0730  1.0679  1.0521  1.0048
##  [4,] -0.2565 -0.2565 -0.2565 -0.2566 -0.2569 -0.2579 -0.2609 -0.2696
##  [5,]  0.0944  0.0944  0.0944  0.0944  0.0944  0.0945  0.0944  0.0940
##  [6,] -0.1089 -0.1089 -0.1089 -0.1088 -0.1088 -0.1086 -0.1082 -0.1066
##  [7,]  1.7489  1.7488  1.7487  1.7482  1.7466  1.7416  1.7261  1.6792
##  [8,]  0.4204  0.4204  0.4204  0.4204  0.4206  0.4209  0.4220  0.4250
##  [9,]  0.2177  0.2177  0.2177  0.2176  0.2172  0.2160  0.2124  0.2014
## [10,] -0.0625 -0.0625 -0.0625 -0.0625 -0.0625 -0.0625 -0.0624 -0.0619
## [11,]  0.2408  0.2408  0.2408  0.2408  0.2409  0.2412  0.2420  0.2445
##             0     0.5       1     1.5       2     2.5       3     3.5
##  [1,]  1.6358  1.8874  2.2002  2.4215  2.5247  2.5634  2.5765  2.5807
##   [2,]  0.6231  0.5429  0.4128  0.2513  0.1150  0.0425  0.0142  0.0046
##   [3,]  0.8776  0.6153  0.2886  0.0825  0.0163  0.0030  0.0007  0.0002
##   [4,] -0.2907 -0.3202 -0.3094 -0.2160 -0.1041 -0.0392 -0.0132 -0.0042
##   [5,]  0.0903  0.0706  0.0256 -0.0072 -0.0097 -0.0045 -0.0016 -0.0005
##   [6,] -0.1010 -0.0841 -0.0519 -0.0229 -0.0085 -0.0029 -0.0009 -0.0003
##   [7,]  1.5497  1.2630  0.8362  0.4359  0.1815  0.0645  0.0213  0.0068
##   [8,]  0.4304  0.4271  0.3728  0.2454  0.1156  0.0432  0.0145  0.0047
##   [9,]  0.1715  0.1084  0.0307 -0.0086 -0.0100 -0.0045 -0.0016 -0.0005
##  [10,] -0.0584 -0.0418 -0.0062  0.0162  0.0129  0.0056  0.0020  0.0006
##  [11,]  0.2504  0.2576  0.2405  0.1686  0.0821  0.0311  0.0105  0.0034
##              4     4.5       5    5.5      6    6.5      7    7.5      8
##   [1,]  2.5821  2.5825  2.5826 2.5827 2.5827 2.5827 2.5827 2.5827 2.5827
##   [2,]  0.0015  0.0005  0.0001 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##   [3,]  0.0001  0.0000  0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##   [4,] -0.0014 -0.0004 -0.0001 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##   [5,] -0.0002 -0.0001  0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##   [6,] -0.0001  0.0000  0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##   [7,]  0.0022  0.0007  0.0002 0.0001 0.0000 0.0000 0.0000 0.0000 0.0000
##   [8,]  0.0015  0.0005  0.0001 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##   [9,] -0.0002 -0.0001  0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##  [10,]  0.0002  0.0001  0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
##  [11,]  0.0011  0.0003  0.0001 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

```r
## compare the fast and slow methods to compute the
## solution when lambda=10^(-4)
H=diag(out$d[1:q]/(out$d[1:q]^2 + 10^(-4)))
bhatm1=out$v%*%H%*%t(out$u)%*%ytilde
bhat1=ybar - sum(xbar*bhatm1)
bhat.fast=c(bhat1, bhatm1)

## the slow method uses the solution from equation (3):
M=diag(c(0, rep(1,p)))
bhat.slow=qr.solve(crossprod(X) + 10^(-4)*M, crossprod(X,y))
res=cbind(bhat.slow, bhat.fast)
colnames(res)=c("bhat.slow","bhat.fast")
res
```

```
##          bhat.slow   bhat.fast
##  [1,]  1.44198157  1.44198157
##  [2,]  0.67994944  0.67994944
##  [3,]  1.07534770  1.07534770
##  [4,] -0.25646676 -0.25646676
##  [5,]  0.09444195  0.09444195
##  [6,] -0.10886480 -0.10886480
##  [7,]  1.74888048  1.74888048
##  [8,]  0.42038601  0.42038601
##  [9,]  0.21774106  0.21774106
## [10,] -0.06249465 -0.06249465
## [11,]  0.24079322  0.24079322
```

We plot the solution paths of the ridge regression in Figure 3.

5.3 Relationship between ridge and OLS

(This section is from the book “The Element of Statistical Learning” by Trevor Hastie, Robert Tibshirani and Jerome Friedman)

The SVD of the centered input matrix X gives us some additional insight into the nature of ridge regression. Using the singular value decomposition we can write the least squares fitted vector as

$$\begin{aligned} X\hat{\beta}^{OLS} &= X(X'X)^{-1}X'Y \\ &= UDV'(VDU'UDV')^{-1}VDU'Y \\ &= UU'Y \end{aligned}$$

after some simplification. Note that $U'Y$ are the coordinates of Y with respect to the orthonormal basis U.

Now the ridge solutions are

$$\begin{aligned} X\hat{\beta}^{ridge} &= X(X'X+\lambda I)^{-1}X'Y \\ &= UD(D^2+\lambda I)^{-1}DU'Y \\ &= \sum_{j=1}^{p} \mathbf{u}_j \frac{d_{jj}^2}{d_{jj}^2+\lambda}\mathbf{u}_j'Y \end{aligned}$$

where the $\mathbf{u}_j$ are the columns of $\mathbf{U}$. Note that since $\lambda \geq 0$, we have $d_{jj}^2/(d_{jj}^2+\lambda) \leq 1$. Like linear regression, ridge regression computes the coordinates of Y with respect to the orthonormal basis U.

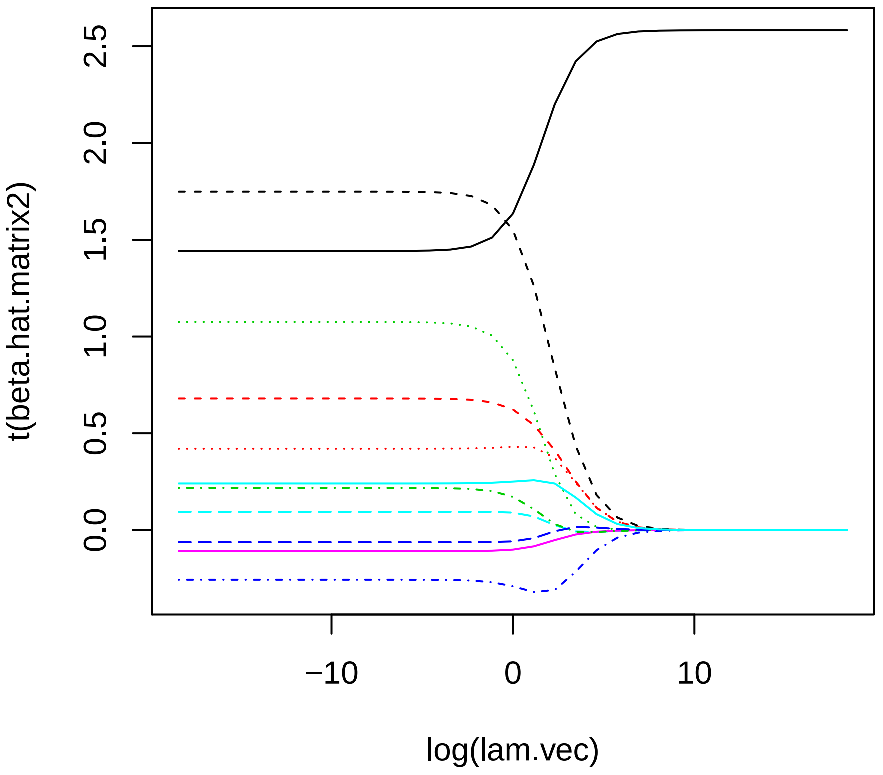

Figure 3: Solution paths of the ridge regression for $\lambda \in \{10^{-8}, 10^{-7.5}, \ldots, 10^{7.5}, 10^{8}\}$.

It then shrinks these coordinates by the factors $d_{jj}^2/(d_{jj}^2+\lambda)$. This means that a greater amount of shrinkage is applied to the coordinates of basis vectors with smaller d_{jj}^2.

What does a small value of d_{jj}^2 mean? The SVD of the centered matrix X is another way of expressing the principal components of the variables in X. We get eigen-decomposition of the sample covariance matrix $X'X/n$

$$X'X/n = VD'U'UDV'/n = VD'DV'/n$$

The first principal component direction v_1 (v_1 is the first column of eigenvectors V) has the property that $z_1 = Xv_1$ has the largest sample variance amongst all normalized linear combinations of the columns of X. This sample variance is easily seen to be

$$\begin{aligned} Var(z_1) &= Var(Xv_1) = v_1'Var(X)v_1 \\ &= v_1'X'Xv_1/n \\ &= v_1'VD'DV'v_1/n \\ &= \frac{d_{11}^2}{n} \end{aligned}$$

and in fact

$$z_1 = Xv_1 = \mathbf{u}_1 d_{11}$$

The derived variable z_1 is called the first principal component of X, and hence $\mathbf{u}_1$ is the normalized first principal component. Subsequent principal components z_j have maximum variance d_{jj}^2/N, subject to being orthogonal to the earlier ones. Conversely the last principal component has minimum variance. Hence the small singular values d_{jj} correspond to directions in the column space of X having small variance, and ridge regression shrinks these directions the most.

Figure 4 illustrates the principal components of some data points in two dimensions. If we consider fitting a linear surface over this domain (the Y-axis is sticking out of the page), the configuration of the data allow us to determine its gradient more accurately in the long direction than the short. Ridge regression protects against the potentially high variance of gradients estimated in the short directions. The implicit assumption is that the response will tend to vary most in the directions of high variance of the inputs. This is often a reasonable assumption, since predictors are often chosen for study because they vary with the response variable, but need not hold in general.

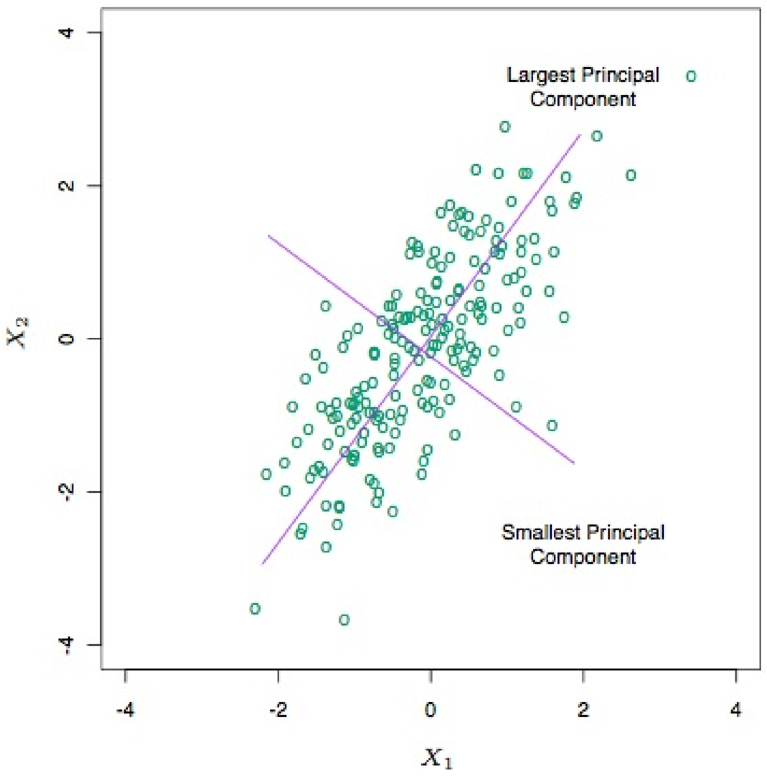

Figure 4: Principal components of some input data points. The largest principal component is the direction that maximizes the variance of the projected data, and the smallest principal component minimizes that variance. Ridge regression projects Y onto these components, and then shrinks the coefficients of the low–variance components more than the high-variance components.