MTH3045: Statistical Computing

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

3/2/2025

Matrix-based computing

Week 4 lecture 1

Example: evaluating the multivariate Normal pdf I

- Let $\mathbf{Y} \sim MVN_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ denote a random p-vector with a multivariate Normal (MVN) distribution that has mean vector $\boldsymbol{\mu}$ and variance-covariance matrix $\boldsymbol{\Sigma}$
- Its probability density function (pdf) is then

$$f(\mathbf{y} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}|}} \exp\left\{ -\frac{1}{2} (\mathbf{y} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y} - \boldsymbol{\mu}) \right\}$$

- So, note that to compute the MVN pdf, we need to consider both the determinant and inverse of $\boldsymbol{\Sigma}$, amongst other calculations
- Write a function `dmvn1()` to evaluate its pdf in `R`, and then evaluate $\log f(\mathbf{y} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ for

$$\mathbf{y} = \begin{pmatrix} 0.7 \\ 1.3 \\ 2.6 \end{pmatrix}, \ \boldsymbol{\mu} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \text{ and } \boldsymbol{\Sigma} = \begin{pmatrix} 4 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 2 \end{pmatrix}$$

Example: evaluating the multivariate Normal pdf II

- The function `dmvn1()` below evaluates the multivariate Normal pdf

$$f(\mathbf{y} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}|}} \exp\left\{ -\frac{1}{2} (\mathbf{y} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y} - \boldsymbol{\mu}) \right\}$$

```
dmvn1 <- function(y, mu, Sigma, log = TRUE) {
  # Function to evaluate multivariate Normal pdf
  # at y given mean mu and variance-covariance
  # matrix Sigma.
  # Returns 1x1 matrix, on log scale, if log == TRUE.
  p <- length(y)
  res <- y - mu
  out <- - 0.5 * determinant(Sigma)$modulus - 0.5 * p * log(2 * pi) -
             0.5 * t(res) %*% solve(Sigma) %*% res
  if (!log)
    out <- exp(out)
  out
}
```

- Note that above `determinant()$modulus` directly calculates `log(det())`, and is usually more reliable, so should be used when possible
- We'll later see that this is a crude attempt

Example: evaluating the multivariate Normal pdf III

- The following create $\mathbf{y}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ as objects `y`, `mu` and `Sigma`, respectively.

```
y <- c(.7, 1.3, 2.6)
mu <- 1:3
Sigma <- matrix(c(4, 2, 1, 2, 3, 2, 1, 2, 2), 3, 3)
```

- Then we evaluate $\log f(\mathbf{y} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with

```
dmvn1(y, mu, Sigma)
```

```
##              [,1]
## [1,] -3.654535
## attr(,"logarithm")
## [1] TRUE
```

- *Remark*: It's usually much more sensible to work with log-likelihoods, and then if the likelihood itself is actually sought, simply exponentiate the log-likelihood at the end
  - this has been implemented for `dmvn1()`
  - this will sometimes avoid underflow

Challenges I

- Go to Challenges I of the week 4 lecture 1 challenges at https://byoungman.github.io/MTH3045/challenges

Special matrices

Diagonal, band-diagonal and triangular matrices I

- The following gives examples of various special types of square matrix, which we sometimes encounter in statistical computing
- These are diagonal (as defined above), tridiagonal, block diagonal, band and lower triangular matrices
- Instead of defining them formally, we'll just show schematics of each
- These are plotted with `image()`
  - it plots the rows along the x-axis and columns across the y-axis
  - to visualise actually looking at the matrix written down on paper, each plot should be considered rotated clockwise through 90 degrees

Diagonal, band-diagonal and triangular matrices II

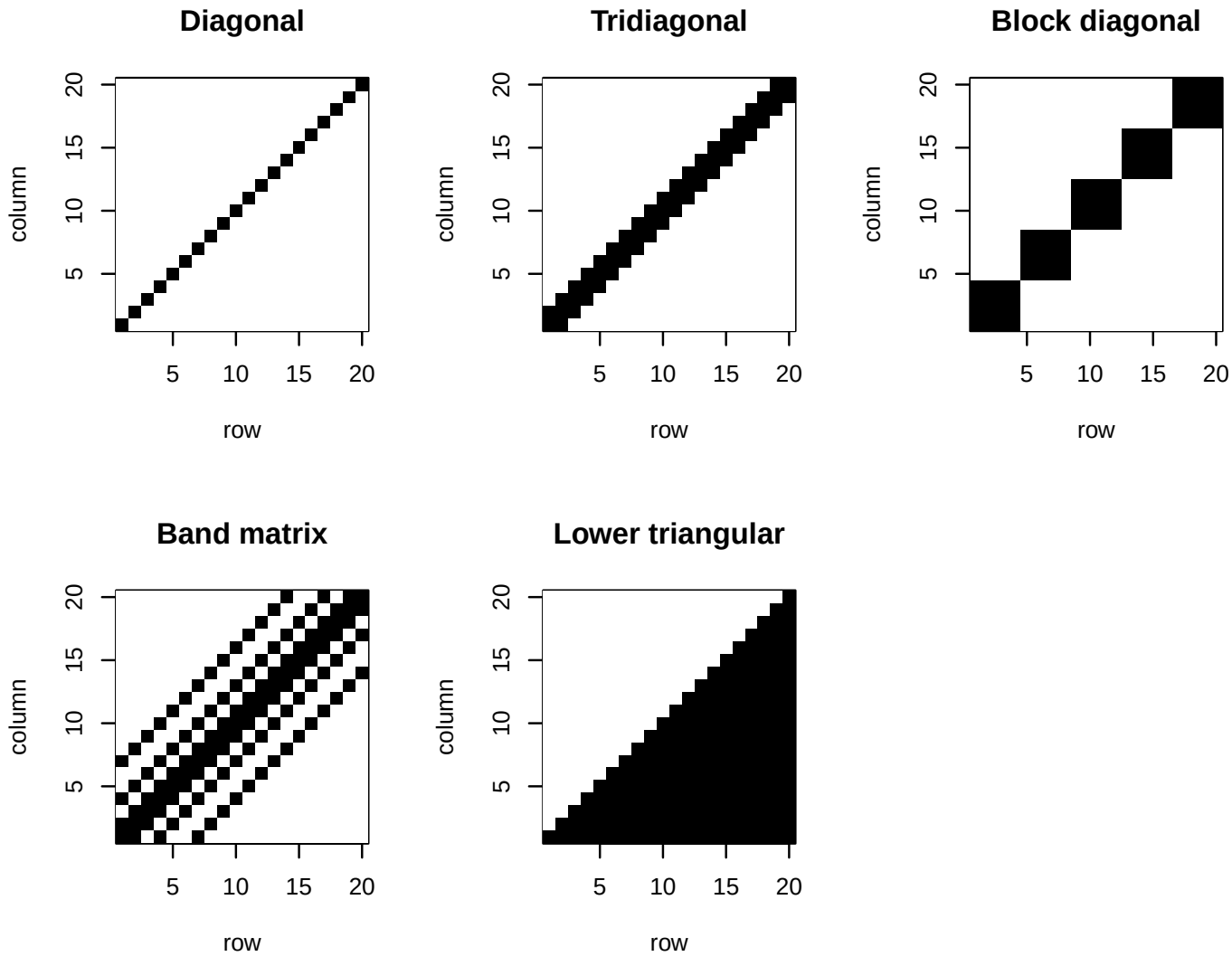

Figure 1: Schematics of diagonal, tridiagonal, block diagonal, band and lower triangular matrices.

Sparse matrices

- **Definition**: A matrix is **sparse** if most of its elements are zero.
- *Remark*: The definition of a sparse matrix is rather vague
  - although no specific criterion exists in terms of the proportion of zeros, some consider that the number of non-zero elements should be similar to the number of rows or columns
- A diagonal matrix is sparse.

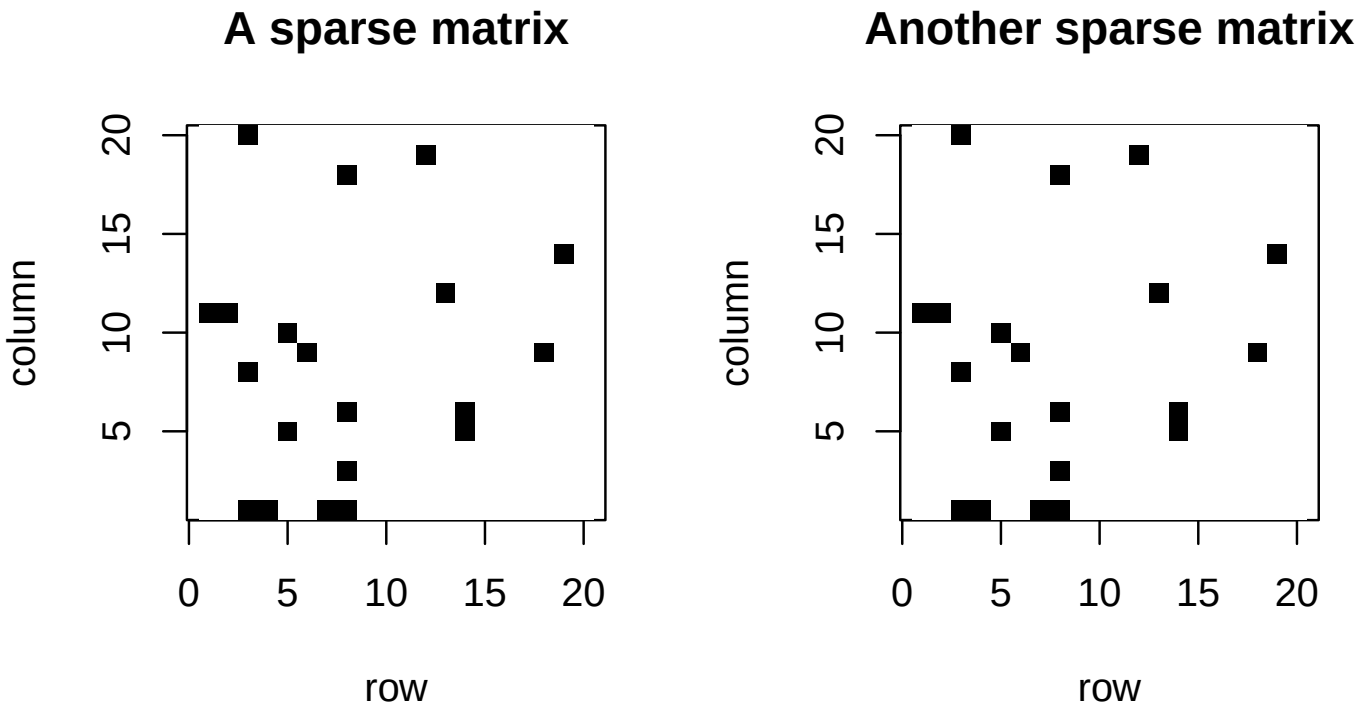

Figure 2: Schematics of two sparse matrices.

Systems of linear equations I

- Systems of linear equations of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

  where $\mathbf{A}$ is an $n \times n$ matrix and $\mathbf{x}$ and $\mathbf{b}$ are n-vectors are often encountered in statistical computing
- The multivariate Normal pdf is one example: we don't need to compute $\mathbf{\Sigma}^{-1}$ and then calculate $\mathbf{z} = \mathbf{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu})$
  - instead, left-multiplying by $\mathbf{\Sigma}$, we can recognise that $\mathbf{z}$ is the solution to $\mathbf{\Sigma}\mathbf{z} = \mathbf{y} - \boldsymbol{\mu}$
- `R`'s `solve()` function can not only invert a matrix, but can also solve a system of linear equations
- If $\mathbf{A}\mathbf{x} = \mathbf{b}$ and we have $\mathbf{A}$ and $\mathbf{b}$ stored as `A` and `b`, respectively, then we obtain $\mathbf{x}$, which we'll store as `x`, with `x <- solve(A, b)`
- *Remark*: In general, solving a system of linear equations is faster and more numerically stable than inverting and multiplying
  - the latter essentially results from reducing numerical errors; see Ch. 2

Example: return of the multivariate Normal pdf

- Modify the function `dmvn1()` used before to give a new function `dmvn2()` in which, instead of inverting $\mathbf{\Sigma}$, the system of linear equations $\mathbf{\Sigma z} = \mathbf{y} - \boldsymbol{\mu}$ is solved for $\mathbf{z}$
- We simply need to replace `solve(Sigma) %*% res` with `solve(Sigma, res)`, giving `dmvn2()` as follows

```
dmvn2 <- function(y, mu, Sigma, log = TRUE) {
# Function to evaluate multivariate Normal pdf
# at y given mean mu and variance-covariance
# matrix Sigma.
# Returns scalar, on log scale if log == TRUE
p <- length(y)
res <- y - mu
out <- - 0.5 * determinant(Sigma)$modulus - 0.5 * p * log(2 * pi) -
           0.5 * t(res) %*% solve(Sigma, res)
if (!log)
  out <- exp(out)
as.vector(out)
}
```

which reassuringly gives the same answer as `dmvn1()`.

```
dmvn2(y, mu, Sigma)
```

```
## [1] -3.654535
```

Systems of linear equations II

- **Definition**: An **elementary row operation** on a matrix is any one of the following.
  - Type-I: interchange two rows of the matrix;
  - Type-II: multiply a row by a nonzero scalar;
  - Type-III: replace a row by the sum of that row and a scalar multiple of another row.

- **Definition**: A matrix $\mathbf{U}$ is said to be in **row echelon form** if the following two conditions hold.
  - If a row $\mathbf{u}_{i*}^{\mathsf{T}}$ comprises all zeros, i.e. $\mathbf{u}_{i*}^{\mathsf{T}} = \mathbf{0}^{\mathsf{T}}$, then all rows below also comprise all zeros.
  - If the first nonzero element of $\mathbf{u}_{i*}^{\mathsf{T}}$ is the jth element, then the jth element in all rows below is zero.

Example: Gaussian elimination – the full rank case I

- **Gaussian elimination** is perhaps the best established method for solving systems of linear equations
- In MTH3045 you won't be examined on Gaussian elimination
  - but it will be useful to be familiar with how it works
  - then the virtues of the matrix decompositions that follow will become apparent
- The system of linear equations $\mathbf{Ax} = \mathbf{b}$ may be verbosely written as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n \end{aligned}$$

Example: Gaussian elimination – the full rank case II

- Gaussian elimination aims to use elementary row operations to transform the previous set of equations into an equivalent but triangular system
- Instead of algebraically writing the algorithm for Gaussian elimination, it will be simpler to consider a numerical example in which we want to solve the following system of four equations in four variables

$$\begin{array}{rcrcrcrcl} 2x_1 & + & 3x_2 & & & & & = & 1 \\ 4x_1 & + & 7x_2 & + & 2x_3 & & & = & 2 \\ -6x_1 & - & 10x_2 & & & + & x_4 & = & 1 \\ 4x_1 & + & 6x_2 & + & 4x_3 & + & 5x_4 & = & 0 \end{array}$$

and for which we'll write the coefficients of the x_is and $\mathbf{b}$ in the augmented and more convenient matrix form

$$\left[\begin{array}{rrrr|r} 2 & 3 & 0 & 0 & 1 \\ 4 & 7 & 2 & 0 & 2 \\ -6 & -10 & 0 & 1 & 1 \\ 4 & 6 & 4 & 5 & 0 \end{array}\right]$$

Example: Gaussian elimination – the full rank case III

- We start by choosing the **pivot**
- Our first choice is a coefficient of x_1
  - We can choose any nonzero coefficient
- Anything below this is set to zero through elementary operations
- We'll choose a_{11} as the pivot and then perform the following elementary matrix operations
  - row 2 $\rightarrow$ 2 $\times$ row 1 + - 1 $\times$ row 2
  - row 3 $\rightarrow$ 3 $\times$ row 1 + 1 $\times$ row 3
  - row 4 $\rightarrow$ -2 $\times$ row 1 + row 4
- These give the following transformation of the above augmented matrix

$$\left[\begin{array}{rrrr|r} 2 & 3 & 0 & 0 & 1 \\ 4 & 7 & 2 & 0 & 2 \\ -6 & -10 & 0 & 1 & 1 \\ 4 & 6 & 4 & 5 & 0 \end{array}\right] \rightarrow \left[\begin{array}{rrrr|r} 2 & 3 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 1 & 4 \\ 0 & 0 & 4 & 5 & -2 \end{array}\right]$$

Example: Gaussian elimination – the full rank case IV

- Repeating this with the element in the position of a_{22} we get

$$\left[\begin{array}{cccc|c} 2 & 3 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 1 & 4 \\ 0 & 0 & 4 & 5 & -2 \end{array}\right] \rightarrow \left[\begin{array}{cccc|c} 2 & 3 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 & 4 \\ 0 & 0 & 4 & 5 & -2 \end{array}\right]$$

- And then again with the element in the position of a_{33} we get

$$\left[\begin{array}{cccc|c} 2 & 3 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 1 & 4 \\ 0 & 0 & 4 & 5 & -2 \end{array}\right] \rightarrow \left[\begin{array}{cccc|c} 2 & 3 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 & 4 \\ 0 & 0 & 0 & 3 & -10 \end{array}\right]$$

Example: Gaussian elimination – the full rank case V

- The previous operations have **triangularised** the system of linear equations, i.e. produced an augmented matrix of the form

$$\left[\begin{array}{cccc|c} u_{11} & u_{12} & \dots & u_{1n} & b_1^* \\ 0 & u_{22} & \dots & u_{2n} & b_2^* \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & u_{nn} & b_n^* \end{array}\right]$$

- This can be tidily written as $\mathbf{U}\mathbf{x} = \mathbf{b}^*$
- It is straightforward to find $\mathbf{x}$ from such a system
  - because $x_4 = -10/3$, which can be substituted to give $x_3 = 11/3$, and so forth gives $x_2 = -22/3$ and $x_1 = 23/2$
  - i.e. $\mathbf{x} = (23/2, -22/3, 11/3, -10/3)^\mathsf{T}$
- The above is an example of **backward substitution**

Challenges II

- Go to Challenges II of the week 4 lecture 1 challenges at https://byoungman.github.io/MTH3045/challenges

Backward substitution

- **Definition**: Consider the system of linear equations given by $\mathbf{Ux} = \mathbf{b}$, where $\mathbf{U}$ is an $n \times n$ upper triangular matrix
- We can find $\mathbf{x}$ by **backward substitution** through the following steps.

1. Calculate $x_n = b_n/u_{nn}$.
2. For $i = n-1, n-2, \ldots, 2, 1$, recursively compute

$$x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right).$$

- Gaussian elimination is the two-stage process of forming the triangular matrix and then performing backward substitution.
- Note that **forward substitution** is simply the analogous process of backward substitution where we find a lower triangular matrix, and then solve for x_1, x_2 given x_1, and so forth.

Backward and forward substitution in R

- *Remark*: If we want to perform backward or forward substitution in R we should use `backsolve()` and `forwardsolve()`, respectively
- These have usage

```
backsolve(r, x, k = ncol(r), upper.tri = TRUE, transpose = FALSE)
forwardsolve(l, x, k = ncol(l), upper.tri = FALSE, transpose = FALSE)
```

- `backsolve()` expects a right upper-triangular matrix
- `forwardsolve()` expects a left lower-triangular matrix.

Example: Backward and forward substitution in R

- Confirm that $\mathbf{x} = (23/2, -22/3, 11/3, -10/3)^\mathsf{T}$ using `backsolve()`
- We need to input the upper-triangular matrix $\mathbf{U}$ and $\mathbf{b}^*$, which we'll call `U` and `bstar`, respectively.

```
U <- rbind(
  c(2, 3, 0, 0),
  c(0, 1, 2, 0),
  c(0, 0, 2, 1),
  c(0, 0, 0, 3)
)
bstar <- c(1, 0, 4, -10)
backsolve(U, bstar)
```

```
## [1] 11.500000 -7.333333  3.666667 -3.333333
```

- *Remark*: We may want to solve multiple systems of linear equations of the form $\mathbf{A}\mathbf{x}_1 = \mathbf{b}_1$, $\mathbf{A}\mathbf{x}_2 = \mathbf{b}_2$, ..., $\mathbf{A}\mathbf{x}_p = \mathbf{b}_p$, which can be written with matrices as $\mathbf{A}\mathbf{X} = \mathbf{B}$ for $n \times p$ matrices $\mathbf{X}$ and $\mathbf{B}$
  - here we only triangularise $\mathbf{A}$ once and then use that triangularisation to go through the back substitution algorithm p times
- *Remark*: We can find the inverse of $\mathbf{A}$ by solving $\mathbf{A}\mathbf{X} = \mathbf{I}_n$ for $\mathbf{X}$ and then setting $\mathbf{A}^{-1} = \mathbf{X}$.

Reduced row echelon form

- **Definition**: A $m \times n$ matrix **U** is said to be in **reduced row echelon form** if the following two conditions hold.

1. It is in row echelon form;
2. The first nonzero element of each row is one;
3. All entries above each pivot are zero.

Week 4 lecture 2

Example: Triangular, forward and backward solving I

- Use `R` to find $\mathbf{x}$ such that $\mathbf{Dx} = \mathbf{b}$ and $\mathbf{Lx} = \mathbf{b}$ and $\mathbf{Ux} = \mathbf{b}$ where

$$\mathbf{D} = \begin{pmatrix} -0.75 & 0.00 & 0.00 \\ 0.00 & -0.61 & 0.00 \\ 0.00 & 0.00 & -0.28 \end{pmatrix}, \quad \mathbf{U} = \begin{pmatrix} 1.00 & -0.19 & 0.89 \\ 0.00 & 0.43 & 0.02 \\ 0.00 & 0.00 & -0.20 \end{pmatrix},$$

$$\mathbf{L} = \begin{pmatrix} -0.72 & 0.00 & 0.00 \\ 0.00 & 2.87 & 0.00 \\ -1.94 & -2.04 & 0.81 \end{pmatrix} \text{ and } \mathbf{b} = \begin{pmatrix} -2.98 \\ 0.39 \\ 0.36 \end{pmatrix}.$$

- We'll start by loading $\mathbf{b}$, $\mathbf{D}$, $\mathbf{U}$ and $\mathbf{L}$ and which we'll store as `b`, `D`, `U` and `L`, respectively.

```
D <- diag(c(-0.75, -0.61, -0.28))
L <- matrix(c(-0.72, 0, -1.94, 0, 2.87, -2.04, 0, 0, 0.81), 3, 3)
U <- matrix(c(1, 0, 0, -0.19, 0.43, 0, 0.89, 0.02, -0.2), 3, 3)
b <- c(-2.98, 0.39, 0.36)
```

Example: Triangular, forward and backward solving II

- We can solve $\mathbf{Dx} = \mathbf{b}$ for $\mathbf{x}$ with the following two lines of code

```
solve(D, b)
```

```
## [1]  3.9733333 -0.6393443 -1.2857143
```

```
b / diag(D)
```

```
## [1]  3.9733333 -0.6393443 -1.2857143
```

  but should note that the latter uses fewer calculations, and so is more efficient and hence better

Example: Triangular, forward and backward solving III

- Then we can solve $\mathbf{Ux} = \mathbf{b}$ for $\mathbf{x}$ with the following two lines of code

```
solve(U, b)
```

```
## [1] -1.1897674  0.9906977 -1.8000000
```

```
backsolve(U, b)
```

```
## [1] -1.1897674  0.9906977 -1.8000000
```

- We can then solve $\mathbf{Lx} = \mathbf{b}$ for $\mathbf{x}$ with the following two lines of code

```
solve(L, b)
```

```
## [1]  4.1388889  0.1358885 10.6995765
```

```
forwardsolve(L, b)
```

```
## [1]  4.1388889  0.1358885 10.6995765
```

- On both these occasions the latter is more efficient because it uses fewer calculations.

Challenges I

- Go to Challenges I of the week 4 lecture 2 challenges at https://byoungman.github.io/MTH3045/challenges

Example: Gaussian elimination – rank deficiant case I

- Consider Gaussian elimination of the 3×3 matrix $\mathbf{A}$ given by

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix}$$

- We can go through the following steps to transform the matrix to reduced row echelon form.

$$\begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & -1 & -3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & -5 \\ 0 & 1 & 3 \\ 0 & 0 & 0 \end{bmatrix}$$

- We've ended up with only two nonzero rows, and hence $\mathbf{A}$ has rank 2, and because it has three rows it is therefore **rank deficient**.

Matrix decompositions

Cholesky decomposition I

- **Definition**: Any positive definite real matrix $\mathbf{A}$ can be factorised as

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$$

  where $\mathbf{L}$ is a real lower-triangular matrix of the same dimension as $\mathbf{A}$ with positive diagonal entries

- The factorisation above is the **Cholesky decomposition**[1].

[1] André-Louis Cholesky (15 Oct 1875 – 31 Aug 1918) was a French military officer and mathematician. He worked in geodesy and cartography, and was involved in the surveying of Crete and North Africa before World War I. He is primarily remembered for the development of a matrix decomposition known as the Cholesky decomposition which he used in his surveying work. His discovery was published posthumously by his fellow officer Commandant Benoît in the Bulletin Géodésique.

Cholesky decomposition II

- We won't usually be concerned with algorithms for computing matrix decompositions in MTH3045
- However, the algorithm for computing the Cholesky decomposition is rather elegant, and so is given below
- You won't, however, be expected to use it
- Consider the following matrices

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \qquad \mathbf{L} = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix}$$

  where $\mathbf{A}$ is symmetric and non-singular
- The entries of $\mathbf{L}$ are given by

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}, \qquad l_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk}}{l_{ii}}, \text{ for } i > j.$$

Cholesky decomposition III

- To compute the Cholesky decomposition in `R` we use

```
chol(A) # computes the Cholesky decomposition of a square matrix A
```

- *Remark*: The `chol()` function in `R` returns an upper-triangular decomposition
  - i.e. returns $\mathbf{U}$ for $\mathbf{A} = \mathbf{U}^\top \mathbf{U}$
  - to obtain $\mathbf{L}$ we just use `t(chol())`

Example: Cholesky decomposition in R

- Compute the Cholesky decomposition of $\boldsymbol{\Sigma}$ from Example 3.2 in upper- and lower-triangular form, and verify that both are Cholesky decompositions of $\boldsymbol{\Sigma}$

```
U <- chol(Sigma) # upper-triangular form
all.equal(crossprod(U), Sigma)
```

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

```
L <- t(U) # lower-triangular form
all.equal(tcrossprod(L), Sigma)
```

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

- *Remark*: Above, instead of `L <- t(chol(Sigma))` we've used `L <- t(U)` to avoid repeated calculation of `chol(Sigma)`
- In this example, where we compute the Cholesky decomposition of a 3×3 matrix, the calculation is trivial
  - however, for much larger matrices, calculating the Cholesky decomposition can be expensive, and so we could gain significant time by only calculating it once

Properties of the Cholesky decomposition

- Once a Cholesky decomposition has been calculated, it can be used to calculate determinants and inverses
  - $\det(\mathbf{A}) = \left(\prod_{i=1}^{n} l_{ii}\right)^2$
  - $\mathbf{A}^{-1} = \mathbf{L}^{-\mathsf{T}}\mathbf{L}^{-1}$, where $\mathbf{L}^{-\mathsf{T}}$ denotes the inverse of $\mathbf{L}^{\mathsf{T}}$

Example: Determinant and inverse via the Cholesky decomposition I

- For $\mathbf{\Sigma}$ from Example 3.2, compute $\det(\mathbf{\Sigma})$ and $\mathbf{\Sigma}^{-1}$ based on either Cholesky decomposition computed in Example 3.8. Verify your results.
- We'll start with the determinant

```
det1 <- det(Sigma)
det2 <- prod(diag(L))^2
all.equal(det1, det2)
```

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

Example: Determinant and inverse via the Cholesky decomposition II

- We then have various options for the inverse

```
inv1 <- solve(Sigma)
inv2 <- crossprod(solve(L), solve(L))
all.equal(inv1, inv2)
```

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

```
inv3 <- crossprod(solve(L))
all.equal(inv1, inv3)
```

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

```
inv4 <- solve(t(L), solve(L))
all.equal(inv1, inv4)
```

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

```
inv5 <- chol2inv(t(L))
all.equal(inv1, inv5)
```

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

which all give the same answer, although `chol2inv()` should be our default

Challenges II

- Go to Challenges II of the week 4 lecture 2 challenges at https://byoungman.github.io/MTH3045/challenges

Week 4 lecture 3

Solving systems of linear equations

- We can also use a Cholesky decomposition to solve a system of linear equations
- Solving $\mathbf{Ax} = \mathbf{b}$ is equivalent to solving $\mathbf{Ly} = \mathbf{b}$ for $\mathbf{y}$ and then $\mathbf{L}^{\mathsf{T}}\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$
- This might seem inefficient at first glance, because we're having to solve two systems of linear equations
  - however, $\mathbf{L}$ being triangular means that forward elimination is efficient for $\mathbf{L}$, and backward elimination is for $\mathbf{L}^{\mathsf{T}}$

Example: Solving linear systems with Cholesky decompositions I

- Recall the multivariate Normal pdf of Example 3.2 in which we needed $\mathbf{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu})$, with $\mathbf{y}$, $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ as given in Example 3.2
- Compute $\mathbf{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu})$ by solving $\mathbf{\Sigma z} = \mathbf{y} - \boldsymbol{\mu}$ for $\mathbf{z}$ using the Cholesky decomposition of $\mathbf{\Sigma}$ from Example 3.8
- Verify your answer

Example: Solving linear systems with Cholesky decompositions II

- We'll first use `solve()` on $\mathbf{\Sigma}$, as in Example 3.3

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

- Then we'll solve $\mathbf{Lx} = \mathbf{y} - \boldsymbol{\mu}$ for $\mathbf{x}$ followed by $\mathbf{L}^\top \mathbf{z} = \mathbf{x}$ for $\mathbf{z}$.

```
x <- solve(L, y - mu)
res2 <- solve(t(L), x)
```

  which we can confirm gives the same as above, i.e. `res1`, with `all.equal()`

```
all.equal(res1, res2)
```

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

Cholesky decomposition and backward and forward substitution

- We can tell `R` to use forward substitution, by calling function `forwardsolve()` instead of `solve()`
- Or we can tell `R` to use backward substitution, by calling function `backsolve()`
- Then `R` knows that one triangle of the supplied matrix comprises zeros, which speeds up solving the system of linear equations
- Solving via the Cholesky decomposition is also more stable than without it
- Otherwise, if we just used `solve()`, `R` performs a lot of needless calculations on zeros, because it doesn't know that they're zeros

Example: Solving triangular linear systems with Cholesky decompositions I

- Repeat Example 3.10 by recognising that the Cholesky decomposition of $\mathbf{\Sigma}$ is triangular
- We want to use `forwardsolve()` to solve $\mathbf{L}\mathbf{x} = \mathbf{y} - \boldsymbol{\mu}$ and then `backsolve()` to solve $\mathbf{L}^\mathsf{T}\mathbf{z} = \mathbf{x}$
- However, `backsolve()` expects an upper-triangular matrix, so we must use $\mathbf{L}^\mathsf{T}$, hence `t(L)` below

```
x2 <- forwardsolve(L, y - mu)
res3 <- backsolve(t(L), x2)
all.equal(res1, res3)
```

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

Example: Solving triangular linear systems with Cholesky decompositions II

- We can avoid the transpose operation by letting `backsolve()` know the format of Cholesky decomposition that we're supplying
- We're supplying a lower-triangular matrix, hence `upper.tri = FALSE`, which needs transposing to be upper-triangular, hence `transpose = TRUE`

```
res4 <- backsolve(L, forwardsolve(L, y - mu),
                  upper.tri = FALSE, transpose = TRUE)
all.equal(res1, res4)
```

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

- If we begin with an upper-triangular matrix, `U = t(L)`, then we'd want to use either of the following

```
res5 <- forwardsolve(U, backsolve(U, y - mu, transpose = TRUE),
                     upper.tri = TRUE)
all.equal(res1, res5)
```

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

```
res6 <- backsolve(U, forwardsolve(U, y - mu, upper.tri = TRUE,
                     transpose = TRUE))
all.equal(res1, res6)
```

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

Challenges I

- Go to Challenges I of the week 4 lecture 3 challenges at https://byoungman.github.io/MTH3045/challenges

Mahalanobis distance

- **Definition**: Given the p-vectors $\mathbf{x}$ and $\mathbf{y}$ and a variance-covariance matrix $\mathbf{\Sigma}$, the **Mahalanobis distance** is defined as

$$D_M(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{y} - \mathbf{x})^\top \mathbf{\Sigma}^{-1} (\mathbf{y} - \mathbf{x})}.$$

Example: Mahalanobis distance via Cholesky decomposition

- We can efficiently compute the Mahalanobis distance using the Cholesky decomposition
- Consider $\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$ so that $\mathbf{\Sigma}^{-1} = \mathbf{L}^{-\mathsf{T}}\mathbf{L}^{-1}$
- Then

$$\begin{aligned}\left[D_M(\mathbf{x}, \mathbf{y})\right]^2 &= (\mathbf{y} - \mathbf{x})^{\mathsf{T}}\mathbf{L}^{-\mathsf{T}}\mathbf{L}^{-1}(\mathbf{y} - \mathbf{x}) \\ &= [\mathbf{L}^{-1}(\mathbf{y} - \mathbf{x})]^{\mathsf{T}}\mathbf{L}^{-1}(\mathbf{y} - \mathbf{x}) \\ &= \mathbf{z}^{\mathsf{T}}\mathbf{z}\end{aligned}$$

where $\mathbf{z}$ is the solution of $\mathbf{L}\mathbf{z} = \mathbf{y} - \mathbf{x}$.

Example: Mahalanobis distance via Cholesky decomposition

- If we have $\mathbf{x}$, $\mathbf{y}$ and the lower-triangular Cholesky decomposition of $\mathbf{\Sigma}$ stored as `x`, `y` and `L`, respectively, then we can efficiently compute the Mahalanobis distance in `R` with

```
sqrt(crossprod(forwardsolve(L, y - x)))
```

- Note that we may want to simplify the use of `crossprod()` and use `sqrt(sum(forwardsolve(L, y - x)^2))` instead

Example: Evaluating the multivariate Normal pdf using the Cholesky decomposition I

- Create a function `dmvn3()` that evaluates the multivariate Normal pdf, as in examples 3.2 and 3.3, based on a Cholesky decomposition of the variance-covariance matrix $\mathbf{\Sigma}$
- Verify your function using $\mathbf{y}$, μ, and $\mathbf{\Sigma}$ given in Example 3.2

Example: Evaluating the multivariate Normal pdf using the Cholesky decomposition II

- We first recognise that, if matrix $\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$, then $\log(\det(\mathbf{\Sigma})) = 2\sum_{i=1}^{n} \log(l_{ii})$, which we'll incorporate in `dmvn3()`

```
dmvn3 <- function(y, mu, Sigma, log = TRUE) {
# Function to evaluate MVN pdf
# y, mu vectors
# Sigma matrix
# log is logical indicating whether logarithm
# returns a scalar
p <- length(y)
res <- y - mu
L <- t(chol(Sigma))
out <- - sum(log(diag(L))) - 0.5 * p * log(2 * pi) -
           0.5 * sum(forwardsolve(L, res)^2)
if (!log)
  out <- exp(out)
out
}
```

together with the result on evaluating the Mahalanobis distance

```
dmvn3(y, mu, Sigma)
```

```
## [1] -3.654535
```

Example: Generating multivariate Normal random vectors I

- We can generate $\mathbf{Y} \sim MVN_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, i.e. multivariate Normal random vectors with mean $\boldsymbol{\mu}$ and variance-covariance matrix $\boldsymbol{\Sigma}$, using the following algorithm.
- Step 1. Find some matrix $\mathbf{L}$ such that $\mathbf{L}\mathbf{L}^{\mathsf{T}} = \boldsymbol{\Sigma}$.
- Step 2. Generate $\mathbf{Z}^{\mathsf{T}} = (Z_1, \ldots, Z_p)$, where Z_i, $i = 1, \ldots, p$, are independent $N(0, 1)$ random variables.
- Step 3. Set $\mathbf{Y} = \boldsymbol{\mu} + \mathbf{L}\mathbf{Z}$.
- The Cholesky decomposition clearly meets the criterion for $\mathbf{L}$ in Step 1.

Example: Generating multivariate Normal random vectors II

- In R, we can write a function, `rmvn()`, to implement generating multivariate Normal random vectors
- Write a function in R to generate n independent $MVN_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ random vectors.
- Suppose that $n =$ `n`, $\boldsymbol{\mu} =$ `mu` and $\boldsymbol{\Sigma} =$ `Sigma`, then we can use function `rmvn()` below.

```
rmvn <- function(n, mu, Sigma) {
# Function to generate n MVN random vectors
# mean vector mu
# variance-covariance matrix Sigma
# integer n
# returns p x n matrix
p <- length(mu)
L <- t(chol(Sigma))
Z <- matrix(rnorm(p * n), nrow = p)
as.vector(mu) + L %*% Z
}
```

Example: Generating multivariate Normal random vectors III

- Generate six with $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ as in Example 3.2.

```
rmvn(6, mu, Sigma)
```

```
##             [,1]       [,2]     [,3]     [,4]      [,5]      [,6]
## [1,] -3.8209259 -0.5535068 1.619784 2.366657 3.0029217 0.4362218
## [2,]  0.2070095 -1.1829378 4.062268 3.420571 0.6487868 3.2778557
## [3,]  2.8769105  0.9298969 3.546394 4.268793 2.5136741 6.0221883
```

Bibliography