Lab 5: Numerical linear algebra

MCHA4400

Semester 2 2025

Introduction

In this lab, you will perform matrix and array manipulation and some linear algebra operations using the C++ template library, Eigen. In addition, you will write some unit tests using doctest to validate the solutions.

The lab should be completed within 4 hours. The assessment will be done in the lab at the end of your enrolled lab session(s). Once you complete the tasks, call the lab demonstrator to start your assessment.

The lab is worth 5% of your course grade and is graded from 0–5 marks.

GenAI Tip

You are strongly encouraged to explore the use of Large Language Models (LLMs), such as GPT, Gemini or Claude, to assist in completing this activity. If you do not already have access to an LLM, bots are available to use via the UoN Mechatronics Slack team, which you can find a link to from Canvas. If you are are unsure how to make the best use of these tools, or are not getting good results, please ask your lab demonstrator for advice.

1 Preliminaries (1 mark)

In this task you will become acquainted with basic Eigen matrix and array operations. Complete the following tasks in the `src/main.cpp` file.

Tip

As you complete the tasks below, keep the quick reference guide open as you will be frequently referring to it. If you are already familiar with MATLAB, also keep the ASCII quick reference open to find many of the equivalent operations in Eigen.

Tasks

a) Create a vector $\mathbf{x} \in \mathbb{R}^3$, of type `Eigen::VectorXd`, such that $\mathbf{x} = \begin{bmatrix} 1 & 3.2 & 0.01 \end{bmatrix}^\mathsf{T}$. Use the `<<` assignment operator to populate $\mathbf{x}$. Generate the following output:

Terminal

```
nerd@basement:~/MCHA4400/lab5/build$ ninja && ./lab5
Output from other tasks
Create a column vector:
x =
[   1;
  3.2;
 0.01]
Output from other tasks
nerd@basement:~/MCHA4400/lab5/build$
```

b) Create a matrix $\mathbf{A} \in \mathbb{R}^{4\times 3}$ with elements $A_{i,j} = ij$ for $i = 1, 2, 3, 4$ and $j = 1, 2, 3$. Print the `size`, `rows`, `cols` of $\mathbf{A}$ as well as $\mathbf{A}$ and $\mathbf{A}^\mathsf{T}$ to the console.

Terminal

```
nerd@basement:~/MCHA4400/lab5/build$ ninja && ./lab5
Output from other tasks
Create a matrix:
A.size() = ?
A.rows() = ?
A.cols() = ?
A =
[?  ?  ?;
 ?  ?  ?;
 ?  ?  ?;
 ?  ?  ?]

A.transpose() =
[?  ?  ?  ?;
 ?  ?  ?  ?;
 ?  ?  ?  ?]
Output from other tasks
nerd@basement:~/MCHA4400/lab5/build$
```

c) Use the `*` operator to compute the matrix-vector product $\mathbf{Ax}$ to generate the following output:

Terminal

```
nerd@basement:~/MCHA4400/lab5/build$ ninja && ./lab5
Output from other tasks
Matrix multiplication:
A*x =
[ 7.43;
 14.86;
 22.29;
 29.72]
Output from other tasks
nerd@basement:~/MCHA4400/lab5/build$
```

d) Using the `<<` operator, create a matrix $\mathbf{B} \in \mathbb{R}^{4\times 6}$ such that $\mathbf{B} = \begin{bmatrix}\mathbf{A} & \mathbf{A}\end{bmatrix}$ and create a matrix $\mathbf{C} \in \mathbb{R}^{8\times 3}$ such that $\mathbf{C} = \begin{bmatrix}\mathbf{A} \\ \mathbf{A}\end{bmatrix}$. Output the both $\mathbf{B}$ and $\mathbf{C}$ to the console.

e) Extract a submatrix $\mathbf{D} \in \mathbb{R}^{1\times 3}$ from $\mathbf{B}$ such that

$$D_{i,j} = B_{1+i,\ 2+j}, \qquad \forall i, j, \tag{1}$$

and output $\mathbf{D}$ to the console using each of the following methods:

i) a block operation,

ii) a slicing operation.

f) Use `rowwise` to compute the matrix $\mathbf{E}$ such that,

$$E_{i,j} = B_{i,j} + v_j, \qquad \forall i, j \tag{2}$$

where $\mathbf{v} = \begin{bmatrix}1 & 3 & 5 & 7 & 4 & 6\end{bmatrix}^\mathsf{T}$. Output $\mathbf{E}$ to the console.

g) Generate the matrix $\mathbf{F} \in \mathbb{R}^{4\times 6}$ from $\mathbf{B}$ such that

$$F_{i,j} = B_{r_i,c_j}, \qquad \forall i,j, \tag{3}$$

where $\mathbf{r} = \begin{bmatrix}1 & 3 & 2 & 4\end{bmatrix}$ and $\mathbf{c} = \begin{bmatrix}1 & 4 & 2 & 5 & 3 & 6\end{bmatrix}$ are arrays of indices, without using a `for` loop. Output $\mathbf{F}$ to the console.

h) An existing array has elements `array` $= \begin{bmatrix}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9\end{bmatrix}$ in memory. Use an `Eigen::Map` to create a view of this memory as a 3×3 matrix, $\mathbf{G}$, assuming row-major storage. Then, write -3 to the third element of `array` and write -7 to $G_{3,1}$ and print $\mathbf{G}$ to the console to verify that $\mathbf{G}$ and `array` use the same memory.

Info

`Eigen::Map` can also be used to create a view of OpenCV matrices via `cv::Mat::ptr` (for dynamically-sized matrices) or `cv::Matx<>::val` (for fixed-size matrices, e.g., `cv::Matx33d` or `cv::Vec3d`). This is convenient, since it enables non-Eigen objects access to the full power of the Eigen library and its interface without unnecessarily duplicating data.

2 Sum of Gaussian random variables (1 mark)

Definition

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, the QR decomposition produces matrices $\mathbf{Q} \in \mathbb{R}^{m\times m}$ and $\mathbf{R} \in \mathbb{R}^{m\times n}$ such that $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix.

The QR decomposition can be used to propagate a square-root covariance matrix through affine transformations without needing to explicitly compute a matrix square root. As an example of such a transformation, consider the sum of two independent n-dimensional Gaussian random variables, $\mathbf{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \mathbf{P}_1)$ and $\mathbf{x}_2 \sim \mathcal{N}(\boldsymbol{\mu}_2, \mathbf{P}_2)$. The sum $\mathbf{x}_1 + \mathbf{x}_2 \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{P})$ where,

$$\begin{aligned}\boldsymbol{\mu} &= \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2, \\ \mathbf{P} &= \mathbf{P}_1 + \mathbf{P}_2,\end{aligned}$$

and $\mathbf{P}, \mathbf{P}_1, \mathbf{P}_2 \in \mathbb{R}^{n\times n}$ are symmetric positive-definite matrices. To implement this operation in square-root form, we seek the upper-triangular matrix $\mathbf{S}$ such that $\mathbf{S}^\mathsf{T}\mathbf{S} = \mathbf{P}$. Assuming that similar factorisations are available for $\mathbf{P}_1$ and $\mathbf{P}_2$, we obtain the following matrix quadratic equation:

$$\begin{aligned}\underbrace{\mathbf{S}^\mathsf{T}\mathbf{S}}_{\mathbf{P}} &= \underbrace{\mathbf{S}_1^\mathsf{T}\mathbf{S}_1}_{\mathbf{P}_1} + \underbrace{\mathbf{S}_2^\mathsf{T}\mathbf{S}_2}_{\mathbf{P}_2} \\ &= \begin{bmatrix}\mathbf{S}_1^\mathsf{T} & \mathbf{S}_2^\mathsf{T}\end{bmatrix}\begin{bmatrix}\mathbf{S}_1 \\ \mathbf{S}_2\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{S}_1 \\ \mathbf{S}_2\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{S}_1 \\ \mathbf{S}_2\end{bmatrix},\end{aligned} \tag{4}$$

where $\mathbf{S}_1$ and $\mathbf{S}_2$ are any matrices with n columns[1] such that $\mathbf{P}_1 = \mathbf{S}_1^\mathsf{T}\mathbf{S}_1$ and $\mathbf{P}_2 = \mathbf{S}_2^\mathsf{T}\mathbf{S}_2$.

[1]The matrices $\mathbf{S}_1$ and $\mathbf{S}_2$ need not be upper triangular nor even square in this example, but they do need the same number of columns.

To find an upper triangular solution for $\mathbf{S}$, let $\mathbf{A} = \begin{bmatrix}\mathbf{S}_1\\ \mathbf{S}_2\end{bmatrix} \in \mathbb{R}^{m\times n}$, where m is the sum of the number of rows in $\mathbf{S}_1$ and $\mathbf{S}_2$. Typically, $\mathbf{A}$ has more rows than columns ($m > n$), which we call a *tall* matrix, and the QR decomposition $\mathbf{A} = \mathbf{QR}$ also has a tall $\mathbf{R}$. Since $\mathbf{R}$ is upper triangular and tall, it must contain (at least) $m-n$ rows of zeros at the bottom. Therefore, we can partition the QR decomposition as follows:

$$\underbrace{\begin{bmatrix}\mathbf{S}_1\\ \mathbf{S}_2\end{bmatrix}}_{\mathbf{A}} = \underbrace{\begin{bmatrix}\mathbf{Y} & \mathbf{Z}\end{bmatrix}}_{\mathbf{Q}} \underbrace{\begin{bmatrix}\mathbf{R}_1\\ \mathbf{0}\end{bmatrix}}_{\mathbf{R}},$$

where $\mathbf{Y} \in \mathbb{R}^{m\times n}$, $\mathbf{Z} \in \mathbb{R}^{m\times(m-n)}$ and $\mathbf{R}_1 \in \mathbb{R}^{n\times n}$. Evaluating $\mathbf{A}^\mathsf{T}\mathbf{A}$ reveals

$$\mathbf{A}^\mathsf{T}\mathbf{A} = \mathbf{R}^\mathsf{T}\mathbf{Q}^\mathsf{T}\mathbf{Q}\mathbf{R} = \mathbf{R}^\mathsf{T}\mathbf{R},$$

where $\mathbf{Q}^\mathsf{T}\mathbf{Q} = \mathbf{I}$ since $\mathbf{Q}$ is orthogonal, and substituting this result into (4) yields

$$\mathbf{S}^\mathsf{T}\mathbf{S} = \begin{bmatrix}\mathbf{S}_1\\ \mathbf{S}_2\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{S}_1\\ \mathbf{S}_2\end{bmatrix} = \begin{bmatrix}\mathbf{R}_1\\ \mathbf{0}\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{R}_1\\ \mathbf{0}\end{bmatrix} = \mathbf{R}_1^\mathsf{T}\mathbf{R}_1.$$

Since $\mathbf{R}_1$ is a square upper triangular matrix, a solution to (4) is given by setting $\mathbf{S} = \mathbf{R}_1$.

The operation of finding a matrix square root of a sum of squared matrices[2] can be considered a matrix generalisation of Pythagoras' theorem. We colloquially refer to the solution of this problem using QR composition the *Pythagorean QR* algorithm. This operation appears frequently when implementing numerically robust state estimators that propagate covariance via a matrix square root.

In this task, you will create a function that implements Pythagorean QR to solve (4) by finding a QR decomposition using Householder reflections (see both the `householderQr` member function and the `HouseholderQR` class).

Tip

Since the Pythagorean QR algorithm only needs the $\mathbf{R}$ matrix from the QR decomposition, we do not need to evaluate the $\mathbf{Q}$ matrix.

Tasks

a) Read through the very short doctest tutorial and then review the BDD-style unit tests you were given in `lab4/test/src/camera.cpp`.

Info

In the BDD-style `SCENARIO`s, note that

- `GIVEN` blocks enclose data,
- `WHEN` blocks enclose the execution of operations,
- `THEN` blocks enclose the assertion of expected results.

In addition to the assertions used to test the expected results, further `REQUIRE` assertions

[2]not to be confused with square matrices

> are used throughout the test scenario to terminate tests early to avoid *crashing* at runtime. This can occur, for example, if an expected file isn't present or if a non-existent element of an `std::vector` is accessed.

b) Place a declaration for `Gaussian::add` in the `public` section of the `Gaussian` class definition in `src/Gaussian.h` as follows:

```cpp
class Gaussian
{
public:
    // ...
    Gaussian add(const Gaussian & other) const;
    // ...
}
```

c) Create the empty member function `Gaussian::add` in `src/Gaussian.cpp` as follows:

```cpp
Gaussian Gaussian::add(const Gaussian & other) const
{
    Gaussian out;
    // Don't implement this function yet
    return out;
}
```

d) Review the `GIVEN` blocks within the `SCENARIO` block in `test/src/GaussianAdd.cpp` that correspond to the following test data:

i) $\boldsymbol{\mu}_1 = \begin{bmatrix}1\\1\\1\end{bmatrix}, \mathbf{S}_1 = \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix}2\\2\\2\end{bmatrix}, \mathbf{S}_2 = \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}.$

ii) $\boldsymbol{\mu}_1 = \begin{bmatrix}1\\1\\1\end{bmatrix}, \mathbf{S}_1 = \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix}2\\2\\2\end{bmatrix}, \mathbf{S}_2 = \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}.$

iii) $\boldsymbol{\mu}_1 = \begin{bmatrix}1\\1\\1\end{bmatrix}, \mathbf{S}_1 = \begin{bmatrix}3&0&0\\0&3&0\\0&0&3\end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix}2\\2\\2\end{bmatrix}, \mathbf{S}_2 = \begin{bmatrix}4&0&0\\0&4&0\\0&0&4\end{bmatrix}.$

iv) $\boldsymbol{\mu}_1 = \begin{bmatrix}1\\1\\1\\1\end{bmatrix}, \mathbf{S}_1 = \begin{bmatrix}1&2&3&4\\0&5&6&7\\0&0&8&9\\0&0&0&16\end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix}2\\2\\2\\2\end{bmatrix}, \mathbf{S}_2 = \begin{bmatrix}0&10&-1&-3\end{bmatrix}.$

e) For each of the cases above, create BDD-style unit tests using the `doctest` API to assert that

i) $\boldsymbol{\mu}$ has expected dimensions

ii) $\boldsymbol{\mu} = \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2$,

Tip

Testing that the elements of a matrix `M` match expected values in `M_expected` may be performed using the `isApprox()` member function and the `CHECK` macro in the following way,

```
CAPTURE_EIGEN(M);           // Capture M to print on test failure
CAPTURE_EIGEN(M_expected);  // Capture M_expected to print on test failure
CHECK(M.isApprox(M_expected));
```

In the event that this test fails, the knowledge that `M` is not equal to `M_expected` alone doesn't help diagnose which elements in those matrices are different. To provide extra diagnostic information, the `CAPTURE_EIGEN` macro ensures that both the `M` and `M_expected` matrices are printed in full only if the test fails.

iii) $\mathbf{S}$ has expected dimensions,

iv) $\mathbf{S}$ is upper triangular,

Tip

Testing that a matrix `S` is upper triangular can be achieved using the `isUpperTriangular()` member function and the `CHECK` macro as follows:

```
CAPTURE_EIGEN(S);       // Capture S to be printed on test failure
CHECK(S.isUpperTriangular());
```

In the event that this test fails, the knowledge that `S` isn't upper triangular alone doesn't help identify which elements are non-zero in the lower triangle. To provide extra diagnostic information, the `CAPTURE_EIGEN` macro ensures that the entire `S` matrix is printed in full only if the test fails.

v) $\mathbf{P} = \mathbf{P}_1 + \mathbf{P}_2$.

f) Run `ninja` and verify that appropriate the tests *fail*, since `Gaussian::add` has not yet been implemented. If your tests *crash*, include appropriate `REQUIRE` assertions in the tests as needed.

g) Implement `Gaussian::add`.

h) Run `ninja` and verify that all the tests pass.

3 Marginal distributions of a Gaussian (1 mark)

Let $\mathbf{x} \in \mathbb{R}^n$ and let

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{P}) \tag{5}$$

be a multivariate Gaussian distribution, where $\boldsymbol{\mu} \in \mathbb{R}^n$ is the mean vector and $\mathbf{P} \in \mathbb{R}^{n\times n}$ is the symmetric positive-definite covariance matrix. If we partition

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{bmatrix}, \tag{6}$$

where the head $\mathbf{x}_a \in \mathbb{R}^{n_a}$ and the tail $\mathbf{x}_b \in \mathbb{R}^{n_b}$ with $n_a + n_b = n$, then we can write (5) as

$$p\left(\begin{bmatrix}\mathbf{x}_a \\ \mathbf{x}_b\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{x}_a \\ \mathbf{x}_b\end{bmatrix}; \begin{bmatrix}\boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b\end{bmatrix}, \begin{bmatrix}\mathbf{P}_{aa} & \mathbf{P}_{ab} \\ \mathbf{P}_{ba} & \mathbf{P}_{bb}\end{bmatrix}\right), \tag{7}$$

where $\boldsymbol{\mu}_a \in \mathbb{R}^{n_a}$, $\boldsymbol{\mu}_b \in \mathbb{R}^{n_b}$, $\mathbf{P}_{aa} \in \mathbb{R}^{n_a \times n_a}$, $\mathbf{P}_{ab} \in \mathbb{R}^{n_b \times n_a}$, $\mathbf{P}_{ba} \in \mathbb{R}^{n_b \times n_a}$ and $\mathbf{P}_{bb} \in \mathbb{R}^{n_b \times n_b}$.

The marginal distributions of the head $\mathbf{x}_a$ and the tail $\mathbf{x}_b$ are then given as follows:

$$\begin{aligned} p(\mathbf{x}_a) &= \mathcal{N}(\mathbf{x}_a; \boldsymbol{\mu}_a, \mathbf{P}_{aa}), \\ p(\mathbf{x}_b) &= \mathcal{N}(\mathbf{x}_b; \boldsymbol{\mu}_b, \mathbf{P}_{bb}). \end{aligned}$$

We can also write (5) using a upper-triangular square-root parameterisation of the covariance matrix, denoted as

$$p(\mathbf{x}) = \mathcal{N}^{\frac{1}{2}}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{S}), \tag{8}$$

where $\mathbf{S} \in \mathbb{R}^{n \times n}$ is an upper triangular matrix such that $\mathbf{S}^\mathsf{T}\mathbf{S} = \mathbf{P}$.

Then, under the partition (6), we have the following relations between the block elements of $\mathbf{P}$ and $\mathbf{S}$:

$$\begin{aligned} \mathbf{P} &= \mathbf{S}^\mathsf{T}\mathbf{S} \\ \begin{bmatrix}\mathbf{P}_{aa} & \mathbf{P}_{ab} \\ \mathbf{P}_{ba} & \mathbf{P}_{bb}\end{bmatrix} &= \begin{bmatrix}\mathbf{S}_1 & \mathbf{S}_2 \\ \mathbf{0} & \mathbf{S}_3\end{bmatrix}^\mathsf{T} \begin{bmatrix}\mathbf{S}_1 & \mathbf{S}_2 \\ \mathbf{0} & \mathbf{S}_3\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{S}_1^\mathsf{T} & \mathbf{0} \\ \mathbf{S}_2^\mathsf{T} & \mathbf{S}_3^\mathsf{T}\end{bmatrix} \begin{bmatrix}\mathbf{S}_1 & \mathbf{S}_2 \\ \mathbf{0} & \mathbf{S}_3\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{S}_1^\mathsf{T}\mathbf{S}_1 & \mathbf{S}_1^\mathsf{T}\mathbf{S}_2 \\ \mathbf{S}_2^\mathsf{T}\mathbf{S}_1 & \mathbf{S}_2^\mathsf{T}\mathbf{S}_2 + \mathbf{S}_3^\mathsf{T}\mathbf{S}_3\end{bmatrix}, \end{aligned} \tag{9}$$

where $\mathbf{S}_1 \in \mathbb{R}^{n_a \times n_a}$, $\mathbf{S}_2 \in \mathbb{R}^{n_a \times n_b}$, $\mathbf{S}_3 \in \mathbb{R}^{n_b \times n_b}$ and $\mathbf{S}_1$ and $\mathbf{S}_3$ are upper triangular.

The marginal distribution of $\mathbf{x}_a$ is given by

$$p(\mathbf{x}_a) = \mathcal{N}(\mathbf{x}_a; \boldsymbol{\mu}_a, \mathbf{S}_1^\mathsf{T}\mathbf{S}_1) = \mathcal{N}^{\frac{1}{2}}(\mathbf{x}_a; \boldsymbol{\mu}_a, \mathbf{S}_a), \tag{10}$$

where $\mathbf{S}_a = \mathbf{S}_1$, since $\mathbf{S}_1$ is already upper triangular. The marginal distribution of $\mathbf{x}_b$ is given by

$$p(\mathbf{x}_b) = \mathcal{N}(\mathbf{x}_b; \boldsymbol{\mu}_b, \mathbf{S}_2^\mathsf{T}\mathbf{S}_2 + \mathbf{S}_3^\mathsf{T}\mathbf{S}_3) = \mathcal{N}^{\frac{1}{2}}(\mathbf{x}_b; \boldsymbol{\mu}_b, \mathbf{S}_b), \tag{11}$$

for some upper triangular matrix $\mathbf{S}_b \in \mathbb{R}^{n_b \times n_b}$ such that $\mathbf{S}_b^\mathsf{T}\mathbf{S}_b = \mathbf{S}_2^\mathsf{T}\mathbf{S}_2 + \mathbf{S}_3^\mathsf{T}\mathbf{S}_3$. To find such a matrix, we first factorise the sum as follows:

$$\begin{aligned} \mathbf{S}_b^\mathsf{T}\mathbf{S}_b &= \mathbf{S}_2^\mathsf{T}\mathbf{S}_2 + \mathbf{S}_3^\mathsf{T}\mathbf{S}_3 \\ &= \begin{bmatrix}\mathbf{S}_2^\mathsf{T} & \mathbf{S}_3^\mathsf{T}\end{bmatrix} \begin{bmatrix}\mathbf{S}_2 \\ \mathbf{S}_3\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{S}_2 \\ \mathbf{S}_3\end{bmatrix}^\mathsf{T} \begin{bmatrix}\mathbf{S}_2 \\ \mathbf{S}_3\end{bmatrix}. \end{aligned}$$

Then, let

$$\underbrace{\begin{bmatrix}\mathbf{S}_2\\ \mathbf{S}_3\end{bmatrix}}_{\mathbf{A}} = \underbrace{\begin{bmatrix}\mathbf{Y} & \mathbf{Z}\end{bmatrix}}_{\mathbf{Q}}\underbrace{\begin{bmatrix}\mathbf{R}_1\\ \mathbf{0}\end{bmatrix}}_{\mathbf{R}}$$

be a QR factorisation, where $\mathbf{A} \in \mathbb{R}^{n\times n_b}$, $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is an orthogonal matrix and $\mathbf{R} \in \mathbb{R}^{n\times n_b}$ is an upper triangular matrix. Then,

$$\begin{aligned}\mathbf{A}^\mathsf{T}\mathbf{A} &= \mathbf{R}^\mathsf{T}\mathbf{Q}^\mathsf{T}\mathbf{Q}\mathbf{R} = \mathbf{R}^\mathsf{T}\mathbf{R} && \text{(since } \mathbf{Q} \text{ is orthogonal)}\\ \begin{bmatrix}\mathbf{S}_2\\ \mathbf{S}_3\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{S}_2\\ \mathbf{S}_3\end{bmatrix} &= \begin{bmatrix}\mathbf{R}_1\\ \mathbf{0}\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{R}_1\\ \mathbf{0}\end{bmatrix}\\ &= \mathbf{R}_1^\mathsf{T}\mathbf{R}_1.\end{aligned}$$

Therefore, $\mathbf{S}_b = \mathbf{R}_1$, since $\mathbf{R}_1$ is upper triangular.

Note

As we have seen, when working with an upper-triangular square-root factorisation of a joint covariance matrix, it is much easier to extract the marginal distribution of the head (10) than the tail (11) of a partition. The former requires only extracting the top left block of $\mathbf{S}$ to find $\mathbf{S}_a$, whereas the latter requires extra computation to upper-triangularise the rightmost columns of $\mathbf{S}$ via a Q-less QR decomposition and then extract the top rows of the result to find $\mathbf{S}_b$.

Tasks

a) Place a declaration for `Gaussian::marginalHead` in the `public` section of the `Gaussian` class definition in `src/Gaussian.h` as follows:

```
class Gaussian
{
public:
    // ...
    Gaussian marginalHead(int na) const;
    // ...
}
```

b) Create the empty member function `Gaussian::marginalHead` in `src/Gaussian.cpp` as follows:

```
Gaussian Gaussian::marginalHead(int na) const
{
    Gaussian out;
    // Don't implement this function yet
    return out;
}
```

c) Review the `GIVEN` blocks within the `SCENARIO` block in `test/src/GaussianMarginal.cpp` that correspond to the following cases:

i) `na` $= 1$ and `nb` $= 1$

ii) `na` $= 2$ and `nb` $= 2$

d) For each of the cases above, create BDD-style unit tests to assert that

i) The dimensions of $\boldsymbol{\mu}_a$ and $\mathbf{S}_a$ are valid.

ii) $\mathbf{S}_a$ is upper triangular.

iii) $\boldsymbol{\mu}_a$ matches the head of the joint mean $\boldsymbol{\mu}$.

iv) $\mathbf{P}_a = \mathbf{S}_a^\mathsf{T}\mathbf{S}_a$ matches the top-left block of the joint covariance $\mathbf{P}$.

e) Run `ninja` and verify that appropriate the tests *fail*, since `Gaussian::marginalHead` has not yet been implemented. If your tests *crash*, include appropriate `REQUIRE` assertions in the tests as needed.

f) Implement `Gaussian::marginalHead` from (10).

g) Run `ninja` and verify that all unit tests pass.

4 Conditional distributions of a Gaussian (1 mark)

If we consider the same head and tail partitions as given in (6) and joint distribution (7), then the conditional distributions of the head given the tail and the tail given the head, respectively, are given as follows:

$$p(\mathbf{x}_a|\mathbf{x}_b) = \mathcal{N}(\mathbf{x}_a; \underbrace{\boldsymbol{\mu}_a + \mathbf{P}_{ab}\mathbf{P}_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b)}_{\boldsymbol{\mu}_{a|b}}, \underbrace{\mathbf{P}_{aa} - \mathbf{P}_{ab}\mathbf{P}_{bb}^{-1}\mathbf{P}_{ba}}_{\mathbf{P}_{a|b}}), \tag{12a}$$

$$p(\mathbf{x}_b|\mathbf{x}_a) = \mathcal{N}(\mathbf{x}_b; \underbrace{\boldsymbol{\mu}_b + \mathbf{P}_{ba}\mathbf{P}_{aa}^{-1}(\mathbf{x}_a - \boldsymbol{\mu}_a)}_{\boldsymbol{\mu}_{b|a}}, \underbrace{\mathbf{P}_{bb} - \mathbf{P}_{ba}\mathbf{P}_{aa}^{-1}\mathbf{P}_{ab}}_{\mathbf{P}_{b|a}}). \tag{12b}$$

From the block relationships previously established (9), we have

$$\mathbf{P}_{aa} = \mathbf{S}_1^\mathsf{T}\mathbf{S}_1, \tag{13a}$$
$$\mathbf{P}_{ab} = \mathbf{S}_1^\mathsf{T}\mathbf{S}_2, \tag{13b}$$
$$\mathbf{P}_{ba} = \mathbf{S}_2^\mathsf{T}\mathbf{S}_1, \tag{13c}$$
$$\mathbf{P}_{bb} = \mathbf{S}_2^\mathsf{T}\mathbf{S}_2 + \mathbf{S}_3^\mathsf{T}\mathbf{S}_3. \tag{13d}$$

Substituting (13c) and (13a) into (12b) yields

$$\mathbf{P}_{ba}\mathbf{P}_{aa}^{-1} = \mathbf{S}_2^\mathsf{T}\mathbf{S}_1(\mathbf{S}_1^\mathsf{T}\mathbf{S}_1)^{-1} = \mathbf{S}_2^\mathsf{T}\mathbf{S}_1\mathbf{S}_1^{-1}\mathbf{S}_1^{-\mathsf{T}} = \mathbf{S}_2^\mathsf{T}\mathbf{S}_1^{-\mathsf{T}},$$

and therefore

$$\begin{aligned}\boldsymbol{\mu}_{b|a} &= \boldsymbol{\mu}_b + \mathbf{P}_{ba}\mathbf{P}_{aa}^{-1}(\mathbf{x}_a - \boldsymbol{\mu}_a) \\ &= \boldsymbol{\mu}_b + \mathbf{S}_2^\mathsf{T}\mathbf{S}_1^{-\mathsf{T}}(\mathbf{x}_a - \boldsymbol{\mu}_a).\end{aligned}$$

Similarly, substituting (13) into (12b) yields

$$\begin{aligned}
\mathbf{P}_{b|a} &= \mathbf{P}_{bb} - \mathbf{P}_{ba}\mathbf{P}_{aa}^{-1}\mathbf{P}_{ab} \\
&= \mathbf{S}_2^\mathsf{T}\mathbf{S}_2 + \mathbf{S}_3^\mathsf{T}\mathbf{S}_3 - \mathbf{S}_2^\mathsf{T}\mathbf{S}_1(\mathbf{S}_1^\mathsf{T}\mathbf{S}_1)^{-1}\mathbf{S}_1^\mathsf{T}\mathbf{S}_2 \\
&= \mathbf{S}_2^\mathsf{T}\mathbf{S}_2 + \mathbf{S}_3^\mathsf{T}\mathbf{S}_3 - \mathbf{S}_2^\mathsf{T}\mathbf{S}_1\mathbf{S}_1^{-1}\mathbf{S}_1^{-\mathsf{T}}\mathbf{S}_1^\mathsf{T}\mathbf{S}_2 \\
&= \mathbf{S}_2^\mathsf{T}\mathbf{S}_2 + \mathbf{S}_3^\mathsf{T}\mathbf{S}_3 - \mathbf{S}_2^\mathsf{T}\mathbf{S}_2 \\
&= \mathbf{S}_3^\mathsf{T}\mathbf{S}_3.
\end{aligned}$$

The conditional distribution of $\mathbf{x}_b$ given $\mathbf{x}_a$ is given by

$$p(\mathbf{x}_b|\mathbf{x}_a) = \mathcal{N}^{\frac{1}{2}}(\mathbf{x}_b; \underbrace{\boldsymbol{\mu}_b + \mathbf{S}_2^\mathsf{T}\mathbf{S}_1^{-\mathsf{T}}(\mathbf{x}_a - \boldsymbol{\mu}_a)}_{\boldsymbol{\mu}_{b|a}}, \underbrace{\mathbf{S}_3}_{\mathbf{S}_{b|a}}). \tag{14}$$

Info

The computation of the conditional mean $\boldsymbol{\mu}_{b|a}$ and upper triangular square-root covariance $\mathbf{S}_{b|a}$ that appears in (14) involves a trivial triangular solve and extraction of the 3 non-zero blocks of $\mathbf{S}$. This is significantly cheaper than the computation involved in (12b).

To find the conditional distribution of $\mathbf{x}_a$ given $\mathbf{x}_b$, we swap the head and tail partitions as follows:

$$p\left(\begin{bmatrix}\mathbf{x}_b \\ \mathbf{x}_a\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mathbf{x}_b \\ \mathbf{x}_a\end{bmatrix}; \begin{bmatrix}\boldsymbol{\mu}_b \\ \boldsymbol{\mu}_a\end{bmatrix}, \begin{bmatrix}\mathbf{P}_{bb} & \mathbf{P}_{ba} \\ \mathbf{P}_{ab} & \mathbf{P}_{aa}\end{bmatrix}\right). \tag{15}$$

Swapping the row and column blocks in $\mathbf{P}$ is equivalent to swapping the column blocks of $\mathbf{S}$, since

$$\begin{bmatrix}\mathbf{P}_{bb} & \mathbf{P}_{ba} \\ \mathbf{P}_{ab} & \mathbf{P}_{aa}\end{bmatrix} = \begin{bmatrix}\mathbf{S}_2 & \mathbf{S}_1 \\ \mathbf{S}_3 & \mathbf{0}\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{S}_2 & \mathbf{S}_1 \\ \mathbf{S}_3 & \mathbf{0}\end{bmatrix}.$$

Since swapping the block columns of $\mathbf{S}$ leads to a non-upper-triangular matrix, let

$$\begin{bmatrix}\mathbf{S}_2 & \mathbf{S}_1 \\ \mathbf{S}_3 & \mathbf{0}\end{bmatrix} = \mathbf{Q}\underbrace{\begin{bmatrix}\mathbf{R}_1 & \mathbf{R}_2 \\ \mathbf{0} & \mathbf{R}_3\end{bmatrix}}_{\mathbf{R}}$$

be a QR factorisation, where $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is an orthogonal matrix and $\mathbf{R} \in \mathbb{R}^{n\times n}$ is an upper triangular matrix with blocks $\mathbf{R}_1 \in \mathbb{R}^{n_b\times n_b}$, $\mathbf{R}_2 \in \mathbb{R}^{n_b\times n_a}$ and $\mathbf{R}_3 \in \mathbb{R}^{n_a\times n_a}$, where $\mathbf{R}_1$ and $\mathbf{R}_3$ are upper triangular. Then,

$$\begin{aligned}
\begin{bmatrix}\mathbf{S}_2 & \mathbf{S}_1 \\ \mathbf{S}_3 & \mathbf{0}\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{S}_2 & \mathbf{S}_1 \\ \mathbf{S}_3 & \mathbf{0}\end{bmatrix} &= \mathbf{R}^\mathsf{T}\mathbf{Q}^\mathsf{T}\mathbf{Q}\mathbf{R} = \mathbf{R}^\mathsf{T}\mathbf{R} && \text{(since } \mathbf{Q} \text{ is orthogonal)} \\
&= \begin{bmatrix}\mathbf{R}_1 & \mathbf{R}_2 \\ \mathbf{0} & \mathbf{R}_3\end{bmatrix}^\mathsf{T}\begin{bmatrix}\mathbf{R}_1 & \mathbf{R}_2 \\ \mathbf{0} & \mathbf{R}_3\end{bmatrix}.
\end{aligned}$$

Therefore, the partition swapped distribution can be parameterised with an upper triangular square-root covariance as follows:

$$p\left(\begin{bmatrix}\mathbf{x}_b \\ \mathbf{x}_a\end{bmatrix}\right) = \mathcal{N}^{\frac{1}{2}}\left(\begin{bmatrix}\mathbf{x}_b \\ \mathbf{x}_a\end{bmatrix}; \begin{bmatrix}\boldsymbol{\mu}_b \\ \boldsymbol{\mu}_a\end{bmatrix}, \begin{bmatrix}\mathbf{R}_1 & \mathbf{R}_2 \\ \mathbf{0} & \mathbf{R}_3\end{bmatrix}\right). \tag{16}$$

Finally, the conditional distribution of $\mathbf{x}_a$ given $\mathbf{x}_b$ is given by the following:

$$p(\mathbf{x}_a|\mathbf{x}_b) = \mathcal{N}^{\frac{1}{2}}(\mathbf{x}_a; \underbrace{\boldsymbol{\mu}_a + \mathbf{R}_2^\mathsf{T}\mathbf{R}_1^{-\mathsf{T}}(\mathbf{x}_b - \boldsymbol{\mu}_b)}_{\boldsymbol{\mu}_{a|b}}, \underbrace{\mathbf{R}_3}_{\mathbf{S}_{a|b}}). \tag{17}$$

Note

As we have seen, when working with an upper-triangular square-root factorisation of a joint covariance matrix, it is much easier to compute the conditional distribution of the tail given the head (14) rather than than the conditional distribution of the head given the tail (17). The former requires only a simple triangular solve using the upper left block of $\mathbf{S}$ and extraction of the lower right block of $\mathbf{S}$, whereas the latter requires a extra computation to upper triangularise the block column swapped $\mathbf{S}$ via a Q-less QR decomposition.

Tasks

a) Place a declaration for `Gaussian::conditionalTailGivenHead` in the `public` section of the `Gaussian` class definition in `src/Gaussian.h` as follows:

```
class Gaussian
{
public:
    // ...
    Gaussian conditionalTailGivenHead(const Eigen::VectorXd & xa) const;
    // ...
}
```

b) Create the empty member function `Gaussian::conditionalTailGivenHead` in `src/Gaussian.cpp` as follows:

```
Gaussian Gaussian::conditionalTailGivenHead(const Eigen::VectorXd & xa) const
{
    Gaussian out;
    // Don't implement this function yet
    return out;
}
```

c) Review the `GIVEN` blocks within the `SCENARIO` block in `test/src/GaussianConditional.cpp` that correspond to the following cases:

i) `na` $= 1$ and `nb` $= 1$

ii) `na` $= 3$ and `nb` $= 1$

iii) `na` $= 1$ and `nb` $= 3$

iv) `na` $= 3$ and `nb` $= 3$

d) For each of the cases above, create BDD-style unit tests to assert that

i) The dimensions of $\boldsymbol{\mu}_{b|a}$ and $\mathbf{S}_{b|a}$ are valid.

ii) $\mathbf{S}_{b|a}$ is upper triangular.

iii) $\boldsymbol{\mu}_{b|a}$ and $\mathbf{P}_{b|a} = \mathbf{S}_{b|a}^{\mathsf{T}}\mathbf{S}_{b|a}$ match expected values `mub_a_expected` and `Pb_a_expected` from (12b), respectively.

e) Run `ninja` and verify that appropriate the tests *fail*, since `Gaussian::conditionalTailGivenHead` has not yet been implemented. If your tests *crash*, include appropriate `REQUIRE` assertions in the tests as needed.

f) Implement `Gaussian::conditionalTailGivenHead` from (14).

Tip

Use the solve member function for triangular views to implement the conditional mean.

g) Run `ninja` and verify that all unit tests pass.

5 Permutations of a Gaussian random variable (1 mark)

In Problems 3 and 4, we only considered the marginal and conditional distributions of simple partitions of $\mathbf{x} = \left[\begin{smallmatrix}\mathbf{x}_a\\ \mathbf{x}_b\end{smallmatrix}\right]$. To generalise these operations to a non-contiguous subset of $\mathbf{x}$, let us introduce an index vector $\mathcal{I} \in \mathbb{N}^{n_\mathcal{I}}$ to select the desired elements and denote this as $\mathbf{x}_\mathcal{I}$. Each element of $\mathbf{x}_\mathcal{I}$ is given by

$$(\mathbf{x}_\mathcal{I})_i = x_{\mathcal{I}_i}, \quad \text{for } i = 1, 2, \ldots, n_\mathcal{I}.$$

Example

Let $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ and let $\mathcal{I} = \begin{bmatrix} 2 & 3 & 1 \end{bmatrix}$. Then $\mathbf{x}_\mathcal{I} = \begin{bmatrix} x_2 \\ x_3 \\ x_1 \end{bmatrix}$.

The permuted distribution of $\mathbf{x}_\mathcal{I}$ is given by

$$p(\mathbf{x}_\mathcal{I}) = \mathcal{N}(\mathbf{x}_\mathcal{I}; \boldsymbol{\mu}_\mathcal{I}, \mathbf{P}_{\mathcal{I}\mathcal{I}}), \tag{18}$$

where $\boldsymbol{\mu}_\mathcal{I} \in \mathbb{R}^{n_\mathcal{I}}$ with elements given by $(\boldsymbol{\mu}_\mathcal{I})_i = \mu_{\mathcal{I}_i}$ for $i = 1, 2, \ldots, n_\mathcal{I}$ and $\mathbf{P}_{\mathcal{I}\mathcal{I}} \in \mathbb{R}^{n_\mathcal{I} \times n_\mathcal{I}}$ with elements given by $(\mathbf{P}_{\mathcal{I}\mathcal{I}})_{i,j} = P_{\mathcal{I}_i,\mathcal{I}_j}$ for $i, j = 1, 2, \ldots, n_\mathcal{I}$.

Example

Let

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix}, \quad \mathbf{P} = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix},$$

and let $\mathcal{I} = \begin{bmatrix} 3 & 5 & 1 \end{bmatrix}$. Then,

$$\boldsymbol{\mu}_{\mathcal{I}} = \begin{bmatrix} \mu_2 \\ \mu_3 \\ \mu_1 \end{bmatrix}, \quad \mathbf{P}_{\mathcal{I}\mathcal{I}} = \begin{bmatrix} P_{22} & P_{23} & P_{21} \\ P_{32} & P_{33} & P_{31} \\ P_{12} & P_{13} & P_{11} \end{bmatrix}.$$

Selecting $\mathcal{I}$-rows and $\mathcal{I}$-columns of $\mathbf{P}$ to form $\mathbf{P}_{\mathcal{I}\mathcal{I}}$ is equivalent to selecting *all*-rows and $\mathcal{I}$-columns of $\mathbf{S}$, i.e.,

$$\mathbf{P}_{\mathcal{I}\mathcal{I}} = \mathbf{S}_{:,\mathcal{I}}^{\mathsf{T}} \mathbf{S}_{:,\mathcal{I}},$$

where $\mathbf{S}_{:,\mathcal{I}} \in \mathbb{R}^{n \times n_{\mathcal{I}}}$.

The marginal distribution of $\mathbf{x}_{\mathcal{I}}$ is given by

$$p(\mathbf{x}_{\mathcal{I}}) = \mathcal{N}(\mathbf{x}_{\mathcal{I}}; \boldsymbol{\mu}_{\mathcal{I}}, \mathbf{S}_{:,\mathcal{I}}^{\mathsf{T}} \mathbf{S}_{:,\mathcal{I}}) = \mathcal{N}^{\frac{1}{2}}(\mathbf{x}_{\mathcal{I}}; \boldsymbol{\mu}_{\mathcal{I}}, \mathbf{S}_{\mathcal{I}}), \tag{19}$$

for some upper triangular matrix $\mathbf{S}_{\mathcal{I}} \in \mathbb{R}^{n_{\mathcal{I}} \times n_{\mathcal{I}}}$ such that $\mathbf{S}_{\mathcal{I}}^{\mathsf{T}} \mathbf{S}_{\mathcal{I}} = \mathbf{S}_{:,\mathcal{I}}^{\mathsf{T}} \mathbf{S}_{:,\mathcal{I}}$. Since $\mathbf{S}_{:,\mathcal{I}}$ is a tall matrix that is not necessarily upper triangular, we can let

$$\mathbf{S}_{:,\mathcal{I}} = \underbrace{\begin{bmatrix} \mathbf{Y} & \mathbf{Z} \end{bmatrix}}_{\mathbf{Q}} \underbrace{\begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix}}_{\mathbf{R}}$$

be a QR factorisation, where $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is an orthogonal matrix and $\mathbf{R} \in \mathbb{R}^{n \times n_{\mathcal{I}}}$ is an upper triangular matrix. Then,

$$\begin{aligned} \mathbf{S}_{:,\mathcal{I}}^{\mathsf{T}} \mathbf{S}_{:,\mathcal{I}} &= \mathbf{R}^{\mathsf{T}} \mathbf{Q}^{\mathsf{T}} \mathbf{Q} \mathbf{R} = \mathbf{R}^{\mathsf{T}} \mathbf{R} && \text{(since } \mathbf{Q} \text{ is orthogonal)} \\ &= \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} \\ &= \mathbf{R}_1^{\mathsf{T}} \mathbf{R}_1. \end{aligned}$$

Therefore, $\mathbf{S}_{\mathcal{I}} = \mathbf{R}_1$, since $\mathbf{R}_1$ is already upper triangular.

Info

Note that the closer $\mathcal{I}$ is to the sequence $\mathcal{I} = \begin{bmatrix} 1 & 2 & 3 & \cdots & n_{\mathcal{I}} \end{bmatrix}$, the cheaper the QR decomposition becomes, becoming free in the case where $\mathcal{I}_j = j$ for each j, since $\mathbf{S}_{:,\mathcal{I}}$ is already upper triangular.

Note

To generalise the previous results for the marginal head (10) and conditional tail (14) of joint Gaussian random variable to an arbitrary partition of $\mathbf{x}$, let us introduce the index vectors $\mathcal{A} \in \mathbb{N}^{n_\mathcal{A}}$ and $\mathcal{B} \in \mathbb{N}^{n_\mathcal{B}}$ to select the desired elements of each set and denote these as $\mathbf{x}_\mathcal{A}$ and $\mathbf{x}_\mathcal{B}$, respectively. Then, let $\mathcal{I} = \begin{bmatrix}\mathcal{A} & \mathcal{B}\end{bmatrix}$ so that $\mathbf{x}_\mathcal{I} = \begin{bmatrix}\mathbf{x}_\mathcal{A} \\ \mathbf{x}_\mathcal{B}\end{bmatrix}$. Then, (19) is in the required form for (10) or (14) to be used directly without any further QR decomposition.

Tasks

a) Place a declaration for `Gaussian::permute` in the `public` section of the `Gaussian` class definition in `src/Gaussian.h` as follows:

```
class Gaussian
{
public:
    // ...
    Gaussian permute(const Eigen::ArrayXi & idx) const;
    // ...
};
```

b) Create the empty member function `Gaussian::permute` in `src/Gaussian.cpp` as follows:

```
Gaussian Gaussian::permute(const Eigen::ArrayXi & idx) const
{
    Gaussian out;
    // Don't implement this function yet
    return out;
}
```

c) Review the `GIVEN` blocks within the `SCENARIO` block in `test/src/GaussianPermute.cpp` that correspond to the following cases:

   i) `n` $= 4$

   ii) `n` $= 2$

d) For each of the cases above, create BDD-style unit tests to assert that:

   i) $\boldsymbol{\mu}_\mathcal{I}$ has correct dimensions and is correctly permuted

   ii) $\mathbf{S}_{:,\mathcal{I}}$ has correct dimensions and is upper triangular

   iii) $\mathbf{P}_{\mathcal{I}\mathcal{I}}$ is correctly permuted

e) Run `ninja` and verify that the appropriate tests fail, since `Gaussian::permute` has not yet been implemented. If your tests crash, include appropriate `REQUIRE` assertions in the tests as needed.

f) Implement `Gaussian::permute` from (19).

g) Run `ninja` and verify that all unit tests pass.