Phoneme Classification Using Kernel Principal Component Analysis

András Kocsor[1], László Tóth[2] and András Kuba[3]

February 28, 2000

Research Group on Artificial Intelligence of the Hungarian Academy of Sciences
and of the University of Szeged,
H-6720 Szeged, Aradi vértanúk tere 1., Hungary
e-mail:{[1]kocsor, [2]tothl, [3]andkuba}@inf.u-szeged.hu

Extended Abstract

1 Introduction

In an earlier paper[4] we compared the effect of the linear feature transformation methods Principal Component Analysis(*PCA*), Linear Discriminant Analysis(*LDA*) and Independent Component Analysis(*ICA*) on several learning algorithms. The algorithms compared were *TiMBL*(the *IB1* algorithm), *C4.5*(*ID3* tree learning), *OC1*(oblique tree learning), Artificial Neural Nets(*ANN*), Gaussian Mixture Modeling(*GMM*) and Hidden Markov Modeling(*HMM*). The domain of the comparison was phoneme classification using a certain segmental phoneme model, and each learner was tested with each transformation in order to find the best combination. Furthermore, in that paper we experimented with several feature sets such as filter bank energies, mel-frequency cepstral coefficients(*MFCC*) and gravity centers. This paper reports on our experiments to extend these investigations towards nonlinear methods. Namely, we show how the well-known Principal Component Analysis(*PCA*) can be non-linearized using the so-called "kernel-idea"[5]. Besides presenting the "Kernel-idea" we also give formulas both for the original *PCA* and the *Kernel-PCA*. In this paper we thoroughly examine how this nonlinear feature transformation effects the efficiency of several learning algorithms. As we mentioned previously in our earlier study we experimented with several feature sets, and we found the best one to be the critical band log-energies. So in this study we use only this quite traditional technique to extract frame-based features from the speech signal. We also learned from our previous investigations that from the learning algorithms *PCA*[2][6] was the most beneficial for *GMM*[1] and *ANN*[1]. Thus, in this paper we present classification results only for these two methods (apart from those of an *HMM* recognizer, which are given to serve as a reference point). Since the crucial point of this study is the *Kernel-PCA*, we give a brief look at it in this extended abstract.

2 Feature Space Transformation Methods with Kernels

Before executing a learning algorithm, additional vector space transformations may be applied on the extracted features. The role of these methods is twofold. Firstly they may improve classification performance, and secondly they may also reduce the dimensionality of the data. This is due to the fact that these techniques search for a transformation which emphasizes more important features and supresses or even eliminates less desirable ones.

2.1 Linear Feature Space Transformation Methods

Without loss of generality we will assume that the original data set lies in $\mathbb{R}^n$, and that we have l elements $\mathbf{x_1}, \ldots, \mathbf{x_l}$ in the training set and t elements $\mathbf{y_1}, \ldots, \mathbf{y_t}$ in the testing set. The feature transformation methods in many cases require certain preprocessing steps, which usually mean simple linear transformations. The results of this preprocessing step will be denoted by $\mathbf{\hat{x}_1}, \dots, \mathbf{\hat{x}_l}$ and $\mathbf{\hat{y}_1}, \dots, \mathbf{\hat{y}_t}$ for the training and testing vectors, respectively. After applying a feature space transformation method for the preprocessed data, the new data set lies in $\mathbb{R}^m \ \ (m \le n)$, the transformed training and testing vectors being denoted by $\mathbf{x_1'}, \dots, \mathbf{x_l'}$ and $\mathbf{y_1'}, \dots, \mathbf{y_r'}$ respectively. With the linear feature space transformation methods, we search for an optimal (in some cases orthogonal) linear transformation $\mathbb{R}^n \to \mathbb{R}^m$ of the form $\mathbf{x_i'} = \mathbf{A}^\top \mathbf{\hat{x}_i}, \ \ i \in \{1, \dots, l\}$, noting that the precise definition of optimality can vary from method to method. The column vectors $\mathbf{a_1}, \dots, \mathbf{a_m}$ of the $n \times m$ matrix $\mathbf{A}$ supposed to be normalized. These algorithms use various objective functions $\tau() : \mathbb{R}^n \to \mathbb{R}$ which serves as a measure for selecting one optimal direction (i.e. a new base vector). Usually, linear feature space transformation methods search for m optimal directions. Although it is possible to define functions that measure the optimality of all the m directions *together*, we will find the directions of the optimal transformations *one-by-one*, employing the τ measure for each direction separately. One, but quite heuristic way of this is to look for unit vectors which form the stationary points of $\tau()$. Intuitively, if larger values of $\tau()$ indicate better directions and the chosen directions needs to be independent in some ways, then choosing stationary points that have large values is a reasonable strategy.

2.2 Kernel Transformation Methods

In this subsection the symbols $\mathcal{H}$ and $\mathcal{F}$ denote real vector spaces that might as well be finite of infinite in dimension. Also, we suppose to have a mapping $\Phi : \mathbb{R}^n \to \mathcal{H}$, which is not necesseraly linear, and $\dim(\mathcal{H})$ is either finite or infinite. Furthermore, we suppose to have given an algorithm $\mathcal{P}$, with its input formed by preprocessed training points $\mathbf{\hat{x}_1}, \dots, \mathbf{\hat{x}_l}$ of the vector space $\mathbb{R}^n$. (In our case this algorithm is the *PCA*.) The output of the algorithm $\mathcal{P}$ is a linear transformation $\mathbb{R}^n \to \mathbb{R}^m$, where both the degree of the dimension reduction (represented by m) and the $n \times m$ transformation matrix $\mathbf{A}$ are determined by the algorithm itself. We will denote the transformation matrix $\mathbf{A}$ resulting for the training data by $\mathcal{P}(\mathbf{\hat{x}_1}, \dots, \mathbf{\hat{x}_l})$.

The goal of the nonlinearization methods is to transform the training vectors into a point set in $\mathcal{H}$ by a mapping Φ, and instead of the original ones in $\mathbb{R}^n$, we apply the algorithm $\mathcal{P}$ on these transformed points in $\mathcal{H}$. Thus, employing the algorithm $\mathcal{P}$ on the input elements $\Phi(\mathbf{\hat{x}_1}), \dots, \Phi(\mathbf{\hat{x}_l}) \in \mathcal{H}$ we gain a linear trasformation $\Psi : \mathcal{H} \to \mathcal{F}$. Similarly as before, we will denote the matrix of the resulting linear mapping Ψ with $\mathcal{P}(\oplus(\mathbf{\hat{x}_1}), \dots, \oplus(\mathbf{\hat{x}_l}))$. Since Φ is not linear in general, the composite transformation $\Psi \circ \Phi$ of Φ and Ψ will not necesseraly be linear either.

In the case of the Kernel transformation methods the algorithm $\mathcal{P}$ is turned into an equivalent algorithm $\mathcal{P}'$ for which the following holds: $\mathcal{P}(\mathbf{\hat{x}_1}, \dots, \mathbf{\hat{x}_l}) = \mathcal{P}'(\mathbf{\hat{x}_1}^\top \mathbf{\hat{x}_1}, \dots, \mathbf{\hat{x}_i}^\top \mathbf{\hat{x}_j}, \ \dots, \ \mathbf{\hat{x}_l}^\top \mathbf{\hat{x}_l})$, for arbitrary $\mathbf{\hat{x}_1}, \dots, \mathbf{\hat{x}_l}$. This will, of course, hold in $\mathcal{H}$ too, that is: $\mathcal{P}(\oplus(\mathbf{\hat{x}_1}), \dots, \oplus(\mathbf{\hat{x}_l})) = \mathcal{P}'(\oplus(\mathbf{\hat{x}_1})^\top \oplus(\mathbf{\hat{x}_1}), \dots, \ \Phi(\mathbf{\hat{x}_i})^\top \Phi(\mathbf{\hat{x}_j}), \ \dots, \ \Phi(\mathbf{\hat{x}_l})^\top \Phi(\mathbf{\hat{x}_l}))$, for arbitrary $\Phi(\mathbf{\hat{x}_1}), \dots, \Phi(\mathbf{\hat{x}_l})$. Thus, the point of the kernel methods is to form an algorithm $\mathcal{P}'$, which is equivalent to $\mathcal{P}$, but its inputs are the dot products of the inputs of $\mathcal{P}$.

The complexity of the linear *PCA*($\mathcal{P}$) is a non-linear function of the dimensionality of the input vectors. Thus if $\dim(\mathcal{H})$ is much larger than n, then the corresponding $\mathcal{P}'$ algorithm may become practically infeasible. This problem can be alleviated if we have a low-complexity (for example linear) Kernel function $\kappa() : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ for which $\Phi(\mathbf{x})^\top \Phi(\mathbf{y}) = \kappa(\mathbf{x}, \mathbf{y}), \ \ \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. In this case the value of $\Phi(\mathbf{\hat{x}_i})^\top \Phi(\mathbf{\hat{x}_j})$ can also be computed with few (for example $O(n)$) operations, even if the dimension of $\Phi(\mathbf{\hat{x}_i})$ and $\Phi(\mathbf{\hat{x}_j})$ are infinite. In practice, however, we usually face the problem just the opposite way: given a $\kappa() : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ functional as Kernel, we are looking for a mapping Φ for which $\Phi(\mathbf{x})^\top \Phi(\mathbf{y}) = \kappa(\mathbf{x}, \mathbf{y}), \ \ \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. There are many good publications about the proper choice of the Kernel functions, and also about their theory in general[5].

In our studies we employed the following Kernels: $\kappa_1(\mathbf{x}, \mathbf{y}) = \left(\mathbf{x}^\top \mathbf{y}\right)^p, \ 0 < p \in \mathbb{R}$ and $\kappa_2(\mathbf{x}, \mathbf{y}) = \exp\left(-||\mathbf{x} - \mathbf{y}||^2/r\right), 0 < r \in \mathbb{R}$. Thus, after choosing a Kernel function the only thing left is to take the $\mathcal{P}'$ version of the algorithm and replace the input elements $\mathbf{\hat{x}_1}^\top \mathbf{\hat{x}_1}, \dots, \mathbf{\hat{x}_i}^\top \mathbf{\hat{x}_j}, \dots, \mathbf{\hat{x}_l}^\top \mathbf{\hat{x}_l}$ with the elements $\kappa(\mathbf{\hat{x}_1}, \mathbf{\hat{x}_1}), \dots, \kappa(\mathbf{\hat{x}_i}, \mathbf{\hat{x}_j}), \dots, \kappa(\mathbf{\hat{x}_l}, \mathbf{\hat{x}_l})$. The algorithm that results from this substitution can perform the *PCA* transformation with practically acceptable complexity, even in a space infinite in dimension. This transformation together with a properly chosen Kernel function results in the non-linear feature space transformation, i.e. *Kernel-PCA*.

2.3 Steps of the Methods

In the following sections the discussion of the methods *PCA* and *Kernel-PCA* will be decomposed into three steps:

- **Preprocessing Step** Describes the preprocessing that might be required by the method.
- **Transformation Step** Here we derive the algorithms themselves.
- **Transformation of Test Vectors** Here we discuss that, having obtained a transformation based on the training vectors, what kind of processing it implies on the test vectors.

3 Principal Component Analysis

Preprocessing Step:

- **Centering:** We shift the original sample set $\mathbf{x_1}, \dots, \mathbf{x_l}$ with its mean $\boldsymbol{\mu}$, to obtain a set $\hat{\mathbf{x}}_\mathbf{1}, \dots, \hat{\mathbf{x}}_\mathbf{l}$, with a mean of $\mathbf{0}$: $\hat{\mathbf{x}}_\mathbf{1} = \mathbf{x_1} - \boldsymbol{\mu}, \dots, \hat{\mathbf{x}}_\mathbf{l} = \mathbf{x_l} - \boldsymbol{\mu}, \quad \boldsymbol{\mu} = l^{-1}\sum_{i=1}^{l} \mathbf{x_i}$.

Transformation Step:
Normally in *PCA* $\tau(\mathbf{a})$ is $\mathbf{a}^\top \mathbf{C}\mathbf{a}/\mathbf{a}^\top\mathbf{a}$ $(\mathbf{a} \in \mathbb{R}^n \setminus \{\mathbf{0}\})$, where $\mathbf{C}$ is the sample covariance matrix for the standardized data $(\mathbf{C} = l^{-1}\sum_{i=1}^{l} \hat{\mathbf{x}}_\mathbf{i}\hat{\mathbf{x}}_\mathbf{i}^\top)$. Practically speaking, $\mathbf{a}^\top \mathbf{C}\mathbf{a}/\mathbf{a}^\top\mathbf{a}$ defines $\tau(\mathbf{a})$ as the variance of the $\{\hat{\mathbf{x}}_\mathbf{1}, \dots, \hat{\mathbf{x}}_\mathbf{l}\}$ n-dimensional point-set projected onto the vector $\mathbf{a}$. Therefore this method prefers directions having a large variance. It can be shown that stationary points of $\tau(\mathbf{a})$ correspond to the right eigenvectors of the sample covariance matrix $\mathbf{C}$ where the eigenvalues form the corresponding function values. Thus it is worth defining *PCA* based on the stationary points where the function $\tau()$ has dominant values. If we assume that the eigenpairs of $\mathbf{C}$ are $(\mathbf{c_1}, \lambda_1), \dots, (\mathbf{c_n}, \lambda_n)$ and $\lambda_1 \geq \dots \geq \lambda_n$, then the transformation matrix $\mathbf{A}$ will be $[\mathbf{c_1}, \dots, \mathbf{c_m}]$, i.e. the eigenvectors with the largest m eigenvalues. Since the sample covariance matrix $\mathbf{C}$ is symmetric positive semidefinite, the eigenvectors are orthogonal and the corresponding real eigenvalues are nonnegative. After this orthogonal linear transformation the dimensionality of the data will be m. It is easy to check that the sample $\mathbf{x'_i} = \mathbf{A}^\top \hat{\mathbf{x}}_\mathbf{i}, \;\; i \in \{1, \dots, l\}$ represented in the new orthogonal basis will be uncorrelated, *i.e.* the covariance matrix $\mathbf{C}'$ of it is diagonal. The diagonal elements of $\mathbf{C}'$ are the m dominant eigenvalues of $\mathbf{C}$. In our experiments, m (the dimensionality of the transformed space) was chosen to be the smallest integer for which $(\lambda_1 + \dots + \lambda_m)/(\lambda_1 + \dots + \lambda_n) > 0.99$ holds. Note that there are many other alternatives, however, for finding a reasonable m.
Transformation of test vectors:
For an arbitrary test vector $\mathbf{y}$ the transformation is $\mathbf{y}' = \mathbf{A}^\top \hat{\mathbf{y}}$, where $\hat{\mathbf{y}}$ denotes the preporecessed $\mathbf{y}$.

4 Formulas for *Kernel-PCA*

Having chosen a proper κ Kernel function for which $\kappa(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x})^\top \Phi(\mathbf{y}), \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ holds for a mapping $\Phi : \mathbb{R}^n \to \mathcal{H}$, we now give the *LDA* transformation in $\mathcal{H}$.
Preprocessing Step:

- **Kernel Centering:** We shift the data $\Phi(\mathbf{x_1}), \dots, \Phi(\mathbf{x_l})$ with its mean $\boldsymbol{\mu}^\Phi$, to obtain a set $\hat{\Phi}(\mathbf{x_1}), \dots,$ $\hat{\Phi}(\mathbf{x_l})$ with a mean of $\mathbf{0}$: $\hat{\Phi}(\mathbf{x_1}) = \Phi(\mathbf{x_1}) - \boldsymbol{\mu}^{\boldsymbol{\Phi}}, \dots, \hat{\Phi}(\mathbf{x_l}) = \Phi(\mathbf{x_l}) - \boldsymbol{\mu}^{\boldsymbol{\Phi}}, \quad \boldsymbol{\mu}^{\boldsymbol{\Phi}} = \frac{1}{l}\sum_{i=1}^{l} \Phi(\mathbf{x_i})$.

Transformation Step:
We employed the following measure in $\mathcal{H}$: $\quad \tau^{\hat{\Phi}}(\mathbf{a}) = \mathbf{a}^\top \mathbf{C}^{\hat{\boldsymbol{\Phi}}}\mathbf{a}/\mathbf{a}^\top\mathbf{a}, \quad \mathbf{a} \in \mathcal{H} \setminus \{\mathbf{0}\}$, where $\mathbf{C}^{\hat{\boldsymbol{\Phi}}}$ $(= l^{-1}\sum_{i=1}^{l} \hat{\Phi}(\mathbf{x_i})\hat{\Phi}(\mathbf{x_i})^\top)$ is the covariance matrix of the sample $\hat{\Phi}(\mathbf{x_1}), \dots, \hat{\Phi}(\mathbf{x_l})$. Analogously to *PCA*, we define *Kernel-PCA* based on the stationary points of $\tau^{\hat{\Phi}}(\mathbf{a})$, which are given as the eigenvectors of the symmetric positive semidefinite matrix $\mathbf{C}^{\hat{\boldsymbol{\Phi}}}$. Because of the special form of $\mathbf{C}^{\hat{\boldsymbol{\Phi}}}$ we can suppose that $\mathbf{a} = \sum_{i=1}^{l} \alpha_i \hat{\Phi}(x_i)$. The following formulas give $\tau^{\hat{\Phi}}(\mathbf{a})$ as the function of α_t and $\kappa(\mathbf{x_i}, \mathbf{x_j})$

$$\tau^{\hat{\Phi}}(\mathbf{a}) = \frac{\mathbf{a}^\top \mathbf{C}^{\hat{\boldsymbol{\Phi}}}\mathbf{a}}{\mathbf{a}^\top\mathbf{a}} = \frac{\left(\sum_{t=1}^{l} \alpha_t \hat{\Phi}(x_t)^\top\right) \mathbf{C}^{\hat{\boldsymbol{\Phi}}} \left(\sum_{s=1}^{l} \alpha_s \hat{\Phi}(x_s)\right)}{\left(\sum_{t=1}^{l} \alpha_t \hat{\Phi}(x_t)^\top\right) \left(\sum_{s=1}^{l} \alpha_s \hat{\Phi}(x_s)\right)} = \frac{\boldsymbol{\alpha}^\top \frac{1}{l} \mathbf{K}^{\hat{\boldsymbol{\Phi}}} \mathbf{K}^{\hat{\boldsymbol{\Phi}}} \boldsymbol{\alpha}}{\boldsymbol{\alpha}^\top \mathbf{K}^{\hat{\boldsymbol{\Phi}}} \boldsymbol{\alpha}}, \tag{1}$$

where[1] $\mathbf{K}^{\hat{\mathbf{\Phi}}}{}_{ts} = \left(\Phi(\mathbf{x_t})^\top - \left(\frac{1}{l}\sum_{i=1}^{l}\Phi(\mathbf{x_i})^\top\right)\right)\left(\Phi(\mathbf{x_s}) - \left(\frac{1}{l}\sum_{i=1}^{l}\Phi(\mathbf{x_i})\right)\right) =$
$\kappa(\mathbf{x_t},\mathbf{x_s}) - \left(\frac{1}{l}\sum_{i=1}^{l}\left(\kappa(\mathbf{x_i},\mathbf{x_s}) + \kappa(\mathbf{x_t},\mathbf{x_i})\right)\right) + \frac{1}{l^2}\sum_{i=1}^{l}\sum_{j=1}^{l}\kappa(\mathbf{x_i},\mathbf{x_j})$. From differentiating $\tau^{\hat{\Phi}}()$ with respect to $\boldsymbol{\alpha}$ we get that the stationary points are the solution vectors of the general eigenvalue problem $\frac{1}{l}\mathbf{K}^{\hat{\mathbf{\Phi}}}\mathbf{K}^{\hat{\mathbf{\Phi}}}\boldsymbol{\alpha} = \lambda\mathbf{K}^{\hat{\mathbf{\Phi}}}\boldsymbol{\alpha}$, which in this case is obviously equivalent to the problem $\frac{1}{l}\mathbf{K}^{\hat{\mathbf{\Phi}}}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}$. Furthermore, since $\kappa(\mathbf{x_t},\mathbf{x_s}) = \kappa(\mathbf{x_s},\mathbf{x_t})$ and[2] $\boldsymbol{\alpha}^\top\frac{1}{l}\mathbf{K}^{\hat{\mathbf{\Phi}}}\boldsymbol{\alpha} = \frac{1}{l}\mathbf{a}^\top\mathbf{a} \geq 0$, the matrix $\frac{1}{l}\mathbf{K}^{\hat{\mathbf{\Phi}}}$ is symmetric positive semidefinite, and thus its eigenvectors are orthogonal and the corresponding real eigenvalues are non-negative. Let the m positive dominant eigenvalues of $\frac{1}{l}\mathbf{K}^{\hat{\mathbf{\Phi}}}$ be denoted by $\lambda_1 \geq \ldots \geq \lambda_m > 0$ and the correcponding normalized eigenvectors by $\boldsymbol{\alpha}^{\mathbf{1}},\ldots,\boldsymbol{\alpha}^{\mathbf{m}}$. Then the orthogonal matrix of the transformation we need can be calculated as below.

$$\mathbf{A}_{\hat{\mathbf{\Phi}}} := \left[\frac{1}{\sqrt{l\lambda_1}}\sum_{i=1}^{l}\alpha_i^1\hat{\Phi}(\mathbf{x_i}),\ldots,\frac{1}{\sqrt{l\lambda_m}}\sum_{i=1}^{l}\alpha_i^m\hat{\Phi}(\mathbf{x_i})\right], \tag{2}$$

where the factors $1/\sqrt{l\lambda}$ are needed to keep the columnvectors of $\mathbf{A}_{\hat{\mathbf{\Phi}}}$ normalized.
Transformation of Test Vectors:
Let $\mathbf{y}$ be an arbitrary test vector. After preprocessing $\Phi(\mathbf{y})$ we get that $\hat{\Phi}(\mathbf{y}) = \Phi(\mathbf{y}) - \boldsymbol{\mu}^\phi$. Then

$$\mathbf{y}' = \mathbf{A}_{\hat{\mathbf{\Phi}}}{}^\top\hat{\Phi}(\mathbf{y}) == \left[\frac{1}{\sqrt{l\lambda_1}}\sum_{i=1}^{l}\alpha_i^1 c_i,\ldots,\frac{1}{\sqrt{l\lambda_m}}\sum_{i=1}^{l}\alpha_i^m c_i\right]^\top, \tag{3}$$

where $c_i = \hat{\Phi}(\mathbf{x_i})^\top\hat{\Phi}(\mathbf{y}) = \kappa(\mathbf{x_i},\mathbf{y}) - \left(\frac{1}{l}\sum_{j=1}^{l}\left(\kappa(\mathbf{x_i},\mathbf{x_j}) + \kappa(\mathbf{x_j},\mathbf{y})\right)\right) + \frac{1}{l^2}\sum_{j=1}^{l}\sum_{k=1}^{l}\kappa(\mathbf{x_j},\mathbf{x_k})$. In our experience the strategy for obtaining a suitable m was the same as in PCA.

References

[1] Alder, M. D. *Principles of Pattern Classification: Statistical, Neural Net and Syntactic Methods of Getting Robots to See and Hear,* http://ciips.ee.uwa.edu.au/˜mike/PatRec, 1994.

[2] Battle, E., Nadeu, C. and Fonollosa, J. A. R. Feature Decorrelation Methods in Speech Recognition. A Comparative Study. In *Proceedings of ICSLP'98*, 1998.

[3] Kocsor, A., Kuba, A. Jr. and Tóth, L. An Overview of the OASIS speech recognition project, *In Proceedings of ICAI'99*, 1999(in press).

[4] Kocsor, A., Tóth, L., Kuba, A. Jr., Kovács, K., Jelasity, M., Gyimóthy, T. and Csirik, J. A Comparative Study of Several Feature Transformation and Learning Methods for Phoneme Classification submitted to *International Journal of Speech and Technology*.

[5] Schölkopf, B., Mika, S., Burges, C. J. C., Knirsch, P., Müller, K., Rätsch, G. and Smola, A. J. Input Space vs. Feature Space in Kernel-Based Methods, *IEEE Transactions on Neural Networks*, 1999(in press).

[6] Tipping, M. E. and Bishop, C. M. Probabilistic Principal Component Analysis, Technical Report NCRG/97/010, Neural Computing Research Group, Aston University, Birmingham, United Kingdom, 1997.

[1]Schölkopf et al. give $\mathbf{K}^{\hat{\mathbf{\Phi}}}$ in a matrix form using additional matrices. Our formula, however, turned out to be easier to code, and resulted in a more effective program.

[2]Here we temporarily disregard the constraint $\mathbf{a} \neq 0$.