MTH3045: Statistical Computing

Dr. Ben Youngman
b.youngman@exeter.ac.uk
Laver 817; ext. 2314

10/02/2025

Matrix-based computing

Matrix-based computing

Week 5 lecture 1

Eigen-decomposition

Eigen-decomposition: definition

- **Definition**: We can write any symmetric matrix $\mathbf{A}$ in the form

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}$$

  where $\mathbf{U}$ is an orthogonal matrix and $\mathbf{\Lambda}$ is a diagonal matrix
- We will denote the diagonal elements of $\mathbf{\Lambda}$ by $\lambda_1 \geq \ldots \geq \lambda_n$
- Post-multiplying both sides of the decomposition by $\mathbf{U}$ we have $\mathbf{AU} = \mathbf{U\Lambda}$
- Let $\mathbf{u}_i$ denote the ith column of $\mathbf{U}$
  - then $\mathbf{A}\mathbf{u}_i = \lambda_i \mathbf{u}_i$
  - the λ_is are the **eigenvalues** of $\mathbf{A}$
  - the columns of $\mathbf{U}$ are the corresponding **eigenvectors**
- We call the decomposition above the **eigen-decomposition** (or some use *spectral decomposition*) of $\mathbf{A}$.

Eigen-decomposition: properties

- If $\mathbf{A}$ is symmetric, the following properties of eigen-decompositions hold
  - $\mathbf{U}^{-1} = \mathbf{U}^{\mathsf{T}}$
  - $\mathbf{A}^{-1} = \mathbf{U}\mathbf{\Lambda}^{-1}\mathbf{U}^{-1}$
  - because $\mathbf{\Lambda}$ is diagonal, so too is $\mathbf{\Lambda}^{-1}$, and its elements are $(\mathbf{\Lambda}^{-1})_{ii} = 1/\lambda_i$.
  - $\det(\mathbf{A}) = \prod_{i=1}^{n} \lambda_i$.

```
eigen(A) # computes the eigen-decomposition of a matrix A
```

Example: Eigen-decomposition in R I

- Use `eigen()` in `R` to give the eigen-decomposition of the 3×3 Hilbert matrix.
- We've already calculated the 3×3 Hilbert matrix and stored it as `H`, so we just need

```
eigen(H)
```

```
## eigen() decomposition
## $values
## [1] 1.40831893 0.12232707 0.00268734
##
## $vectors
##           [,1]       [,2]       [,3]
## [1,] 0.8270449  0.5474484  0.1276593
## [2,] 0.4598639 -0.5282902 -0.7137469
## [3,] 0.3232984 -0.6490067  0.6886715
```

Example: Eigen-decomposition in R II

- Note that `eigen()` returns a `list` comprising...
  - `values`, a vector of the eigenvalues in descending order
  - `vectors`, a matrix of the eigenvectors, in column order corresponding to the eigenvalues
- We can ask `eigen()` to return only the eigenvalues by specifying `only.values = TRUE`
- We can stipulate that the supplied matrix is symmetric by specifying `symmetric = TRUE` (which avoids checking symmetry, and can save a bit of time for large matrices)

Example: Orthogonality of the eigen-decomposition in R

- Confirm that the eigenvectors of the eigen-decomposition of the 3×3 Hilbert matrix form an orthogonal matrix.
- If $\mathbf{U}$ denotes the matrix of eigenvectors, then we need to show that $\mathbf{U}^{\mathsf{T}}\mathbf{U} = \mathbf{I}$.

```
eH <- eigen(H)
U <- eH$vectors
crossU <- crossprod(U) # should be 3 x 3 identity matrix
all.equal(crossU, diag(3))
```

```
## [1] TRUE
```

Challenges I

- Go to Challenges I of the week 5 lecture 1 challenges at https://byoungman.github.io/MTH3045/challenges

Example: Powers of matrices

- Consider $n \times n$ matrix $\mathbf{A}$ with eigen-decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}$
- The second power of $\mathbf{A}$ is $\mathbf{A}^2 = \mathbf{A}\mathbf{A}$
- Show that the mth power of is given by $\mathbf{A}^m = \mathbf{U}\mathbf{\Lambda}^m\mathbf{U}^{\mathsf{T}}$
- We note that

$$\mathbf{A}^m = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}\ldots\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}.$$

  Because $\mathbf{U}^{\mathsf{T}}\mathbf{U} = \mathbf{I}_n$ this reduces to

$$\mathbf{A}^m = \mathbf{U}\mathbf{\Lambda}\mathbf{\Lambda}\ldots\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}} = \mathbf{U}\mathbf{\Lambda}^m\mathbf{U}^{\mathsf{T}}.$$

Challenges II

- Go to Challenges II of the week 5 lecture 1 challenges at https://byoungman.github.io/MTH3045/challenges

Properties of eigenvalues and eigenvectors

- **Theorem**: Let $\mathbf{A}$ be a positive definite matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ and corresponding eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \ldots \mathbf{u}_n$
- Then $\mathbf{u}_1$ maximises $\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x}$ *and* $\mathbf{u}_1^\mathsf{T}\mathbf{A}\mathbf{u}_1 = \lambda_1$
- Furthermore, for $k = 1, \ldots, p < n$, given $\mathbf{u}_1, \ldots, \mathbf{u}_k$, $\mathbf{u}_{k+1}$ maximises $\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x}$, subject to $\mathbf{x}$ being orthogonal to $\mathbf{u}_1, \ldots, \mathbf{u}_k$ *and* $\mathbf{u}_{k+1}^\mathsf{T}\mathbf{A}\mathbf{u}_{k+1} = \lambda_{k+1}$
- *Proof*: For a proof see, e.g. Johnson and Wichern (2007, 80), but note that knowledge of the proof is beyond the scope of MTH3045

Example: Principal component analysis I

- Consider a random vector $\mathbf{Y} = (Y_1, \ldots, Y_n)^\mathsf{T}$ with variance-covariance matrix $\mathbf{\Sigma}$
- Then consider taking linear combinations of $\mathbf{Y}$ so that

$$\begin{array}{rcccl} Z_1 & = & \mathbf{a}_1^\mathsf{T}\mathbf{Y} & = & a_{11}Y_1 + a_{12}Y_2 + \ldots + a_{1n}Y_n, \\ Z_2 & = & \mathbf{a}_2^\mathsf{T}\mathbf{Y} & = & a_{21}Y_1 + a_{22}Y_2 + \ldots + a_{2n}Y_n, \\ \vdots & = & \vdots & = & \vdots \\ Z_n & = & \mathbf{a}_n^\mathsf{T}\mathbf{Y} & = & a_{n1}Y_1 + a_{n2}Y_2 + \ldots + a_{nn}Y_n, \end{array}$$

  where $\mathbf{a}_1, \ldots, \mathbf{a}_n$ are coefficient vectors

- Then

$$\begin{array}{rll} \mathrm{Var}(Z_i) & = \mathbf{a}_i^\mathsf{T}\mathbf{\Sigma}\mathbf{a}_i & \text{for } i = 1, \ldots, n \\ \mathrm{Cov}(Z_j, Z_k) & = \mathbf{a}_j^\mathsf{T}\mathbf{\Sigma}\mathbf{a}_k & \text{for } j, k = 1, \ldots, n. \end{array}$$

Example: Principal component analysis II

- The **principal components** are the *uncorrelated* linear combinations of $Y_1, \ldots, Y_n$ that maximise $\mathrm{Var}(\mathbf{a}_i^\mathsf{T}\mathbf{Y})$, for $i = 1, \ldots, n$
- Hence the first principal component maximises $\mathrm{Var}(\mathbf{a}_1\mathbf{Y})$ subject to $\mathbf{a}_1^\mathsf{T}\mathbf{a}_1 = 1$
- The second maximises $\mathrm{Var}(\mathbf{a}_2\mathbf{Y})$ subject to $\mathbf{a}_2^\mathsf{T}\mathbf{a}_2 = 1$ *and* $\mathrm{Cov}(\mathbf{a}_1^\mathsf{T}\mathbf{Y}, \mathbf{a}_2^\mathsf{T}\mathbf{Y}) = 0$, and so forth
- More generally, the ith principal component, $i > 1$, maximises $\mathrm{Var}(\mathbf{a}_i^\mathsf{T}\mathbf{Y})$ subject $\mathbf{a}_i^\mathsf{T}\mathbf{a}_i = 1$ *and* $\mathrm{Cov}(\mathbf{a}_i^\mathsf{T}\mathbf{Y}, \mathbf{a}_j^\mathsf{T}\mathbf{Y}) = 0$ for $j < i$
- Now suppose that we form the eigen-decomposition $\mathbf{\Sigma} = \mathbf{U}^\mathsf{T}\mathbf{\Lambda}\mathbf{U}$. The eigenvectors of $\mathbf{\Sigma}$ therefore meet the criteria described above for principal components, and hence give one definition for principal components

Example: Principal component analysis of the `iris` data I

- Fisher's iris data are distributed with `R` as object `iris`
- From the dataset's help file the data comprise "*the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are* `Iris setosa`, `versicolor`, *and* `virginica`." Compute the principal components for the variables `Sepal.Length`, `Sepal.Width`, `Petal.Length` and `Petal.Width`.
- To compute the principals components, we'll load the `iris` data, which we'll quickly look at

```
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Example: Principal component analysis of the `iris` data II

- Then we'll extract the relevant variables, and form their empirical correlation matrix

```
vbls <- c('Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width')
species <- as.factor(iris$Species)
Y <- as.matrix(iris[, vbls]) # data corresponding to variables under study
corY <- cor(Y) # correlation matrix of variables under study
```

- We can then use the eigen-decomposition of the correlation matrix to form the eigenvectors, which are also the coefficient vectors, and which we'll store as `A`
- Note that we use the correlation matrix because we're measuring different features of the plant, which we don't necessarily expect to be directly comparable

```
A <- eigen(corY)$vectors
A
```

```
##             [,1]        [,2]       [,3]       [,4]
## [1,]  0.5210659 -0.37741762  0.7195664  0.2612863
## [2,] -0.2693474 -0.92329566 -0.2443818 -0.1235096
## [3,]  0.5804131 -0.02449161 -0.1421264 -0.8014492
## [4,]  0.5648565 -0.06694199 -0.6342727  0.5235971
```

Example: Principal component analysis of the `iris` data III

- The principal components are then obtained by calculating $\mathbf{a}_j^T \mathbf{y}$ for $j = 1, \ldots, 4$. We'll store these as `pcs` and then use `head()` to show the first few.

```
pcs <- Y %*% A
colnames(pcs) <- paste('PC', 1:4, sep = '')
head(pcs)
```

```
##            PC1       PC2      PC3        PC4
## [1,] 2.640270 -5.204041 2.488621 -0.1170332
## [2,] 2.670730 -4.666910 2.466898 -0.1075356
## [3,] 2.454606 -4.773636 2.288321 -0.1043499
## [4,] 2.545517 -4.648463 2.212378 -0.2784174
## [5,] 2.561228 -5.258629 2.392226 -0.1555127
## [6,] 2.975946 -5.707321 2.437245 -0.2237665
```

- *Remark*: Principal component analysis is a commonly used statistical method, often as a method of *dimension reduction*, when a finite number of principal components, below the dimension of the data, are used to capture most of what's in the original data.

Example: Principal component analysis of the `iris` data IV

- In practice, if we want to perform PCA then we'd usually use one of `R`'s built in functions, such as `prcomp()` or `princomp()`
- For example

```
prcomp(Y, center = TRUE, scale = TRUE)
```

```
## Standard deviations (1, .., p=4):
## [1] 1.7083611 0.9560494 0.3830886 0.1439265
##
## Rotation (n x k) = (4 x 4):
##                     PC1         PC2        PC3        PC4
## Sepal.Length  0.5210659 -0.37741762  0.7195664  0.2612863
## Sepal.Width  -0.2693474 -0.92329566 -0.2443818 -0.1235096
## Petal.Length  0.5804131 -0.02449161 -0.1421264 -0.8014492
## Petal.Width   0.5648565 -0.06694199 -0.6342727  0.5235971
```

gives the same principal component coefficients as in Example 3.17

- A benefit of working with `prcomp()` or `princomp()` is that `R` interprets the objects as relating to PCA, and then `summary()` and `plot()`, for example, perform useful actions
- Principal component regression is a statistical model in which we perform regression on principal components

Week 5 lecture 2

Singular value decomposition (SVD)

- **Theorem**: For an $m \times n$ matrix $\mathbf{A}$ with real elements and $m \geq n$, there exist orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ such that

$$\mathbf{U}^{\mathsf{T}}\mathbf{A}\mathbf{V} = \mathbf{D},$$

  where $\mathbf{D}$ is a diagonal matrix with elements $d_1 \geq d_2 \geq \ldots \geq d_m$.

- **Definition**: The **singular value decomposition** (SVD) of $\mathbf{A}$ is

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}}$$

- The diagonal entries of $m \times n$ matrix $\mathbf{D}$ are the singular values of $\mathbf{A}$
- We can form a $m \times m$ matrix $\mathbf{U}$ from the eigenvectors of $\mathbf{A}^{\mathsf{T}}\mathbf{A}$ and a $n \times n$ matrix $\mathbf{V}$ from the eigenvectors of $\mathbf{A}\mathbf{A}^{\mathsf{T}}$
- The singular values are the square roots of the positive eigenvalues of $\mathbf{A}^{\mathsf{T}}\mathbf{A}$

Example: SVD of the Hilbert matrix in R I

```
svd(A) # calculates the SVD of a matrix A
```

- Compute a SVD of the 3×3 Hilbert matrix, $\mathbf{H}_3$, in R using `svd()`
- We have $\mathbf{H}_3$ stored as `H` already, so we'll now calculate its SVD

```
(H.svd <- svd(H))
```

```
## $d
## [1] 1.40831893 0.12232707 0.00268734
##
## $u
##             [,1]       [,2]       [,3]
## [1,] -0.8270449  0.5474484  0.1276593
## [2,] -0.4598639 -0.5282902 -0.7137469
## [3,] -0.3232984 -0.6490067  0.6886715
##
## $v
##             [,1]       [,2]       [,3]
## [1,] -0.8270449  0.5474484  0.1276593
## [2,] -0.4598639 -0.5282902 -0.7137469
## [3,] -0.3232984 -0.6490067  0.6886715
```

Example: SVD of the Hilbert matrix in R II

- From `svd()` we get a three-element list where `d` is a vector of the diagonal elements of **D**, `u` is **U** and `v` is **V**
- We can quickly confirm that $\mathbf{H}_3 = \mathbf{UDV}^\mathsf{T}$

```
all.equal(H, H.svd$u %*% tcrossprod(diag(H.svd$d), H.svd$v))
```

```
## [1] TRUE
```

Solving systems of linear equations using a SVD

- *Remark*: One application of the SVD is solving systems of linear equations
- Let $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\mathsf{T}$ be the SVD of $\mathbf{A}$
- Consider again solving $\mathbf{A}\mathbf{x} = \mathbf{b}$
- Then

$$\begin{aligned}
\mathbf{U}^\mathsf{T}\mathbf{A}\mathbf{x} &= \mathbf{U}^\mathsf{T}\mathbf{b} && \text{(premultiplying by } \mathbf{U}^\mathsf{T}\text{)} \\
\mathbf{U}^\mathsf{T}\mathbf{U}\mathbf{D}\mathbf{V}^\mathsf{T}\mathbf{x} &= \mathbf{U}^\mathsf{T}\mathbf{b} && \text{(substituting } \mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\mathsf{T}\text{)} \\
\mathbf{D}\mathbf{V}^\mathsf{T}\mathbf{x} &= \mathbf{U}^\mathsf{T}\mathbf{b} && \text{(as } \mathbf{U}^\mathsf{T}\mathbf{U} = \mathbf{I}_n\text{)} \\
\mathbf{D}\tilde{\mathbf{x}} &= \tilde{\mathbf{b}}. && \text{(setting } \tilde{\mathbf{x}} = \mathbf{V}^\mathsf{T}\mathbf{x} \text{ and } \tilde{\mathbf{b}} = \mathbf{U}^\mathsf{T}\mathbf{b}\text{)}
\end{aligned}$$

- As $\mathbf{D}$ is diagonal, we see that setting $\tilde{\mathbf{x}} = \mathbf{V}^\mathsf{T}\mathbf{x}$ and $\tilde{\mathbf{b}} = \mathbf{U}^\mathsf{T}\mathbf{b}$ results in a diagonal solve, i.e. essentially n divisions

Challenges I

- Go to Challenges I of the week 5 lecture 2 challenges at https://byoungman.github.io/MTH3045/challenges

Example: Solving systems of linear equations using a SVD in R I

- Solve $\mathbf{\Sigma z} = \mathbf{y} - \boldsymbol{\mu}$ in `R` using a SVD with $\mathbf{y}$, $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ as in Example 3.2
- Following the above remark, we want to calculate $\tilde{\mathbf{x}} = \mathbf{V}^\mathsf{T}\mathbf{x}$ and $\tilde{\mathbf{b}} = \mathbf{U}^\mathsf{T}(\mathbf{y} - \boldsymbol{\mu})$
- We'll start by computing the SVD of $\mathbf{\Sigma}$, and then extract diag($\mathbf{D}$) and $\mathbf{V}$, which we'll call `S.d` and `S.V`, respectively.

```
S.svd <- svd(Sigma)
S.d <- S.svd$d
S.V <- S.svd$v
S.U <- S.svd$u
```

Then we obtain $\tilde{\mathbf{b}} = \mathbf{U}^\mathsf{T}(\mathbf{y} - \boldsymbol{\mu})$, which we'll call `b2`.

```
b2  <- crossprod(S.U, y - mu)
```

Then $\tilde{\mathbf{x}}$ is the solution of $\mathbf{D}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$, which we'll call `x2`, and can be computed with

```
x2 <- b2 / S.svd$d
```

since $\mathbf{D}$ is diagonal

Example: Solving systems of linear equations using a SVD in R II

- Finally, $\mathbf{z}$ is the solution of $\mathbf{V}^T\mathbf{z} = \tilde{\mathbf{x}}$, which we'll call `res6` and can obtain with

```
z <- S.V %*% x2
```

since $\mathbf{V}^{-\mathsf{T}} = \mathbf{V}$. Vectorising this and renaming it to `res6`, we confirm that we get the same result as in Example 3.8

```
res6 <- as.vector(z)
all.equal(solve(Sigma, y - mu), res6)
```

```
## [1] TRUE
```

Generalized inverse

- So far we have considered systems of linear equations where $\mathbf{A}$ is non-singular, which means that $\mathbf{A}^{-1}$ is unique
- Now we'll consider the case where $\mathbf{A}$ is singular, although this won't be examined in MTH3045
- **Definition**: A **generalized inverse** matrix of the matrix $\mathbf{A}$ is any matrix $\mathbf{A}^{-}$ such that

$$\mathbf{A}\mathbf{A}^{-}\mathbf{A} = \mathbf{A}$$

- Note that $\mathbf{A}^{-}$ is not unique

Moore-Penrose inverse I

- The **Moore-Penrose pseudo-inverse**[1] of a matrix $\mathbf{A}$ is a generalized inverse that is unique by virtue of stipulating that it must satisfy the following four properties.

1. $\mathbf{A}\mathbf{A}^{-}\mathbf{A} = \mathbf{A}$.
2. $\mathbf{A}^{-}\mathbf{A}\mathbf{A}^{-} = \mathbf{A}^{-}$.
3. $(\mathbf{A}\mathbf{A}^{-})^{\mathsf{T}} = \mathbf{A}\mathbf{A}^{-}$.
4. $(\mathbf{A}^{-}\mathbf{A})^{\mathsf{T}} = \mathbf{A}^{-}\mathbf{A}$.

[1] The Moore-Penrose pseudoinverse is named after E. H. Moore and Sir Roger Penrose. Moore first worked in abstract algebra, proving in 1893 the classification of the structure of finite fields (also called Galois fields). He then worked on various topics, including the foundations of geometry and algebraic geometry, number theory, and integral equations. Penrose has made contributions to the mathematical physics of general relativity and cosmology. He has received several prizes and awards, including the 1988 Wolf Prize in Physics, which he shared with Stephen Hawking for the Penrose–Hawking singularity theorems, and one half of the 2020 Nobel Prize in Physics "for the discovery that black hole formation is a robust prediction of the general theory of relativity".

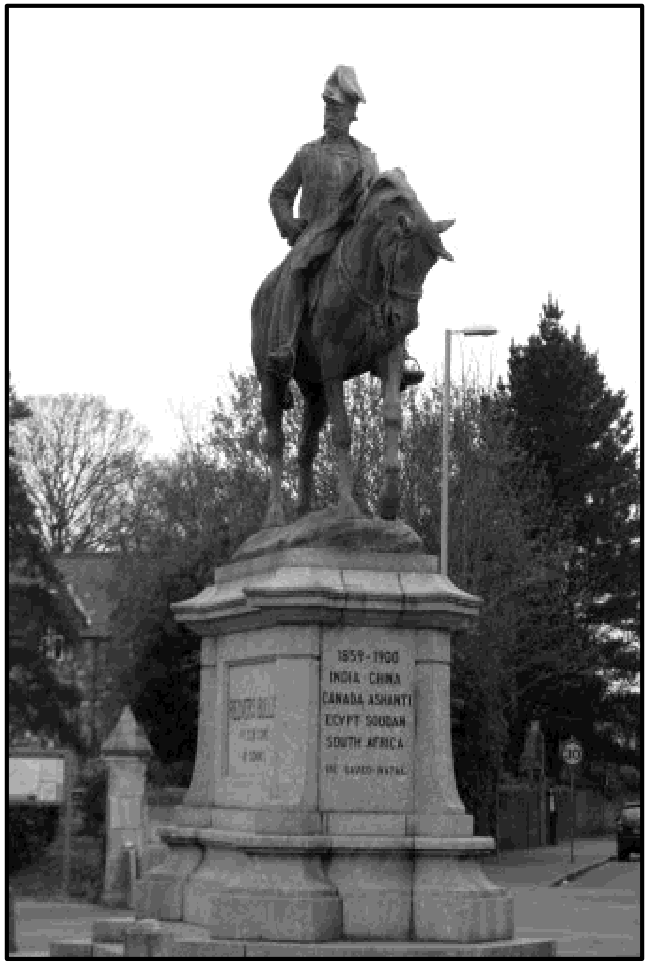
Moore-Penrose inverse II

- We can construct a Moore-Penrose pseudo-inverse via the SVD
- Consider $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}}$, the SVD of $\mathbf{A}$
- Let $\mathbf{D}^{-}$ denote the generalised inverse of $\mathbf{D}$, which is simply obtained by taking reciprocals of the positive diagonal values, with zeros left as zeros
- Then we have

$$\mathbf{A}^{-} = \mathbf{U}\mathbf{D}^{-}\mathbf{V}^{\mathsf{T}}$$

Example: Image processing via an SVD I

- Consider the following $n \times m = 338 \times 450$ pixel greyscale image



Example: Image processing via an SVD II

- The image can be represented as a matrix, $\mathbf{A}$, which we'll store as `A`, comprising values on $[0, 1]$
  - 0 is white and 1 is black
  - the bottom left 7×7 pixels take the following values

```
round(A[1:7, 1:7], 4)
```

```
##        [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]
## [1,] 0.3516 0.4719 0.4484 0.2719 0.2523 0.2758 0.2758
## [2,] 0.4261 0.5229 0.4719 0.2915 0.2562 0.2719 0.2680
## [3,] 0.4915 0.5464 0.4876 0.3033 0.2444 0.2601 0.2562
## [4,] 0.5346 0.5503 0.4915 0.3150 0.2327 0.2601 0.2758
## [5,] 0.5739 0.5699 0.5111 0.3647 0.2641 0.3033 0.3503
## [6,] 0.5895 0.5739 0.5490 0.4471 0.3503 0.3856 0.4405
## [7,] 0.5895 0.5817 0.5647 0.5163 0.4340 0.4536 0.4837
```

Example: Image processing via an SVD III

- Suppose we compute the SVD $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathsf{T}}$
- A finite-rank representation of $\mathbf{A}$ is given by

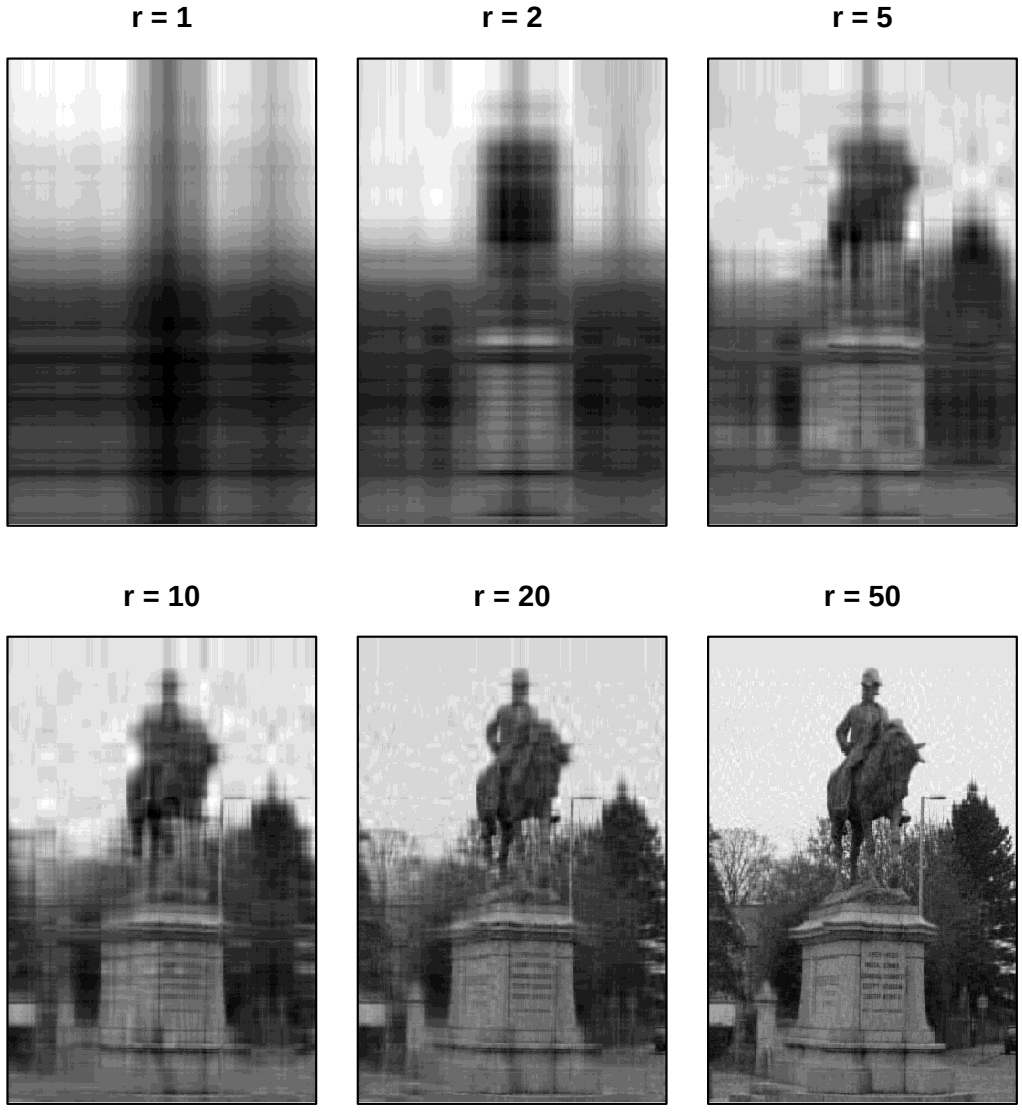
$$\mathbf{A}_r = \mathbf{U}_r\mathbf{D}_r\mathbf{V}_r^{\mathsf{T}}$$

where

- $\mathbf{U}_r$ is the $n \times r$ matrix comprising the first r columns of $\mathbf{U}$
- $\mathbf{D}_r$ is the $r \times r$ matrix comprising the first r rows and columns of $\mathbf{D}$
- $\mathbf{V}_r$ is the $m \times r$ matrix comprising the first r columns of $\mathbf{V}$

Example: Image processing via an SVD IV

- The following shows the resulting greyscale images obtained by plotting $\mathbf{A}_r$ for $r = 1,\ 2,\ 5,\ 10,\ 20$ and 50



Example: Image processing via an SVD V

- You might wonder how SVD has compressed our image
- The image itself takes

```
format(object.size(A), units = 'Kb')
```

```
## [1] "1188.5 Kb"
```

bytes (and there are eight bits in a byte)

- However, if we consider the $r = 20$ case of Example 3.20, then we need to store the r diagonal elements of $\mathbf{D}_r$ and the first r columns of $\mathbf{U}_r$ and $\mathbf{V}_r$, which we could store in a list

```
r <- 20
ind <- 1:r
A_r <- list(diag(D[ind, ind]), U[, ind], V[, ind])
format(object.size(A_r), units = 'Kb')
```

```
## [1] "123.8 Kb"
```

and takes about 10% of the memory of the original image

Example: Image processing via an SVD VI

- Of course, there is computational cost of computing the singular value decomposition, i.e. compressing the image, and then later decompressing the image, which should be taken into account when considering image compression

Challenges II

- Go to Challenges II of the week 5 lecture 2 challenges at https://byoungman.github.io/MTH3045/challenges

Week 5 lecture 3

QR decomposition

- The **QR decomposition** is often used in the background of functions in `R`
- **Definition**: Any real square matrix **A** may be decomposed as

$$\mathbf{A} = \mathbf{QR},$$

  where **Q** is an orthogonal matrix and **R** is an upper triangular matrix
- This is its **QR decomposition**
- If **A** is non-singular, then the QR decomposition is unique

```
qr(A) # computes the QR decomposition of a matrix A
```

- *Remark*: When `R` computes the QR decomposition, **R** is simply the upper triangle of `qr()$qr`
  - however, **Q** is rather more complicated to obtain, but fortunately `qr.Q()` does all the calculations for us

Properties of the QR decomposition

- $|\det(\mathbf{A})| = |\det(\mathbf{Q})|\det(\mathbf{R}) = \det(\mathbf{R})$ since $\det(\mathbf{Q}) = \pm 1$ as $\mathbf{Q}$ is orthogonal
- $\det(\mathbf{R}) = \prod_{i=1}^{n} |r_{ii}|$, since $\mathbf{R}$ is triangular
- $\mathbf{A}^{-1} = (\mathbf{QR})^{-1} = \mathbf{R}^{-1}\mathbf{Q}^{-1} = \mathbf{R}^{-1}\mathbf{Q}^{\mathsf{T}}$

Example: QR decomposition of the Hilbert matrix in R I

- Compute the QR decomposition of $\mathbf{H}_3$, the 3×3 Hilbert matrix, in `R`
- Then use `qr.Q()` and `qr.R()` to extract $\mathbf{Q}$ and $\mathbf{R}$ from the output of `qr()` to confirm that $\mathbf{QR} = \mathbf{H}_3$

Example: QR decomposition of the Hilbert matrix in R II

- We'll start with the QR decomposition of $\mathbf{H}_3$, which we'll store as `qr.H`

```
(qr.H <- qr(H))
```

```
## $qr
##             [,1]       [,2]          [,3]
## [1,] -1.1666667 -0.6428571 -0.450000000
## [2,]  0.4285714 -0.1017143 -0.105337032
## [3,]  0.2857143  0.7292564  0.003901372
##
## $rank
## [1] 3
##
## $qraux
## [1] 1.857142857 1.684240553 0.003901372
##
## $pivot
## [1] 1 2 3
##
## attr(,"class")
## [1] "qr"
```

Example: QR decomposition of the Hilbert matrix in R III

- Then `qr.Q()` and `qr.R()` will give **Q** and **R**, which we'll store as `Q.H` and `R.H`

```
Q.H <- qr.Q(qr.H)
Q.H
```

```
##             [,1]       [,2]       [,3]
## [1,] -0.8571429  0.5016049  0.1170411
## [2,] -0.4285714 -0.5684856 -0.7022469
## [3,] -0.2857143 -0.6520864  0.7022469
```

```
R.H <- qr.R(qr.H)
R.H
```

```
##           [,1]       [,2]          [,3]
## [1,] -1.166667 -0.6428571 -0.450000000
## [2,]  0.000000 -0.1017143 -0.105337032
## [3,]  0.000000  0.0000000  0.003901372
```

Example: QR decomposition of the Hilbert matrix in R IV

- Finally we'll compute **QR**

```
Q.H %*% R.H
```

```
##           [,1]      [,2]      [,3]
## [1,] 1.0000000 0.5000000 0.3333333
## [2,] 0.5000000 0.3333333 0.2500000
## [3,] 0.3333333 0.2500000 0.2000000
```

```
all.equal(Q.H %*% R.H, H)
```

```
## [1] TRUE
```

which does indeed give $\mathbf{H}_3$.

Example: Solving systems of linear equations via the QR decomposition in R I

- Solve $\mathbf{\Sigma z} = \mathbf{y} - \boldsymbol{\mu}$ in R using the QR decomposition and with $\mathbf{y}$, $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ as in Example 3.2
- Suppose that $\mathbf{\Sigma}$ has QR decomposition $\mathbf{\Sigma} = \mathbf{QR}$
- The following calculates the QR decomposition of $\mathbf{\Sigma}$ as `S.qr`, and then $\mathbf{Q}$ and $\mathbf{R}$ as `S.Q` and `S.R`, respectively

```
S.qr <- qr(Sigma)
S.Q <- qr.Q(S.qr)
S.R <- qr.R(S.qr)
```

- From Example 3.2 we have that $\mathbf{z}$ is `res1`, i.e.

```
res1 <- solve(Sigma, y - mu)
res1
```

```
## [1]  0.08 -0.38  0.14
```

Example: Solving systems of linear equations via the QR decomposition in R II

- Solving $\mathbf{\Sigma z} = \mathbf{y} - \boldsymbol{\mu}$ is then equivalent to solving $\mathbf{QRz} = \mathbf{y} - \boldsymbol{\mu}$
- So we solve $\mathbf{Qx} = \mathbf{y} - \boldsymbol{\mu}$ for $\mathbf{x}$ with the following

```
x <- crossprod(S.Q, y - mu)
```

  as $\mathbf{Q}^{-1} = \mathbf{Q}^{\mathsf{T}}$ because $\mathbf{Q}$ is orthogonal

- Then we solve $\mathbf{Rz} = \mathbf{x}$ for $\mathbf{z}$,

```
z1 <- solve(S.R, x)
```

  or, because $\mathbf{R}$ is upper triangular,

```
z2 <- backsolve(S.R, x)
```

  which are both the same

```
all.equal(z1, z2)
```

```
## [1] TRUE
```

Example: Solving systems of linear equations via the QR decomposition in R III

- We'll take the second, z2, and convert it from a one-column matrix to a vector called `res7`

```
res7 <- as.vector(z2)
res7
```

```
## [1]  0.08 -0.38  0.14
```

and see that this gives the same as before

```
all.equal(res1, res7)
```

```
## [1] TRUE
```

QR decomposition in R

- If we have obtained a QR decomposition in R using `qr()`, then we can use `qr.solve(A, b)` to solve $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{x}$
- This avoids having to find $\mathbf{Q}$ and $\mathbf{R}$ with `qr.Q()` and `qr.R()`, and requires only one function call in R

Challenges I

- Go to Challenges I of the week 5 lecture 3 challenges at https://byoungman.github.io/MTH3045/challenges

QR decomposition of a rectangular matrix

- **Definition**: Definition 3.16 extends to a $m \times n$ matrix $\mathbf{A}$, with $m \geq n$, so that

$$\mathbf{A} = \mathbf{QR} = \mathbf{Q} \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{Q}_1 \mathbf{R}_1,$$

  $\mathbf{Q}$ is an $m \times m$ unitary matrix, and $\mathbf{R}$ and an $m \times n$ upper triangular matrix
- then
  - $\mathbf{Q}_1$ is $m \times n$
  - $\mathbf{Q}_2$ is $m \times (m - n)$
  - $\mathbf{Q}_1$ and $\mathbf{Q}_2$ both have orthogonal columns
  - $\mathbf{R}_1$ is an $n \times n$ upper triangular matrix, followed by $(m - n)$ rows of zeros

Example: Linear modelling via QR decomposition

- Consider a linear model with response vector $\mathbf{y}$ and design matrix $\mathbf{X}$, where $\mathbf{y} = \mathbf{X}\beta + \mathbf{e}$ and β is the vector of regression coefficients and $\mathbf{e}$ is the observed residual vector
- The least squares estimate of β, denoted $\hat{\beta}$, satisfies

$$\mathbf{X}^\mathsf{T}\mathbf{X}\beta = \mathbf{X}^\mathsf{T}\mathbf{y}$$

- This is achieved in R by forming the QR decomposition of $\mathbf{X}$, i.e. $\mathbf{X} = \mathbf{QR}$
- Then $\hat{\beta}$, satisfies

$$(\mathbf{QR})^\mathsf{T}\mathbf{QR}\hat{\beta} = (\mathbf{QR})^\mathsf{T}\mathbf{y}$$

  which can be re-written as

$$\mathbf{R}^\mathsf{T}\mathbf{Q}^\mathsf{T}\mathbf{QR}\hat{\beta} = \mathbf{R}^\mathsf{T}\mathbf{Q}^\mathsf{T}\mathbf{y}$$

- Given $\mathbf{Q}$ is orthogonal, this simplifies to

$$\mathbf{R}^\mathsf{T}\mathbf{R}\hat{\beta} = \mathbf{R}^\mathsf{T}\mathbf{Q}^\mathsf{T}\mathbf{y}$$

Example: MTH2006 cement factory data I

- Recall the cement factory data from MTH2006 and the linear model

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \varepsilon_i$$

- Y_i denotes the output of the cement factory in month i for x_{i1} days at a temperature of x_{i2} degrees Fahrenheit
- $\varepsilon_i \sim \mathrm{N}(0, \sigma^2)$ and are i.i.d
- Use the QR decomposition to find $\hat{\beta}$ and confirm your answer against that given by `lm()`

Example: MTH2006 cement factory data II

- We'll first input the data

```
# operating temperatures
temp <- c(35.3, 29.7, 30.8, 58.8, 61.4, 71.3, 74.4, 76.7, 70.7, 57.5,
46.4, 28.9, 28.1, 39.1, 46.8, 48.5, 59.3, 70, 70, 74.5, 72.1,
58.1, 44.6, 33.4, 28.6)
# number of operational days
days <- c(20, 20, 23, 20, 21, 22, 11, 23, 21, 20, 20, 21, 21, 19, 23,
20, 22, 22, 11, 23, 20, 21, 20, 20, 22)
# output from factory
output <- c(10.98, 11.13, 12.51, 8.4, 9.27, 8.73, 6.36, 8.5, 7.82, 9.14,
8.24, 12.19, 11.88, 9.57, 10.94, 9.58, 10.09, 8.11, 6.83, 8.88,
7.68, 8.47, 8.86, 10.36, 11.08)
```

and put them into a `data.frame` called `prod` (as in MTH2006).

```
prod <- data.frame(temp = temp, days = days, output = output)
```

Example: MTH2006 cement factory data III

- Then we'll fit the linear model with `lm()`, extract the regression coefficients, and store them as `b0`.

```
m0 <- lm(output ~ days + temp, data = prod)
b0 <- coef(m0)
```

- Next we'll store the response data as `y` and the design matrix as `X`.

```
y <- output
X <- cbind(1, days, temp)
```

- The QR decomposition of `X` can be obtained with `qr()`, which is stored as `X.qr`, and then $\mathbf{Q}$ and $\mathbf{R}$ of the QR decomposition $\mathbf{X} = \mathbf{QR}$ stored as `Q.qr` and `R.qr`, respectively

```
X.qr <- qr(X)
Q.qr <- qr.Q(X.qr)
R.qr <- qr.R(X.qr)
```

Example: MTH2006 cement factory data IV

- Next we'll compute $\mathbf{w} = \mathbf{R}^\mathsf{T}\mathbf{Q}^\mathsf{T}\mathbf{y}$, say, then solve $\mathbf{R}^\mathsf{T}\mathbf{z} = \mathbf{w}$ for $\mathbf{z}$ and then solve $\mathbf{R}\hat{\beta} = \mathbf{z}$ for $\hat{\beta}$

```
w <- crossprod(R.qr, crossprod(Q.qr, y))
z <- forwardsolve(R.qr, w, transpose = TRUE, upper.tri = TRUE)
b1 <- drop(backsolve(R.qr, z))
```

- Finally we'll check that our regression coefficients calculated through the QR decomposition are the same as those extracted from our call to `lm()`

```
all.equal(b0, b1, check.attributes = FALSE)
```

```
## [1] TRUE
```

which they are

Example: MTH2006 cement factory data V

```
b1 <- drop(backsolve(R.qr, z))
all.equal(b0, b1, check.attributes = FALSE)
```

- *Remark*: Note two things above

1. We've used `drop()` because `backsolve()` will by default return a $(p+1) \times 1$ matrix whereas `coef()` returns an $(p+1)$-vector

- the two aren't considered identical by `all.equal()`, even if their values are the same
- calling `drop()` will simplify a matrix to a vector *if* possible, and hence we're comparing like with like

2. Setting `all.equal(..., check.attributes = FALSE)` should check only the supplied objects and not any attributes, such as names

- Without this, because `b0` has names and `b1` doesn't, `all.equal()` wouldn't consider them identical

Challenges II

- Go to Challenges II of the week 5 lecture 3 challenges at https://byoungman.github.io/MTH3045/challenges

Computational costs of matrix decompositions

- Each of the four matrix decompositions can be used for various purposes, such as calculating the determinant of a matrix, or solving a system of linear equations
- We might ask ourselves which one we should use
- We'll start by comparing the computational cost of each, i.e. the number of flops each takes
- The Cholesky algorithm above requires $n^3/3$ flops (and n square roots)
  - hence its dominant cost is its $n^3/3$ flops
- The dominant cost for the QR decomposition, if computed with householder reflections, is $2n^3/3$
  - so it's roughly twice as expensive as the Cholesky decomposition
- The eigen-decomposition and SVD both require kn^3 flops, for some value k, but $k \gg 1/3$, as in Cholesky algorithm, making them considerably slower for large n

Sherman-Morrison formula / Woodbury matrix identity

- In general, unless we actually need the inverse of a matrix (such as for standard errors of regression coefficients in a linear model), we should solve systems of linear equations
- Sometimes, though, we do need – or might just have – an inverse, and want to calculate something related to it
- The following are a set of formulae, which go by various names, that can be useful in this situation

Woodbury's formula

- Consider an $m \times m$ matrix $\mathbf{A}$, with m large, and for which we have the inverse, $\mathbf{A}^{-1}$
- Suppose $\mathbf{A}$ receives a small update of the form $\mathbf{A} + \mathbf{U}\mathbf{V}^{\mathsf{T}}$, for $m \times n$ matrices $\mathbf{U}$ and $\mathbf{V}$ where $n \ll m$
- Then, by **Woodbury's formula**,

$$(\mathbf{A} + \mathbf{U}\mathbf{V}^{\mathsf{T}})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I}_n + \mathbf{V}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^{\mathsf{T}}\mathbf{A}^{-1}$$

- *Remark*: What's important to note here is that we're looking to calculate an $m \times m$ inverse with m large, and so in general this will be an $O(m^3)$ calculation based on the LHS
- However, in the above, the RHS only requires that we to invert an $n \times n$ matrix, at cost $O(n^3)$, which is much less that of the LHS

Sherman-Morrison-Woodbury formula

- Woodbury's formula above generalises to the so-called **Sherman-Morrison-Woodbury formula** by introducing the $n \times n$ matrix $\mathbf{C}$, so that

$$(\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V}^{\mathsf{T}})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{V}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^{\mathsf{T}}\mathbf{A}^{-1}$$

Challenges III

- Go to Challenges III of the week 5 lecture 1 challenges at https://byoungman.github.io/MTH3045/challenges

Sherman-Morrison formula

- The **Sherman-Morrison formula** is the special case of Woodbury's formula (and hence the Woodbury-Sherman-Morrison formula) in which the update to $\mathbf{A}$ can be considered in terms of m-vectors $\mathbf{u}$ and $\mathbf{v}$, so that

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^{\mathsf{T}})^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^{\mathsf{T}}\mathbf{A}^{-1}}{1 + \mathbf{v}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{u}}$$

- *Remark*: The Sherman-Morrison formula is particularly useful because it requires no matrix inversion

Example: Bayesian linear regression I

- Recall from MTH2006 the normal linear model where

$$Y_i \sim N(\mathbf{x}_i^{\mathsf{T}}\boldsymbol{\beta}, \sigma^2)$$

  with independent errors $\varepsilon_i = Y_i - \mathbf{x}_i^{\mathsf{T}}\boldsymbol{\beta}$s, for $i = 1, \ldots, n$, where $\mathbf{x}_i^{\mathsf{T}} = (1, x_{i1}, \ldots, x_{ip})$ is the ith row of design matrix $\mathbf{X}$ and where $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^{\mathsf{T}}$
- Hence

$$\mathbf{Y} \mid \mathbf{X}\boldsymbol{\beta} \sim MVN_n(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I}_n)$$

- In Bayesian linear regression, the elements of $\boldsymbol{\beta}$ and σ^2 are not fixed, unknown parameters: they are random variables, and we must declare *a priori* our beliefs about their distribution
- The conjugate prior is that

$$\boldsymbol{\beta} \sim MVN_{p+1}(\boldsymbol{\mu}_{\boldsymbol{\beta}}, \boldsymbol{\Sigma}_{\boldsymbol{\beta}}^{-1})$$

- Integrating out $\boldsymbol{\beta}$ gives

$$\mathbf{Y} \sim MVN_n(\mathbf{X}\boldsymbol{\mu}_{\boldsymbol{\beta}}, \sigma^2\mathbf{I}_n + \mathbf{X}\boldsymbol{\Sigma}_{\boldsymbol{\beta}}^{-1}\mathbf{X}^{\mathsf{T}})$$

Example: Bayesian linear regression II

- Now suppose that we want to evaluate the marginal likelihood for an observation, $\mathbf{y}$, say. Recall the MVN pdf
- The Mahalanobis distance then involves the term

$$(\mathbf{y} - \mathbf{X}\boldsymbol{\mu}_\beta)^\mathsf{T}(\sigma^2\mathbf{I}_n + \mathbf{X}\boldsymbol{\Sigma}_\beta^{-1}\mathbf{X}^\mathsf{T})^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\mu}_\beta)$$

- The covariance of the marginal distribution, $\sigma^2\mathbf{I}_n + \mathbf{X}\boldsymbol{\Sigma}_\beta^{-1}\mathbf{X}^\mathsf{T}$, is typically dense, expensive to form, and leads to expensive solutions to systems of linear equations
- Its inverse, however, can be computed through the Sherman-Morrison-Woodbury formula with $\mathbf{A}^{-1} = \sigma^{-2}\mathbf{I}_n$, $\mathbf{U} = \mathbf{V} = \mathbf{X}$, and $\mathbf{C} = \boldsymbol{\Sigma}_\beta^{-1}$

Bibliographic notes

- For more details on matrix decompositions, consider Wood (2015, Appendix B) for a concise overview
- For fuller details consider Monahan (2011, chaps. 3, 4 and 6) or Press et al. (2007, chap. 2 and 11)

Johnson, R. A., and D. W. Wichern. 2007. *Applied Multivariate Statistical Analysis*. 6th ed. Applied Multivariate Statistical Analysis. Pearson Prentice Hall. https://books.google.co.uk/books?id=gFWcQgAACAAJ.

Monahan, John F. 2011. *Numerical Methods of Statistics*. 2nd ed. Cambridge University Press. https://doi.org/10.1017/CBO9780511977176.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. 2007. *Numerical Recipes: The Art of Scientific Computing*. 3rd ed. Cambridge University Press. https://books.google.co.uk/books?id=1aAOdzK3FegC.

Wood, Simon N. 2015. *Core Statistics*. Institute of Mathematical Statistics Textbooks. Cambridge University Press. https://doi.org/10.1017/CBO9781107741973.