Fast Operations for Low-Rank + Diagonal Matrices

Frederik Kunstner

Abstract

Given a p.d. matrix $\mathbf{A}$ of size n with a low-rank + diagonal structure, $\mathbf{A} = \mathbf{U}\mathbf{U}^\top + \mathbf{D}$, where $\mathbf{U}$ is a $[n \times k]$ matrix with $k \ll n$ and $\mathbf{D}$ is a diagonal matrix, it is possible to obtain efficient implementations of common matrix operations by leveraging the structure of the matrix. These notes show how to implement some operations that arise when working with Multivariate Gaussian distributions with covariance matrices exhibiting this structure, such as sampling, computing the probability distribution function or the Kullback-Leibler divergence to other Gaussians. The linear algebra operations involved are computing inverses, symmetric factorizations and determinants.

1 Introduction

Consider a positive definite matrix $\mathbf{A}$ of dimension $[n \times n]$ that can be written as a sum of a low-rank and diagonal matrix, $\mathbf{A} = \mathbf{U}\mathbf{U}^\top + \mathbf{D}$, where $\mathbf{U}$ is of dimension $[n \times k]$ and $\mathbf{D}$ is a diagonal $[n \times n]$ matrix with positive entries. The goal of this document is to show how to do common operations involving $\mathbf{A}$ efficiently, by leveraging the low-rank + diagonal structure. The naive way of handling operations with this matrix, by computing $\mathbf{U}\mathbf{U}^\top$ and using $\mathbf{A}$ directly, involves a high memory cost - of order $O(n^2)$ instead of $O(nk)$ - and a high computational cost, depending on the type of operation. This is prohibitive if n is large and unnecessary if k is much smaller. Instead of working with $\mathbf{A}$, working directly with $\mathbf{U}$ and $\mathbf{D}$ can lead to significant improvements.

One motivating use case of structured matrices can be found in probabilistic models, for example as the covariance matrix of a Gaussian probability distribution learned from data. Structured covariance estimation aims at finding a middle ground between the cheap but crude diagonal approximation commonly used in practice and the expressive but expensive full covariance matrix estimation. By modeling the k most important eigenvectors of the covariance matrix in a low-rank matrix $\mathbf{U}\mathbf{U}^\top$ and the remaining $n - k$ eigenvectors as a diagonal $\mathbf{D}$, it is possible to trade computational complexity with the precision of the non-diagonal elements of the covariance matrix.

Most of the theory used to get efficient implementation is more general than the low-rank + diagonal structure shown here. However, those notes only aim at giving a short exposition on the simple case and do not go into proof, nor much details. Section 2 present each operation and Section 3 shows the algorithms in a more compact format. Compagnon implementations can be found at `https://github.com/fkunstner/low-rank`, based on `Numpy` or `PyTorch`.

2 Efficient operations

Most of the improvements in efficiency come from a better ordering of operations. As a simple illustrative example, consider the matrix-vector multiplication with an arbitrary vector $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{A}\mathbf{x}$. The naive implementation, assuming $\mathbf{A}$ is precomputed (which in itself already costs $O(n^2 k)$), leads to a computational cost of $O(n^2)$. A simple reordering of operations, assuming $\mathbf{U}$ and $\mathbf{D}$ are accessible in memory,

$$\mathbf{A}\mathbf{x} = \left(\mathbf{U}\left(\mathbf{U}^\top\mathbf{x}\right)\right) + \mathbf{D}\mathbf{x}, \tag{1}$$

leads to a computational cost of $O(nk^2)$.

For more complex operations, such as matrix inversion and factorization, we will first need to obtain closed form solutions. For notational simplicity, consider the parametrization using $\mathbf{V} = \mathbf{D}^{-1/2}\mathbf{U}$, giving

$$\mathbf{A} = \mathbf{D}^{1/2}\left(\mathbf{V}\mathbf{V}^\top + \mathbf{I}_n\right)\mathbf{D}^{1/2}, \tag{2}$$

where $\mathbf{I}_n$ is the $[n \times n]$ identity matrix. Most of complexity of operating with $\mathbf{A}$ arises from the operations involving $\mathbf{V}\mathbf{V}^\top + \mathbf{I}_n$, as the multiplication by diagonal matrices should be easy to deal with.

2.1 Inversion

Inverting $\mathbf{A}$, using the $\mathbf{V}$ parametrization, involves computing the inverse of $\mathbf{V}\mathbf{V}^\top + \mathbf{I}_n$. This can be done efficiently using (a special case of) the Woodbury inversion formula.

Theorem 1 (Structured Inversion: Woodbury's inversion formula)**.** *Let* $\mathbf{A}, \mathbf{U}, \mathbf{C}, \mathbf{V}$ *be matrices with sizes* $\mathbf{A} : [n \times n], \mathbf{U} : [n \times k], \mathbf{C} : [k \times k], \mathbf{V} : [k \times n]$*, where* $\mathbf{A}$ *and* $\mathbf{C}$ *are full rank. Then,*

$$(\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{C}^{-1} + \mathbf{V}\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}\mathbf{A}^{-1} \tag{3}$$

Applying Woodbury's inversion formula to the low-rank + identity matrix leads to

$$\left(\mathbf{V}\mathbf{V}^\top + \mathbf{I}_n\right)^{-1} = \mathbf{I}_n - \mathbf{V}\left(\mathbf{I}_k + \mathbf{V}^\top\mathbf{V}\right)^{-1}\mathbf{V}^\top. \tag{4}$$

Computing the inverse of the $[k \times k]$ matrix costs $O(nk^2 + k^3)$, and the necessary matrix multiplications to get $\mathbf{A}^{-1}$ add $O(nk^2 + n^2k)$ operations, leading to a total computational complexity of $O(n^2k)$ instead of $O(n^3)$;

$$\mathbf{A}^{-1} = \mathbf{D}^{-1/2}\left(\mathbf{V}\mathbf{V}^\top + \mathbf{I}_n\right)^{-1}\mathbf{D}^{-1/2} = \mathbf{D}^{-1/2}\left(\mathbf{I}_n - \mathbf{V}\left(\mathbf{I}_k + \mathbf{V}^\top\mathbf{V}\right)^{-1}\mathbf{V}^\top\right)\mathbf{D}^{-1/2}. \tag{5}$$

Further optimization is possible if one is only interested in the product of $\mathbf{A}^{-1}$ with a vector $\mathbf{x}$ instead of $\mathbf{A}^{-1}$ itself. The $O(nk^2 + k^3)$ cost of computing and inverting the $[k \times k]$ matrix is unavoidable, but the matrix multiplication operations can be transformed into matrix-vector multiplications by careful ordering. In terms of $\mathbf{U}$ and $\mathbf{D}$, letting $\mathbf{K} = \left(\mathbf{I}_k + \mathbf{U}\mathbf{D}^{-1}\mathbf{U}^\top\right)^{-1}$, the computational cost can be limited to $O(nk^2)$ by using the ordering

$$\mathbf{A}^{-1}\mathbf{x} = \mathbf{D}^{-1}\mathbf{x} - \mathbf{D}^{-1}\left(\mathbf{U}\left(\mathbf{K}\left(\mathbf{U}^\top\mathbf{D}^{-1}\mathbf{x}\right)\right)\right),$$

giving Algorithm 1.

Additional details
Also referred to as the Sherman-Morrison-Woodbury formula, the Woodbury's inversion formula is a generalization of the [Sherman-Morrison formula - Wikipedia] for rank-1 modifications, which states that given vectors $\mathbf{u}, \mathbf{v}$ and an arbitrary matrix $\mathbf{X}$,

$$\left(\mathbf{X} + \mathbf{u}\mathbf{v}^\top\right)^{-1} = \mathbf{X}^{-1} - \frac{\mathbf{X}^{-1}\mathbf{u}\mathbf{v}^\top\mathbf{X}^{-1}}{1 + \mathbf{v}^\top\mathbf{X}\mathbf{u}}. \tag{6}$$

Generalizations of the Woodbury formula include the [Binomial Inverse Theorem - Wikipedia], which allows for non-square $\mathbf{C}$ in Theorem 1.

Algorithm 1 Inverse: $\mathbf{y} = \mathbf{A}^{-1}\mathbf{x}$

```
$\mathbf{K} \leftarrow \left(\mathbf{I}_k + \mathbf{U}\mathbf{D}^{-1}\mathbf{U}^\top\right)^{-1}$
$\mathbf{y} \leftarrow \mathbf{D}^{-1}\left(\mathbf{U}\left(\mathbf{K}\left(\mathbf{U}^\top\left(\mathbf{D}^{-1}\mathbf{X}\right)\right)\right)\right) + \mathbf{D}\mathbf{x}$
```

2.2 Symmetric factorization

In the context of using a low-rank + diagonal structure for covariance matrices, having access to efficient matrix factorization enables efficient sampling algorithms. Assume that you want to sample from $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A})$, and have access to a square matrix $\mathbf{B}$ such that $\mathbf{B}\mathbf{B}^\top = \mathbf{A}$. Sampling $\boldsymbol{\epsilon}$ from $\mathcal{N}(\mathbf{0}, \mathbf{I}_n)$ and computing $\mathbf{B}\boldsymbol{\epsilon} + \boldsymbol{\mu}$ gives a sample from $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A})$.

To obtain a symmetric factorization for $\mathbf{A}$, we need a symmetric factorization for $\mathbf{V}\mathbf{V}^\top + \mathbf{I}_n$. This can be achieved using Theorem 3.1 in Ambikasaran et al. [2014].

Theorem 2 (Symmetric Factorization - Thm 3.1 from Ambikasaran et al. [2014])**.**
Let $\mathbf{U}, \mathbf{C}$ be matrices with sizes $\mathbf{U} : [n \times k], \mathbf{C} : [k \times k]$ and let $\mathbf{I}_n$ be the $[n \times n]$ identity matrix. Assuming that $\mathbf{I}_n + \mathbf{U}\mathbf{C}\mathbf{U}^\top$ is positive definite, a symmetric factorization is given by $\mathbf{W}\mathbf{W}^\top$ where

- *$\mathbf{L}$ is the symmetric factor $\mathbf{U}^\top\mathbf{U}$ and $\mathbf{M}$ is the symmetric factor of $\mathbf{I}_k + \mathbf{L}^\top\mathbf{C}\mathbf{L}$*
- *$\mathbf{K} = \mathbf{L}^{-\top}(\mathbf{M} - \mathbf{I}_k)\mathbf{L}^{-1}$ and $\mathbf{W} = \mathbf{I}_n + \mathbf{U}\mathbf{K}\mathbf{U}^\top$*

Computing the necessary matrices to construct $\mathbf{W}$,

$$\mathbf{L} = \text{Cholesky}(\mathbf{V}^\top\mathbf{V}), \qquad \mathbf{M} = \text{Cholesky}(\mathbf{I}_k + \mathbf{L}^\top\mathbf{L}), \qquad \mathbf{K} := \mathbf{L}^{-\top}(\mathbf{M} - \mathbf{I}_k)\mathbf{L}^{-1}, \tag{7}$$

costs $O(nk^2)$ computing $\mathbf{W} = \mathbf{I}_n + \mathbf{U}\mathbf{K}\mathbf{U}^\top$ adds $O(n^2k)$ operations in matrix multiplications, leading to a total cost in $O(n^2k)$. As for the matrix inversion, if one is not interested in $\mathbf{W}$ directly but only in its multiplication with a vector $\mathbf{x}$, as is the case in sampling, computing $\mathbf{W}$ is not efficient, especially if k is smaller than $\sqrt{n}$. Given $\mathbf{W}$, each matrix-vector multiplication will cost $O(n^2)$, but working with $\mathbf{V}, \mathbf{D}, \mathbf{K}$ directly leads to an $O(nk^2)$ algorithm by computing

$$\mathbf{D}^{1/2}\mathbf{W}\mathbf{x} = \mathbf{D}^{1/2}\left(\mathbf{x} + \mathbf{V}\left(\mathbf{K}\left(\mathbf{V}^\top\mathbf{x}\right)\right)\right), \tag{8}$$

giving Algorithm 3.

Symmetric factorization of the inverse
If instead the Gaussian distribution is parametrized with a low-rank + diagonal precision matrix, $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A}^{-1})$, we can combine inversion and factorization. Given $\mathbf{W} = \mathbf{I}_n + \mathbf{V}\mathbf{K}\mathbf{V}$, using Woodbury's inversion formula gives

$$\mathbf{W}^{-1} = \left(\mathbf{I}_n + \mathbf{V}\mathbf{K}\mathbf{V}^\top\right)^{-1} = \mathbf{I}_n - \mathbf{V}\left(\mathbf{K}^{-1} + \mathbf{V}\mathbf{V}^\top\right)^{-1}\mathbf{V}^\top. \tag{9}$$

Clever ordering leads again to a $O(nk^2)$ algorithm to compute $\mathbf{D}^{-1/2}\mathbf{W}^{-1}\mathbf{x}$, giving Algorithm 4.

Algorithm 2 $\mathbf{K} =$ `computeK`$(\mathbf{V})$

```
$\mathbf{V} \leftarrow \mathbf{D}^{-1/2}\mathbf{U}$
$\mathbf{L} \leftarrow \texttt{Cholesky}(\mathbf{V}^\top\mathbf{V})$
$\mathbf{M} \leftarrow \texttt{Cholesky}(\mathbf{I}_k + \mathbf{L}^\top\mathbf{L})$
$\mathbf{K} \leftarrow \mathbf{L}^{-\top}(\mathbf{M} - \mathbf{I}_k)\mathbf{L}^{-1}$
```

Algorithm 3 Factorization $\mathbf{y} = \mathbf{B}\mathbf{x}$, $\mathbf{B}\mathbf{B}^\top = \mathbf{A}$

```
$\mathbf{V} \leftarrow \mathbf{D}^{-1/2}\mathbf{U}$
$\mathbf{K} \leftarrow \texttt{computeK}(\mathbf{V})$
$\mathbf{w} \leftarrow \mathbf{x} + \mathbf{V}\left(\mathbf{K}\left(\mathbf{V}^\top\mathbf{x}\right)\right)$
$\mathbf{y} \leftarrow \mathbf{D}^{1/2}\mathbf{w}$
```

Algorithm 4 Inv. Fact. $\mathbf{y} = \mathbf{C}\mathbf{x}$, $\mathbf{C}\mathbf{C}^\top = \mathbf{A}^{-1}$

```
$\mathbf{V} \leftarrow \mathbf{D}^{-1/2}\mathbf{U}$
$\mathbf{K} \leftarrow \texttt{computeK}(\mathbf{V})$
$\mathbf{w} \leftarrow \mathbf{x} - \mathbf{V}\left(\left(\mathbf{K}^{-1} + \mathbf{V}^\top\mathbf{V}\right)^{-1}\left(\mathbf{V}^\top\mathbf{x}\right)\right)$
$\mathbf{y} \leftarrow \mathbf{D}^{-1/2}\mathbf{w}$
```

2.3 Determinant and Trace

Using the multiplicity of the determinant, $\det(\mathbf{XY}) = \det(\mathbf{X})\det(\mathbf{Y})$, we have that

$$\det(\mathbf{A}) = \det\left(\mathbf{D}^{1/2}\left(\mathbf{V}\mathbf{V}^\top + \mathbf{I}_n\right)\mathbf{D}^{1/2}\right) = \det(\mathbf{D})\det\left(\mathbf{V}\mathbf{V}^\top + \mathbf{I}_n\right). \tag{10}$$

To compute the determinant of $\mathbf{V}\mathbf{V}^\top + \mathbf{I}_n$, we can use Sylvester's Determinant Identity [Wikipedia] to get

$$\det(\mathbf{V}\mathbf{V}^\top + \mathbf{I}_n) = \det(\mathbf{V}^\top\mathbf{V} + \mathbf{I}_k). \tag{11}$$

Computing this determinant this way costs $O(nk^2)$ instead of $O(n^3)$. As a side note, the computation of the determinant is unstable in high dimension due to the large multiplications involved. It is preferable to compute the log-determinant directly, as done in Algorithm 5,

$$\log\det(\mathbf{A}) = \sum_{i=1}^{n}\log(\mathbf{D}_{ii}) + \log\det(\mathbf{I}_k + \mathbf{V}^\top\mathbf{V}). \tag{12}$$

Additional info
Similar to the relation between Woodbury's inversion formula and the Sherman-Morrison formula, there is a generalization of Sylvester's Determinant Identity to compute the determinant of matrices matching Theorem 1, the [Matrix Determinant Lemma - Wikipedia]

The trace
The trace of a matrix is typically not computationally intensive, as it reduces to a sum of the diagonal elements. For this setting, the challenge is in computing $\mathbf{A}$ itself, which already costs $O(n^2k)$ but is unnecessary since we are only interested in the diagonal. Given the linearity of the trace, we have that

$$\mathrm{Tr}(\mathbf{A}) = \mathrm{Tr}(\mathbf{U}\mathbf{U}^\top + \mathbf{D}) = \mathrm{Tr}(\mathbf{U}\mathbf{U}^\top) + \mathrm{Tr}(\mathbf{D}), \tag{13}$$

and the trace of the low-rank matrix reduces to

$$\mathrm{Tr}(\mathbf{U}\mathbf{U}^\top) = \sum_{i=1}^{n}\left(\mathbf{U}\mathbf{U}^\top\right)_{ii} = \sum_{i=1}^{n}\sum_{j=1}^{k}\mathbf{U}_{ij}^2, \tag{14}$$

giving an $O(nk)$ algorithm to compute the trace. and the cyclic property of the trace gives

$$\mathrm{Tr}(\mathbf{U}\mathbf{U}^\top) = \mathrm{Tr}(\mathbf{U}^\top\mathbf{U}). \tag{15}$$

If $\mathbf{A}$ is already computed, this procedure is more expensive than simply taking the trace. However, if only $\mathbf{U}$ and $\mathbf{D}$ are available, this procedure is equivalent in computational complexity as computing only the diagonal of the $\mathbf{U}\mathbf{U}^\top$ matrix. leading to a $O(nk^2)$ computational complexity to get the trace instead of $O(n^2k)$ to obtain $\mathbf{A}$ first.

Algorithm 5 Determinant: $\mathbf{y} = \det(\mathbf{A})$

```
$\mathbf{V} \leftarrow \mathbf{D}^{-1/2}\mathbf{U}$
$\mathbf{K} \leftarrow \mathbf{I}_k + \mathbf{V}^\top\mathbf{V}$
$\mathbf{y} \leftarrow \sum_{i=1}^{n}\log(\mathbf{D}_{ii}) + \log\det(\mathbf{K})$
```

2.4 Kullback-Leibler divergence

The efficient trace and determinant operation also makes it possible to compute the Kullback-Leibler divergence efficiently. Given two Multivariate Gaussian distributions with mean $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$ and covariance matrices $\mathbf{A}$ and $\boldsymbol{\Sigma}$, we have that

$$\mathbb{D}_{KL}\left(\mathcal{N}(\boldsymbol{\mu}_1, \mathbf{A}) || \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\right) = \frac{1}{2}\left(\mathrm{Tr}(\mathbf{A}^{-1}\boldsymbol{\Sigma}) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^\top\mathbf{A}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - n + \log\left(\frac{\det(\mathbf{A})}{\det(\boldsymbol{\Sigma})}\right)\right). \tag{16}$$

Application of the previous properties lead to a more efficient implementation, with computational complexity depending on the structure of $\boldsymbol{\Sigma}$.

3 Algorithms (Compact form)

The following algorithms show how to do the following operations

$\mathbf{A}^{-1}\mathbf{x}$,	$\mathbf{B}\mathbf{x}$ for $\mathbf{B}\mathbf{B}^\top = \mathbf{A}$,	$\mathbf{C}\mathbf{x}$ for $\mathbf{C}\mathbf{C}^\top = \mathbf{A}^{-1}$,	$\log\det(\mathbf{A})$,
Alg. 6	Alg. 8	Alg. 9	Alg. 10

when $\mathbf{A} = \mathbf{U}\mathbf{U}^\top + \mathbf{D}$ for $\mathbf{U} : [n \times k]$ and $\mathbf{D}$ a $[n \times n]$ diagonal matrix with positive elements.

Algorithm 6 Inverse: $\mathbf{y} = \mathbf{A}^{-1}\mathbf{x}$

```
  $\mathbf{K} \leftarrow \left(\mathbf{I}_k + \mathbf{U}\mathbf{D}^{-1}\mathbf{U}^\top\right)^{-1}$
  $\mathbf{y} \leftarrow \mathbf{D}^{-1}\left(\mathbf{U}\left(\mathbf{K}\left(\mathbf{U}^\top\left(\mathbf{D}^{-1}\mathbf{X}\right)\right)\right)\right) + \mathbf{D}\mathbf{x}$
```

Algorithm 7 $\mathbf{K} =$ `computeK`$(\mathbf{V})$

```
  $\mathbf{V} \leftarrow \mathbf{D}^{-1/2}\mathbf{U}$
  $\mathbf{L} \leftarrow \texttt{Cholesky}(\mathbf{V}^\top\mathbf{V})$
  $\mathbf{M} \leftarrow \texttt{Cholesky}(\mathbf{I}_k + \mathbf{L}^\top\mathbf{L})$
  $\mathbf{K} \leftarrow \mathbf{L}^{-\top}(\mathbf{M} - \mathbf{I}_k)\mathbf{L}^{-1}$
```

Algorithm 8 Factorization $\mathbf{y} = \mathbf{B}\mathbf{x}$, $\mathbf{B}\mathbf{B}^\top = \mathbf{A}$

```
  $\mathbf{V} \leftarrow \mathbf{D}^{-1/2}\mathbf{U}$
  $\mathbf{K} \leftarrow \texttt{computeK}(\mathbf{V})$
  $\mathbf{w} \leftarrow \mathbf{x} + \mathbf{V}\left(\mathbf{K}\left(\mathbf{V}^\top\mathbf{x}\right)\right)$
  $\mathbf{y} \leftarrow \mathbf{D}^{1/2}\mathbf{w}$
```

Algorithm 9 Inv. Fact. $\mathbf{y} = \mathbf{C}\mathbf{x}$, $\mathbf{C}\mathbf{C}^\top = \mathbf{A}^{-1}$

```
  $\mathbf{V} \leftarrow \mathbf{D}^{-1/2}\mathbf{U}$
  $\mathbf{K} \leftarrow \texttt{computeK}(\mathbf{V})$
  $\mathbf{w} \leftarrow \mathbf{x} - \mathbf{V}\left(\left(\mathbf{K}^{-1} + \mathbf{V}^\top\mathbf{V}\right)^{-1}\left(\mathbf{V}^\top\mathbf{x}\right)\right)$
  $\mathbf{y} \leftarrow \mathbf{D}^{-1/2}\mathbf{w}$
```

Algorithm 10 Determinant: $\mathbf{y} = \det(\mathbf{A})$

```
  $\mathbf{V} \leftarrow \mathbf{D}^{-1/2}\mathbf{U}$
  $\mathbf{K} \leftarrow \mathbf{I}_k + \mathbf{V}^\top\mathbf{V}$
  $\mathbf{y} \leftarrow \sum_{i=1}^{n} \log(\mathbf{D}_{ii}) + \log\det(\mathbf{K})$
```

References

S. Ambikasaran, M. O'Neil, and K. R. Singh. Fast symmetric factorization of hierarchical matrices with applications. 2014. URL `https://arxiv.org/abs/1405.0223v2`.