Handout 05: Solving the Normal Equations

2 Solving least squares with the singular value decomposition

Many numerical methods for solving the normal equations rely on decomposing the data matrix X or the Gram matrix X^tX into factors using a variety of standard matrix decomposition algorithms. Here, we specifically make use of the singular value decomposition (SVD). Although more computationally intensive than some other techniques, the SVD gives us detailed insight into properties of the matrix useful for the development of numerically stable solution methods.

Let $X \in \mathbb{R}^{n\times p}$ and let $k = \min\{n, p\}$. Then there exist matrices $U \in \mathbb{R}^{n\times k}$ and $V \in \mathbb{R}^{p\times k}$ with orthonormal columns $U^tU = V^tV = I$ such that

$$U^tXV = \Sigma, \tag{2.1}$$

where Σ is a $k \times k$ diagonal matrix with non-negative entries $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$ along its main diagonal. This is sometimes called the *thin* SVD. In the case where $n > p$ it is possible to extend the matrix U into a square orthonormal $n \times n$ matrix $\bar{U}$ by adding $n - p$ additional orthonormal columns. Similarly, when $n < p$ we can extend V to a square orthonormal $p \times p$ matrix $\bar{V}$. The extended version is sometimes called the *full* SVD or just the SVD in many references and $\bar{U}^TX\bar{V} = \bar{\Sigma}$ results in an $n \times p$ rectangular diagonal matrix with the same main diagonal entries $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$ as the thin version. The full SVD is especially useful for analysis, while the thin SVD is more commonly used in computation.

The columns of $\bar{U}$ are called the *left singular vectors* of X and the columns of $\bar{V}$ are called the *right singular vectors*. The σ_i are called *singular values* of X. The SVD breaks matrix vector multiplication into three steps: rotation, scaling, then another rotation. Consider an $n \times p$ matrix X and its product y with a vector $b \in \mathbb{R}^p$. Using the full SVD $y = Xb = \bar{U}\bar{\Sigma}\bar{V}^Tb$:

1. Let $\hat{b} = \bar{V}^Tb \in \mathbb{R}^p$. Since $\bar{V}$ is an orthonormal matrix, $\hat{b}$ is simply a rotation of the vector b.
2. Now let $s = \bar{\Sigma}\hat{b} \in \mathbb{R}^n$, scaling each entry of $\hat{b}$ by the corresponding σ_i.
3. Finally let $y = \bar{U}s$, simply another rotation by the orthonormal matrix $\bar{U}$.

The SVD reveals a lot of information about the structure of the matrix X. Step 2 tells us how much a vector can be scaled by X, and together with the rotations in

steps 1 and 3 about its range and null space. The number of nonzero singular values of X is equal to the *rank* of X – the dimension of the range of X (range means the set of all linear combinations of the columns of X, that is, the span of the columns of X). Section **??** llustrates the sensitivity to noise of the solution of least squares problems involving X in terms of the singular values of X.

The SVD can be used to solve general ordinary least squares problems. The following result is adapted from Golub and Van Loan [1, Theorem 5.5.1], a recipe for computing the *unique* ordinary least squares solution of minimal Euclidean norm. Let X be a real $n \times p$ matrix, with full SVD $\bar{U}^T X \bar{V} = \bar{\Sigma}$ using extended matrices $\bar{U} = [u_1, u_2, \ldots, u_n] \in \mathbb{R}^{n \times n}$, $\bar{\Sigma} \in \mathbb{R}^{n \times p}$, and $\bar{V} = [v_1, v_2, \ldots, v_p] \in \mathbb{R}^{p \times p}$, and let $r \leq \min\{n, p\}$ be the rank of X. Then

$$b_{LS} = \sum_{i=1}^{r} \frac{u_i^T y}{\sigma_i} v_i \tag{2.2}$$

minimizes $\|Xb - y\|^2$ and has the smallest Euclidean norm of all such minimizers.

The proof of the above statement relies on properties of the orthogonal matrices produced by the SVD. For any vector $b \in \mathbb{R}^p$,

$$\begin{aligned}
\|Xb - y\|^2 &= \|\bar{U}\bar{\Sigma}\bar{V}^T b - y\|^2 && \text{(replacing X with its full SVD)} \\
&= \|\bar{U}^T(\bar{U}\bar{\Sigma}\bar{V}^T b - y)\|^2 && \text{(by **??**)} \\
&= \|\bar{\Sigma}\bar{V}^T b - \bar{U}^T y\|^2 \\
&= \sum_{i=1}^{p} (\sigma_i v_i^T b - u_i^T y)^2 + \sum_{i=p+1}^{n} (u_i^T y)^2. && (2.3)
\end{aligned}$$

The columns of $\bar{V}$ for an orthonormal basis of $\mathbb{R}^p$. Express the solution b as a linear combination of the column vectors v_i, $b = \sum_{i=1}^{p} \gamma_i v_i$ (that is, $\gamma_i = v_i^T b$). Since $\text{rank}(X) = r$ then $\sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_p = 0$ and the corresponding coefficients γ_i may take on any value without affecting the residual norm. The specific choice $\gamma_{r+1} = \gamma_{r+2} = \cdots \gamma_p = 0$ minimizes the norm of any possible solution b. Then the residual norm in Equation 2.3 is minimized by setting the remaining coefficients $v_i^T b = \gamma_i = (u_i^T y) v_i / \sigma_i$ for $i = 1, 2, \ldots, r$.

We can implement an algorithm to solve ordinary least squares using the SVD by calling R's function `svd`.

```
# Compute OLS estimate using SVD decomposition.
#
# Args:
#     X: A numeric data matrix.
#     y: Response vector.
#
# Returns:
```

```
#     Regression vector beta of length ncol(X).
casl_ols_svd  <-
function(X, y)
{
  svd_output <- svd(X)
  r <- sum(svd_output$d > .Machine$double.eps)
  U <- svd_output$u[, 1:r]
  V <- svd_output$v[, 1:r]
  beta <- V %*% (t(U) %*% y / svd_output$d[1:r])
  beta
}
```

To test this function, we will first create some random data and set a regression vector β.

```
n <- 1e4; p <- 4
X <- matrix(rnorm(n*p), ncol = p)
beta <- c(1,2,3,4)
epsilon <- rnorm(n)
y <- X %*% beta + epsilon
```

From here, we compute the estimated $\widehat{\beta}$ from `casl_ols_svd`.

```
beta_h_svd <- casl_ols_svd(X, y)
beta_h_svd
```

```
          [,1]
[1,] 0.9816599
[2,] 1.9938207
[3,] 2.9941449
[4,] 4.0062232
```

The result closely reconstructs the true β, which was set to the vector $(1, 2, 3, 4)$. We should not expect to get the exact solution due to the presence of the noise vector `epsilon`. We can verify that this is the same solution given by R using the `lm` function and extracting the coefficients with `coef`.

```
coef(lm(y ~ X - 1))
```

```
       X1        X2        X3        X4
0.9816599 1.9938207 2.9941449 4.0062232
```

This result matches, at least to the 7th decimal place, with the result from our function.

References

[1] Golub, G. H., and Van Loan, C. F. *Matrix Computations*, vol. 3. JHU Press, New York, NY, 2012.

LAB QUESTIONS

1. Here is a thing!