2013

Computational Methods for Support Vector Machine Classification and Large-Scale Kalman Filtering

Marylesa Howard
The University of Montana

COMPUTATIONAL METHODS FOR SUPPORT VECTOR MACHINE CLASSIFICATION AND LARGE-SCALE KALMAN FILTERING

By

Marylesa Marie Howard

B.S., George Fox University, Newberg, OR, 2007
M.A., The University of Montana, Missoula, MT, 2009

Dissertation

presented in partial fulfillment of the requirements
for the degree of

Doctorate of Philosophy
in Mathematics

The University of Montana
Missoula, MT

May 2013

Approved by:

Sandy Ross, Associate Dean of the Graduate School
Graduate School

Dr. Johnathan Bardsley, Chair
Mathematical Sciences

Dr. Jon Graham
Mathematical Sciences

Dr. David Patterson
Mathematical Sciences

Dr. Jesse Johnson
Computer Science

Dr. Albert Parker
Center for Biofilm Engineering
Montana State University, Bozeman, MT

Howard, Marylesa, Doctorate of Philosophy, May 2013 Mathematics

Computational Methods for Support Vector Machine Classification and Large-Scale Kalman Filtering

Committee Chair: Johnathan Bardsley, Ph.D.

The first half of this dissertation focuses on computational methods for solving the constrained quadratic program (QP) within the support vector machine (SVM) classifier. One of the SVM formulations requires the solution of bound and equality constrained QPs. We begin by describing an augmented Lagrangian approach which incorporates the equality constraint into the objective function, resulting in a bound constrained QP. Furthermore, all constraints may be incorporated into the objective function to yield an unconstrained quadratic program, allowing us to apply the conjugate gradient (CG) method. Lastly, we adapt the scaled gradient projection method of [10] to the SVM QP and compare the performance of these methods with the state-of-the-art sequential minimal optimization algorithm and MATLAB's built in constrained QP solver, `quadprog`. The augmented Lagrangian method outperforms other state-of-the-art methods on three image test cases.

The second half of this dissertation focuses on computational methods for large-scale Kalman filtering applications. The Kalman filter (KF) is a method for solving a dynamic, coupled system of equations. While these methods require only linear algebra, standard KF is often infeasible in large-scale implementations due to the storage requirements and inverse calculations of large, dense covariance matrices. We introduce the use of the CG and Lanczos methods into various forms of the Kalman filter for low-rank approximations of the covariance matrices, with low-storage requirements. We also use CG for efficient Gaussian sampling within the ensemble Kalman filter method. The CG-based KF methods perform similarly in root-mean-square error when compared to the standard KF methods, when the standard implementations are feasible, and outperform the limited-memory Broyden-Fletcher-Goldfarb-Shanno approximation method.

Acknowledgments

I would like to thank my advisor, Dr. Johnathan Bardsley, for being an outstanding example of what it means to be an avid researcher. You have undoubtedly provided me with the skills I will need to succeed as I venture forth from this institution. This past six years you have worked to mold me into a well-rounded mathematician and your efforts do not go unnoticed. I will forever be thankful for the time I have spent as your student.

Thank you, Dr. Jon Graham, for your strong example of teaching and work ethic. From day one, you demand respect from your students and show them respect in return, creating a wonderful class dynamic. I like to think that some of my teaching methodologies have been developed by observing you. Additionally, you have demonstrated that sometimes in life we can have it all if we are willing to sacrifice sleep.

I would like to thank Dr. David Patterson for reminding me to think outside the box, even in statistics. I will never forget that one can estimate the number of blades of grass on the Oval by taking a sample of mowed grass and weighing it. I will always strive to attain your level of understanding, as you always find a way to ask intelligent questions in every seminar you attend.

To Dr. Jesse Johnson, thank you for sharing your passion of the sciences, love for teaching, and enthusiasm to develop the sciences in young minds. You demand a lot from your students, but the depth of understanding we gain from it is absolutely worthwhile.

Thank you, Dr. Albert Parker for your post-graduate advice, contribution to this dissertation, collaborative research, and for not being afraid to step on Grizzly territory, by which you have taught me courage. I hope we may continue collaborating on research in the future.

To my husband, Kaleb, thank you for your strength and support. You never once asked me to choose between education and you – that means more to me than I can express. Thank you. Lastly, through it all, my family has always been a wealth of encouragement to me and has always kept me laughing. We are nothing if we don't have laughter.

Thank you, everyone.

Notations

$'$ ..... Prime denotes transposition of matrix/vector
$\mathbb{R}$ ..... Real numbers
$\boldsymbol{x}_k$ ..... A vector indicating the k^{th} iterate of vector $\boldsymbol{x}$
x_i ..... A scalar indicating the i^{th} component of vector $\boldsymbol{x}$
p ..... Training set size
$\forall$ ..... For all
$\in$ ..... Element of a set
$|T|$ ..... The cardinality of set T
$E[\cdot]$ ..... Expected value
$\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ ..... Gradient of $f(\boldsymbol{x})$
ξ ..... Slack variable
L_p ..... Primal Lagrangian
L_d ..... Dual Lagrangian
$< \boldsymbol{x}, \boldsymbol{y} >$ ..... Inner product of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$
$K(\boldsymbol{x}, \boldsymbol{z})$ ..... Kernel $\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle$
$\boldsymbol{x} \perp \boldsymbol{y}$ ..... Vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are independent
I ..... Identity matrix

Contents

List of Tables

List of Figures

Preface

This dissertation consists of two parts: a chapter on constrained optimization methods for classification with the support vector machine and a chapter on conjugate gradient implementation within the Kalman filter.

Chapter 1 opens with a review of classification techniques, followed by derivations for the support vector machine classifier. Optimization methods developed by the author are presented for the bound and linear equality constrained quadratic program. Numerical examples demonstrate the improvement of the newly designed algorithms over state-of-the-art techniques.

A discussion of various Kalman filter methods is given in Chapter 2, followed by our implementation of the conjugate gradient (CG) method as a means for optimization and to build low-storage, low-rank approximations of covariance and inverse-covariance matrices. Examples compare the Kalman filter methods with CG to that with limited-memory BFGS, with improvement in root-mean-square error demonstrated by the methods utilizing CG. The implementation of CG within the Kalman filter, variational Kalman filter, and extended Kalman filter has been published in [6]. A second paper regarding implementation of CG within the ensemble Kalman filter has also been published [7].

Chapter 1

Constrained Optimization Methods for Support Vector Machine Classification

The first chapter of this dissertation presents our work with the support vector machine classification method. Within the support vector classifier framework, we introduce new algorithms for solving the bound and equality constrained quadratic program.

1.1 Introduction

Solving practical and mathematical problems on the computer often requires a written program that calculates the desired outputs explicitly from the inputs. As the problem becomes more and more complex, the computation may become too expensive, or there may be no known method for calculating outputs directly from inputs. One method to circumvent this problem is to have computers learn by example.

Computers have demonstrated a great ability to learn from experience and through examples, though the extent of this ability has not yet been determined [22]. Reliable learning tactics are necessary as many tasks may not be solved via classical programming techniques. For example, computer programmers have not written a program that can automatically recognize handwritten characters without first learning to recognize the characters through examples, just as a child learns to read [22]. This is referred to as *statistical learning*. Furthermore, computers detect spam emails by learning from examples [49].

Supervised learning uses the input/output examples provided by the user to build a prediction model for new data. The output variable serves to aide in the creation of the learner during the learning process. The *training data* are the paired input/output data used to train the machine. For example, in imaging problems, the training data are sets of pixels that correspond to each class in the image. A classifier is built from the training data, which is then applied to the *test set*. To continue with the imaging example, the test set is the set of remaining pixels in the image for which the classes are unknown. While it would be most beneficial to learn the *target function*–the underlying function that maps inputs to outputs–this function does not always exist, such as for cases in which outputs are noisy [22]. Rather, the solution, or *decision function*, is the estimate of the target function, which is output by the learning algorithm.

The learning task in *unsupervised learning* is to gain understanding of the process which generated the data, describing how the data are clustered and organized, and therefore does not use the output training data in the learning process. Some common methods of unsupervised learning include clustering, density estimation, and learning the support of a distribution. Unsupervised learning is less developed in the literature [49] and is not a focus of this thesis.

In classical statistical literature, the inputs are known as predictors or independent variables and the outputs are referred to as responses or dependent variables. The type of output depends on the problem. Determining digitized numbers on a mailed envelope is a type of

qualitative output, for which the set of outputs has the form $\mathcal{G} = \{0, 1, \ldots, 9\}$. Here we have categories, or *classes*, as outputs. These qualitative variables are also referred to as categorical variables, discrete variables, or factors. Quantitative variables have a numerical value and measurements closer in value are closer in nature than measurements further in value. Some numerical outputs, such as the binary output set $\{0, 1\}$, can be considered as either quantitative or qualitative outputs, depending on the meaning behind the numbers.

When learning from data, it is typical to have an outcome measurement, typically qualitative (heart attack/no heart attack, diabetes/no diabetes) or quantitative (stock price, atmospheric pressure), that we wish to predict based on a set of *features* (such as diet and clinical measurements). The *training set* is the paired observed feature measurements and outcomes for a set of objects (people, pixels, etc). Using the training data we build a prediction model, or learner, which allows us to predict outcomes for new and unseen objects. The goal of a good learner is to accurately predict the output from the inputs.

This distinction between the types of output variables gave rise to different names for the set of predictions from learners. When requiring qualitative responses, the process is referred to as *classification* and when the output is quantitative, the process is called *regression*. Here we focus primarily on classification methods.

A learning problem with binary outputs, such as the detection of disease, is a *binary classification* problem and the outcomes are typically represented numerically by 0/1 or by -1/+1. These numeric codes for categorical responses are often referred to as *targets*. In the binary case, the target may also be treated as a quantitative output, generally lying in the interval $[0, 1]$. A learning problem with a finite number of categorical outputs, such as identifying letters of the alphabet, is *multi-class classification*. For example, the famous iris data set presented by R. A. Fisher uses quantitative inputs (sepal length, sepal width, petal length, petal width) to predict iris species with class set $\mathcal{G} = \{\text{Setosa, Versicolor, Virginica}\}$.

The method in which the training data are presented to the learner determines the type of learning. With *batch* learning, the learner is provided with all the training data at the start of the learning process, by which the classifier is learned. For *on-line* learning, a single training datum is given to the learner, from which the learner determines its estimate of the output and then receives the corresponding correct output value to determine how it performed. This process is repeated for the entire training data set, adding one datum at a time to determine its estimated output so that the learner can update its hypothesis after each new training datum is presented to obtain the final classifier.

On-line learning lends itself to a simple form of assessment: totaling the number of mistakes made during learning. For batch learning, a good assessment of the classifier on unknown data is not immediately as clear, however the goal of early machine learning was to perform a correct classification of the training data [22] and problems may arise when such a hypothesis is created to be consistent with the training data. Also, if the training data are noisy, then there may be no decision function to correctly map inputs to outputs.

If we can find a decision function that is consistent with the training data, there is no guarantee it will correctly classify the unseen data. When the decision function becomes too complex in order to remain consistent with the training data, it has become overfit. For example, decision trees may be grown large enough that each leaf is a training example [49]. Therefore, one may choose to prune the decision tree to reduce overfitting.

The remainder of this chapter is organized as follows. We introduce notation in Section 1.2, followed by a discussion on widely-used batch learning methods in Section 1.3. Methods for determining misclassification error are discussed in Section 1.4. The support vector classifier is derived in Section 1.5 and the heart of the research contributed in this chapter by the author on constrained quadratic programming is given in Section 1.6. Section 1.7 introduces three image test cases to compare the constrained quadratic optimization methods and conclusions are given in Section 1.8.

1.2 Notation

The most common notation used in this chapter is presented here. Input variables are typically vectors, and are denoted by $\boldsymbol{X}$. Outputs are represented by Y and qualitative outputs are represented by G (categorical group). We let $\mathcal{G}$ represent the set of categorical outputs. The i^{th} observation of variable $\boldsymbol{X}$ is denoted $\boldsymbol{x}_i$ and the j^{th} component of vector $\boldsymbol{x}$ is denoted x_j. Bold lowercase letters represent vectors while matrices will be represented with bold uppercase letters. The transposes of $\boldsymbol{x}$ and $\mathbf{A}$ are denoted $\boldsymbol{x}'$ and $\mathbf{A}'$, respectively. We assume p pairs of training data are provided for batch learning, and are represented by $(\boldsymbol{x}_i, y_i)$, with $i = 1, \ldots, p$.

The prediction for quantitative outputs is $\hat{y}(\boldsymbol{x})$ and for categorical outputs is $\widehat{G}$. Thus if Y takes on values in $\mathbb{R}$, then so should $\hat{y}(\boldsymbol{x})$. Similarly, if the outputs take on values from the set of classes $\mathcal{G}$, then so should the prediction, $\widehat{G}(\boldsymbol{x})$. We denote an observation $\boldsymbol{x}$ belonging to the k^{th} class of $\mathcal{G}$ by $\boldsymbol{x} \in g_k$.

Typically, in the support vector machine (SVM) literature, the number of features of observation $\boldsymbol{x}$ is n and the number of training observations is p; however, in the statistical literature, it is the opposite. Since this chapter focuses on the SVM implementation, we use the common SVM notation, even for the literature review of other classification methods.

1.3 A Review of Supervised Classification Methods

The purpose of this section is to introduce the reader to widely used classification methods as a background to the support vector machine method introduced in Section 1.5. A variety of algorithms exist for supervised classification, but they all consist of the same basic steps. First, a set of output categories must be chosen for the data set. For example, if working with a satellite image of a floodplain, the categories may include water, cobble, regenerative material, deciduous forest, shrubbery, etc. Second, training data are chosen for each output category.

Continuing with the satellite image example, the training data are sets of pixels corresponding to each category chosen. Third, the training data are used to estimate parameters that describe either the probability model or the partitions in feature space. Lastly, once the classifier has been learned, it is used to classify the test set. Returning to the satellite image example, this consists of classifying the remaining pixels not in the training set. Preferably some method to assess accuracy would then be applied.

1.3.1 Least Squares

The least squares method places structural assumptions on the data in feature space (linear separability) and produces stable, though possibly inaccurate, predictions [49]. We begin by describing the linear model which leads to least squares. Linear models have become an important tool of classical statistics [49]. Considering an input $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)'$, where n is the number of features, the quantitative output variable Y is predicted using the linear model

$$\hat{y}(\boldsymbol{x}) = \hat{\beta}_0 + \sum_{i=1}^{n} x_i \hat{\beta}_i, \tag{1.1}$$

where the intercept $\hat{\beta}_0$ is known as the *bias* in machine learning [49, p. 11]. If we choose to rewrite $\boldsymbol{x} = (1, x_1, \ldots, x_n)'$, then (1.1) becomes the inner product

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{x}'\widehat{\boldsymbol{\beta}},$$

where $\widehat{\boldsymbol{\beta}} = (\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_n)'$. If the constant of 1 is included in $\boldsymbol{x}$, then $(\boldsymbol{x}, \hat{y})$ represents a hyperplane that passes through the origin in $(n+1)$-dimensional space. When the constant is not included, the n-dimensional hyperplane $(\boldsymbol{x}, \hat{y})$ intercepts the y-axis at $(\mathbf{0}, \hat{\beta}_0)$. We assume the constant is included. Here $\hat{y}$ is a scalar and $\boldsymbol{x}$ is a vector, but we note that we can use this model to predict the outcome of many realizations at one time by letting $\mathbf{X}$ be a matrix of observational input columns and $\hat{\boldsymbol{y}}$ a vector.

The least squares method chooses the vector $\boldsymbol{\beta}$ that minimizes the residual sum of squares (RSS)

$$\begin{aligned} \mathrm{RSS}(\boldsymbol{\beta}) &= \sum_{i=1}^{p} (y_i - \boldsymbol{x}_i'\boldsymbol{\beta})^2 \\ &= (\boldsymbol{y} - \mathbf{X}'\boldsymbol{\beta})'(\boldsymbol{y} - \mathbf{X}'\boldsymbol{\beta}), \end{aligned} \tag{1.2}$$

where $\mathbf{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p]$, the matrix of p observations. The *normal equations* are obtained through differentiation of (1.2) with respect to $\boldsymbol{\beta}$,

$$\mathbf{X}(\boldsymbol{y} - \mathbf{X}'\boldsymbol{\beta}) = \mathbf{0}.$$

Assuming $\mathbf{X}\mathbf{X}'$ is nonsingular, or the rows of $\mathbf{X}$ are linearly independent, the unique least squares solution is

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}\mathbf{X}')^{-1}\mathbf{X}\boldsymbol{y}.$$

The vector $\widehat{\boldsymbol{\beta}}$ is built from training data inputs $\mathbf{X}$ and outputs $\boldsymbol{y}$, from which we calculate $\hat{y}_i = \boldsymbol{x}_i{}'\widehat{\boldsymbol{\beta}}$ to be the predicted (fitted) value for any input observation $\boldsymbol{x}_i$. It is mentioned in [49, p. 12] that a large training data set is not needed to fit the least squares model.

Note that the least squares model may be fit to binary output data of 0's and 1's using the rule

$$\widehat{G}(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \hat{y}(\boldsymbol{x}) \leq 0.5, \\ 1 & \text{if } \hat{y}(\boldsymbol{x}) > 0.5. \end{cases} \tag{1.3}$$

In this case the *decision boundary*–the boundary in input feature space which divides the classes–is $\{\boldsymbol{x} : \boldsymbol{x}'\boldsymbol{\beta} = 0.5\}$. This is one possible approach for binary output data in other models as well [22], though logistic regression is another common technique [49].

When the training data are generated from Gaussian distributions, linear decision boundaries are most appropriate; however, should the data be mixtures of tightly clustered Gaussians, a nonlinear and disjoint decision boundary is more optimal [49, p. 14].

1.3.2 k-Nearest Neighbor

Nearest neighbor methods are cluster methods that use training observations closest in input feature space to a new observation $\boldsymbol{x}$ to predict $Y(\boldsymbol{x})$. While the k-nearest neighbor (k-NN) learner is supervised, it requires no structural assumption on the data as inference via least squares does, and thus predictions may be more accurate, but are also sometimes unstable due to the decision boundary which is heavily influenced training data [49, p. 16]. Moreover, this algorithm is typically computationally expensive [48, 79].

Essentially, when a prediction is desired for a new data vector $\boldsymbol{x}$, k-NN averages the k closest training data in input feature space to observation $\boldsymbol{x}$. Mathematically, this prediction is

$$\hat{y}(\boldsymbol{x}) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} y_i,$$

where $N_k(\boldsymbol{x})$, the *neighborhood* of $\boldsymbol{x}$, is defined as the k training points $\boldsymbol{x}_i$ closest in feature space to unknown observation $\boldsymbol{x}$. To determine closeness, the Euclidean distance metric is typically used.

This can also be adapted for classification, in addition to regression learning, by using an equation such as (1.3) for binary classification, or for multi-class classification, determining which class dominates (the mode function) amongst the k closest training samples. The decision boundaries for k-NN are far more irregular than the linear decision boundary produced in least squares, which allows for the learner to respond to local areas of clustered data in which one class dominates over another [49].

1.3.3 Bayes' Classifier

The formulation of Bayes' classifier is intuitive and its decision rule places an observation into the most probable class, based on some assumed probability distribution. Bayes' classifier does not make distributional assumptions on the data $\boldsymbol{x}$ and solely describes the decision rule. Classification methods such as linear discriminant analysis and quadratic discriminant analysis make distributional assumptions and use the Bayes' classifier decision rule to classify observations.

We begin by considering a loss function, $L(G, \widehat{G}(\boldsymbol{X}))$, which penalizes prediction errors. The loss function indicates $L(g_k, \hat{g}_l)$ is the price paid for an observation belonging to g_k predicted (classified) as being in g_l. It is typical for categorical output data to use the *zero-one loss function* [49], which can be represented by matrix $\mathbf{L}_{K\times K}$ where $K = |\mathcal{G}|$ and the ij^{th} entry of $\mathbf{L}$ is defined as

$$[\mathbf{L}(G, \widehat{G}(\boldsymbol{X})]_{i,j} = \begin{cases} 0 & \text{if } i = j, \\ 1 & \text{if } i \neq j. \end{cases} \tag{1.4}$$

The expected prediction error (EPE) is the expectation of the prediction error function, or

$$EPE = E\left[L\left(G, \widehat{G}(\boldsymbol{X})\right)\right], \tag{1.5}$$

where the expectation is with respect to the joint distribution $p(G, \boldsymbol{X})$. Conditioning on $\boldsymbol{X}$ and using iterated expectations, the expectation in (1.5) can be rewritten as

$$\begin{aligned} EPE &= E_{\boldsymbol{X}}\left[E_{G|\boldsymbol{X}}\left[L(g_k, \widehat{G}(\boldsymbol{X}))|\boldsymbol{X}\right]\right] \\ &= E_{\boldsymbol{X}}\left[\sum_{k=1}^{K} L(g_k, \widehat{G}(\boldsymbol{X}))p(g_k|\boldsymbol{X})\right]. \end{aligned}$$

Minimizing the expected prediction error can be done point-wise [49, p. 21], yielding

$$\widehat{G}(\boldsymbol{X} = \boldsymbol{x}) = g_k \quad \text{if} \quad p(g_k|\boldsymbol{X} = \boldsymbol{x}) \geq p(g_l|\boldsymbol{X} = \boldsymbol{x}), \quad \forall\, l \neq k. \tag{1.6}$$

Equation (1.6) defines the Bayes' classifier. In simplest terms, the observation $\boldsymbol{x}$ is placed into the most probable class defined using the conditional distribution $p(G|\boldsymbol{X})$. Now that we have Bayes' classifier, it remains to define $p(G|\boldsymbol{X})$.

Considering the probability $p(g_k|\boldsymbol{x})$, Bayes' theorem yields

$$p(g_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|g_k)p(g_k)}{p(\boldsymbol{x})}, \tag{1.7}$$

where $p(\boldsymbol{x}|g_k)$ is the probability of data vector $\boldsymbol{x}$ given class g_k, $p(g_k)$ is the relative frequency with which the k^{th} class occurs in the data set and $p(\boldsymbol{x})$ is the probability of observing data vector $\boldsymbol{x}$. In Bayes' terminology, $p(g_k)$ is referred to as the prior probability and is used to model *a priori* knowledge, while $p(g_k|\boldsymbol{x})$ is referred to as the class posterior.

Substituting (1.7) into (1.6), Bayes' classifier places $\boldsymbol{x}$ into class g_k when

$$p(\boldsymbol{x}|g_k)p(g_k) > p(\boldsymbol{x}|g_l)p(g_l), \quad \forall\, l{\neq}k. \tag{1.8}$$

To simplify (1.8), we define the discriminant function $\delta_k(\boldsymbol{x})$ to be the natural logarithm of the left-hand side of (1.8). Performing this transform on both sides of (1.8) preserves the inequality since the logarithm is a monotone increasing function:

$$\delta_k(\boldsymbol{x}) = \ln p(\boldsymbol{x}|g_k) + \ln p(g_k), \quad k = 1, \ldots, K. \tag{1.9}$$

Thus the Bayes' classifier places $\boldsymbol{x}$ into class g_k when

$$\delta_k(\boldsymbol{x}) > \delta_l(\boldsymbol{x}), \quad \forall\, l{\neq}k. \tag{1.10}$$

It remains to specify $p(\boldsymbol{x}|g_k)$ and $p(g_k)$. Two widely-used techniques for determining a model of $p(\boldsymbol{x}|g_k)$ are linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA), both of which perform well on large data sets [49]. While derivations for both techniques are determined from Gaussian assumptions and LDA assumes equal covariance matrices between classes, it is suggested in [49, p. 111] that their wide use is due to the fact that data often have linear or quadratic decision boundaries, in which case, the estimates provided by LDA and/or QDA are accurate, as well as stable.

A Gaussian assumption on $p(\boldsymbol{x}|g_k)$ leads to

$$p(\boldsymbol{x}|g_k) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}_k|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_k)'\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{m}_k)},$$

where n is the number of features and $\boldsymbol{m}_k$ and $\boldsymbol{\Sigma}_k$ are the mean vector and covariance matrix, respectively, of the data belonging to class g_k. In practice, we estimate $\boldsymbol{m}_k$ and $\boldsymbol{\Sigma}_k$ using the training data. The prior probability $p(g_k)$ is determined using a priori knowledge of the data set. Here we assume a particular vector $\boldsymbol{x}$ has the same probability of being classified into any one of the classes. However, if one has reason to believe that a data vector is more likely to be in one class than another, then different priors may be assigned.

Linear Discriminant Analysis

Linear discriminant analysis (LDA) is derived from the assumption that the classes have equal covariance matrices, that is $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$ for all k. Given this covariance assumption and the Gaussian assumption on $p(\boldsymbol{x}|g_k)$, the discriminant function in (1.9) is reduced to

$$\delta_k(\boldsymbol{x}) = -\frac{n}{2}\ln 2\pi - \frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{m}_k)'\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{m}_k) + \ln p(g_k).$$

The terms $-\frac{n}{2}\ln 2\pi$, $-\frac{1}{2}\ln|\boldsymbol{\Sigma}|$ and $\boldsymbol{x}'\boldsymbol{\Sigma}\boldsymbol{x}$ appear in all $\delta_k(\boldsymbol{x})$ and the term $\ln p(g_k)$ is assumed equal for all k, thus they can be ignored. Moreover, since it is (1.10) that is of interest, we

can equivalently use the discriminant function

$$\delta_k(\boldsymbol{x}) = \boldsymbol{x}'\boldsymbol{\Sigma}^{-1}\boldsymbol{m}_k - \frac{1}{2}\boldsymbol{m}_k'\boldsymbol{\Sigma}^{-1}\boldsymbol{m}_k. \tag{1.11}$$

Note that $\delta_k(\boldsymbol{x})$ is linear in $\boldsymbol{x}$, and finally that we have derived the LDA technique, which uses the discriminant functions of (1.11) to classify the data using decision rule (1.10).

Quadratic Discriminant Analysis

If instead we assume the covariance matrices between classes are not equal, we obtain the quadratic discriminant analysis (QDA) classifier. Then, canceling constant terms the discriminant function takes the form

$$\delta_k(\boldsymbol{x}) = -\ln|\boldsymbol{\Sigma}_k| - (\boldsymbol{x}-\boldsymbol{m}_k)'\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{m}_k). \tag{1.12}$$

Note that $\delta_k(\boldsymbol{x})$ is quadratic in $\boldsymbol{x}$, and we use the discriminant functions defined in (1.12) to classify data using the decision rule in (1.10).

Minimum Distance

While QDA is a popular technique [79], it requires sufficient training data to estimate $\boldsymbol{m}_k$ and $\boldsymbol{\Sigma}_k$. In the case when less training data are available, it may be wiser to use a method which does not require a covariance matrix, rather only the mean positions of the classes. In such a case, one can use the minimum distance classifier.

The minimum distance classifier is faster than QDA, but because it does not utilize the covariance information, it is less accurate in its classification if the covariance can be estimated [79]. Using training data to estimate a class mean, the minimum distance classifier then places

each training data vector in the class corresponding to the nearest mean in Euclidean distance.

The squared Euclidean distance of unknown $\boldsymbol{x}$ to each of the class means is computed by

$$\begin{aligned} d(\boldsymbol{x}, \boldsymbol{m}_k)^2 &= (\boldsymbol{x} - \boldsymbol{m}_k)'(\boldsymbol{x} - \boldsymbol{m}_k) \\ &= \boldsymbol{x}'\boldsymbol{x} - 2\boldsymbol{m}_k'\boldsymbol{x} + \boldsymbol{m}_k'\boldsymbol{m}_k, \quad \forall\, k \end{aligned} \tag{1.13}$$

Thus classification places $\boldsymbol{x} \in g_k$ if

$$d(\boldsymbol{x}, \boldsymbol{m}_k)^2 < d(\boldsymbol{x}, \boldsymbol{m}_l)^2, \quad \forall\, l \neq k.$$

Simplifying, we notice the first term in the right hand side of (1.13) is the same regardless of class, and by negating the expression we arrive at the discriminant function for the minimum distance classifier,

$$\delta_k(\boldsymbol{x}) = 2\boldsymbol{m}_k'\boldsymbol{x} - \boldsymbol{m}_k'\boldsymbol{m_k}, \quad \forall\, k,$$

which is used to classify data using the decision function in (1.10).

1.4 Error

The ability of a learning method to correctly predict new observations is called its *generalization*. While there are many ways to approach error in the statistical modeling setting, considering batch classification, two suggested methods for estimating prediction error are cross-validation and bootstrapping [13].

1.4.1 K-Fold Cross-Validation

Cross-validation is a simple and widely-used method for estimating prediction error [49]. The expected error is directly estimated with

$$Err = E[L(G, \widehat{G}(\boldsymbol{X}))],$$

where the expectation is with respect to the unconditional joint distribution of $(\boldsymbol{X}, G)$ and L is the loss function of the model applied to a observation $\boldsymbol{X}$ [49, p. 220]. Again, it is typical to use the zero-one loss function described in Section 1.3.3 for categorical data.

In an ideal world with an excess of training data, one would build the model with a majority of the training data, while the remaining training data are saved to be used as a validation set. However, training data may be difficult or expensive to obtain, in which case this is an infeasible option. K-fold cross-validation is one solution to this problem and measures the performance of the model [12, 49, 79].

The general process of K-fold cross-validation begins by randomly partitioning the training data set into K approximately equal-sized parts. The first part is placed aside to become the new testing set and the remaining $K - 1$ parts become the new training set. The classifier is created with this adjusted training set and is used to classify the new testing set. For example, when $K = 5$, 95% of the training data is used to build a classifier, from which the remaining 5% of the training data is classified. Since the true classification of the training set is known, the classification of the new testing set can be compared to the truth with a loss function. This process is repeated K times (folds) so that each of the K sets is used exactly once as a testing set. Note that this process can be repeated and the results averaged for a more stable estimate of error.

In the case of classification, the zero-one loss function may be used to give the estimate of

prediction error as

$$\widehat{Err}_{\text{CV}} = \frac{1}{p}\sum_{i=1}^{p} L(g(\boldsymbol{x}_i), \widehat{\mathcal{G}}_{-i}(\boldsymbol{x}_i)),$$

where L is defined in (1.4), $g(\boldsymbol{x}_i)$ is the group that training point $\boldsymbol{x}_i$ belongs to, and $\widehat{\mathcal{G}}_{-i}(\boldsymbol{x}_i)$ is the predicted group for observation $\boldsymbol{x}_i$ using the classifier built without the k^{th} training set to which $\boldsymbol{x}_i$ belongs. In this case, the estimated prediction error is the proportion of misclassified observations. Furthermore, the case when $K = p$ is known as *leave-one-out cross-validation* because, for each fold, all observations but one are used to build the classifier, with the remaining observation used as the testing set. It follows that p-fold cross-validation has the highest computational cost when compared to any other chosen K. Since the p training sets are very similar to one another, the cross-validation estimator has high variance, though it is nearly unbiased ([31], [49, p. 242]).

The recommended K for K-fold cross validation is $K = 5$ or 10 [13, 54]. With a smaller number of folds, such as $K = 5$, the estimator has lower variance but perhaps higher bias than in the case of more folds. The bias depends on how the learning method performance is affected by the size of the training set.

1.4.2 Bootstrapping

Another method by which we can estimate prediction error is known as bootstrapping. The bootstrap procedure takes samples of size p with replacement from the original training set. This is repeated B times. Nowadays with larger computational power available, B can be taken to be in the thousands or tens of thousands.

For each bootstrap sample b, we can build a classifier and compare it to the known input

labels for all of the training data. That is, we calculate

$$\widehat{Err}_{\text{boot}} = \frac{1}{p \cdot B} \sum_{b=1}^{B} \sum_{i=1}^{p} L(g(\boldsymbol{x}_i), \widehat{\mathcal{G}}_b(\boldsymbol{x}_i)), \tag{1.14}$$

where $\widehat{\mathcal{G}}_b(\boldsymbol{x}_i)$ is the predicted group for observation $\boldsymbol{x}_i$ for the classifier built from the b^{th} bootstrap sample. Unfortunately, the estimated error in (1.14) will typically be too low due to the fact that the data in the bootstrap samples also occur in the training sample [49]. To circumvent this issue, recall that K-fold cross-validation creates the training and test sets for each fold with partitioned data.

The *leave-one-out bootstrap* estimate of prediction error only considers calculating the loss function, in each bootstrap sample, for predictions of training observations not selected in the bootstrap sample. Letting C^{-i} be the set of indices of bootstrap samples b such that observation $\boldsymbol{x}_i$ is not in bootstrap sample b, then the estimate of prediction error for the leave-one-out bootstrap is

$$\widehat{Err}_{(1)} = \frac{1}{p} \sum_{i=1}^{p} \frac{1}{|C^{-i}|} \sum_{b \in C^{-i}} L(g(\boldsymbol{x}_i), \widehat{\mathcal{G}}_b(\boldsymbol{x}_i)). \tag{1.15}$$

To account for the fact that $|C^{-i}|$ could be zero, [49, p. 251] suggests either choosing B large enough to ensure this does not occur or ignoring the zero terms in (1.15). Furthermore, the estimate in (1.15) resolves the overfitting issue that occurs in (1.14) but has bias due to training set size, just as with K-fold cross-validation for smaller K.

The complex *.632 estimator* overcomes the training-set-size bias of the leave-one-out bootstrap estimate in (1.15) and is defined as

$$\widehat{Err}_{.632} = 0.632 \, \widehat{Err}_{(1)} + 0.368 \, \overline{err},$$

where

$$\overline{err} = \frac{1}{p}\sum_{i=1}^{p} L(g(\boldsymbol{x}_i), \widehat{\mathcal{G}}(\boldsymbol{x}_i))$$

is the training error. This weighted average estimator aims to alleviate the upward bias in the $\widehat{Err}_{(1)}$ estimator by averaging it with the downwardly biased estimator $\overline{err}$ [32]. While any weight could be used, the value .632 comes from noting that any observation $\boldsymbol{x}_i$ has probability near 0.632 of being selected in any given bootstrap sample [31]:

$$\begin{aligned} p(\boldsymbol{x}_i \in b) &= 1 - \left(1 - \frac{1}{p}\right)^p \\ &\approx 1 - e^{-1} \\ &\approx 0.632. \end{aligned}$$

Thus the probability of any given observation not being chosen in a sample (with replacement) of size p is 0.368.

In [31], Efron compares leave-one-out cross-validation to the .632 bootstrap estimator and found that, particularly for small samples, the .632 bootstrap estimator performed best; that is, it was closer to the true prediction error. Efron and Tibshirani describe a variation to the .632 estimator, designed to be a less biased compromise between $\widehat{Err}_{(1)}$ and $\overline{err}$ by putting more weight on $\widehat{Err}_{(1)}$ for the case of extensive overfitting, i.e., when $\widehat{Err}_{(1)} - \overline{err}$ is large [32]. In addition, [49] suggests that both bootstrap and cross-validation perform fairly equivalently in their accuracy of estimating prediction error, except in fitting methods such as trees where the estimators can underestimate true error by 10%.

1.5 Support Vector Classifiers

The support vector machine (SVM) is a well-known method for supervised classification and is well documented throughout the literature; see, e.g., [15, 22, 36, 49, 94, 95]. It requires the

solution of a constrained quadratic minimization problem, whose solution is our focus in this chapter. Specifically, for computing the SVM classifier with bound and linear equality constraints, we introduce an augmented Lagrangian technique similar to that of [65, 70]. Furthermore, we adapt the scaled gradient projection algorithm of [10] for use on the constrained SVM quadratic program.

We now give a brief introduction to support vector classifiers. The idea behind the support vector classifier (SVC) is to seek a separating hyperplane of the training data in feature space and then apply that hyperplane to the test set, giving a classification of the data set.

For simplicity and visualization purposes, we begin with a data set whose response variable consists of two classes. Later we discuss the extension of SVC to multi-class classification. Following the work of [15, 22, 49], let the training data be the set $\{\boldsymbol{x}_i, y_i\},\ i = 1, \ldots, p$, where $\boldsymbol{x}_i \in \mathbb{R}^n$ is the set of n measurements for observation i and $y_i \in \{-1, +1\}$ represents its inclusion into one of the two classes. Let a linear classifier for this training data be defined by a vector $\boldsymbol{w} \in \mathbb{R}^n$, a scalar b, and the decision rule

$$G(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}'\boldsymbol{x} + b), \tag{1.16}$$

which places $\boldsymbol{x}$ into the -1 class if $G(\boldsymbol{x})$ is negative and into the $+1$ class if it is positive.

Assuming completely separable data, there exist $\boldsymbol{w}$ and b such that the hyperplane $\boldsymbol{w}'\boldsymbol{x}+b = 0$ separates the training data. Figure 1.1 shows an example of the feature space for a two class data set with $n = 2$ and a separating hyperplane. Let d^+ (d^-) be the shortest distance from the separating hyperplane to the closest positive (negative) y_i. The *margin* is defined as $d^+ + d^-$.

Maximizing the margin is an intuitive approach that simultaneously distances all of the training data as far from the hyperplane as possible, thereby minimizing the likelihood of a misclassification. See Figure 1.2 for a visual demonstration of the optimality of the maximum

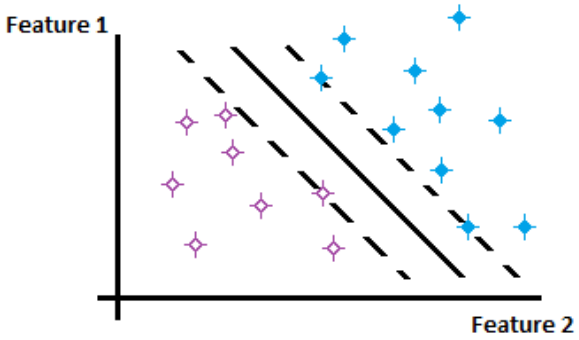

Figure 1.1: An example of a two class data set in $\mathbb{R}^2$ with a separating hyperplane (solid line) in its feature space. The shortest distance between the dotted lines is the margin, and training data that fall on the margin are called *support vectors*.

margin hyperplane in terms of misclassification error.

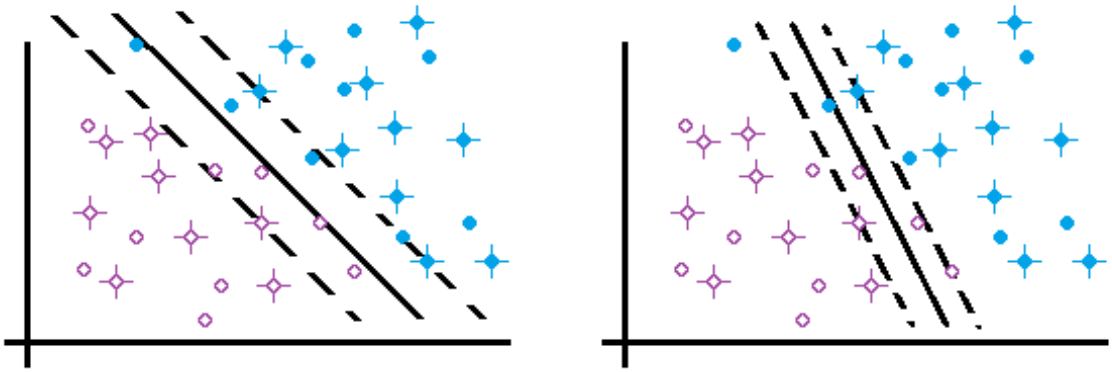

Figure 1.2: An example of a two class data set in $\mathbb{R}^2$ with and without maximizing the margin of a separating hyperplane. Training data appear as stars, test data appear as closed and open circles for the two classes. Left: The maximized margin separating hyperplane from Figure 1.1 applied to the test data. Note that one data point is misclassified. Right: A separating hyperplane in which the margin is not maximized. Note that four data points are misclassified.

Next, we show $\boldsymbol{w}$ is perpendicular to the hyperplane $\boldsymbol{w}'\boldsymbol{x} + b = 0$, and then use this fact to compute the margin, i.e., the distance between the two hyperplanes

$$\boldsymbol{w}'\boldsymbol{x} + b = -1 \quad \text{and} \quad \boldsymbol{w}'\boldsymbol{x} + b = 1. \tag{1.17}$$

For the hyperplane surface defined by $f(\boldsymbol{x}) = 0$, the vector $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ is a normal vector. Then for the hyperplane $f(\boldsymbol{x}) = \boldsymbol{w}'\boldsymbol{x} + b = 0$, $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{w}$ is a normal vector.

Now, two vectors $\boldsymbol{x}^- = c^-\boldsymbol{w}$ and $\boldsymbol{x}^+ = c^+\boldsymbol{w}$ lie on the two planes, respectively, in (1.17). Plugging into (1.17) and solving for c^- and c^+ we obtain

$$c^- = \frac{-1-b}{||\boldsymbol{w}||^2} \text{ and } c^+ = \frac{1-b}{||\boldsymbol{w}||^2},$$

from which the margin is calculated as

$$\begin{aligned} ||\boldsymbol{x}^+ - \boldsymbol{x}^-|| &= ||(c^+ - c^-)\boldsymbol{w}|| \\ &= \frac{2}{||\boldsymbol{w}||}. \end{aligned}$$

Thus maximizing the margin corresponds to minimizing $||\boldsymbol{w}||$, or in what follows, $||\boldsymbol{w}||^2$.

1.5.1 Support Vector Machine Variations

Many variations on the SVC formulation exist [22, 36], and each formulation has its own merits. Three derivations of support vector classifiers based on different assumptions are presented here. These derivations lead to two types of quadratic optimization problems: those with bound and equality constraints and those simply with bound constraints. The methods we developed are used to solve the bound and equality constrained quadratic program; however, we include the SVC formulations for the bound constrained quadratic programs for completeness. Following the derivations, we discuss the enlargement of the feature space using kernels, which allows for the nonlinear separating hyperplane classification within the support vector machine.

Derivation of a Bound-Plus-Equality Constrained SVC

We assume a non-separable data set such that a hyperplane cannot partition the training data in feature space into its classes without misclassification. For non-separable data, the SVC seeks the hyperplane of maximal margin but introduces a nonnegative slack variable ξ_i for each training point $\boldsymbol{x}_i$ to measure the degree of the misclassification of $\boldsymbol{x}_i$. The slack variable has value 0 if the training point is on the correct side of the margin, but when the training point is on the wrong side of the margin, the perpendicular (shortest) distance to its margin is given by $\xi_i/||\boldsymbol{w}||$. Figure 1.3 gives an example of a non-separable training set, where the red lines indicate positive slack variables for those training points on the wrong side of their margin.

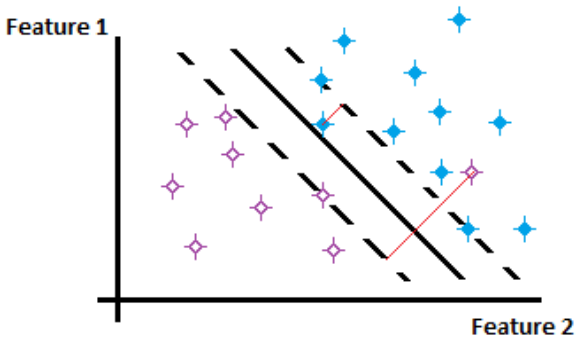

Figure 1.3: An example of a non-separable training set. For those training data on the wrong side of the margin, the red line indicates its perpendicular distance to the margin and has a length of $\xi_i/||\boldsymbol{w}||$. For training points on the correct side of their margin, the corresponding slack variable has value 0.

Cortes and Vapnik [21] suggest a *soft margin* method in which the objective function penalizes positive values of ξ_i, resulting in a trade off between a large margin and small penalty errors in the optimization problem:

$$\min_{\boldsymbol{w},\, b} \ \tfrac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{p}\xi_i \quad \text{subject to} \quad \left\{ \begin{array}{c} y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - (1-\xi_i) \geq 0 \quad \forall\, i \\ \xi_i \geq 0 \quad \forall\, i \end{array} \right\}. \tag{1.18}$$

Definition 1.5.1. *For a general optimization problem of the form*

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \textit{subject to} \quad \left\{ \begin{array}{ll} c_i(\boldsymbol{x}) = 0, & i \in \mathcal{E} \\ d_i(\boldsymbol{x}) \leq 0, & i \in \mathcal{I} \end{array} \right\},$$

with solution $\boldsymbol{x}^*$*, where* $\mathcal{E}$ *and* $\mathcal{I}$ *are the equality and inequality constraint index sets, respectively, the* primal objective function *is given by*

$$L_p(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i \in \mathcal{E}} \lambda_i c_i(\boldsymbol{x}) + \sum_{i \in \mathcal{I}} \mu_i d_i(\boldsymbol{x}),$$

where $\boldsymbol{\lambda}$*,* $\boldsymbol{\mu}$ *are the Lagrange multiplier vectors. This function is also referred to as the* Lagrangian *[71].*

Definition 1.5.2. *At the solution* $\boldsymbol{x}^*$*, there are Lagrange multiplier vectors* $\boldsymbol{\lambda}^*$ *and* $\boldsymbol{\mu}^*$ *such that the following* Karush-Kuhn-Tucker (KKT) conditions *are satisfied at* $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$*:*

$$\begin{array}{lrcll} \textit{Stationarity} & \nabla_{\boldsymbol{x}} L_p(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) & = & \mathbf{0}, & \\ \textit{Primal Feasibility} & c_i(\boldsymbol{x}^*) & = & 0, & \forall\, i \in \mathcal{E}, \\ \textit{Primal Feasibility} & d_i(\boldsymbol{x}^*) & \leq & 0, & \forall\, i \in \mathcal{I}, \\ \textit{Dual Feasibility} & \mu_i^* & \geq & 0, & \forall\, i \in \mathcal{I}, \\ \textit{Complementary Slackness} & \mu_i^* d_i(\boldsymbol{x}^*) & = & 0, & \forall\, i \in \mathcal{I}. \end{array}$$

The gradient of L_p at the solution $\boldsymbol{x}^*$ is $\mathbf{0}$, giving us the stationarity condition. In other words,

$$\nabla_{\boldsymbol{x}} L_p = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^*) + \sum_{i \in \mathcal{A}(\boldsymbol{x}*)} \mu_i^* \nabla_{\boldsymbol{x}} d_i(\boldsymbol{x}^*) = 0,$$

where $i \in \mathcal{A}(\boldsymbol{x}^*)$ when the constraint $d_i(\boldsymbol{x}) \leq 0$ is active at the solution $\boldsymbol{x}^*$. The primal feasibility KKT conditions are simply the constraints on the primal problem. The dual of the primal [40] has non-negativity constraints on the Lagrange multipliers, yielding the dual feasibility KKT condition. The complementary slackness KKT condition states that for all

$i \in \mathcal{I}$, either $\mu_i^* = 0$ or $d_i(\boldsymbol{x}^*) = 0$, indicating when the inequality constraint is active ($\mu_i^* \neq 0$) and when it is inactive ($\mu_i^* = 0$).

The primal objective function for (1.18) is given by

$$L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}||\boldsymbol{w}||^2 + \sum_{i=1}^{p}(C - \mu_i)\xi_i - \sum_{i=1}^{p}\alpha_i(y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - (1 - \xi_i)), \tag{1.19}$$

where α_i and μ_i, for $i = 1, \ldots, p$, are the Lagrange multipliers [49]. Solving the primal problem (1.19) is equivalent to solving (1.18) [49, Theorem 12.1].

The stationarity KKT condition then implies

$$\begin{aligned}
\mathbf{0} &= \nabla_{(\boldsymbol{w},b,\boldsymbol{\xi})} L_p \\
&= \begin{bmatrix} \nabla_{\boldsymbol{w}} L_p \\ \nabla_b L_p \\ \nabla_{\boldsymbol{\xi}} L_p \end{bmatrix} \\
&= \begin{bmatrix} \nabla_{\boldsymbol{w}} \left(\frac{1}{2}\boldsymbol{w}'\boldsymbol{w} - \sum_{i=1}^{p} \alpha_i y_i (\boldsymbol{w}'\boldsymbol{x}_i) + \text{ non-}\boldsymbol{w} \text{ terms} \right) \\ \nabla_b \left(- \sum_{i=1}^{p} \alpha_i y_i b + \text{ non-}b \text{ terms} \right) \\ \nabla_{\boldsymbol{\xi}} \left(\sum_{i=1}^{p} \alpha_i (1 - \xi_i) + C \sum_{i=1}^{p} \xi_i - \sum_{i=1}^{p} \mu_i \xi_i + \text{ non-}\boldsymbol{\xi} \text{ terms} \right) \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{w} - \sum_{i=1}^{p} \alpha_i y_i \boldsymbol{x}_i \\ - \sum_{i=1}^{p} \alpha_i y_i \\ -\boldsymbol{\alpha} + \boldsymbol{C} - \boldsymbol{\mu} \end{bmatrix},
\end{aligned}$$

where $\boldsymbol{C} = (C, \ldots, C)'$

The KKT conditions for (1.18) can therefore be stated as follows:

$$\text{Stationarity} \qquad \begin{bmatrix} \boldsymbol{w} \\ 0 \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{p} \alpha_i y_i \boldsymbol{x}_i \\ \sum_{i=1}^{p} \alpha_i y_i \\ \boldsymbol{C} - \boldsymbol{\mu} \end{bmatrix},$$

$$\begin{array}{lrcll}
\text{Primal Feasibility} & y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - (1 - \xi_i) & \geq & 0, & i = 1, \ldots, p, \\
\text{Primal Feasibility} & \xi_i & \geq & 0, & i = 1, \ldots, p, \\
\text{Dual Feasibility} & (\alpha_i,\ \mu_i) & \geq & 0, & i = 1, \ldots, p, \\
\text{Complementary Slackness} & \alpha_i(y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - (1 - \xi_i)) & = & 0, & i = 1, \ldots, p, \\
\text{Complementary Slackness} & \mu_i \xi_i & = & 0, & i = 1, \ldots, p.
\end{array}$$

The Wolfe dual [40], L_d, is derived from maximizing the primal objective function, L_p, with respect to the Lagrange multipliers subject to $\nabla_{(\boldsymbol{w},b,\xi_i)} L_p = \mathbf{0}$ (i.e., stationarity) and $\alpha_i,\ \mu_i \geq 0$. By substituting the stationarity conditions into the primal objective function (1.19) we obtain:

$$\begin{aligned}
L_d(\boldsymbol{\alpha}) &= L_p\left(\sum_{i=1}^{p} \alpha_i y_i \boldsymbol{x}_i, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}\right) \\
&= \frac{1}{2}\left(\sum_{i=1}^{p} \alpha_i y_i \boldsymbol{x}_i\right)'\left(\sum_{i=1}^{p} \alpha_i y_i \boldsymbol{x}_i\right) + \sum_{i=1}^{p} (C - \mu_i)\,\xi_i \\
&\quad - \sum_{i=1}^{p} \alpha_i \left[y_i \left(\left(\sum_{j=1}^{p} \alpha_j y_j \boldsymbol{x}_j \right)' \boldsymbol{x}_i + b \right) - (1 - \xi_i) \right] \\
&= \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j + \sum_{i=1}^{p} \underbrace{(C - \mu_i - \alpha_i)}_{=0} \xi_i \\
&\quad - \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j - b \underbrace{\sum_{i=1}^{p} \alpha_i y_i}_{=0} + \sum_{i=1}^{p} \alpha_i \\
&= -\frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j - b \underbrace{\sum_{i=1}^{p} \alpha_i y_i}_{=0} + \sum_{i=1}^{p} \alpha_i
\end{aligned}$$

$$= \sum_{i=1}^{p} \alpha_i - \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j.$$

Notice that the Lagrange multipliers μ_i, slack variables ξ_i, hyperplane vector $\boldsymbol{w}$, and intercept b do not appear in this final formulation. The dual optimization problem is then written as

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j - \sum_{i=1}^{p} \alpha_i \right\} \text{ subject to } \left\{ \begin{array}{c} 0 \leq \alpha_i \leq C \ \ \forall \, i \\ \sum_{i=1}^{p} \alpha_i y_i = 0 \end{array} \right\}, \tag{1.20}$$

where the constraint $0 \leq \alpha_i \leq C$ comes from the fact that $\alpha_i = C - \mu_i$ and $\mu_i \geq 0$. We will focus in this manuscript on solving (1.20). However, we note that it is also possible to solve the primal problem [11, 18, 41, 51, 87].

Solving (1.20) for the Lagrange multipliers is equivalent to solving (1.19) for $\boldsymbol{w}$, b, and $\boldsymbol{\xi}$ [71, Theorems 12.5, 12.6], as well as (1.18). Once (1.20) is solved for Lagrange multiplier vector $\boldsymbol{\alpha}$, it remains to find $\boldsymbol{w}$ and b that define the hyperplane and decision rule. These are obtained from the KKT conditions. First we note that for each $\alpha_i \neq 0$, we have that $\boldsymbol{x}_i$ is a *support vector*, indicating that it lies on the margin (the dotted lines in Figure 1.1), in which case $\xi_i = 0$. Letting $\mathcal{S} = \{i | \alpha_i^* \neq 0\}$, then by the stationarity KKT condition we have

$$\boldsymbol{w}^* = \sum_{i \in \mathcal{S}} \alpha_i^* y_i \boldsymbol{x}_i. \tag{1.21}$$

Using the KKT complementary slackness condition $\alpha_i(y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - (1 - \xi_i)) = 0$, we solve for b to obtain

$$b = \frac{1 - y_i \boldsymbol{w}' \boldsymbol{x}_i}{y_i},$$

for all $i \in \mathcal{S}$. We take b^* to be the average of those hyperplane intercepts suggested by the

support vectors:

$$b^* = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \frac{1 - y_i(\boldsymbol{w}^*)'\boldsymbol{x}_i}{y_i}.$$

Using the estimated hyperplane, $(\boldsymbol{w}^*)'\boldsymbol{x} + b^*$, the decision rule for classification is given by (1.16).

Generalizing to more classes is a simple extension of the two class problem. Classification with k classes requires an associated hyperplane $(\boldsymbol{w}_j^*, b_j^*)$ for each class, for $j = 1, \ldots, k$. To build the j^{th} hyperplane, if $\boldsymbol{x}_i \in g_j$, then we assign $y_i = 1$, otherwise $y_i = -1$ for $\boldsymbol{x}_i \notin g_j$. That is, we build a hyperplane for each training class versus the rest of the training set. In a two class problem, it is unnecessary to build two hyperplanes since class 1 against class 2 would yield the same hyperplane as class 2 against class 1.

The decision rule is given by

$$\widehat{G}(\boldsymbol{x}) = \arg \max_{1 \leq j \leq k} ((\boldsymbol{w}_j^*)'\boldsymbol{x} + b_j^*).$$

Geometrically, this creates a hyperplane for each class and assigns new data observation $\boldsymbol{x}$ to the class for which $\boldsymbol{x}$ is furthest from the associated hyperplane if the observation is on the "correct" side of the hyperplane for that class, i.e., $\boldsymbol{x}$ is on the "correct" side of the hyperplane for class g_j if $(\boldsymbol{w}_j^*)'\boldsymbol{x} + b_j$ is positive. Otherwise, if the observation is on the "incorrect" side of the hyperplane for all hyperplanes, $\boldsymbol{x}$ is placed in the class for which it is closest to the associated hyperplane.

Derivation of a Bound Constrained SVC Without Intercept

Consider again the minimization problem (1.18) but with the assumption that the hyperplane does not have an intercept, i.e., $b = 0$. Then our primal objective function is

$$L_p(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{p} \alpha_i (y_i \boldsymbol{w}' \boldsymbol{x}_i - (1 - \xi_i)) + C \sum_{i=1}^{p} \xi_i - \sum_{i=1}^{p} \mu_i \xi_i.$$

Since b does not appear in the objective function anymore, we no longer have the constraint $\sum_{i=1}^{p} \alpha_i y_i = 0$. Thus the dual optimization problem becomes

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j - \sum_{i=1}^{p} \alpha_i \right\} \quad \text{subject to } \ 0 \leq \alpha_i \leq C, \quad \forall\, i. \tag{1.22}$$

Notice that (1.22) has the same objective as (1.20), but that the equality constraint has been removed. This simplification of the constraint set is a motivation for the assumption that the intercept is zero. Once again, after (1.22) is solved for Lagrange multipliers $\boldsymbol{\alpha}^*$, the associated $\boldsymbol{w}^*$ is defined by (1.21) and the decision rule by (1.16) with $b = 0$. The result of assuming $b = 0$ forces the hyperplane to pass through the origin and reduces the degrees of freedom by one.

Derivation of a Bound Constrained SVC with Penalized Intercept

The third SVC formulation considered here is one that suggests maximizing the margin with a penalty placed on large values of b if it is suspected that b should remain small. We now consider the problem

$$\min_{\boldsymbol{w}, b} \frac{1}{2} \left|\left|[\boldsymbol{w}', \ \delta b]\right|\right|^2 + C \sum_{i=1}^{p} \quad \text{subject to } \left\{ \begin{array}{c} y_i(\boldsymbol{w}' \boldsymbol{x}_i + b) - (1 - \xi_i) \geq 0 \\ \xi_i \geq 0 \end{array} \right\}, \tag{1.23}$$

where δ is a penalty parameter specified by the user. The primal objective function is given by

$$L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \frac{1}{2}|||[\boldsymbol{w}',\ \delta b]||^2 - \sum_{i=1}^{p} \alpha_i (y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - (1 - \xi_i)) + C\sum_{i=1}^{p} \xi_i - \sum_{i=1}^{p} \mu_i \xi_i.$$

The KKT conditions derived are similar to the bound-plus-equality constrained SVC above and are given by

$$\begin{array}{lrcll}
\text{Stationarity} & \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix} & = & \sum\limits_{i \in S} \alpha_i y_i \begin{bmatrix} \boldsymbol{x}_i \\ \frac{1}{\delta^2} \end{bmatrix}, & \\
\text{Stationarity} & \boldsymbol{\alpha} & = & \boldsymbol{C} - \boldsymbol{\mu}, & \\
\text{Primal Feasibility} & y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - (1 - \xi_i) & \geq & 0, & i = 1, \ldots, p, \\
\text{Primal Feasibility} & \xi_i & \geq & 0, & i = 1, \ldots, p, \\
\text{Dual Feasibility} & (\alpha_i,\ \mu_i) & \geq & 0, & i = 1, \ldots, p, \\
\text{Complementary Slackness} & \alpha_i(y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - (1 - \xi_i)) & = & 0, & i = 1, \ldots, p, \\
\text{Complementary Slackness} & \mu_i \xi_i & = & 0, & i = 1, \ldots, p.
\end{array}$$

The corresponding Wolfe dual [40] is given by

$$\begin{aligned}
L_d(\boldsymbol{\alpha}) &= L_p\left(\sum_{i=1}^{p} \alpha_i y_i \boldsymbol{x}_i, \sum_{i=1}^{p} \alpha_i y_i \tfrac{1}{\delta\delta^2}, \boldsymbol{\xi}\right) \\
&= \frac{1}{2}\left(\sum_{i=1}^{p} \alpha_i y_i \boldsymbol{x}_i\right)'\left(\sum_{i=1}^{p} \alpha_i y_i \boldsymbol{x}_i\right) + \frac{1}{2}\delta^2\left(\frac{1}{\delta^2}\sum_{i=1}^{p} \alpha_i y_i\right)\left(\frac{1}{\delta^2}\sum_{i=1}^{p} \alpha_i y_i\right) \\
&\quad - \sum_{i=1}^{p} \alpha_i \left[y_i\left(\left(\sum_{j=1}^{p} \alpha_j y_j \boldsymbol{x}_j\right)' \boldsymbol{x}_i + \left(\frac{1}{\delta^2}\sum_{j=1}^{p} \alpha_j y_j\right)\right) - (1 - \xi_i)\right] + \sum_{i=1}^{p} (C - \mu_i)\,\xi_i \\
&= \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i'\boldsymbol{x}_j + \frac{1}{2\delta^2}\sum_{i=1}^{p}\sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j - \sum_{i=1}^{p}\sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i'\boldsymbol{x}_j \\
&\quad - \frac{1}{\delta^2}\sum_{i=1}^{p}\sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j + \sum_{i=1}^{p} \alpha_i + \sum_{i=1}^{p} \underbrace{(C - \mu_i - \alpha_i)}_{=0} \xi_i
\end{aligned}$$

$$
\begin{aligned}
&= -\frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j y_i y_j \begin{bmatrix} \boldsymbol{x}_i \\ \frac{1}{\delta} \end{bmatrix}' \begin{bmatrix} \boldsymbol{x}_j \\ \frac{1}{\delta} \end{bmatrix} + \sum_{i=1}^{p}\alpha_i \\
&= \sum_{i=1}^{p}\alpha_i - \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j y_i y_j \begin{bmatrix} \boldsymbol{x}_i \\ \frac{1}{\delta} \end{bmatrix}' \begin{bmatrix} \boldsymbol{x}_j \\ \frac{1}{\delta} \end{bmatrix}.
\end{aligned}
$$

The Wolfe dual formulation of (1.23) then simplifies to

$$
\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j y_i y_j \begin{bmatrix} \boldsymbol{x}_i' & \frac{1}{\delta} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_j \\ \frac{1}{\delta} \end{bmatrix} - \sum_{i=1}^{p}\alpha_i \right\} \quad \text{subject to } 0 \le \alpha_i \le C \quad \forall\, i. \tag{1.24}
$$

Notice that this formulation again removes the equality constraint that appeared in (1.20), reducing the dual to a bound constrained quadratic program. Note also that as $\delta \to \infty$, the quadratic program converges to (1.22).

Solving the dual yields the Lagrange multipliers α_i^* from which we would use the nonzero α_i^* to compute $\boldsymbol{w}^*$ and b^* using the KKT stationarity equation

$$
\begin{bmatrix} \boldsymbol{w}^* \\ b^* \end{bmatrix} = \sum_{i\in\mathcal{S}}\alpha_i^* y_i \begin{bmatrix} \boldsymbol{x}_i \\ \frac{1}{\delta^2} \end{bmatrix},
$$

where we recall that $\{\boldsymbol{x}_i | i \in S\}$ are the support vectors. The decision rule is then given by (1.16) with $\boldsymbol{w} = \boldsymbol{w}^*$ and $b = b^*$.

1.5.2 Expanding the Feature Space with Kernels

For the SVC, the decision function is a linear function of $\boldsymbol{x}$, creating linear decision boundaries in the input feature space. This is not always optimal and by enlarging the feature space using basis expansions, the classifier has better training class separation in the original feature space,

i.e., the boundaries become nonlinear in feature space.

Nonlinear boundaries arise when the Euclidean inner-product $\boldsymbol{x}_i'\boldsymbol{x}_j$, appearing in the above dual objective functions, is replaced by the evaluation of a kernel function, $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, yielding the support vector machine (SVM). An advantage of using kernels when the data is not linearly separable is that the nonlinear decision boundaries are computed cheaply [49].

The underlying idea is illustrated as follows. First, we transform the training data using m basis functions: $\boldsymbol{h}(\boldsymbol{x}_i) = (h_1(\boldsymbol{x}_i), \ldots, h_m(\boldsymbol{x}_i))$, for $i = 1, \ldots, p$. We then apply the SVC to the transformed data, yielding the dual function

$$L_d(\boldsymbol{\alpha}) = \sum_{i=1}^{p} \alpha_i - \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{h}(\boldsymbol{x}_i), \boldsymbol{h}(\boldsymbol{x}_j) \rangle ,$$

which can be used in the Wolfe-dual optimization problems (1.20) and (1.22), and the dual function for (1.24) is given by

$$L_d(\boldsymbol{\alpha}) = \sum_{i=1}^{p} \alpha_i - \frac{1}{2} \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j y_i y_j \left(\langle \boldsymbol{h}(\boldsymbol{x}_i), \boldsymbol{h}(\boldsymbol{x}_j) \rangle + \frac{1}{\delta^2} \right),$$

where the inner product is defined as

$$\langle \boldsymbol{h}(\boldsymbol{x}_i), \boldsymbol{h}(\boldsymbol{x}_j) \rangle = \boldsymbol{h}(\boldsymbol{x}_i)' \boldsymbol{h}(\boldsymbol{x}_j) = \sum_{k=1}^{m} h_k(\boldsymbol{x}_i) h_k(\boldsymbol{x}_j).$$

Then the corresponding hyperplane solution has the form

$$G(\boldsymbol{x}) = (\boldsymbol{w}^*)' \boldsymbol{h}(\boldsymbol{x}) + b^* = \sum_{i=1}^{p} \alpha_i y_i \langle \boldsymbol{h}(\boldsymbol{x}_i), \boldsymbol{h}(\boldsymbol{x}) \rangle + b^*.$$

Notice that in all of the above equations, the basis functions appear only within an inner

product, and hence it suffices to know the kernel function defined by

$$K(\boldsymbol{u}, \boldsymbol{v}) = \langle \boldsymbol{h}(\boldsymbol{u}), \boldsymbol{h}(\boldsymbol{v}) \rangle \,,$$

which computes inner products in the transformed space. The following three kernels were the first ones used in pattern recognition and are still popular:

$$\begin{aligned} d^{th} \text{ Degree polynomial:} \quad & K(\boldsymbol{u}, \boldsymbol{v}) = (1 + \langle \boldsymbol{u}, \boldsymbol{v} \rangle)^d, \\ \text{Radial basis:} \quad & K(\boldsymbol{u}, \boldsymbol{v}) = e^{-\sigma ||\boldsymbol{u} - \boldsymbol{v}||^2}, \\ \text{Neural Network:} \quad & K(\boldsymbol{u}, \boldsymbol{v}) = \tanh(\kappa_1 \langle \boldsymbol{u}, \boldsymbol{v} \rangle + \kappa_2). \end{aligned}$$

In enlarged feature space, the cost parameter C (the upper-bound on the inequality constraint in (1.20), (1.22) and (1.24)), is more understandable in the role it plays. With complete separation often plausible in the enlarged feature space, a larger value of C corresponds to a greater number of support vectors (fewer positive slack variables ξ_i) and thus a wiggly, possibly overfit boundary in the original feature space [49]. A smaller value of C corresponds to fewer support vectors and a smoother boundary, allowing more positive values of ξ_i.

Figure 1.4 visually demonstrates the improvement in classification error for non-separable data using the radial basis kernel versus no kernel. In the left image, no kernel was used, resulting in linear boundaries for the classification. In this case, we see there are four misclassifications: two triangles are misclassified as circles, a circle is misclassified as a square, and a square is misclassified as a triangle. In the right image where a radial basis kernel was used to create nonlinear boundaries for classification, none of the training data were misclassified.

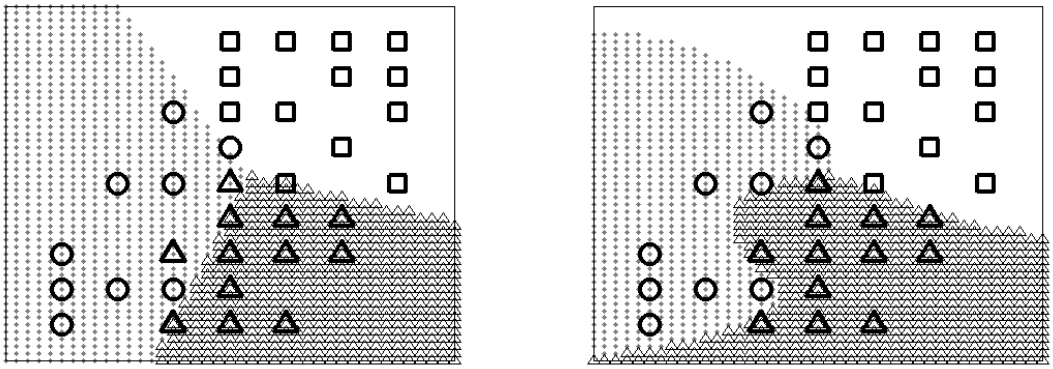

Figure 1.4: Training data for three classes are indicated by the large circles, triangles, and squares. Classification for $\mathbb{R}^2$ is given, where the smaller dots correspond to the large circle class, the smaller triangles correspond to the large triangle class, and the empty space corresponds to the square class. The left image gives the classification for the training set using linear boundaries (no kernel) while the right image gives the classification using a radial basis kernel with $\sigma = 0.2$. Both classifications use $C = 10^5$. Notice the linear decision boundaries misclassify four of the training data points due to non-separability while the nonlinear decision boundaries correctly classify the training data.

1.6 Constrained Quadratic Programming Methods

We have presented two basic forms of optimization problems for solving SVMs and they are rewritten here in matrix notation for implementation in optimization algorithms. The bound and equality constrained quadratic program (1.20) can be written in the form

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \ \left\{ f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}'\mathbf{A}\boldsymbol{\alpha} - \boldsymbol{\alpha}'\boldsymbol{b} \right\} \\ &\text{subject to } \ \boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u} \ \text{ and } \ \boldsymbol{y}'\boldsymbol{\alpha} = 0, \end{aligned} \tag{1.25}$$

where the inequality is component-wise, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_p)'$, $\boldsymbol{b} = (1, \ldots, 1)'$, $\boldsymbol{l} = (0, \ldots, 0)'$, $\boldsymbol{u} = (C, \ldots, C)'$, $\boldsymbol{y}' = (y_1, \ldots, y_p)'$, and

$$\mathbf{A} = \begin{bmatrix} y_1 \boldsymbol{x}_1' \\ \vdots \\ y_p \boldsymbol{x}_p' \end{bmatrix} \begin{bmatrix} y_1 \boldsymbol{x}_1 & \ldots & y_p \boldsymbol{x}_p \end{bmatrix}. \tag{1.26}$$

The bound constrained quadratic programs (1.22) and (1.24) can be written in the form

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \ \left\{ f(\boldsymbol{\alpha}) = \frac{1}{2} \boldsymbol{\alpha}' \mathbf{A} \boldsymbol{\alpha} - \boldsymbol{\alpha}' \boldsymbol{b} \right\} \\ &\text{subject to } \ \boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}, \end{aligned} \tag{1.27}$$

where all vectors and matrices are given as above with $\mathbf{A}$ as in (1.26) for (1.22) and

$$\mathbf{A} = \begin{bmatrix} y_1 \boldsymbol{x}_1' & \frac{1}{\delta} y_1 \\ \vdots & \vdots \\ y_p \boldsymbol{x}_p' & \frac{1}{\delta} y_p \end{bmatrix} \begin{bmatrix} y_1 \boldsymbol{x}_1 & \ldots & y_p \boldsymbol{x}_p \\ \frac{1}{\delta} y_1 & \ldots & \frac{1}{\delta} y_p \end{bmatrix}$$

for (1.24).

We note that matrix $\mathbf{A}$ is symmetric positive definite if the training data $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p)$ are linearly independent, otherwise it is symmetric semipositive definite.

While there are many approaches for solving the SVM problem as mentioned above, here we focus solely on the dual formulations provided in Section 1.5.1. Methods such as cutting planes and the stochastic sub-gradient take advantage of the primal formulation [11, 18, 41, 51, 87] but are not discussed here. Interior point methods which exploit the low-rank structure of the problem are another common approach but are not discussed here [29, 37]. While we incorporate the use of projected Newton and conjugate gradient algorithms into the augmented Lagrangian algorithms, various other active set methods could also be used [69, 83, 88].

The focus of this section is on numerical algorithms for the bound and equality constrained SVM quadratic program (QP) (1.25). Methods for solving (1.27) are well-documented and well-studied. In particular, gradient projection (GP) [53], projected Newton (PN) [53] and gradient projection conjugate gradient (GPCG) [69] are methods that may be used to solve (1.27). The remainder of this section focuses on developing numerical methods for solving the bound-plus-equality constrained QP.

1.6.1 The Augmented Lagrangian

The augmented Lagrangian (AL) method is also known as the method of multipliers and solves the constrained optimization problem

$$\min_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}) \quad \text{subject to} \quad \left\{ \begin{array}{ll} h_i(\boldsymbol{\alpha}) = 0 & \forall\, i \in \mathcal{E} \\ g_i(\boldsymbol{\alpha}) \leq 0 & \forall\, i \in \mathcal{I} \end{array} \right\}, \tag{1.28}$$

where $\mathcal{E}$ and $\mathcal{I}$ are the equality and inequality constraint index sets, respectively. The general idea behind the AL method is to incorporate some or all of the constraints into the objective function $f(\boldsymbol{\alpha})$ [9, 71]. We present AL formulations for the SVM quadratic programming problem (1.25) here.

In [65, 70], the equality constraint is incorporated into the Lagrangian function, leaving only the bound constraints. In [70], gradient projection conjugate gradient (GPCG) [69] is used to solve the resulting bound constrained quadratic program, whereas in [65] an approximate problem is solved analytically. Here we follow the approach of [70] but use projected Newton [53] in place of GPCG as it reduces computational time and is easier to implement. We also take the AL algorithm one step further and incorporate both the bound and equality constraints into the Lagrangian, allowing us to construct an iterative algorithm requiring the solution of a sequence of unconstrained quadratic programs, which can be solved with less sophisticated methods, and whose solutions converge to the solution of (1.25).

Before continuing, we note that the LANCELOT algorithm is a well-known method for solving general problems of the form (1.28) using the AL technique [20]. Due to the size of the class of problems in (1.28), the algorithm is necessarily complex. Since our focus is on the much smaller subclass of problems of the form (1.25), we are able to present two AL algorithms that are much simpler than LANCELOT and that exploit the specific structure of the optimization problem.

The Bound Constrained Augmented Lagrangian

In the bound constrained AL technique, we add a quadratic penalty to the Lagrangian function. For (1.25), the AL function is given by

$$\mathcal{L}(\boldsymbol{\alpha}, \lambda, \mu) = f(\boldsymbol{\alpha}) + \lambda \boldsymbol{y}'\boldsymbol{\alpha} + \frac{\mu}{2}(\boldsymbol{y}'\boldsymbol{\alpha})^2, \tag{1.29}$$

where λ is a Lagrange multiplier and $\mu > 0$ is a penalty parameter.

Note that as μ increases, failure to satisfy the equality constraint $\boldsymbol{y}'\boldsymbol{\alpha} = 0$ is increasingly penalized, forcing the optimization towards the feasible region. One might think that adding the quadratic penalty term alone to $f(\boldsymbol{\alpha})$ would be enough, however the resulting problem is ill-conditioned in the sense that as $\mu \to \infty$, a systematic perturbation to the minimizers results [71]. Adding only the Lagrange multiplier term and not the quadratic penalty term would be using the Lagrangian method again, which was not explored here.

To create the bound constrained AL algorithm, at iteration k, we fix $\lambda = \lambda_k$ and $\mu = \mu_k$, and define $\boldsymbol{\alpha}^k$ to be the minimizer of $\mathcal{L}(\boldsymbol{\alpha}, \lambda_k, \mu_k)$ with respect to $\boldsymbol{\alpha}$ and subject to $\boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}$.

The KKT conditions state that the minimizer $(\boldsymbol{\alpha}^*, \lambda^*, \mu^*)$ of $\mathcal{L}(\boldsymbol{\alpha}, \lambda, \mu)$ satisfies

$$\begin{aligned} \mathbf{0} &= \nabla_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha}^*, \lambda^*, \mu^*) \\ &= \nabla_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}^*) + \lambda^* \boldsymbol{y} + \mu^* (\underbrace{\boldsymbol{y}'\boldsymbol{\alpha}^*}_{=0}) \boldsymbol{y} \\ &= \nabla_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}^*) + \lambda^* \boldsymbol{y}. \end{aligned} \tag{1.30}$$

Moreover, we have that at each iteration k,

$$\mathbf{0} = \nabla_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha}^k, \lambda_k, \mu_k) = \nabla_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}^k) + (\lambda_k + \mu_k(\boldsymbol{y}'\boldsymbol{\alpha}^k))\boldsymbol{y}. \tag{1.31}$$

Equating (1.30) and (1.31) motivates the update

$$\lambda_{k+1} = \lambda_k + \mu_k \boldsymbol{y}'\boldsymbol{\alpha}^k. \tag{1.32}$$

Bertsekas discusses selecting a penalty parameter in [9], noting that μ_k needs to increase fast enough to have a decent convergence rate but slow enough to reduce ill-conditioning. Various updates for penalty parameter μ_k are given in [9], one of which we use here: let $\beta > 1$ and $0 < \nu < 1$, then

$$\mu_{k+1} = \begin{cases} \beta\mu_k & \text{if } ||\boldsymbol{y}'\boldsymbol{\alpha}^k|| > \nu ||\boldsymbol{y}'\boldsymbol{\alpha}^{k-1}||, \\ \mu_k & \text{if } ||\boldsymbol{y}'\boldsymbol{\alpha}^k|| \leq \nu ||\boldsymbol{y}'\boldsymbol{\alpha}^{k-1}||, \end{cases} \tag{1.33}$$

with a recommended choice of $\nu = 0.25$, and we take $\beta = 3$ [9]. This increases the penalty parameter by a factor of β only if the constraint violation has not decreased by a factor of ν from the previous minimization.

We can now present the first AL method, which solves (1.25) by cyclically minimizing $\mathcal{L}(\boldsymbol{\alpha}, \lambda_k, \mu_k)$ with respect to $\boldsymbol{\alpha}$ subject to $\boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}$ using projected Newton [53], or one of many other bound constrained optimizers [69–71], and by updating λ_k and μ_k via (1.32) and (1.33), respectively. Pseudocode for projected Newton is given in Section 1.6.3. Pseudocode for this augmented Lagrangian approach is presented in Algorithm 1.

Algorithm 1 The Bound Constrained Augmented Lagrangian Method

Given $\mu_0 \geq 0$ and initial points $\boldsymbol{\alpha}^0$ and λ_0, set $k = 0$.

1. Use projected Newton to find the approximate minimizer $\boldsymbol{\alpha}^{k+1}$ of $\mathcal{L}(\boldsymbol{\alpha}, \lambda_k, \mu_k)$ in (1.29) subject to $\boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}$ with starting point $\boldsymbol{\alpha}^k$.
2. If $|\boldsymbol{y}'\boldsymbol{\alpha}^{k+1}| < 10^{-6}$, stop with approximate solution $\boldsymbol{\alpha}^{k+1}$.
3. Update the Lagrange multiplier via (1.32) to obtain λ_{k+1}, and choose a new penalty parameter μ_{k+1} via (1.33).
4. Set $k = k + 1$ and return to Step 1.

Bertsekas gives a basic convergence result in [9], restated here for completeness.

Theorem 1. *Assume that f and h are continuous functions and that the constraint set $\{\boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u} | h(\boldsymbol{\alpha}) = 0\}$ is nonempty. For $k = 0, 1, \ldots,$ let $\boldsymbol{\alpha}^k$ be a global minimum of the problem*

$$\begin{aligned} &\textit{minimize} && \mathcal{L}(\boldsymbol{\alpha}, \lambda_k, \mu_k) \\ &\textit{subject to} && \boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}, \end{aligned}$$

where $\{\lambda_k\}$ is bounded, $0 < \mu_k < \mu_{k+1}$ for all k, and $\mu_k \to \infty$. Then every limit point of the sequence $\{\boldsymbol{\alpha}^k\}$ is a global minimum of the original problem

$$\begin{aligned} &\textit{minimize} && f(\boldsymbol{\alpha}) \\ &\textit{subject to} && h(\boldsymbol{\alpha}) = 0, \;\; \boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}. \end{aligned} \tag{1.34}$$

Proof. Let $\bar{\boldsymbol{\alpha}}$ be a limit point of $\{\boldsymbol{\alpha}^k\}$. We have by definition of $\boldsymbol{\alpha}^k$

$$\mathcal{L}(\boldsymbol{\alpha}^k, \lambda_k, \mu_k) \leq \mathcal{L}(\boldsymbol{\alpha}, \lambda_k, \mu_k), \quad \forall \;\; \boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}. \tag{1.35}$$

Let f^* denote the the optimal value of the original problem (1.34). We have

$$\begin{aligned} f^* &= \inf_{h(\boldsymbol{\alpha})=0,\, \boldsymbol{l}\leq\boldsymbol{\alpha}\leq\boldsymbol{u}} f(\boldsymbol{\alpha}) \\ &= \inf_{h(\boldsymbol{\alpha})=0,\, \boldsymbol{l}\leq\boldsymbol{\alpha}\leq\boldsymbol{u}} \left\{f(\boldsymbol{\alpha}) + \lambda_k h(\boldsymbol{\alpha}) + \tfrac{\mu_k}{2}||h(\boldsymbol{\alpha})||^2\right\} \\ &= \inf_{h(\boldsymbol{\alpha})=0,\, \boldsymbol{l}\leq\boldsymbol{\alpha}\leq\boldsymbol{u}} \mathcal{L}(\boldsymbol{\alpha}, \lambda_k, \mu_k). \end{aligned}$$

Hence, by taking the infimum of the right-hand side of (1.35) over $\boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}$, $h(\boldsymbol{\alpha}) = 0$, we obtain

$$\mathcal{L}(\boldsymbol{\alpha}^k, \lambda_k, \mu_k) = f(\boldsymbol{\alpha}^k) + \lambda_k h(\boldsymbol{\alpha}^k) + \tfrac{\mu_k}{2}||h(\boldsymbol{\alpha}^k)||^2 \;\leq\; f^*.$$

The sequence $\{\lambda_k\}$ is bounded and hence it has a limit point $\bar{\lambda}$. Without loss of generality, we may assume that $\lambda_k \to \bar{\lambda}$. By taking the supremum in the relation above and by using the continuity of f and h, we obtain

$$f(\bar{\boldsymbol{\alpha}}) + \bar{\lambda} h(\bar{\boldsymbol{\alpha}}) + \limsup_{k\to\infty} \frac{\mu_k}{2}||h(\boldsymbol{\alpha}^k)||^2 \;\leq\; f^*. \tag{1.36}$$

Since $||h(\boldsymbol{\alpha}^k)||^2 \geq 0$ and $\mu_k \to \infty$, it follows that $h(\boldsymbol{\alpha}^k) \to 0$ and

$$h(\bar{\boldsymbol{\alpha}}) = 0, \tag{1.37}$$

for otherwise the left-hand side of (1.36) would equal ∞, while $f^* < \infty$ (since the constraint set is assumed nonempty). Since $\boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u}$ is a closed set, we also obtain that $\boldsymbol{l} \leq \bar{\boldsymbol{\alpha}} \leq \boldsymbol{u}$. Hence, $\bar{\boldsymbol{\alpha}}$ is feasible, and since from (1.36) and (1.37) we have $f(\bar{\boldsymbol{\alpha}}) \leq f^*$, it follows that $\bar{\boldsymbol{\alpha}}$ is optimal. □

The Unconstrained Augmented Lagrangian Method

Bertsekas indicates in [9] that bound constraints may be included into the AL function using nonzero slack variables s_i and t_i satisfying $l_i - \alpha_i + s_i^2 = 0$ and $\alpha_i - u_i + t_i^2 = 0$. Then (1.25)

can be alternatively written

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \;\; & f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}'\mathbf{A}\boldsymbol{\alpha} - \boldsymbol{\alpha}'\boldsymbol{b} \\ \text{subject to} \;\; & \boldsymbol{y}'\boldsymbol{\alpha} = 0, \\ & l_i - \alpha_i + s_i^2 = 0, \quad \text{for } i = 1, \ldots, p, \\ & \alpha_i - u_i + t_i^2 = 0, \quad \text{for } i = 1, \ldots, p. \end{aligned} \tag{1.38}$$

Next, we incorporate all of the constraints into the augmented Lagrangian, using one penalty parameter as per [9], to obtain

$$\begin{aligned} \mathcal{L}(\boldsymbol{\alpha}, \lambda, \boldsymbol{\gamma}, \mu, \boldsymbol{s}, \boldsymbol{t}) = f(\boldsymbol{\alpha}) &+ \lambda \boldsymbol{y}'\boldsymbol{\alpha} + \tfrac{\mu}{2}(\boldsymbol{y}'\boldsymbol{\alpha})^2 \\ &+ \sum_{i=1}^{p} \left[\gamma_{l_i}(l_i - \alpha_i + s_i^2) + \gamma_{u_i}(\alpha_i - u_i + t_i^2)\right] \\ &+ \tfrac{\mu}{2} \sum_{i=1}^{p} \left[(l_i - \alpha_i + s_i^2)^2 + (\alpha_i - u_i + t_i^2)^2\right], \end{aligned} \tag{1.39}$$

with $\boldsymbol{s} = (s_1, \ldots, s_p)'$, $\boldsymbol{t} = (t_1, \ldots, t_p)'$. Each iteration of the AL method updates the Lagrange multiplier parameters λ and

$$\boldsymbol{\gamma} = (\boldsymbol{\gamma}_l', \boldsymbol{\gamma}_u')' = (\gamma_{l_1}, \ldots, \gamma_{l_p}, \gamma_{u_1}, \ldots, \gamma_{u_p})',$$

and quadratic penalty parameter μ.

To remove $\boldsymbol{s}$ from (1.39), we consider minimizing the augmented Lagrangian with respect to $\boldsymbol{s}$ and setting it to equal to $\mathbf{0}$:

$$\begin{aligned} \mathbf{0} &= \nabla_{\boldsymbol{s}} \mathcal{L}(\boldsymbol{\alpha}, \lambda, \boldsymbol{\gamma}, \mu, \boldsymbol{s}, \boldsymbol{t}) \\ &= 2\boldsymbol{\gamma}_l \odot \boldsymbol{s} + \mu(\boldsymbol{l} - \boldsymbol{\alpha} + \boldsymbol{s}^2) \odot 2\boldsymbol{s} \\ &= 2\boldsymbol{s} \odot (\boldsymbol{\gamma}_l + \mu(\boldsymbol{l} - \boldsymbol{\alpha} + \boldsymbol{s}^2)), \end{aligned} \tag{1.40}$$

where '$\odot$' indicates component-wise multiplication. Thus by (1.40), we have that either $s_i^* = 0$ or $(s_i^*)^2 = -(\gamma_{l_i}/\mu + l_i - \alpha_i)$. Thus we can write the solution as $(\boldsymbol{s}^*)^2 = \max\left\{\mathbf{0}, -(\frac{\boldsymbol{\gamma}_l}{\mu} + \boldsymbol{l} - \boldsymbol{\alpha})\right\}$, where the maximum function is component-wise. Substituting $(\boldsymbol{s}^*)^2$ into the summation terms in (1.39) involving s_i^2 yields

$$\begin{aligned}
&\sum_{i=1}^{p}\left[\gamma_{l_i}\left(l_i - \alpha_i + \max\left\{0, -\tfrac{1}{\mu}(\gamma_{l_i} + \mu(l_i - \alpha_i))\right\}\right) + \tfrac{\mu}{2}\left(l_i - \alpha_i + \max\left\{0, -\tfrac{1}{\mu}(\gamma_{l_i} + \mu(l_i - \alpha_i))\right\}\right)^2\right] \\
&\qquad = \sum_{i=1}^{p}\begin{cases} \gamma_{l_i}(l_i - \alpha_i) + \frac{\mu}{2}(l_i - \alpha_i)^2 & \text{if } \gamma_{l_i} + \mu(l_i - \alpha_i) > 0 \\ -\frac{1}{2\mu}\gamma_{l_i}^2 & \text{otherwise} \end{cases} \\
&\qquad = \frac{1}{2\mu}\sum_{i=1}^{p}\begin{cases} (2\mu\gamma_{l_i}(l_i - \alpha_i) + \mu^2(l_i - \alpha_i)^2) & \text{if } \gamma_{l_i} + \mu(l_i - \alpha_i) > 0 \\ -\gamma_{l_i}^2 & \text{otherwise} \end{cases} \\
&\qquad = \frac{1}{2\mu}\sum_{i=1}^{p}\begin{cases} (\gamma_{l_i} + \mu(l_i - \alpha_i))^2 - \gamma_{l_i}^2 & \text{if } \gamma_{l_i} + \mu(l_i - \alpha_i) > 0 \\ -\gamma_{l_i}^2 & \text{otherwise} \end{cases} \\
&\qquad = \frac{1}{2\mu}\sum_{i=1}^{p}[\max\{0, \gamma_{l_i} + \mu(l_i - \alpha_i)\}^2 - \gamma_{l_i}^2].
\end{aligned}$$

Performing the analogous computation for the slack variable $\boldsymbol{t}$, the augmented Lagrangian in (1.39) simplifies to

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\alpha}, \lambda, \boldsymbol{\gamma}, \mu) = f(\boldsymbol{\alpha}) &+ \lambda \boldsymbol{y}'\boldsymbol{\alpha} + \tfrac{\mu}{2}(\boldsymbol{y}'\boldsymbol{\alpha})^2 \\
&+ \frac{1}{2\mu}\sum_{i=1}^{p}[\max\{0, \gamma_{l_i} + \mu(l_i - \alpha_i)\}^2 - \gamma_{l_i}^2] \\
&+ \frac{1}{2\mu}\sum_{i=1}^{p}[\max\{0, \gamma_{u_i} + \mu(\alpha_i - u_i)\}^2 - \gamma_{u_i}^2].
\end{aligned} \tag{1.41}$$

Notice the similarity between (1.41) and (1.29), yet we now have included both bound and equality constraints in the augmented Lagrangian.

The update rule for λ and $\boldsymbol{\gamma}$ are derived from (1.41). The KKT conditions state that the

minimizer $(\boldsymbol{\alpha}^*, \lambda^*, \boldsymbol{\gamma}^*, \mu^*)$ of $\mathcal{L}(\boldsymbol{\alpha}, \lambda, \boldsymbol{\gamma}, \mu)$ satisfies

$$\mathbf{0} = \nabla_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha}^*, \lambda^*, \boldsymbol{\gamma}^*, \mu^*) = \nabla_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}^*) + \lambda^* \boldsymbol{y} + \sum_{i=1}^{p} \gamma_{l_i}^* \nabla_{\boldsymbol{\alpha}} g_{l_i}(\boldsymbol{\alpha}^*) + \sum_{i=1}^{p} \gamma_{u_i}^* \nabla_{\boldsymbol{\alpha}} g_{u_i}(\boldsymbol{\alpha}^*), \quad (1.42)$$

where $g_{l_i}(\boldsymbol{\alpha}) = l_i - \alpha_i$ and $g_{u_i}(\boldsymbol{\alpha}) = \alpha_i - u_i$ are the inequality constraints and we used the fact that $\boldsymbol{y}'\boldsymbol{\alpha}^* = 0$. Moreover, we have that at each iteration k,

$$\begin{aligned} \mathbf{0} = \nabla_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha}^k, \lambda_k, \boldsymbol{\gamma}^k, \mu_k) = \nabla_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}^k) + (\lambda_k + \mu_k \boldsymbol{y}'\boldsymbol{\alpha}^k)\boldsymbol{y} \\ + \sum_{\{i | -\gamma_{l_i}^k < \mu^k (l_i - \alpha_i^k)\}} (\gamma_{l_i}^k + \mu_k (l_i - \alpha_i^k)) \nabla_{\boldsymbol{\alpha}} g_{l_i}(\boldsymbol{\alpha}^k) \\ + \sum_{\{i | -\gamma_{u_i}^k < \mu^k (\alpha_i^k - u_i)\}} (\gamma_{u_i}^k + \mu_k (\alpha_i^k - u_i)) \nabla_{\boldsymbol{\alpha}} g_{u_i}(\boldsymbol{\alpha}^k). \end{aligned} \quad (1.43)$$

Equating (1.42) and (1.43), we see the update rule for λ is still given by (1.32), while the update for the Lagrangian vector $\boldsymbol{\gamma}$ is given by

$$\begin{aligned} \boldsymbol{\gamma}_l^{k+1} &= \max\{\mathbf{0}, \boldsymbol{\gamma}_l^k + \mu_k (\boldsymbol{l} - \boldsymbol{\alpha}^k)\}, \\ \boldsymbol{\gamma}_u^{k+1} &= \max\{\mathbf{0}, \boldsymbol{\gamma}_u^k + \mu_k (\boldsymbol{\alpha}^k - \boldsymbol{u})\}. \end{aligned} \quad (1.44)$$

Summarizing thus far, applying this method to the bound and equality constrained program of (1.28) requires the solution of a sequence of minimizations of (1.41). Bertsekas states in [9] that this is equivalent to the equality constrained problem of (1.38), and therefore (1.25), where the convergence result of Theorem 1 still holds, with the addition of Lagrange multiplier vector $\boldsymbol{\gamma}$.

However, in order to apply an unconstrained optimization method such as the conjugate gradient to (1.41), we require that $\mathcal{L}(\boldsymbol{\alpha}, \lambda, \boldsymbol{\gamma}, \mu)$ is quadratic in $\boldsymbol{\alpha}$. This can be done by obtaining linear approximations of the maximum function in (1.41), where we assume $\boldsymbol{\alpha} = \boldsymbol{\alpha}^k$

and define

$$
\begin{aligned}
[\mathbf{I}_{l,k}]_{ii} &= \left\{ \begin{array}{ll} 1 & \text{if } \gamma_{l_i} + \mu(l_i - \alpha_i^k) > 0 \\ 0 & \text{otherwise} \end{array} \right\}, \\
[\mathbf{I}_{u,k}]_{ii} &= \left\{ \begin{array}{ll} 1 & \text{if } \gamma_{u_i} + \mu(\alpha_i^k - u_i) > 0 \\ 0 & \text{otherwise} \end{array} \right\}.
\end{aligned}
\tag{1.45}
$$

Theorem 2. *The augmented Lagrangian in (1.41) can be approximated by the quadratic function*

$$
\mathcal{L}_k(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}'\widehat{\mathbf{A}}_k\boldsymbol{\alpha} - \boldsymbol{\alpha}'\widehat{\boldsymbol{b}}_k, \tag{1.46}
$$

where

$$
\widehat{\mathbf{A}}_k = \mathbf{A} + \mu_k(\boldsymbol{y}\boldsymbol{y}' + \mathbf{I}_{l,k} + \mathbf{I}_{u,k}), \tag{1.47}
$$

and

$$
\widehat{\boldsymbol{b}}_k = \boldsymbol{b} - \lambda_k\boldsymbol{y} + \mathbf{I}_{l,k}(\mu_k\boldsymbol{l} + \boldsymbol{\gamma}_l^k) + \mathbf{I}_{u,k}(\mu_k\boldsymbol{u} - \boldsymbol{\gamma}_u^k). \tag{1.48}
$$

Proof. We begin with the form of the augmented Lagrangian in (1.41) and incorporate the indicator matrices in (1.45).

$$
\begin{aligned}
\mathcal{L}_k(\boldsymbol{\alpha}) &= \mathcal{L}(\boldsymbol{\alpha}, \lambda_k, \boldsymbol{\gamma}^k, \mu_k) \\
&= f(\boldsymbol{\alpha}) + \lambda_k\boldsymbol{y}'\boldsymbol{\alpha} + \tfrac{\mu_k}{2}(\boldsymbol{y}'\boldsymbol{\alpha})^2 \\
&\quad + \frac{1}{2\mu_k}\sum_{i=1}^{p}\left\{ \begin{array}{ll} -(\gamma_{l_i}^k)^2 & \text{if } \gamma_{l_i}^k + \mu_k(l_i - \alpha_i) < 0 \\ (\gamma_{l_i}^k + \mu_k(l_i - \alpha_i))^2 - (\gamma_{l_i}^k)^2 & \text{otherwise} \end{array} \right. \\
&\quad + \frac{1}{2\mu_k}\sum_{i=1}^{p}\left\{ \begin{array}{ll} -(\gamma_{u_i}^k)^2 & \text{if } \gamma_{u_i}^k + \mu_k(\alpha_i - u_i) < 0 \\ (\gamma_{u_i}^k + \mu_k(\alpha_i - u_i))^2 - (\gamma_{u_i}^k)^2 & \text{otherwise} \end{array} \right. \\
&= \frac{1}{2}\boldsymbol{\alpha}'\mathbf{A}\boldsymbol{\alpha} - \boldsymbol{\alpha}'\boldsymbol{b} + \lambda_k\boldsymbol{y}'\boldsymbol{\alpha} + \tfrac{\mu_k}{2}\boldsymbol{\alpha}'\boldsymbol{y}\boldsymbol{y}'\boldsymbol{\alpha} \\
&\quad + \frac{1}{2\mu_k}\sum_{i=1}^{p}\left\{ \begin{array}{ll} 0 & \text{if } \gamma_{l_i}^k + \mu_k(l_i - \alpha_i) < 0 \\ -2\mu_k\gamma_{l_i}^k\alpha_i + \mu_k^2(\alpha_i^2 - 2l_i\alpha_i) & \text{otherwise} \end{array} \right.
\end{aligned}
$$

$$
\begin{aligned}
&+\frac{1}{2\mu_k}\sum_{i=1}^{p}\begin{cases} 0 & \text{if } \gamma_{u_i}^k+\mu_k(\alpha_i-u_i)<0 \\ 2\mu_k\gamma_{u_i}^k\alpha_i+\mu_k^2(\alpha_i^2-2u_i\alpha_i) & \text{otherwise}\end{cases}\\
&+\text{ non-}\boldsymbol{\alpha}\text{ terms}\\
=\;&\frac{1}{2}\boldsymbol{\alpha}'\left(\mathbf{A}+\mu_k\boldsymbol{y}\boldsymbol{y}'\right)\boldsymbol{\alpha}-\boldsymbol{\alpha}'\left(\boldsymbol{b}-\lambda_k\boldsymbol{y}\right)\\
&+\frac{1}{2\mu_k}\left(-2\mu_k(\boldsymbol{\gamma}_l^k)'\mathbf{I}_{l,k}\boldsymbol{\alpha}+\mu_k^2\boldsymbol{\alpha}'\mathbf{I}_{l,k}\boldsymbol{\alpha}-2\mu_k^2\boldsymbol{l}'\mathbf{I}_{l,k}\boldsymbol{\alpha}\right)\\
&+\frac{1}{2\mu_k}\left(2\mu_k(\boldsymbol{\gamma}_u^k)'\mathbf{I}_{u,k}\boldsymbol{\alpha}+\mu_k^2\boldsymbol{\alpha}'\mathbf{I}_{u,k}\boldsymbol{\alpha}-2\mu_k^2\boldsymbol{u}'\mathbf{I}_{u,k}\boldsymbol{\alpha}\right)+\text{ non-}\boldsymbol{\alpha}\text{ terms}\\
=\;&\frac{1}{2}\boldsymbol{\alpha}'\left(\mathbf{A}+\mu_k(\boldsymbol{y}\boldsymbol{y}'+\mathbf{I}_{l,k}+\mathbf{I}_{u,k})\right)\boldsymbol{\alpha}-\boldsymbol{\alpha}'(\boldsymbol{b}-\lambda_k\boldsymbol{y}+\mathbf{I}_{l,k}(\mu_k\boldsymbol{l}+\boldsymbol{\gamma}_l^k)+\mathbf{I}_{u,k}(\mu_k\boldsymbol{u}-\boldsymbol{\gamma}_u^k))\\
&+\text{ non-}\boldsymbol{\alpha}\text{ terms}
\end{aligned}
$$

All non-$\boldsymbol{\alpha}$ terms above may be disregarded as they do not affect the minimization of the augmented Lagrangian with respect to $\boldsymbol{\alpha}$, in which case, we obtain (1.46), where $\widehat{\mathbf{A}}$ and $\widehat{\boldsymbol{b}}$ are defined in (1.47) and (1.48), respectively. □

The benefit of (1.46) is that an unconstrained quadratic optimizer, such as the conjugate gradient method or even a direct solver such as Gaussian elimination for small-scale problems, can be used to compute its minimizer, yielding the updated approximation $\boldsymbol{\alpha}^{k+1}$. Note, however, that the Hessian must be symmetric and positive definite in order to apply CG, which will occur if the training data $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_p)$ are linearly independent. To ensure positive definiteness, we add machine epsilon to the diagonal of the Hessian. Pseudocode for the conjugate gradient method can be found in Section 1.6.3.

The pseudocode for this AL method is given in Algorithm 2, which closely resembles Algorithm 1, with the added update for the Lagrangian parameter vector $\boldsymbol{\gamma}$. We use the conjugate gradient method to minimize (1.46) with relative residual norm stopping tolerance 10^{-6}, where the relative residual norm is defined $r_n=||\boldsymbol{b}-\mathbf{A}\boldsymbol{\alpha}^k||/||\boldsymbol{b}||$.

To the best of our knowledge, the unconstrained augmented Lagrangian method represented by Algorithm 2 has not been developed elsewhere nor applied to the SVM optimization prob-

Algorithm 2 Unconstrained Augmented Lagrangian Method

Given $\mu_0 \geq 0$ and initial points $\boldsymbol{\alpha}^0$, λ_0 and $\boldsymbol{\gamma}_0$, set $k = 0$.

1. Fix constant indicator matrices (1.45) using $\boldsymbol{\alpha}^k$, then use the conjugate gradient method to find the approximate minimizer $\boldsymbol{\alpha}^{k+1}$ of $\mathcal{L}_k(\boldsymbol{\alpha})$ in (1.46), with starting point $\boldsymbol{\alpha}^k$ and with relative residual norm stopping tolerance of 10^{-6}.
2. If $|\boldsymbol{y}'\boldsymbol{\alpha}^k| < 10^{-6}$ and bound constraints hold, stop with approximate solution $\boldsymbol{\alpha}^{k+1}$.
3. Update Lagrange multipliers via (1.32) and (1.44) to obtain λ_{k+1} and $\boldsymbol{\gamma}^{k+1}$, and choose a new penalty parameter μ_{k+1} via (1.33).
4. Set $k = k + 1$ and return to Step 1.

lem.

1.6.2 Scaled Gradient Projection

The scaled gradient projection (SGP) algorithm was proposed for image restoration applications in [10]. We note that to our knowledge, this method has not been adapted, nor implemented, for the solution of SVMs. We do so here.

To apply SGP to (1.25), we must make adjustments to the method presented in [10] and hence we present a majority of the algorithm here. We define the current iterate $\boldsymbol{\alpha}^k$, the feasible set

$$\Omega = \left\{ \boldsymbol{\alpha} \in \mathbb{R}^n \mid \boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u},\ \boldsymbol{y}'\boldsymbol{\alpha} = \boldsymbol{0} \right\},$$

where the inequality is component-wise, and the projection operator

$$P_\Omega(\boldsymbol{\alpha}) \equiv \arg\min_{\boldsymbol{\beta} \in \Omega} ||\boldsymbol{\beta} - \boldsymbol{\alpha}||_2 = \arg\min_{\boldsymbol{\beta} \in \Omega} \left(\frac{1}{2}\boldsymbol{\beta}'\boldsymbol{\beta} - \boldsymbol{\beta}'\boldsymbol{\alpha} \right). \tag{1.49}$$

The authors of [10] suggest a more general scaled projection, but we found (1.49) worked best.

Next we define

$$\boldsymbol{\beta}^k = \mathcal{P}_\Omega(\boldsymbol{\alpha}^k - \lambda_k(\mathbf{A}\boldsymbol{\alpha}^k - \boldsymbol{b})), \tag{1.50}$$

to be the projection of the negative gradient path onto Ω, where λ_k is chosen by the Barzilai and Borwein-like steplength selection method of [10] given in Section 1.6.3. If $\boldsymbol{\beta}^k = \boldsymbol{\alpha}^k$, we have $\boldsymbol{\alpha}^k$ as a stationary point, otherwise the descent direction is taken as $\boldsymbol{d}^k = \boldsymbol{\beta}^k - \boldsymbol{\alpha}^k$, and a backtracking line search is used to ensure a sufficient decrease in the objective function value at the current iterate over the last M iterations. For $M = 1$, this backtracking loop is simply the Armijo rule in (1.55) below [1]. These steps completely describe one iteration of the SGP method in [10].

The projection remains to be defined and is where our derivation differs from [10]. Given the quadratic programming problem (1.25), the SGP projection (1.50) is equivalent to the solution of the constrained and strictly convex quadratic program

$$\boldsymbol{\alpha}^* = \left\{ \arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}'\boldsymbol{\alpha} - \boldsymbol{\alpha}'\boldsymbol{z} \quad \text{subject to} \quad \boldsymbol{l} \leq \boldsymbol{\alpha} \leq \boldsymbol{u} \quad \text{and} \quad \boldsymbol{y}'\boldsymbol{\alpha} = 0 \right\}, \tag{1.51}$$

where $\boldsymbol{z} = \boldsymbol{\alpha}^k - \lambda_k(\mathbf{A}\boldsymbol{\alpha}^k - \boldsymbol{b})$. Then $\boldsymbol{\beta}^k = \boldsymbol{\alpha}^*$.

Letting $\boldsymbol{\alpha}^*$ be the solution of (1.51), using the KKT first order optimality conditions, there exist Lagrange multipliers $\lambda^* \in \mathbb{R}$ and $\boldsymbol{\mu}^*, \boldsymbol{\gamma}^* \in \mathbb{R}^n$ such that we have:

Stationarity	$\boldsymbol{\alpha}^* - \boldsymbol{z} + \lambda^*\boldsymbol{y} - \boldsymbol{\mu}^* + \boldsymbol{\gamma}^*$	$=$	$\mathbf{0}$,	
Primal Feasibility	$\boldsymbol{y}'\boldsymbol{\alpha}^*$	$=$	0,	
Primal Feasibility	$\boldsymbol{\alpha}^*$	$\geq$	$\mathbf{0}$,	
Primal Feasibility	$\boldsymbol{\alpha}^*$	$\leq$	$\boldsymbol{u}$,	
Dual Feasibility	$\boldsymbol{\mu}^*$	$\geq$	$\mathbf{0}$,	
Dual Feasibility	$\boldsymbol{\gamma}^*$	$\geq$	$\mathbf{0}$,	
Complementary Slackness	$\mu_i^*\alpha_i^*$	$=$	0,	$\forall\ i = 1, \ldots, p,$
Complementary Slackness	$\gamma_i^*(\alpha_i^* - u_i)$	$=$	0,	$\forall\ i = 1, \ldots, p.$

For simplicity, let $b(\lambda) = \boldsymbol{z} - \lambda\boldsymbol{y}$. The stationarity condition yields equations for $\boldsymbol{\alpha}^*$, $\boldsymbol{\mu}^*$, and

$\boldsymbol{\gamma}^*$, as functions of one another and λ^*:

$$\alpha_i^*(\lambda^*) = \mu_i^* - \gamma_i^* + [b(\lambda^*)]_i$$
$$\mu_i^*(\lambda^*) = \gamma_i^* + \alpha_i^* - [b(\lambda^*)]_i$$
$$\gamma_i^*(\lambda^*) = \mu_i^* - \alpha_i^* + [b(\lambda^*)]_i.$$

Theorem 3. *The complementary slackness KKT conditions yield $\boldsymbol{\alpha}^*$ as a function solely of λ^*:*

$$\alpha_i^* = \max\{0, \min\{(z_i - \lambda^* y_i), u_i^*\}\}. \tag{1.52}$$

Proof. The two complementary slackness KKT conditions yield two different sets of cases for solving for $\boldsymbol{\alpha}^*$, $\boldsymbol{\mu}^*$, and $\boldsymbol{\gamma}^*$ as functions of λ^*. Condition $\mu_i^*\alpha_i^* = 0$ indicates for all $i = 1, \ldots, p$ either $\alpha_i^* = 0$ (case 1) or $\mu_i^* = 0$ (case 2). Condition $\gamma_i^*(\alpha_i^* - u_i) = 0$ indicates for all $i = 1, \ldots, p$ that either $\alpha_i^* - u_i = 0$ (case A) or $\gamma_i^* = 0$ (case B).

Combining the cases appropriately, we obtain the feasible solutions. Obviously case 1 and case A cannot simultaneously occur as we assume the upper and lower bounds are not identical, i.e., $u_i \neq l_i$. Case 2 and case A have solution

$$\mu_i^* = 0, \quad \alpha_i^* = u_i, \quad \text{and} \quad \gamma_i^*(\lambda^*) = [b(\lambda^*)]_i - u_i.$$

Case 1 and case B yield

$$\alpha_i^* = 0, \quad \gamma_i^* = 0, \quad \text{and} \quad \mu_i^*(\lambda^*) = -[b(\lambda^*)]_i.$$

Lastly, case 2 and case B have solution

$$\mu_i^* = 0, \quad \gamma_i^* = 0, \quad \text{and} \quad \alpha_i^*(\lambda^*) = [b(\lambda^*)]_i.$$

The question that remains is, within the algorithm, how do we know which case we have? The feasible solution depends on the sign, and sometimes magnitude, of $[b(\lambda^*)]_i$.

If $[b(\lambda^*)]_i < 0$, the combination of cases 2 and B contain a contradiction with the nonnegativity constraint on $\boldsymbol{\alpha} \geq \mathbf{0}$, namely $\alpha_i^*(\lambda^*) \neq [b(\lambda^*)]_i < 0$. Considering the combination of cases 2 and A,

$$\gamma_i = \underbrace{[b(\lambda^*)]_i}_{<0} \underbrace{-u_i}_{<0} < 0.$$

But by the nonnegativity constraint on Lagrange multiplier γ_i, this also leads to a contradiction. However, the combination of cases 1 and B are feasible with the KKT conditions and therefore

$$\alpha_i^* = 0, \quad \gamma_i^* = 0, \quad \text{and} \quad \mu_i^*(\lambda^*) = -[b(\lambda^*)]_i$$

whenever $[b(\lambda^*)]_i < 0$.

If $0 < [b(\lambda^*)]_i < u_i$, the combination of cases 1 and B are not feasible due to the nonnegativity constraints on Lagrange multiplier μ_i. Similarly, cases 2 and A lead to the same contradiction on Lagrange multiplier γ_i with

$$\gamma_i = \underbrace{[b(\lambda^*)]_i}_{<u_i} - u_i < 0.$$

However, the combination of cases 2 and B do not contradict the KKT conditions and therefore

$$\mu_i^* = 0, \quad \gamma_i^* = 0, \quad \text{and} \quad \alpha_i^*(\lambda^*) = [b(\lambda^*)]_i$$

whenever $0 < [b(\lambda^*)]_i < u_i$.

If $[b(\lambda^*)]_i > u_i$, then the constraint $\alpha_i \leq u_i$ is violated in the combination of cases 2 and B. As before, $[b(\lambda^*)]_i > u_i$ violates the nonnegativity constraints on Lagrange multiplier μ_i in cases 1 and B. However, the solution in cases 2 and A is feasible with the KKT conditions

and

$$\mu_i^* = 0, \quad \alpha_i^* = u_i, \quad \text{and} \quad \gamma_i^*(\lambda^*) = [b(\lambda^*)]_i - u_i$$

whenever $[b(\lambda^*)]_i > u_i$.

If $[b(\lambda^*)]_i = u_i$, cases 2 and A yield the same results as cases 2 and B:

$$\mu_i^* = 0, \quad \gamma_i^* = 0, \quad \text{and} \quad \alpha_i^* = u_i.$$

Trivially, if $[b(\lambda^*)]_i = 0$, then $\alpha_i^* = 0$, $\mu_i^* = 0$, and $\gamma_i^* = 0$.

Combining all of the above discussion, we can now write $\boldsymbol{\alpha}^*$ as (1.52). □

Thus, to solve the KKT system, we must find λ^* satisfying the remaining constraint

$$\sum_{i=1}^{p} \alpha_i^*(\lambda^*) y_i = 0. \tag{1.53}$$

The solution λ^* of (1.53) can be computed using one of many root finding algorithms. We use MATLAB's `fzero`, while in [24] a secant-based method is introduced. Note, the solution $\boldsymbol{\alpha}^*$ defined by (1.52) and (1.53) yields the desired SGP projection of (1.50), $\boldsymbol{\beta}^k = \boldsymbol{\alpha}^*$.

Convergence of SGP is proven in [10]. Recall that $\boldsymbol{\alpha}^k$ is a stationary point if $\boldsymbol{\beta}^k = \boldsymbol{\alpha}^k$. We consider the following additional termination criteria for SGP: a maximum number of iterations and a projected gradient norm or step norm tolerance of 10^{-6}, with the projected gradient defined in (1.57). Pseudocode for SGP is given in Algorithm 3. In our trials, initial algorithm parameter values are taken as $\eta = 10^{-4}$, $\theta = 0.4$, $\lambda_{min} = 10^{-5}$ and $\lambda_{max} = 10^5$, as suggested in [10]. Pseudocode for the suggested Barzilai and Borwein-like steplength selection method is given in Section 1.6.3.

Algorithm 3 Scaled Gradient Projection (SGP)

Given initial, feasible $\boldsymbol{\alpha}^0$, parameters η, $\theta \in (0,1)$, $0 < \lambda_{min} < \lambda_{max}$, and positive integer M, set $k = 0$. Iterate on k:

1. Select $\lambda_k \in [\lambda_{min}, \lambda_{max}]$ using the Barzilai and Borwein-like steplength selection method defined in [10].
2. Project $\boldsymbol{\beta}^k = P_\Omega(\boldsymbol{\alpha}^k - \lambda_k \nabla f(\boldsymbol{\alpha}^k))$ by solving (1.52) and (1.53). If $\boldsymbol{\beta}^k = \boldsymbol{\alpha}^k$, stop with stationary point $\boldsymbol{\alpha}^k$. If other stopping tolerances are met, stop with approximate solution $\boldsymbol{\alpha}^*$.
3. Compute descent direction $\boldsymbol{d}^k = \boldsymbol{\beta}^k - \boldsymbol{\alpha}^k$.
4. Set $\rho_k = 1$ and $f_{max} = \max_{0 \le j \le \min(k, M-1)} f(\boldsymbol{\alpha}^{k-j})$.
5. Backtracking loop: If $f(\boldsymbol{\alpha}^k + \rho_k \boldsymbol{d}^k) > f_{max} + \eta \rho_k \nabla f(\boldsymbol{\alpha}^k)' \boldsymbol{d}^k$, then set $\rho_k = \theta \rho_k$ and return to step 5.
6. Set $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho_k \boldsymbol{d}^k$ and $k = k + 1$. Return to Step 1.

1.6.3 Supportive Algorithms

This section contains both the projected Newton and conjugate gradient methods used in the augmented Lagrangian algorithms described above, as well as the Barzilai and Borwein-like steplength selection process described in [10], for completeness.

Projected Newton

Bertsekas introduces the projected Newton (PN) algorithm in [8] as being well suited for solving large-scale, bound constrained, quadratic programming problems of the form (1.27). We define the feasible set

$$\Omega = \{\boldsymbol{\alpha} \in \mathbb{R}^n | \boldsymbol{l} \le \boldsymbol{\alpha} \le \boldsymbol{u}\},$$

and the current iterate $\boldsymbol{\alpha}^k$. Then the new PN iterate has the form

$$\boldsymbol{\alpha}^{k+1} = \mathcal{P}(\boldsymbol{\alpha}^k - \lambda_k \mathcal{R}_k^{-1} \nabla f(\boldsymbol{\alpha}^k)),$$

where $\mathcal{P}(\cdot)$ is the projection onto the feasible set Ω defined by

$$\mathcal{P}(\alpha_i) = \left\{ \begin{array}{ll} l_i & \text{if } \alpha_i \leq l_i \\ \alpha_i & \text{if } l_i < \alpha_i < u_i \\ u_i & \text{if } \alpha_i \geq u_i \end{array} \right\},$$

λ_k is the steplength parameter given by the line search method below, and $\mathcal{R}_k$ is the symmetric, positive definite reduced Hessian described below.

For $\lambda > 0$, we define

$$\boldsymbol{\alpha}^k(\lambda) = \mathcal{P}(\boldsymbol{\alpha}^k - \lambda \mathcal{R}_k^{-1} \nabla f(\boldsymbol{\alpha}^k)). \tag{1.54}$$

Note then that $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k(\lambda_k)$. Moreover, for large scale problems, the conjugate gradient algorithm can be applied to efficiently approximate $\mathcal{R}_k^{-1} \nabla f(\boldsymbol{\alpha}^k)$ without directly calculating the inverse of the reduced Hessian.

A common line search method for selecting the steplength parameter λ_k is the Armijo rule [1, 53], which in its simplest form is given by the sufficient decrease condition

$$f(\boldsymbol{\alpha}^k(\lambda_k)) - f(\boldsymbol{\alpha}^k) \leq \frac{-\beta}{\lambda_k} ||\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^k(\lambda_k)||^2, \tag{1.55}$$

where $\lambda_k = \gamma^m$, $\gamma \in (0, 1)$, with m the least positive integer such that (1.55) holds. We take $\gamma = 0.1$, with $\beta = 10^{-4}$ as suggested in [53].

We now define the reduced Hessian, $\mathcal{R}_k$. For $\boldsymbol{\alpha} \in \Omega$, following Kelley [53], we define

$$\epsilon_k = \min \left(||\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^k(1)||, \min_i (u_i - l_i)/2 \right).$$

The ϵ-*active set* at $\boldsymbol{\alpha}^k$ is then defined

$$\mathcal{A}^\epsilon(\boldsymbol{\alpha}^k) = \{i | u_i - \alpha_i^k \leq \epsilon_k \quad \text{or} \quad \alpha_i^k - l_i \leq \epsilon_k\},$$

and the symmetric reduced Hessian matrix takes the form

$$[\mathcal{R}_k]_{ij} = \begin{cases} \delta_{ij} & \text{if } i \text{ or } j \in \mathcal{A}^{\epsilon_k}(\boldsymbol{\alpha}^k), \\ \mathbf{A}_{ij} & \text{otherwise,} \end{cases}$$

for $1 \leq i, j \leq n$, where δ_{ij} is the Kronecker delta:

$$\delta_{ij} = \begin{cases} 1 & i = j, \\ 0 & i \neq j. \end{cases}$$

Termination for PN is related to the measure of stationarity, $||\boldsymbol{\alpha} - \boldsymbol{\alpha}(1)||$, such that termination occurs at iteration k if

$$||\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^k(1)|| \leq \tau_a + \tau_r r_0, \tag{1.56}$$

where τ_a and τ_r are absolute and relative tolerances, respectively, and $r_0 = ||\boldsymbol{\alpha}^0 - \boldsymbol{\alpha}^0(1)||$. In our implementation, we use $\tau_a = 10^{-6}$ and $\tau_r = 0.05$. In addition, the PN algorithm is also set to terminate if either a maximum number of iterations is exceeded or if the tolerance is met for the norm of the step or for the norm of the *projected gradient*, defined by

$$[\nabla_{\mathcal{P}} f(\boldsymbol{\alpha})]_i = \begin{cases} [\mathbf{A}\boldsymbol{\alpha} - \boldsymbol{b}]_i & \text{if } \alpha_i \in (l_i, u_i), \\ \min\{[\mathbf{A}\boldsymbol{\alpha} - \boldsymbol{b}]_i, 0\} & \text{if } \alpha_i = l_i, \\ \max\{[\mathbf{A}\boldsymbol{\alpha} - \boldsymbol{b}]_i, 0\} & \text{if } \alpha_i = u_i. \end{cases} \tag{1.57}$$

We take a projected gradient norm tolerance of 10^{-6}. Pseudocode for the PN algorithm is given in Algorithm 4.

In the first iteration of PN, Step 1(b) requires the computation of $\boldsymbol{\alpha}^1(1)$ via (1.54); however, we note that $\mathcal{R}_1$ has not yet been calculated. Bertsekas suggests in [8] using a fixed diagonal positive definite matrix M, such as the identity matrix, in place of $\mathcal{R}_1$ in that case.

Algorithm 4 Projected Newton (PN)

Given $\boldsymbol{\alpha}^1$, $f(\boldsymbol{\alpha}^1)$, λ_1, τ_a, τ_r, and *maxiter*,

1. For k = 1,. . . ,*maxiter*
   (a) Compute $f(\boldsymbol{\alpha}^k)$ and $\nabla f(\boldsymbol{\alpha}^k)$; test for termination using (1.56).
   (b) Set $\epsilon_k = \min(||\boldsymbol{\alpha}^k - \boldsymbol{\alpha}^k(\lambda)||, \min(u_i - l_i)/2)$.
   (c) Compute and factor $\mathcal{R}_k = \mathcal{R}_k(\nabla^2 f(\boldsymbol{\alpha}^k), \epsilon_k)$. If $\mathcal{R}_k$ is not positive definite, terminate with a failure message.
   (d) Solve $\mathcal{R}_k d = -\nabla f(\boldsymbol{\alpha}^k)$.
   (e) Find the least integer m_k such that (1.55) holds for $\lambda_k = \beta^{m_k}$.
   (f) Set $\boldsymbol{\alpha}^k = \boldsymbol{\alpha}^k(\lambda)$ defined in (1.54).
2. If $k = maxiter$ and the termination test failed, we have signal failure. If step norm or projected gradient norms are met, stop with approximate solution $\boldsymbol{\alpha}^k$. Otherwise, return to Step 1.

Conjugate Gradient

The conjugate gradient algorithm is an iterative algorithm for solving the unconstrained linear system $\mathbf{A}\boldsymbol{\alpha} = \boldsymbol{b}$, or equivalently, minimizing $\frac{1}{2}\boldsymbol{\alpha}'\mathbf{A}\boldsymbol{\alpha} - \boldsymbol{\alpha}'\boldsymbol{b}$, where $\mathbf{A}_{n\times n}$ is symmetric and positive definite. A nice property of the CG algorithm is that it converges to the solution $\mathbf{A}^{-1}\boldsymbol{b}$ in at most n steps [4]. The pseudocode for the CG method is given in Algorithm 5.

Algorithm 5 Conjugate Gradient (CG)

Given $\mathbf{A}$, $\boldsymbol{b}$ and $\boldsymbol{\alpha}^0$, let $\boldsymbol{r}_0 = \boldsymbol{b} - \mathbf{A}\boldsymbol{\alpha}^0$, $\boldsymbol{p}_0 = \boldsymbol{r}_0$, and $k = 1$. Specify some stopping tolerance ϵ. Iterate:

1. $\gamma_{k-1} = \frac{\boldsymbol{r}'_{k-1}\boldsymbol{r}_{k-1}}{\boldsymbol{p}'_{k-1}\mathbf{A}\boldsymbol{p}_{k-1}}$ is the 1-D minimizer of ϕ in the direction $\boldsymbol{\alpha}^{k-1} + \gamma\boldsymbol{p}_{k-1}$.
2. $\boldsymbol{\alpha}^k = \boldsymbol{\alpha}^{k-1} + \gamma_{k-1}\boldsymbol{p}_{k-1}$.
3. $\boldsymbol{r}_k = -\nabla_\alpha\phi(\boldsymbol{\alpha}^k) = \boldsymbol{b} - \mathbf{A}\boldsymbol{\alpha}^k = \boldsymbol{r}_{k-1} - \gamma_{k-1}\mathbf{A}\boldsymbol{p}_{k-1}$ is the residual.
4. $\beta_k = -\frac{\boldsymbol{r}'_k\boldsymbol{r}_k}{\boldsymbol{r}'_{k-1}\boldsymbol{r}_{k-1}}$.
5. $\boldsymbol{p}_k = \boldsymbol{r}_k - \beta_k\boldsymbol{p}_{k-1}$ is the next conjugate search direction.
6. Quit if $||\boldsymbol{r}_k|| < \epsilon$. Else set $k = k + 1$ and return to Step 1.

Barzilai and Borwein Steplength Selection

The steplength λ_k in Step 1 of SGP (Algorithm 3) is selected using the Barzilai and Borwein-like updating method given in Algorithm 6.

Algorithm 6 SGP Steplength Selection

if $k = 0$ **then**
  set $\lambda_0 \in [\lambda_{min}, \lambda_{max}], \tau_1 \in (0, 1)$ and a nonnegative integer M_λ
else
  if $\boldsymbol{s}'_{k-1}\boldsymbol{t}_{k-1} \leq 0$ **then**
    $\lambda_k^{(1)} = \lambda_{max}$
  else
    $\lambda_k^{(1)} = \max\left\{\lambda_{min}, \min\left\{\frac{\boldsymbol{s}'_{k-1}\boldsymbol{s}_{k-1}}{\boldsymbol{s}'_{k-1}\boldsymbol{t}_{k-1}}, \lambda_{max}\right\}\right\}$
  end if
  if $\boldsymbol{s}'_{k-1}\boldsymbol{t}_{k-1} \leq 0$ **then**
    $\lambda_k^{(2)} = \lambda_{max}$
  else
    $\lambda_k^{(2)} = \max\left\{\lambda_{min}, \min\left\{\frac{\boldsymbol{s}'_{k-1}\boldsymbol{t}_{k-1}}{\boldsymbol{t}'_{k-1}\boldsymbol{t}_{k-1}}, \lambda_{max}\right\}\right\}$
  end if
  if $\lambda_k^{(2)}/\lambda_k^{(1)} \leq \tau_k$ **then**
    $\lambda_k = \min\left\{\lambda_j^{(2)}, j = \max\{1, k - M_\lambda\}, \ldots, k\right\}$
    $\tau_{k+1} = 0.9\tau_k$
  else
    $\lambda_k = \lambda_k^{(1)}$
    $\tau_{k+1} = 1.1\tau_k$
  end if
end if

1.7 Numerical Results

We now present three images for SVM classification using the methods presented here for comparison. All programming code was written in MATLAB. Computational time, recorded in seconds, is the average of five trials for the solution of the hyperplane for each class. Initial parameter values for the bound constrained and unconstrained AL algorithms are set at $\lambda_0 = 5$ and $\mu_0 = 10$. Initial Lagrange vector $\boldsymbol{\gamma}^0 = \mathbf{1}$ is used for the unconstrained AL algorithm. For

all algorithms, regardless of constraints, the initial guess for $\boldsymbol{\alpha}^0$ is chosen such that $\boldsymbol{y}'\boldsymbol{\alpha}^0 = 0$.

As previously discussed for multi-class classification, a separating hyperplane is built for each of the k classes in the image; thus the quadratic program must be solved k times for a single classification. To denote the algorithms, we use 'AL w/ PN' for the constrained augmented Lagrangian with projected Newton as the quadratic solver and 'AL w/ CG' for the unconstrained augmented Lagrangian with conjugate gradient as the quadratic solver. The scaled gradient projection is denoted by 'SGP'. The computational time to solve each constrained quadratic program is compared with MATLAB's built-in quadratic program solver `quadprog`, which uses an active-set method described in [44]. Furthermore, we also compare computational time with the widely-known sequential minimal optimization (SMO) algorithm, introduced by Platt in [77] as a simple, fast, and easy-to-implement technique for solving a bound and equality constrained SVM quadratic program. SMO has become a popular algorithm in the SVM literature and is often used as a comparative algorithm to test new methods [17, 22, 28, 58]. SMO solves a sequence of two-dimensional subproblems to obtain asymptotic convergence. Each subproblem requires the computation of updates to two Lagrange multipliers, one of which violates the KKT conditions. This method and its pseudocode are presented in [77].

We inspect the images visually to determine the number of classes. 10-fold cross-validation is used to measure the performance of the algorithms [49, 79], with the average percent of misclassified pixels is reported. If we had selected too few or too many classes during the visual inspection and training data selection, this misjudgment would be reflected in the K-fold cross-validation error. The effect of training set size on the algorithms was not explored in any of the test cases.

1.7.1 Cat

The image of the cat displayed in Figure 1.5 is a three-banded, red-green-blue (RGB) image of size $196 \times 293 \times 3$. Four classes are chosen for the training set as indicated in Figure 1.5: the cat, the black background, the gray background, and the carpet. A total of 181 training pixels are used, as indicated in the right image of Figure 1.5, where the number of training pixels per class, given in order as above, is 32, 54, 35, and 60.

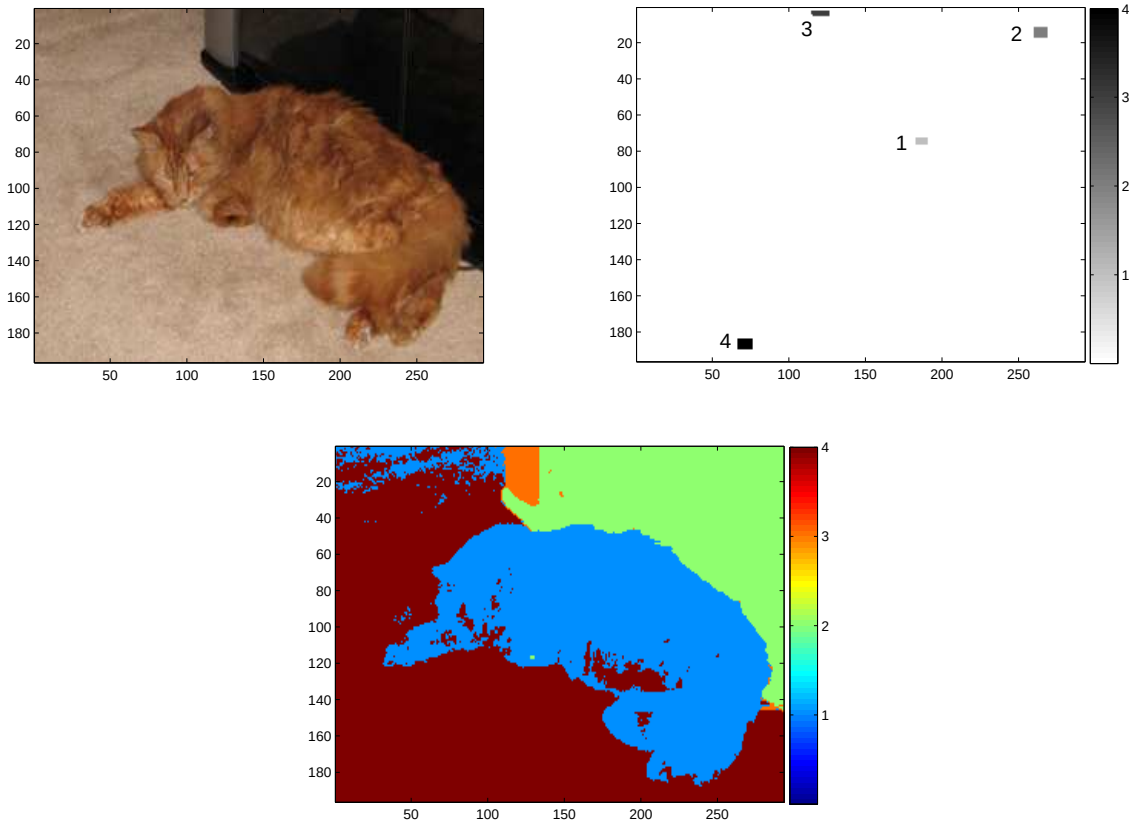

Figure 1.5: Left: A $196 \times 293 \times 3$ image of a cat with four classes. Right: Corresponding training set for the four classes in the cat image indicated in numerical order: the cat, the black background, the gray background, and the carpet. Bottom: The SVM classification of the image.

For trials on the cat image, the radial basis kernel with parameter $\sigma = 0.001$ is used, along

with parameter $C = 10$. The classification scheme is more sensitive to the value of σ than to C. Specifically, a change in σ by an order of magnitude in either direction results in a poor classification of the image, while reducing C by an order of magnitude yields a similar poor classification. Increasing C does not affect the classification.

Classification of the cat image is presented in Figure 1.5. Table 1.1 summarizes the average CPU time and its standard deviations for five trials of the SVM formulation with the optimization techniques presented above. Note that the AL and SMO algorithms classify the image faster than the other two methods, with the unconstrained AL method leading in computational time. The 10-fold cross-validation error for each algorithm is 0.00%, most likely due to the fact that the selected training data were uniform and spatially well-separated. However, note that in the image, not every pixel is correctly classified, e.g. in the upper-left some carpet pixels are classified as cat pixels, which should be physically impossible.

Method	Average Time in Seconds (Standard Deviation)			
	Class 1	Class 2	Class 3	Class 4
AL w/ PN	0.436 (0.005)	0.466 (0.001)	0.431 (0.002)	0.417 (0.001)
AL w/ CG	0.162 (0.024)	0.177 (0.031)	0.166 (0.026)	0.191 (0.031)
SGP	1.558 (0.062)	1.012 (0.007)	0.751 (0.003)	0.895 (0.005)
SMO	0.337 (0.070)	0.439 (0.089)	0.305 (0.061)	0.370 (0.063)
`quadprog`	0.998 (0.011)	0.844 (0.008)	0.926 (0.003)	0.867 (0.005)

Table 1.1: CPU average times and standard deviations over five trials for the optimization method on each class for the SVM classification of the cat image.

1.7.2 Plumeria Flower

The image of the plumeria flower in Figure 1.6 is an RGB image of size $302 \times 369 \times 3$. Six classes are indicated by training data pixels in Figure 1.6: background grass, leaf, stem, bud, flower center and flower petal. There are a total of 300 training data pixels, 50 in each class.

Classification with the bound and equality constrained SVM formulation is shown in Figure

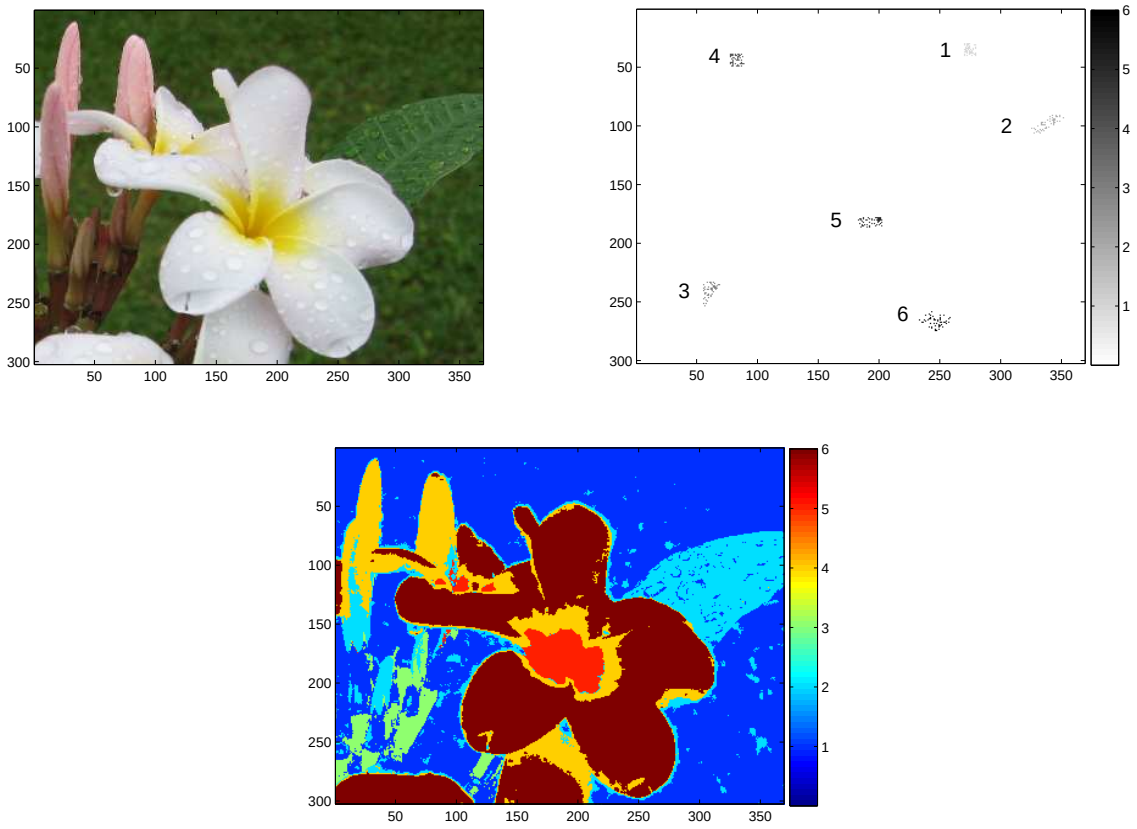

Figure 1.6: Left: A $302 \times 369 \times 3$ image of a plumeria flower. Right: Training data for the six classes in the flower image, indicated in numerical order as background grass, leaf, stem, bud, flower center and flower petal. Bottom: The SVM classification of the image.

1.6. For all SVM classifications, $C = 50$ and a radial basis kernel is used with $\sigma = 0.0005$. Decreasing C by an order of magnitude results in a poor classification while an increase of the same size yields a near identical classification, with classification only changing slightly in the lower left-hand corner of the image. Changing σ by an order of magnitude in either direction results in a poor classification. Thus the classification is somewhat sensitive to the choice of both σ and C.

Results from classification of the flower image are summarized in Table 1.2 with CPU time averaged over five trials, along with corresponding standard deviation. Notice the AL

Method	Average Time in Seconds (Standard Deviation)					
	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
AL w/ PN	0.50 (0.10)	0.45 (0.03)	0.49 (0.01)	0.51 (0.01)	0.49 (0.01)	0.50 (0.01)
AL w/ CG	0.49 (0.13)	0.46 (0.09)	0.41 (0.04)	0.34 (0.03)	0.35 (0.03)	0.35 (0.03)
SGP	1.68 (0.30)	1.07 (0.36)	1.24 (0.48)	2.35 (0.91)	2.18 (0.56)	2.95 (1.27)
SMO	1.14 (0.46)	1.15 (0.49)	1.70 (0.56)	1.61 (0.53)	1.23 (0.44)	1.26 (0.36)
`quadprog`	3.73 (0.40)	7.54 (1.59)	5.20 (0.49)	5.86 (0.69)	5.15 (0.72)	4.58 (0.91)

Table 1.2: CPU average times and standard deviations over five trials for the optimization method on each class for the SVM classification of the flower image.

algorithms outperform the remaining methods in computational time. The 10-fold cross-validation error of the flower image for each algorithm is 0.00%, most likely for the same reasons as the previous example.

1.7.3 Clock Tower

The image of the clock tower in Figure 1.7 is an RGB image of size $995 \times 890 \times 3$. Six classes are indicated by training data pixels in Figure 1.7: sky, green roof, white steeple, gray pole, red brick, black clock background. There are a total of 300 training data pixels, 50 in each class.

Classification with the bound and equality constrained SVM formulation is shown in Figure 1.7. For all SVM classifications, $C = 20$ and a radial basis kernel is used with $\sigma = 0.001$. Decreasing C by an order of magnitude results in a slightly poorer classification while an increase of up to four orders of magnitude yields a near identical classification. Decreasing σ by an order of magnitude or increasing it by two orders of magnitude results in a poor classification. Thus the classification is somewhat sensitive to the choice of σ and less sensitive to the choice of C.

Results from classification of the clock tower image are summarized in Table 1.3 with CPU

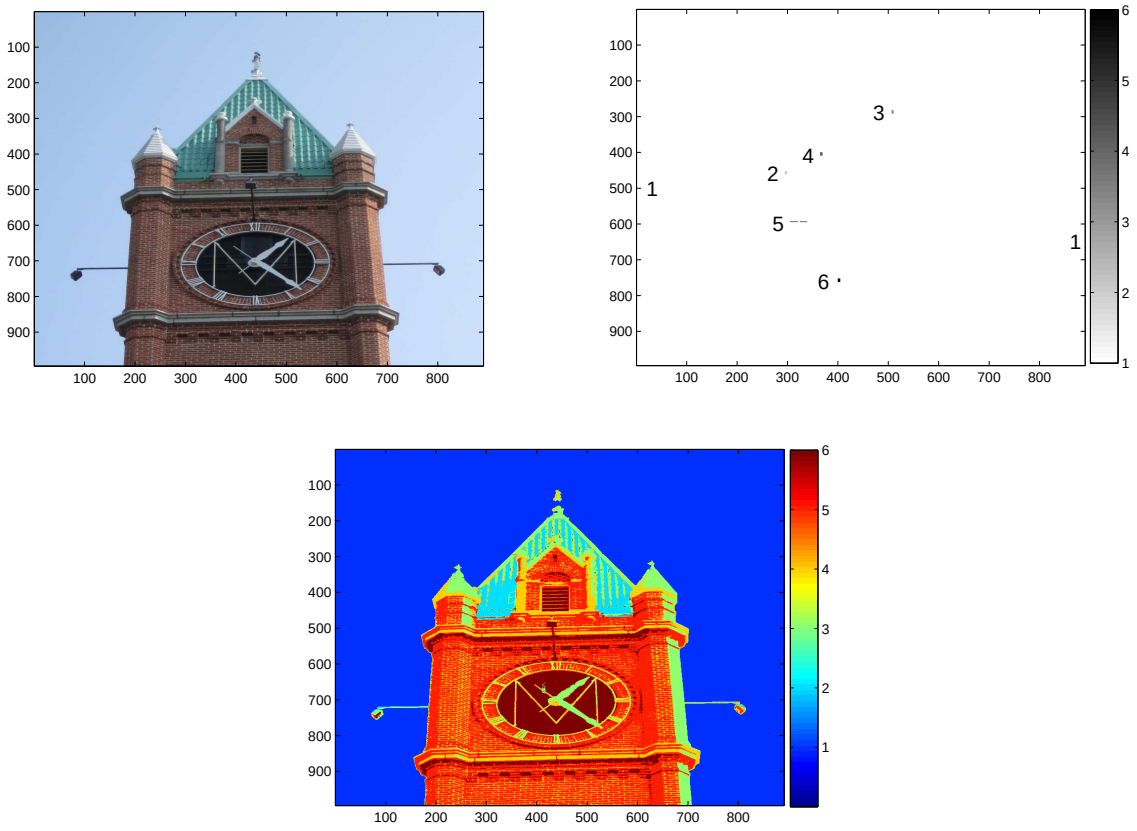

Figure 1.7: Left: A $995 \times 890 \times 3$ image of a clock tower. Right: Training data for the six classes in the clock tower image, indicated in numerical order as sky, green roof, white steeple, gray pole, red brick, black clock background. Bottom: The SVM classification of the image.

time averaged over five trials, along with corresponding standard deviation. Note that, once again, the AL algorithms outperform the remaining methods in computational time. The 10-fold cross-validation error of the clock tower image for each algorithm is 0.00%, most likely for the same reasons as the previous example.

Method	Average Time in Seconds (Standard Deviation)					
	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
AL w/ PN	1.18 (0.60)	0.96 (0.13)	1.00 (0.18)	1.04 (0.09)	0.86 (0.18)	0.81 (0.28)
AL w/ CG	0.71 (0.03)	0.85 (0.01)	0.77 (0.13)	0.54 (0.13)	0.59 (0.11)	0.81 (0.14)
SGP	3.97 (0.92)	4.13 (0.27)	1.73 (0.52)	2.89 (0.46)	2.76 (0.75)	3.04 (0.69)
SMO	1.86 (0.81)	2.16 (0.64)	1.28 (0.49)	1.52 (0.46)	2.47 (0.81)	2.96 (0.69)
`quadprog`	6.83 (1.21)	5.75 (0.28)	5.75 (1.00)	4.93 (0.60)	4.37 (0.34)	4.72 (0.42)

Table 1.3: CPU average times and standard deviations over five trials for the optimization method on each class for the SVM classification of the clock tower image.

1.8 Conclusion

In this chapter we have introduced an unconstrained augmented Lagrangian (AL) technique for numerically solving the bound and equality constrained quadratic program (QP) that arises in support vector machine (SVM) classification problems. We use the term unconstrained because the algorithm requires only the use of the conjugate gradient method or some other unconstrained quadratic minimization solver. We have also presented a bound constrained AL method for solving the SVM QP, which requires the minimization of bound constrained QP subproblems. Finally, we have presented an extension of the scaled gradient projection method of [10] for solving the SVM QP.

We compare these three optimization methods with MATLAB's built-in constrained quadratic solver, `quadprog`, and with the sequential minimal optimization algorithm of [77] on the SVM QP problems arising in three image classification examples. In all three test cases, the unconstrained AL method is the most efficient. Moreover, it is straightforward to implement, suggesting that it will be useful to the wider SVM community.

Chapter 2

Conjugate Gradient Based Kalman Filters for Large-Scale Estimation Problems

The second chapter of this dissertation presents our work with the Kalman filter and discusses how we implement the conjugate gradient method to efficiently solve quadratic minimization tasks and create low-rank, low-storage covariance approximations. This work has been published in [6,7]. My contribution to this work lies in the MATLAB code written for the conjugate gradient and Lanczos methods, as well as writing those sections of the paper.

2.1 Introduction

The Kalman filter was introduced in 1960 by Rudolf E. Kalman [52] as a linear, dynamical method for finding the minimum variance estimator of a temporal variable when provided with observations of a related temporal variable. This statistically optimal method has been

extended for use in nonlinear systems as well, with applications in ocean and weather forecasting [23, 30, 64, 68, 73], autonomous and assisted navigation [5, 19, 27, 59, 76], and fish stock assessment [46, 47, 55].

We begin in the Introduction by briefly describing each of the Kalman filter based methods of interest to us in this chapter: the Kalman filter (KF), the extended Kalman filter (EKF), the variational Kalman filter (VKF), and the ensemble Kalman filter (EnKF). These methods will be discussed in more detail later in the chapter.

2.1.1 The Kalman Filter

The Kalman filter (KF) is a method for solving the following coupled system of discrete, linear, stochastic difference equations,

$$\boldsymbol{x}_k = \mathbf{M}_k \boldsymbol{x}_{k-1} + \boldsymbol{\epsilon}_k^p, \tag{2.1}$$

$$\boldsymbol{y}_k = \mathbf{K}_k \boldsymbol{x}_k + \boldsymbol{\epsilon}_k^o. \tag{2.2}$$

In (2.1), the vector $\boldsymbol{x}_k$ denotes the unknown $n \times 1$ *state* vector of the system at time step k, which is to be estimated by KF. The matrix $\mathbf{M}_k$ is the known $n \times n$ *linear evolution operator*. The $n \times 1$ random vector $\boldsymbol{\epsilon}_k^p$ is the *prediction error*, which comprises stochastic measurement errors, errors in the model, and the corresponding numerical approximations. In (2.2), $\boldsymbol{y}_k$ is the $m \times 1$ *observed data* vector at time step k; the matrix $\mathbf{K}_k$ is the known $m \times n$ *linear observation operator*; and $\boldsymbol{\epsilon}_k^o$ is the $m \times 1$ stochastic *observation error*] vector. We assume that both error terms are independent and normally distributed with zero mean and covariances $\mathbf{C}_{\boldsymbol{\epsilon}_k^p}$ and $\mathbf{C}_{\boldsymbol{\epsilon}_k^o}$, respectively.

The goal of KF is to estimate $\boldsymbol{x}_k$ and its error covariance $\mathbf{C}_k$, denoted $\boldsymbol{x}_k^{est}$ and $\mathbf{C}_k^{est}$, given the evolution and observation operators, observed data vector, error covariance matrices, and estimates for the state and its covariance at the previous time step, $\boldsymbol{x}_{k-1}^{est}$ and $\mathbf{C}_{k-1}^{est}$,

respectively.

Standard implementations of KF require only linear algebra, even in the nonlinear case. However, for large-scale problems, matrix inversion and storage requirements of the full $n \times n$ covariance matrices become too computationally expensive, making standard implementations of KF, as well as its nonlinear extension, infeasible. Oftentimes, approximations can be made to avoid these problems, as will be discussed later.

We will provide a derivation of the Kalman filter below, but first we discuss a nonlinear extension, an equivalent variational method, and an ensemble method for the Kalman filter.

2.1.2 The Extended Kalman Filter

The Kalman filter has been adapted to the nonlinear case, where the linear operators, $\mathbf{M}_k$ of (2.1) and $\mathbf{K}_k$ of (2.2), are replaced with nonlinear functions $\mathcal{M}(\boldsymbol{x}_{k-1})$ and $\mathcal{K}(\boldsymbol{x}_k)$, respectively. The resulting method is known as the extended Kalman filter (EKF) [99], where a linearization of $\mathcal{M}$ and $\mathcal{K}$ is required. The linearization is best computed using adjoint and tangent linear codes, which are model specific and often tedious to develop. Furthermore, like KF for large-scale problems, EKF is computationally prohibitive due to the storage of, and computations with, dense $n \times n$ covariance matrices.

2.1.3 The Variational Kalman Filter

The variational Kalman filter (VKF) is equivalent to KF and results from a sequential application of Bayes' Theorem, whereas KF is derived using minimum variance estimation. Later, we will use both of these approaches to derive the respective filters. VKF also extends to the nonlinear case and is equivalent to EKF; moreover, it is similar to three-dimensional variational data assimilation [61], which is commonly used in weather forecasting. In large-

scale problems, an advantage of VKF over KF is that its formulation suggests the use of an optimization algorithm for finding $\boldsymbol{x}_k^{est}$ and $\mathbf{C}_k^{est}$.

Various methods have been developed for approximating the covariance matrices appearing in KF, VKF and EKF, such as projecting the state space onto a smaller subspace [16, 25, 38, 43, 92, 97]. A downside to this approach is that a chosen subspace is typically fixed in time and cannot capture the temporal dynamics of the system [39]. The iterative method of limited memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) has been used for state estimation and low-storage/full-rank covariance approximation for both regular and variational forms of KF and EKF [2, 3, 96].

Our contribution, which is a focus of this chapter, is to apply the conjugate gradient (CG) method to the quadratic minimization tasks within KF, VKF and EKF, and to use CG iterations to compute low-rank, low-storage approximations of the covariance and inverse-covariance matrices [6]. In one case, we utilize the connection between CG and the Lanczos method to compute the covariance approximations. These low-rank approximations allow for efficient implementation of KF, VKF and EKF, and reduce the storage requirements of the covariance and inverse-covariance matrices. Moreover, we note that the CG covariance approximations change with each iteration of the filter, and hence are better able to capture the temporal dynamics of the system than the projection methods mentioned in the previous paragraph. In our application of CG, we show improvement over the LBFGS implementation of [2, 3].

2.1.4 The Ensemble Kalman Filter

The ensemble Kalman filter (EnKF) is an alternative to EKF, and a full overview of its formulation and practical implementation is given in [34]. Introduced in [33], EnKF is advantageous over EKF in the sense that it does not require the linearization of the nonlinear

evolution operator, nor does it require storage of full-rank covariance matrices.

In EnKF, at each time step, an ensemble of random samples of the state vector $\boldsymbol{x}_k$ is created and the state and covariance estimates are taken to be the empirical mean and covariance of the ensemble, respectively. The ensemble size is typically much less than n, thus greatly decreasing storage requirements. EnKF computes a low-rank, low-storage approximation of the model covariance rather than the full-rank covariance as in EKF. To return the approximation to full-rank, an additional matrix may be added through the technique of *covariance inflation* [72, 100], but we do not do this here.

EnKF is not exempt from problems: sampling errors due to ensemble sample sizes much less than n, underestimation of the covariance due to random perturbations of the model, and ensemble inbreeding are well-documented [60, 66, 82]. Model perturbations are discussed in [14], stating that random errors must be added to the observations to alleviate underestimation of the ensemble covariance, that is, the spread of the finite number of ensemble members always underestimates the mean squared difference between the ensemble mean and the true state [82]. However, [100] discusses that in the case of a finite ensemble size, the errors added to the observations still produce an ensemble which underestimates the covariance; to which end, they suggest building a filter that does not require perturbing the observations. Ensemble inbreeding refers to the fact that we use the same ensemble for analysis as we do to compute the estimate of error [82].

Our contribution to EnKF is similar to that of [90], where the LBFGS method is used to minimize a quadratic function yielding an estimate $\boldsymbol{x}_k^{est}$ of $\boldsymbol{x}_k$ as well as a new ensemble for time step k. In the same fashion, we apply the CG method and utilize the CG sampler of [74] to compute the ensemble, as it is simple and intuitive, requiring minimal added computations [7]. Implementing CG does not invoke the problems associated with random perturbations of the model since the prediction and observation errors are not directly sampled, nor does it require covariance inflation. Our CG implementation within EnKF results in a faster converging

and more accurate filter than classical EnKF or than the implementation using the LBFGS method of [90].

The remainder of this chapter is organized as follows. In Section 2.2, we present each of the four filters mentioned in the Introduction. We then present our CG-based Kalman filters in Section 2.3, along with a description of the CG sampler and an analysis of the CG-based methods. Two commonly used KF test cases are given in Section 2.4 to demonstrate the various CG Kalman filter performances, and we end with conclusions in Section 2.5.

2.2 Deriving the Various Kalman Filters

We begin this section with statistical properties that will become useful later in the derivations of the various Kalman filter methods. We then move on to a discussion of minimum variance estimation, since the classical Kalman filter is derived as a sequential minimum variance estimator, before finally presenting the Kalman filter methods.

2.2.1 Preliminaries

Let $\boldsymbol{x} = (x_1, \ldots, x_n)'$ be a random vector with *mean*

$$E(\boldsymbol{x}) = (E(x_1), \ldots, E(x_n))' = (\mu_1, \ldots, \mu_n)' = \boldsymbol{\mu},$$

and *covariance*

$$[\mathrm{cov}(\boldsymbol{x})]_{ij} = E((x_i - \mu_i)(x_j - \mu_j)), \quad 1 \leq i, j \leq n.$$

Given $\mathbf{A}_{m \times n}$

$$\mathrm{cov}(\mathbf{A}\boldsymbol{x}) = \mathbf{A}\mathrm{cov}(\boldsymbol{x})\mathbf{A}', \tag{2.3}$$

where, typically, $m \leq n$ and $\mathbf{A}$ is full-rank, such that $\mathbf{A}\mathrm{cov}(\boldsymbol{x})\mathbf{A}'$ is positive definite.

The *cross correlation matrix*, $\mathbf{\Gamma}_{\boldsymbol{xy}}$, of random vectors $\boldsymbol{x}_{n\times 1}$ and $\boldsymbol{y}_{m\times 1}$ is defined as

$$\mathbf{\Gamma}_{\boldsymbol{xy}} = E(\boldsymbol{x}\boldsymbol{y}').$$

Note that if $\boldsymbol{x}$ and $\boldsymbol{y}$ are independent with zero means, then $\mathbf{\Gamma}_{\boldsymbol{xy}} = 0$. Furthermore, if $E(\boldsymbol{x}) = \mathbf{0}$, then $\mathbf{\Gamma}_{\boldsymbol{xx}} = \text{cov}(\boldsymbol{x})$.

If we assume $\boldsymbol{x} \sim N(\boldsymbol{\mu}, \mathbf{\Sigma})$, where $E(\boldsymbol{x}) = \boldsymbol{\mu}$ and $\text{cov}(\boldsymbol{x}) = \mathbf{\Sigma}$, the probability density function has the form

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^n \det(\mathbf{\Sigma})}} e^{\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})'\mathbf{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)},$$

where $\det(\mathbf{\Sigma})$ is the determinant of $\mathbf{\Sigma}$.

Finally, we will frequently use the matrix inversion lemma, also known as the Woodbury matrix identity, which states that

$$(\mathbf{S} + \mathbf{UCV})^{-1} = \mathbf{S}^{-1} - \mathbf{S}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{VS}^{-1}\mathbf{U})^{-1}\mathbf{VS}^{-1}, \tag{2.4}$$

where $\mathbf{S}_{n\times n}$, $\mathbf{U}_{n\times m}$, $\mathbf{C}_{m\times m}$ and $\mathbf{V}_{m\times n}$.

2.2.2 Minimum Variance Estimation

Linear models, as previously discussed in Chapter 1, are well-addressed in the statistical literature [78], and have the form

$$\boldsymbol{y} = \mathbf{A}\boldsymbol{x} + \boldsymbol{e}, \tag{2.5}$$

where $\boldsymbol{y}$ is the vector of observed data, $\mathbf{A}$ is the known observation matrix, $\boldsymbol{x}$ is the unknown vector of parameters, and $\boldsymbol{e}$ is typically a zero mean Gaussian random vector, generally referred to as the *error vector*.

Oftentimes, the least squares estimation method is used to solve for the unknown deterministic parameter vector $\boldsymbol{x}$ [42]. If we consider (2.5), the *least squares estimator* is given by

$$\boldsymbol{x}_{ls} = (\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}'\boldsymbol{y}.$$

Here we take a Bayesian approach and assume $\boldsymbol{x} \sim N(\mathbf{0}, \mathbf{C}_{\boldsymbol{x}})$ with $\boldsymbol{x} \perp \boldsymbol{e}$, and use *minimum variance estimation* [91] to solve (2.5) as an extension of least squares.

Definition 2.2.1. *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be random vectors defined on the same probability space, whose components have finite expected squares. Then the* minimum variance estimator *of $\boldsymbol{x}$ from $\boldsymbol{y}$ is*

$$\boldsymbol{x}^{est} = \widehat{\mathbf{B}}\boldsymbol{y}, \tag{2.6}$$

where

$$\widehat{\mathbf{B}} = \arg\min_{\mathbf{B}_{n\times n}} E(||\mathbf{B}\boldsymbol{y} - \boldsymbol{x}||_2^2).$$

We can rewrite the minimum variance estimator using linear algebra, as demonstrated by the following theorem.

Theorem 4. *Let $\boldsymbol{y}$ and $\boldsymbol{x}$ be random vectors defined on the same probability space and whose components have finite expected squares. If $\boldsymbol{\Gamma}_{\boldsymbol{yy}}$ is invertible, then the minimum variance estimator of $\boldsymbol{x}$ from $\boldsymbol{y}$ can be written as*

$$\boldsymbol{x}^{est} = \boldsymbol{\Gamma}_{\boldsymbol{xy}}(\boldsymbol{\Gamma}_{\boldsymbol{yy}})^{-1}\boldsymbol{y}. \tag{2.7}$$

Proof. We begin by rewriting

$$\begin{aligned} E(||\mathbf{B}\boldsymbol{y} - \boldsymbol{x}||_2^2) &= \operatorname{trace}\left(E\left[(\mathbf{B}\boldsymbol{y} - \boldsymbol{x})(\mathbf{B}\boldsymbol{y} - \boldsymbol{x})'\right]\right) \\ &= \operatorname{trace}\left(\mathbf{B}\left[E(\boldsymbol{y}\boldsymbol{y}')\right]\mathbf{B}' - \mathbf{B}\left[E(\boldsymbol{y}\boldsymbol{x}')\right] - E(\boldsymbol{x}\boldsymbol{y}')\mathbf{B}' + E(\boldsymbol{x}\boldsymbol{x}')\right) \\ &= \operatorname{trace}\left(\mathbf{B}\boldsymbol{\Gamma}_{\boldsymbol{yy}}\mathbf{B}'\right) - 2\cdot\operatorname{trace}\left(\mathbf{B}\boldsymbol{\Gamma}_{\boldsymbol{yx}}\right) + \operatorname{trace}\left(\boldsymbol{\Gamma}_{\boldsymbol{xx}}\right). \end{aligned} \tag{2.8}$$

To find the minimum argument of $E(||\mathbf{B}\boldsymbol{y}-\boldsymbol{x}||_2^2)$, we differentiate (2.8) with respect to $\mathbf{B}$, set it equal to $\mathbf{0}$, and solve for $\mathbf{B}$. First it will be beneficial to note a few properties of the trace function, so we define $[\mathbf{B}]_{ij} = b_{ij}$ and $[\mathbf{C}]_{ij} = c_{ij}$ for $1 \leq i, j \leq n$. We can write

$$\operatorname{trace}(\mathbf{BC}) = \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}c_{ji}.$$

Thus

$$\frac{\partial \operatorname{trace}(\mathbf{BC})}{\partial b_{ij}} = c_{ji},$$

which shows that

$$\frac{d(\operatorname{trace}(\mathbf{BC}))}{d\mathbf{B}} = \mathbf{C}'.$$

Similarly,

$$\operatorname{trace}\left(\mathbf{BCB}'\right) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} b_{ij}b_{ik}c_{jk},$$

and hence

$$\begin{aligned}\frac{\partial \operatorname{trace}(\mathbf{BCB}')}{\partial b_{ij}} &= 2b_{ij}c_{jj} + \sum_{k\neq j} b_{ik}(c_{kj}+c_{jk}) \\ &= \left[b_{ij}c_{jj} + \sum_{k\neq j} b_{ik}c_{kj}\right] + \left[b_{ij}c_{jj} + \sum_{k\neq j} b_{ik}c_{jk}\right] \\ &= [\mathbf{BC}]_{ij} + \left[\mathbf{BC}'\right]_{ij},\end{aligned}$$

which shows that

$$\frac{d(\operatorname{trace}(\mathbf{BCB}'))}{d\mathbf{B}} = \mathbf{BC} + \mathbf{BC}'.$$

Applying these properties to the derivative of (2.8) leads to

$$\begin{aligned}\frac{d}{d\mathbf{B}}E(||\mathbf{B}\boldsymbol{y}-\boldsymbol{x}||_2^2) &= \mathbf{B}\boldsymbol{\Gamma}_{\boldsymbol{yy}} + \mathbf{B}(\boldsymbol{\Gamma}_{\boldsymbol{yy}})' - 2(\boldsymbol{\Gamma}_{\boldsymbol{yx}})' \\ &= 2\mathbf{B}\boldsymbol{\Gamma}_{\boldsymbol{yy}} - 2\boldsymbol{\Gamma}_{\boldsymbol{xy}}.\end{aligned}$$

Setting $\frac{d}{d\mathbf{B}}E(||\mathbf{B}\boldsymbol{y}-\boldsymbol{x}||_2^2)=\mathbf{0}$ and solving for $\mathbf{B}$ yields

$$\widehat{\mathbf{B}}=\boldsymbol{\Gamma}_{\boldsymbol{xy}}(\boldsymbol{\Gamma}_{\boldsymbol{yy}})^{-1}. \tag{2.9}$$

Finally, by substituting (2.9) into (2.6), we have (2.7). □

2.2.3 Deriving the Kalman Filter

We now return to the problem of estimating $\boldsymbol{x}_k$ in the discrete, stochastic system at the beginning of the chapter, (2.1) and (2.2). Simply stated, the Kalman filter is the application of minimum variance estimation to the problem of sequentially estimating $\{\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots\}$ in (2.1), given the observed data $\{\boldsymbol{y}_1,\boldsymbol{y}_2,\ldots\}$ in (2.2).

Suppose we have the model as described in (2.1), (2.2), where $\boldsymbol{\epsilon}_k^p\sim N(\mathbf{0},\mathbf{C}_{\boldsymbol{\epsilon}_k^p})$ and $\boldsymbol{\epsilon}_k^o\sim N(\mathbf{0},\mathbf{C}_{\boldsymbol{\epsilon}_k^o})$. We wish to estimate $\boldsymbol{x}_k$ from both $\boldsymbol{y}_k$ and the estimate $\boldsymbol{x}_{k-1}^{est}$, where it is assumed that $\boldsymbol{x}_{k-1}^{est}\sim N(\boldsymbol{x}_{k-1},\mathbf{C}_{k-1}^{est})$. In order to do this, we rewrite (2.1) and (2.2), and then apply minimum variance estimation. We begin by defining

$$\boldsymbol{x}_k^p = \mathbf{M}_k\boldsymbol{x}_{k-1}^{est}, \tag{2.10}$$

$$\boldsymbol{z}_k = \boldsymbol{x}_k-\boldsymbol{x}_k^p, \tag{2.11}$$

$$\boldsymbol{r}_k = \boldsymbol{y}_k-\mathbf{K}_k\boldsymbol{x}_k^p. \tag{2.12}$$

By subtracting (2.10) from (2.1), subtracting $\mathbf{K}_k\boldsymbol{x}_k^p$ from both sides of (2.2), and substituting (2.11) and (2.12) appropriately, we obtain the following coupled, stochastic, linear equations:

$$\boldsymbol{z}_k = \mathbf{M}_k(\boldsymbol{x}_{k-1}-\boldsymbol{x}_{k-1}^{est})+\boldsymbol{\epsilon}_k^p, \tag{2.13}$$

$$\boldsymbol{r}_k = \mathbf{K}_k\boldsymbol{z}_k+\boldsymbol{\epsilon}_k^o. \tag{2.14}$$

The minimum variance estimator of $\boldsymbol{z}_k$ from $\boldsymbol{r}_k$ given (2.13) and (2.14) is then given by Theorem 4 as

$$\boldsymbol{z}_k^{est} = \boldsymbol{\Gamma}_{\boldsymbol{z}_k\boldsymbol{r}_k}(\boldsymbol{\Gamma}_{\boldsymbol{r}_k\boldsymbol{r}_k})^{-1}\boldsymbol{r}_k. \tag{2.15}$$

We assume that random vectors $(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})$ and $\boldsymbol{z}_k$ have zero mean and are independent of $\boldsymbol{\epsilon}_k^p$ and $\boldsymbol{\epsilon}_k^o$, respectively. We can use (2.15) and (2.11) to obtain the KF estimate

$$\boldsymbol{x}_k^{est} = \boldsymbol{x}_k^p + \boldsymbol{\Gamma}_{\boldsymbol{z}_k\boldsymbol{r}_k}(\boldsymbol{\Gamma}_{\boldsymbol{r}_k\boldsymbol{r}_k})^{-1}(\boldsymbol{y}_k - \mathbf{K}_k\boldsymbol{x}_k^p), \tag{2.16}$$

but first we solve for the necessary cross correlation matrices in (2.16).

We note that $\boldsymbol{x}_k$ is not a random variable, so $\mathrm{cov}(\boldsymbol{x}_k) = \mathbf{0}$. Since $\boldsymbol{z}_k$ has zero mean, we begin with

$$\begin{aligned}
\boldsymbol{\Gamma}_{\boldsymbol{z}_k\boldsymbol{z}_k} &= \mathrm{cov}(\boldsymbol{z}_k) \\
&= \mathrm{cov}(\mathbf{M}_k(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est}) + \boldsymbol{\epsilon}_k^p) \\
&= \mathrm{cov}(\mathbf{M}_k(-\boldsymbol{x}_{k-1}^{est})) + \mathrm{cov}(\boldsymbol{\epsilon}_k^p) \\
&= \mathbf{M}_k\mathbf{C}_{k-1}^{est}\mathbf{M}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p} \\
&\stackrel{\mathrm{def}}{=} \mathbf{C}_k^p,
\end{aligned}$$

where, by (2.11), $\mathbf{C}_k^p$ is the covariance matrix of $\boldsymbol{z}_k$, and therefore, the covariance matrix of $\boldsymbol{x}_k^p$.

Since we assumed $\boldsymbol{\epsilon}_k^o \perp \boldsymbol{z}_k$ and $\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est} \perp \boldsymbol{\epsilon}_k^p$, where all sets of variables have zero means, then it follows that $E\left[\boldsymbol{z}_k(\boldsymbol{\epsilon}_k^o)'\right] = \mathbf{0}$ and $E\left[(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})(\boldsymbol{\epsilon}_k^p)'\right] = \mathbf{0}$, so we have

$$\begin{aligned}
\boldsymbol{\Gamma}_{\boldsymbol{z}_k\boldsymbol{r}_k} &= E\left[(\mathbf{M}_k(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est}) + \boldsymbol{\epsilon}_k^p)(\mathbf{K}_k\boldsymbol{z}_k - \boldsymbol{\epsilon}_k^o)'\right] \\
&= E\left[\mathbf{M}_k(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})\boldsymbol{z}_k'\mathbf{K}_k'\right] + E\left[\boldsymbol{\epsilon}_k^p\boldsymbol{z}_k'\mathbf{K}_k'\right] + E\left[\mathbf{M}_k(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est} + \boldsymbol{\epsilon}_k^p)(\boldsymbol{\epsilon}_k^o)'\right]
\end{aligned}$$

$$
\begin{aligned}
&= \mathbf{M}_k E\left[(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})\boldsymbol{z}_k'\right]\mathbf{K}_k' + E\left[\boldsymbol{\epsilon}_k^p \boldsymbol{z}_k'\right]\mathbf{K}_k' + \underbrace{E\left[\boldsymbol{z}_k(\boldsymbol{\epsilon}_k^o)'\right]}_{=0 \text{ by above}} \\
&= \mathbf{M}_k E\left[(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})(\mathbf{M}_k(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est}) + \boldsymbol{\epsilon}_k^p)'\right]\mathbf{K}_k' \\
&\quad + E\left[\boldsymbol{\epsilon}_k^p(\mathbf{M}_k(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est}) + \boldsymbol{\epsilon}_k^p)'\right]\mathbf{K}_k' \\
&= \mathbf{M}_k E\left[(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})'\mathbf{M}_k'\right]\mathbf{K}_k' + \mathbf{M}_k \underbrace{E\left[(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})(\boldsymbol{\epsilon}_k^p)'\right]}_{=0 \text{ by above}}\mathbf{K}_k' \\
&\quad + \underbrace{E\left[\boldsymbol{\epsilon}_k^p(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})'\right]}_{=0 \text{ by above}}\mathbf{M}_k'\mathbf{K}_k' + E[\boldsymbol{\epsilon}_k^p(\boldsymbol{\epsilon}_k^p)']\mathbf{K}_k' \\
&= \mathbf{M}_k E\left[(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})(\boldsymbol{x}_{k-1} - \boldsymbol{x}_{k-1}^{est})'\right]\mathbf{M}_k'\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p}\mathbf{K}_k' \\
&= \mathbf{M}_k \mathbf{C}_{k-1}^{est}\mathbf{M}_k'\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p}\mathbf{K}_k' \\
&= \mathbf{C}_k^p \mathbf{K}_k'.
\end{aligned}
$$

Finally, we have

$$
\begin{aligned}
\boldsymbol{\Gamma}_{\boldsymbol{r}_k\boldsymbol{r}_k} &= E\left[(\mathbf{K}_k\boldsymbol{z}_k + \boldsymbol{\epsilon}_k^o)(\mathbf{K}_k\boldsymbol{z}_k + \boldsymbol{\epsilon}_k^o)'\right] \\
&= \mathbf{K}_k E[\boldsymbol{z}_k\boldsymbol{z}_k']\mathbf{K}_k' + \mathbf{K}_k \underbrace{E[\boldsymbol{z}_k(\boldsymbol{\epsilon}_k^o)']}_{=0} + \underbrace{E[\boldsymbol{\epsilon}_k^o\boldsymbol{z}_k']}_{=0}\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o} \\
&= \mathbf{K}_k \boldsymbol{\Gamma}_{\boldsymbol{z}_k\boldsymbol{z}_k}\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o} \\
&= \mathbf{K}_k \mathbf{C}_k^p \mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o}.
\end{aligned}
$$

Substituting these cross-correlation matrices into (2.16), and defining the *Kalman Gain* matrix as

$$\mathbf{G}_k = \mathbf{C}_k^p\mathbf{K}_k'(\mathbf{K}_k\mathbf{C}_k^p\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}, \tag{2.17}$$

we can now write the Kalman filter estimate of $\boldsymbol{x}_k$ as

$$\boldsymbol{x}_k^{est} = \boldsymbol{x}_k^p + \mathbf{G}_k(\boldsymbol{y}_k - \mathbf{K}_k\boldsymbol{x}_k^p). \tag{2.18}$$

In addition, we wish to have a closed form equation for the covariance of $\boldsymbol{x}_k^{est}$. Using (2.3), we obtain

$$
\begin{aligned}
\mathbf{C}_k^{est} &= \operatorname{cov}(\boldsymbol{x}_k^{est}) \\
&= \operatorname{cov}((\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)\boldsymbol{x}_k^p) + \operatorname{cov}(\mathbf{G}_k\boldsymbol{y}_k) \\
&= (\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)\operatorname{cov}(\boldsymbol{x}_k^p)(\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)' + \mathbf{G}_k\operatorname{cov}(\mathbf{K}_k\boldsymbol{x}_k+\boldsymbol{\epsilon}_k^o)\mathbf{G}_k' \\
&= (\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)\mathbf{C}_k^p(\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)' + \mathbf{G}_k\mathbf{K}_k\underbrace{\operatorname{cov}(\boldsymbol{x}_k)}_{=0}\mathbf{K}_k'\mathbf{G}_k' + \mathbf{G}_k\operatorname{cov}(\boldsymbol{\epsilon}_k^o)\mathbf{G}_k' \\
&= (\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)\mathbf{C}_k^p(\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)' + \mathbf{G}_k\mathbf{C}_{\boldsymbol{\epsilon}_k^o}\mathbf{G}_k' \\
&\quad + \underbrace{\mathbf{G}_k\mathbf{K}_k\mathbf{C}_k^p - \mathbf{G}_k(\mathbf{K}_k\mathbf{C}_k^p\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})(\mathbf{K}_k\mathbf{C}_k^p\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}\mathbf{K}_k\mathbf{C}_k^p}_{\text{adding zero}} \\
&= (\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)\mathbf{C}_k^p(\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)' + \mathbf{G}_k\mathbf{C}_{\boldsymbol{\epsilon}_k^o}\mathbf{G}_k' + \mathbf{G}_k\mathbf{K}_k\mathbf{C}_k^p \\
&\quad -\mathbf{G}_k(\mathbf{K}_k\mathbf{C}_k^p\mathbf{K}_k')(\mathbf{K}_k\mathbf{C}_k^p\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}\mathbf{K}_k\mathbf{C}_k^p - \mathbf{G}_k\mathbf{C}_{\boldsymbol{\epsilon}_k^o}(\mathbf{K}_k\mathbf{C}_k^p\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}\mathbf{K}_k\mathbf{C}_k^p \\
&= (\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)\mathbf{C}_k^p(\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)' + \mathbf{G}_k\mathbf{C}_{\boldsymbol{\epsilon}_k^o}\mathbf{G}_k' + \mathbf{G}_k\mathbf{K}_k\mathbf{C}_k^p \\
&\quad -\mathbf{G}_k\mathbf{K}_k\mathbf{C}_k^p\mathbf{K}_k'\mathbf{G}_k' - \mathbf{G}_k\mathbf{C}_{\boldsymbol{\epsilon}_k^o}\mathbf{G}_k' \qquad \text{(by definition of } \mathbf{G}_k\text{)} \\
&= (\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)\mathbf{C}_k^p(\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)' + \mathbf{G}_k\mathbf{K}_k\mathbf{C}_k^p - \mathbf{G}_k\mathbf{K}_k\mathbf{C}_k^p\mathbf{K}_k'\mathbf{G}_k' \qquad \text{(terms canceled)} \\
&= (\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)\mathbf{C}_k^p(\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)' + \mathbf{G}_k\mathbf{K}_k\mathbf{C}_k^p(\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)' \\
&= \left[(\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)\mathbf{C}_k^p + \mathbf{G}_k\mathbf{K}_k\mathbf{C}_k^p\right](\mathbf{I}-\mathbf{G}_k\mathbf{K}_k)' \\
&= \mathbf{C}_k^p - \mathbf{C}_k^p\mathbf{K}_k'\mathbf{G}_k'.
\end{aligned}
$$

Since $\mathbf{C}_k^{est}$ is a symmetric covariance matrix, we can write $\mathbf{C}_k^{est} = \mathbf{C}_k^p - \mathbf{G}_k\mathbf{K}_k\mathbf{C}_k^p$.

With both the Kalman estimate and covariance in hand, we now present the Kalman filter method in Algorithm 7.

Algorithm 7 The Kalman Filter (KF)

1. Select initial guesses for $\boldsymbol{x}_0^{est}$ and $\mathbf{C}_0^{est}$, and set $k = 1$.
2. Compute the evolution model estimate and covariance:
   (a) Compute $\boldsymbol{x}_k^p = \mathbf{M}_k \boldsymbol{x}_{k-1}^{est}$.
   (b) Define $\mathbf{C}_k^p = \mathbf{M}_k \mathbf{C}_{k-1}^{est} \mathbf{M}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p}$.
3. Compute the Kalman filter and covariance estimates:
   (a) Define the Kalman Gain $\mathbf{G}_k = \mathbf{C}_k^p \mathbf{K}_k' (\mathbf{K}_k \mathbf{C}_k^p \mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}$.
   (b) Compute the Kalman filter estimate $\boldsymbol{x}_k^{est} = \boldsymbol{x}_k^p + \mathbf{G}_k(\boldsymbol{y}_k - \mathbf{K}_k \boldsymbol{x}_k^p)$.
   (c) Define the estimate covariance $\mathbf{C}_k^{est} = \mathbf{C}_k^p - \mathbf{G}_k \mathbf{K}_k \mathbf{C}_k^p$.
4. Set $k = k + 1$ and return to Step 2.

2.2.4 Deriving The Extended Kalman Filter

As was discussed in the introduction, in the presence of nonlinearity, KF has been adapted to the case where (2.1) and (2.2) are replaced by

$$\boldsymbol{x}_k = \mathcal{M}(\boldsymbol{x}_{k-1}) + \boldsymbol{\epsilon}_k^p, \tag{2.19}$$

$$\boldsymbol{y}_k = \mathcal{K}(\boldsymbol{x}_k) + \boldsymbol{\epsilon}_k^o, \tag{2.20}$$

where the possibly nonlinear functions $\mathcal{M}$ and $\mathcal{K}$ are the evolution and observation operators, respectively. The extended Kalman filter (EKF) is the best known extension of KF to (2.19) and (2.20). It has the same form as Algorithm 7, but with Step 2(a) replaced by the nonlinear model forward integration calculation $\boldsymbol{x}_k^p = \mathcal{M}(\boldsymbol{x}_{k-1}^{est})$, and elsewhere using the linear approximations of $\mathcal{M}$ and $\mathcal{K}$,

$$\mathbf{M}_k = \frac{\partial \mathcal{M}(\boldsymbol{x}_{k-1}^{est})}{\partial \boldsymbol{x}} \quad \text{and} \quad \mathbf{K}_k = \frac{\partial \mathcal{K}(\boldsymbol{x}_k^p)}{\partial \boldsymbol{x}}, \tag{2.21}$$

where $\partial / \partial \boldsymbol{x}$ denotes the Jacobian of the function with respect to $\boldsymbol{x}$. The EKF method is given in Algorithm 8.

Algorithm 8 The Extended Kalman Filter (EKF)

1. Select initial guesses for $\boldsymbol{x}_0^{est}$ and C_0^{est}, and set $k = 1$.
2. Compute the evolution model estimate and covariance:
   (a) Compute $\boldsymbol{x}_k^p = \mathcal{M}(\boldsymbol{x}_{k-1}^{est})$.
   (b) Define $\mathbf{M}_k = \partial\mathcal{M}(\boldsymbol{x}_{k-1}^{est})/\partial\boldsymbol{x}$ and $\mathbf{C}_k^p = \mathbf{M}_k\mathbf{C}_{k-1}^{est}\mathbf{M}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p}$.
3. Compute the Kalman filter and covariance estimates:
   (a) Define the Kalman Gain $\mathbf{G}_k = \mathbf{C}_k^p\mathbf{K}_k'(\mathbf{K}_k\mathbf{C}_k^p\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}$, where $\mathbf{K}_k = \partial\mathcal{K}(\boldsymbol{x}_k^p)/\partial\boldsymbol{x}$.
   (b) Compute the Kalman filter estimate $\boldsymbol{x}_k^{est} = \boldsymbol{x}_k^p + \mathbf{G}_k(\boldsymbol{y}_k - \mathbf{K}_k\boldsymbol{x}_k^p)$.
   (c) Define the estimate covariance $\mathbf{C}_k^{est} = \mathbf{C}_k^p - \mathbf{G}_k\mathbf{K}_k\mathbf{C}_k^p$.
4. Set $k = k + 1$ and return to Step 2.

While the use of finite differences is one common approach to computing the linearizations of $\mathcal{M}$ and $\mathcal{K}$ in (2.21), using adjoint and tangent linear codes is more accurate, and is more efficient computationally and in terms of storage. The adjoint and tangent linear codes are model dependent and are difficult to write in many instances, though they are available in many highly-used models such as weather forecasting [26]. Specifically, for the evolution and observation operators, the adjoint code computes multiplication of a vector by $\mathbf{M}_k'$ and $\mathbf{K}_k'$, and the tangent linear code computes multiplication of a vector by $\mathbf{M}_k$ and $\mathbf{K}_k$. The linearization may be infeasible or computationally problematic in the case of large-scale EKF problems, and numerical approximations may yield inaccuracies.

2.2.5 The Variational Kalman Filter

An alternative derivation of KF yields the variational Kalman filter (VKF), which may be formulated for both linear and nonlinear models. While, as we will see, VKF is equivalent to KF (and EKF in the nonlinear case), it has advantages in large-scale problems, where the estimates and covariances can be approximated with iterative methods. However, VKF may still have computational difficulties in large-scale problems due to the storage of, and

multiplication by, dense $n \times n$ covariance matrices.

VKF follows from using Bayes' Theorem rather than minimum variance estimation. From Bayes' formula we can write the posterior density $p(\boldsymbol{x}|\boldsymbol{y})$ as

$$p(\boldsymbol{x}_k|\boldsymbol{y}_k) \propto p(\boldsymbol{y}_k|\boldsymbol{x}_k)p(\boldsymbol{x}_k), \tag{2.22}$$

where $\boldsymbol{x}_k$ is the unknown state vector, $\boldsymbol{y}_k$ is the vector of observation measurements, $p(\boldsymbol{x}_k)$ denotes the prior density and $p(\boldsymbol{y}_k|\boldsymbol{x}_k)$ is the density of $\boldsymbol{y}_k$ given $\boldsymbol{x}_k$. We wish to maximize (2.22), yielding the maximum a posteriori (MAP) estimate of $\boldsymbol{x}_k$. This is equivalent to minimizing the negative log of $p(\boldsymbol{x}_k|\boldsymbol{y}_k)$,

$$\ell(\boldsymbol{x}_k|\boldsymbol{y}_k) = -\log p(\boldsymbol{y}_k|\boldsymbol{x}_k) - \log p(\boldsymbol{x}_k).$$

The prior density is given by (2.1) or (2.19) and has the form $\boldsymbol{x}_k \sim N(\boldsymbol{x}_k^p, \mathbf{C}_k^p)$. From the linear model in (2.2) or (2.20), $\boldsymbol{y}_k|\boldsymbol{x}_k \sim N(\mathbf{K}_k\boldsymbol{x}_k, \mathbf{C}_{\boldsymbol{\epsilon}_k^o})$, where $\mathbf{K}_k$ is the linearization of $\mathcal{K}$ in the nonlinear case. Hence we have

$$\ell(\boldsymbol{x}_k|\boldsymbol{y}_k) = \frac{1}{2}(\boldsymbol{y}_k - \mathbf{K}_k\boldsymbol{x}_k)'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}(\boldsymbol{y}_k - \mathbf{K}_k\boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x}_k - \boldsymbol{x}_k^p)'(\mathbf{C}_k^p)^{-1}(\boldsymbol{x}_k - \boldsymbol{x}_k^p), \tag{2.23}$$

where the normalizing terms (non-$\boldsymbol{x}_k$ terms) for $p(\boldsymbol{x}_k|\boldsymbol{y}_k)$ are dropped because they do not affect the minimization. We obtain the variational Kalman filter estimate by taking the gradient of (2.23), setting it equal to $\mathbf{0}$,

$$\begin{aligned}
\mathbf{0} &= \nabla\ell(\boldsymbol{x}_k|\boldsymbol{y}_k) \\
&= -\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}(\boldsymbol{y}_k - \mathbf{K}_k\boldsymbol{x}_k) + (\mathbf{C}_k^p)^{-1}(\boldsymbol{x}_k - \boldsymbol{x}_k^p) \\
&= -\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\boldsymbol{y}_k + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\boldsymbol{x}_k + (\mathbf{C}_k^p)^{-1}\boldsymbol{x}_k - (\mathbf{C}_k^p)^{-1}\boldsymbol{x}_k^p;
\end{aligned}$$

then solving for $\boldsymbol{x}_k$:

$$
\begin{aligned}
\boldsymbol{x}_k^{est} &= (\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1})^{-1}(\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\boldsymbol{y}_k + (\mathbf{C}_k^p)^{-1}\boldsymbol{x}_k^p) \\
&= (\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1})^{-1}\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\boldsymbol{y}_k + (\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1})^{-1}(\mathbf{C}_k^p)^{-1}\boldsymbol{x}_k^p \\
&\underbrace{+(\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1})^{-1}\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\boldsymbol{x}_k^p - (\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1})^{-1}\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\boldsymbol{x}_k^p}_{\text{adding zero}} \\
&= (\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}K_k + (\mathbf{C}_k^p)^{-1})^{-1}\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\boldsymbol{y}_k - (\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1})^{-1}\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\boldsymbol{x}_k^p \\
&+ (\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1})^{-1}(\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k' + (\mathbf{C}_k^p)^{-1})\boldsymbol{x}_k^p \\
&= \boldsymbol{x}_k^p + (\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1})^{-1}\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}(\boldsymbol{y}_k - \mathbf{K}_k\boldsymbol{x}_k^p) \qquad (2.24) \\
&= \boldsymbol{x}_k^p - [\nabla^2\ell(\boldsymbol{x}_k^p|\boldsymbol{y}_k)]^{-1}\nabla\ell(\boldsymbol{x}_k^p|\boldsymbol{y}_k),
\end{aligned}
$$

where '∇' and '∇^2' denote the gradient and Hessian operators, respectively.

To see that (2.24) is the same as the KF state estimate (2.18), note that

$$
\begin{aligned}
&\left((\mathbf{C}_k^p)^{-1} + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\right)^{-1}\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} \\
&\qquad = \mathbf{C}_k^p\left[\mathbf{I} - \mathbf{I} + (\mathbf{C}_k^p)^{-1}((\mathbf{C}_k^p)^{-1} + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k)^{-1}\right]\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} \\
&\qquad = \mathbf{C}_k^p\left[\mathbf{I} - \underbrace{\left((\mathbf{C}_k^p)^{-1} + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\right)\left((\mathbf{C}_k^p)^{-1} + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\right)^{-1}}_{\text{the identity}}\right]\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} \\
&\qquad\quad + \mathbf{C}_k^p\left[(\mathbf{C}_k^p)^{-1}\left((\mathbf{C}_k^p)^{-1} + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\right)^{-1}\right]\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} \\
&\qquad = \mathbf{C}_k^p\left[\mathbf{I} - \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\left((\mathbf{C}_k^p)^{-1} + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\right)^{-1}\right]\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} \qquad \text{(terms canceled)} \\
&\qquad\quad \left.\begin{aligned}
&+\mathbf{C}_k^p\left[(\mathbf{C}_k^p)^{-1}\left((\mathbf{C}_k^p)^{-1} + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}K_k\right)^{-1}\right]\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} \\
&-\mathbf{C}_k^p\left[(\mathbf{C}_k^p)^{-1}\left((\mathbf{C}_k^p)^{-1} + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}K_k\right)^{-1}\right]\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}
\end{aligned}\right\}\text{adding zero} \\
&\qquad = \left[\mathbf{C}_k^p\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} - \mathbf{C}_k^p\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\left((\mathbf{C}_k^p)^{-1} + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\right)^{-1}\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\right] \\
&\qquad = \mathbf{C}_k^p\mathbf{K}_k'\left[\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} - \mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\left((\mathbf{C}_k^p)^{-1} + \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k\right)^{-1}\mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\right]
\end{aligned}
$$

$$= \mathbf{G}_k,$$

where the last equality follows from the matrix inversion lemma (2.4) and the Kalman Gain $\mathbf{G}_k$ is given in (2.17).

One further useful fact comes from using the matrix inversion lemma (2.4), yet again:

$$\begin{aligned}
\mathbf{C}_k^{est} &= \mathbf{C}_k^p - \mathbf{C}_k^p \mathbf{K}_k' (\mathbf{K}_k \mathbf{C}_k^p \mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1} \mathbf{K}_k \mathbf{C}_k^p, \\
&= (\mathbf{K}_k' \mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} \mathbf{K}_k + (\mathbf{C}_k^p)^{-1})^{-1} \\
&= \left[\nabla^2 \ell(\boldsymbol{x}_k)\right]^{-1}.
\end{aligned}$$

Finally, to avoid repetition, we present the VKF method for the nonlinear model case (2.19), (2.20) in Algorithm 9. In the linear case, the only change is that $\mathcal{M}(\boldsymbol{x}_{k-1}^{est})$ is replaced by $\mathbf{M}_k \boldsymbol{x}_{k-1}^{est}$, and the linearizations in Steps 2(b) and 3 are not needed.

Algorithm 9 The Variational Kalman Filter (VKF)

1. Select initial guesses for $\boldsymbol{x}_0^{est}$ and $\mathbf{C}_0^{est}$, and set $k = 1$.
2. Compute the evolution model estimate and covariance:
   (a) Compute $\boldsymbol{x}_k^p = \mathcal{M}(\boldsymbol{x}_{k-1}^{est})$.
   (b) Define $\mathbf{M}_k = \partial \mathcal{M}(\boldsymbol{x}_{k-1}^{est})/\partial \boldsymbol{x}$ and $\mathbf{C}_k^p = \mathbf{M}_k \mathbf{C}_{k-1}^{est} \mathbf{M}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p}$.
3. Compute the Kalman filter and covariance estimates:

   Compute the minimizer $\boldsymbol{x}_k^{est}$ and inverse Hessian $\mathbf{C}_k^{est}$ of

   $$\ell(\boldsymbol{x}|\boldsymbol{y}_k) = \tfrac{1}{2}(\boldsymbol{y}_k - \mathbf{K}_k \boldsymbol{x})' \mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} (\boldsymbol{y}_k - \mathbf{K}_k \boldsymbol{x}) + \tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k^p)' (\mathbf{C}_k^p)^{-1} (\boldsymbol{x} - \boldsymbol{x}_k^p),$$

   where $\mathbf{K}_k = \partial \mathcal{K}(\boldsymbol{x}_k^p)/\partial \boldsymbol{x}$.
4. Set $k = k + 1$ and return to Step 2.

2.2.6 The Ensemble Kalman Filter

The ensemble Kalman filter (EnKF) was proposed by Geir Evensen in [33] as an alternative to EKF for nonlinear filtering problems of the form (2.19) and (2.20). Unlike EKF, EnKF does not require storage of full covariance matrices nor linearization of the nonlinear function $\mathcal{M}$, and therefore is advantageous for many large-scale, nonlinear problems. Instead, EnKF generates a random sample of the state, called an *ensemble*, at each iteration of the filter, from which the Kalman state and covariance estimate are calculated as the empirical mean and covariance of the ensemble, respectively. However, if $\mathcal{K}$ is nonlinear, then the linearization of $\mathcal{K}$ will be required as previously discussed. The pseudocode for EnKF is provided in Algorithm 10.

Computations for EnKF remain efficient when the covariance $\mathbf{C}_k^p$ and the Kalman Gain matrix are kept in a low-rank 'ensemble form' rather than being explicitly computed at each iteration [35]. The low-rank covariance matrix, however, may need to be regularized using the technique of covariance inflation [72, 100]. Furthermore, the model is not restricted to using Gaussian distributed noise, even though it is most common to do so [34].

It is worth mentioning that a slight error in the initial guess for the ensemble error does not highly influence the results of the filter [34]. The ensemble size, N, needs to be chosen large enough so that the estimator in Step 3(c) is accurate; otherwise the method may perform poorly [34]. When the state vector dimension n is large, this likely means that N must be large as well, which will limit the computational advantage of EnKF. Further inaccuracies may be fostered by random perturbations of the model states in Steps 2(a) and 3(b).

Now that we have finished introductions to, and derivations of, various Kalman filter methods, we move on to discuss our implementation of the conjugate gradient algorithm within these filters.

Algorithm 10 The Ensemble Kalman Filter (EnKF)

1. Select initial guesses for $\boldsymbol{x}_0^{est}$ and $\mathbf{C}_0^{est}$. Sample initial ensemble $\boldsymbol{x}_{0,i}^{est} \sim N(\boldsymbol{x}_0^{est}, \mathbf{C}_0^{est})$ for $i = 1, \ldots, N$, and set $k = 1$.

2. Integrate the ensemble forward in time and compute the evolution model estimate and covariance:

   (a) Sample $\boldsymbol{\epsilon}_{k,i}^p \sim N(\mathbf{0}, \mathbf{C}_{\boldsymbol{\epsilon}_k^p})$, for $i = 1, \ldots, N$ and compute ensemble members $\boldsymbol{x}_{k,i}^p = \mathcal{M}(\boldsymbol{x}_{k-1,i}^{est}) + \boldsymbol{\epsilon}_{k,i}^p$.

   (b) Set $\boldsymbol{x}_k^p = \mathcal{M}(\boldsymbol{x}_{k-1}^{est})$ and compute the model covariance estimate

$$\mathbf{C}_k^p = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_{k,i}^p - \boldsymbol{x}_k^p)(\boldsymbol{x}_{k,i}^p - \boldsymbol{x}_k^p)'.$$

3. Compute a new ensemble using the Kalman filter formulas:

   (a) Define the Kalman Gain $\mathbf{G}_k = \mathbf{C}_k^p\mathbf{K}_k'(\mathbf{K}_k\mathbf{C}_k^p\mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}$, where $\mathbf{K}_k = \partial\mathcal{K}(\boldsymbol{x}_k^p)/\partial\boldsymbol{x}$.

   (b) Sample $\boldsymbol{\epsilon}_{k,i}^o \sim N(\mathbf{0}, \mathbf{C}_{\boldsymbol{\epsilon}_k^o})$, for $i = 1, \ldots, N$ and then compute ensemble members $\boldsymbol{x}_{k,i}^{est} = \boldsymbol{x}_{k,i}^p + \mathbf{G}_k(\boldsymbol{y}_k - \mathbf{K}_k\boldsymbol{x}_{k,i}^p + \boldsymbol{\epsilon}_{k,i}^o)$.

   (c) Compute the state estimate as the mean of the ensemble $\boldsymbol{x}_k^{est} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_{k,i}^{est}$ and the covariance estimate as the empirical covariance

$$\mathbf{C}_k^{est} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_{k,i}^{est} - \boldsymbol{x}_k^{est})(\boldsymbol{x}_{k,i}^{est} - \boldsymbol{x}_k^{est})'.$$

4. Set $k = k + 1$ and return to Step 2.

2.3 Conjugate Gradient Based Kalman Filters

As previously discussed in the case of large-scale problems, KF, VKF and EKF are often computationally prohibitive due to the storage and matrix inversion of the large, non-sparse covariance matrices computed at every filter step. Additionally, EnKF incurs inaccuracies due to perturbations of the measurements and may require covariance inflation. Schneider and Willsky discuss the problems associated with using the conjugate gradient (CG) method to build low-rank covariance and inverse-covariance matrix approximations in [85, 86]. These authors suggest the use of CG within KF in [84] but do not discuss details of its implementation.

We note that the CG-Lanczos connection is used in [93] to build preconditioners for CG iterations within the four-dimensional variational data assimilation algorithm. LBFGS [71] has also been used within the various forms of KF iterations to solve the necessary quadratic minimization tasks, as well as to obtain low-storage, full-rank (low-rank + identity) approximations of covariance matrices [2, 3].

Here we introduce the use of CG within the previously discussed Kalman filters, for both quadratic minimization tasks and low-rank, low-storage covariance and inverse-covariance matrix approximations. We begin with a discussion of how CG and its connection to Lanczos may be used to determine some of the matrix approximations, followed by a discussion of the implementation of CG and Lanczos within KF and VKF. We end with a discussion of the CG sampler [74], which is used to efficiently sample from a Gaussian distribution to create ensembles in the EnKF method.

2.3.1 The Conjugate Gradient and Lanczos Methods

CG is most widely known for solving a quadratic minimization problem of the form

$$\phi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}'\mathbf{A}\boldsymbol{x} - \boldsymbol{x}'\boldsymbol{b}, \tag{2.25}$$

where $\mathbf{A}_{n\times n}$ is symmetric and positive definite. It is mentioned in Chapter 1 that the minimizer of (2.25) is also the solution of $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$. We again give the pseudocode for CG, for completeness, in Algorithm 11.

Algorithm 11 Conjugate Gradient (CG)

Given $\mathbf{A}$, $\boldsymbol{b}$ and $\boldsymbol{x}_0$, let $\boldsymbol{r}_0 = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_0$, $\boldsymbol{p}_0 = \boldsymbol{r}_0$, and $j = 1$. Specify stopping tolerance ϵ and iterate:

1. $\gamma_{j-1} = \frac{\boldsymbol{r}'_{j-1}\boldsymbol{r}_{j-1}}{\boldsymbol{p}'_{j-1}\mathbf{A}\boldsymbol{p}_{j-1}}$ is the 1-D minimizer of ϕ in the direction $\boldsymbol{x}_{j-1} + \gamma\boldsymbol{p}_{j-1}$.
2. $\boldsymbol{x}_j = \boldsymbol{x}_{j-1} + \gamma_{j-1}\boldsymbol{p}_{j-1}$.
3. $\boldsymbol{r}_j = -\nabla_{\boldsymbol{x}}\phi(\boldsymbol{x}_j) = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_j = \boldsymbol{r}_{j-1} - \gamma_{j-1}\mathbf{A}\boldsymbol{p}_{j-1}$ is the residual.
4. $\beta_j = -\frac{\boldsymbol{r}'_j\boldsymbol{r}_j}{\boldsymbol{r}'_{j-1}\boldsymbol{r}_{j-1}}$.
5. $\boldsymbol{p}_j = \boldsymbol{r}_j - \beta_j\boldsymbol{p}_{j-1}$ is the next conjugate search direction.
6. Quit if $||\boldsymbol{r}_j|| < \epsilon$. Else set $j = j + 1$ and return to Step 1.

In addition to using CG for optimization problems, we use it here to efficiently build a q-rank approximation $\mathbf{B}_{k,q} \approx \mathbf{A}^{-1}$, where $q < n$ indicates the stopping iteration of the CG algorithm and k indicates the time step of the Kalman filter. Afterwards we will show how to construct the low-rank (q-rank) approximation $\mathbf{B}^{\dagger}_{k,q} \approx \mathbf{A}$ by exploiting the connection between the CG and Lanczos iterations.

We begin by determining the form of $\mathbf{B}_{k,q}$ from the CG iterations in Algorithm 11. Define $\mathbf{P}_q$ to be the $n \times q$ matrix with $\{\boldsymbol{p}_i\}_{i=0}^{q-1}$ as columns and $\mathbf{P}_B$ to be the $n \times (n-q)$ matrix with $\{\boldsymbol{p}_i\}_{i=q}^{n-1}$ as columns. Then the theory of CG [45] states that in exact arithmetic, when $\mathbf{A}$ has

n distinct eigenvalues,

$$\mathbf{D}_n = \begin{pmatrix} \mathbf{D}_q & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_B \end{pmatrix} = \begin{pmatrix} \mathbf{P}_q'\mathbf{A}\mathbf{P}_q & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_B'\mathbf{A}\mathbf{P}_B \end{pmatrix} = \mathbf{P}_n'\mathbf{A}\mathbf{P}_n$$

is an invertible diagonal matrix with entries $[\mathbf{D}_n]_{ii} = \boldsymbol{p}_i'\mathbf{A}\boldsymbol{p}_i$. Thus we can write the inverse of $\mathbf{A}$ as

$$\mathbf{A}^{-1} = \mathbf{P}_n\mathbf{D}_n^{-1}\mathbf{P}_n' = \mathbf{P}_q\mathbf{D}_q^{-1}\mathbf{P}_q' + \mathbf{P}_B\mathbf{D}_B^{-1}\mathbf{P}_B'. \tag{2.26}$$

For $q < n$, the $q-$rank approximation of $\mathbf{A}^{-1}$ given by the CG algorithm at filter time step k is

$$\begin{aligned} \mathbf{B}_{k,q} &= \mathbf{P}_q\mathbf{D}_q^{-1}\mathbf{P}_q' \\ &= (\mathbf{P}_q\mathbf{D}_q^{-1/2})(\mathbf{P}_q\mathbf{D}_q^{-1/2})' \\ &\stackrel{\text{def}}{=} \mathbf{X}_{k,q}\mathbf{X}_{k,q}'. \end{aligned} \tag{2.27}$$

Storing $\mathbf{X}_{k,q}$ of size $n \times q$, when $q < n$, reduces the storage requirements of the covariance matrix. Thus we now have a way of determining a low-rank, low-storage approximation of $\mathbf{A}^{-1}$; namely $\mathbf{B}_{k,q} = \mathbf{X}_{k,q}\mathbf{X}_{k,q}'$.

Next we consider the connection between Lanczos and CG, which we will use to determine an approximation $\mathbf{B}_{k,q}^{\dagger}$ of $\mathbf{A}$.

Lanczos

The Lanczos algorithm is an iterative method for determining the singular value decomposition of a rectangular matrix, specifically in the case of large, sparse matrices [56, 57], and its performance in finite precision is well-studied [67, 75, 81]. The CG and Lanczos methods are equivalent for symmetric, positive definite matrices, i.e. the iterations of one can be used to obtain the other and vice versa [4, 45, 67].

In exact arithmetic, the Lanczos algorithm approximates the eigenvalue/eigenvector pairs $(\lambda_i, \boldsymbol{w}_i)$ of positive definite matrix $\mathbf{A}_{n\times n}$ [74], so that $\mathbf{A}\boldsymbol{w}_i = \lambda_i \boldsymbol{w}_i$, where the eigenvalues of $\mathbf{A}$ are ordered by

$$\lambda_1 < \lambda_2 < \cdots < \lambda_n.$$

In finite precision, Lanczos finds only a few eigenvalues of $\mathbf{A}$, as well as the extreme and well-separated ones.

For completeness, we present the two-term recurrence version of the Lanczos method due to Paige [67] in Algorithm 12.

Algorithm 12 Lanczos

Given an initial vector $\tilde{\boldsymbol{v}}_0$, let $\boldsymbol{v}_0 = \frac{\tilde{\boldsymbol{v}}_0}{||\tilde{\boldsymbol{v}}_0||}$, $\alpha_0 = \boldsymbol{v}_0'\mathbf{A}\boldsymbol{v}_0$, $\tilde{\boldsymbol{v}}_1 = \mathbf{A}\boldsymbol{v}_0 - \alpha_0\boldsymbol{v}_0$, and $k = 1$. Specify some stopping tolerance ϵ. Iterate:

1. $\eta_k = ||\tilde{\boldsymbol{v}}_k||$. Quit if $\eta_k < \epsilon$.
2. $\boldsymbol{v}_k = \frac{\tilde{\boldsymbol{v}}_k}{\eta_k}$ is a Lanczos vector.
3. $\boldsymbol{u}_k = \mathbf{A}\boldsymbol{v}_k - \eta_k\boldsymbol{v}_{k-1}$.
4. $\alpha_k = \boldsymbol{v}_k'\boldsymbol{u}_k$.
5. $\tilde{\boldsymbol{v}}_{k+1} = \boldsymbol{u}_k - \alpha_k\boldsymbol{v}_k$.
6. Set $k = k + 1$ and return to Step 1.

From Lanczos iteration history, we define the tridiagonal Lanczos matrix $\mathbf{T}_q$ as

$$\mathbf{T}_q = \begin{bmatrix} \alpha_0 & \eta_1 & 0 & 0 & \cdots & 0 \\ \eta_1 & \alpha_1 & \eta_2 & 0 & \cdots & 0 \\ 0 & \eta_2 & \alpha_2 & \eta_3 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \eta_{q-2} & \alpha_{q-2} & \eta_{q-1} \\ 0 & \cdots & 0 & 0 & \eta_{q-1} & \alpha_{q-1} \end{bmatrix},$$

and the matrix $\mathbf{V}_q$ to have orthogonal columns $\{\boldsymbol{v}_i\}_{i=0}^{q-1}$. Then [45, 67] state that we can

rewrite the tridiagonal Lanczos matrix as

$$\mathbf{T}_q = \mathbf{V}_q' \mathbf{A} \mathbf{V}_q. \tag{2.28}$$

By the orthogonality of $\mathbf{V}_q$, we have the following low-rank (q-rank) approximation of $\mathbf{A}$:

$$\mathbf{B}_{k,q}^{\dagger} = \mathbf{V}_q \mathbf{T}_q \mathbf{V}_q'. \tag{2.29}$$

As was stated above, we wish to obtain $\mathbf{B}_{k,q}^{\dagger}$ in terms of the CG iteration history. First we note that, using the CG residuals, $\mathbf{V}_q$ can be calculated with the relationship ([45], [67] p. 50)

$$\boldsymbol{v}_i = (-1)^i \frac{\boldsymbol{r}_i}{||\boldsymbol{r}_i||},$$

where the residual $\boldsymbol{r}_i$ from CG is calculated as $\boldsymbol{r}_i = -\nabla_{\boldsymbol{x}} \phi(\boldsymbol{x}_i)$. This, together with (2.28), defines (2.29).

Now, since $\mathbf{T}_q$ is symmetric, it can be diagonalized:

$$\mathbf{T}_q = \mathbf{U}_q \mathbf{S}_q \mathbf{U}_q'$$

where $\mathbf{U}_q = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_q)$ is the matrix with the orthonormal eigenvectors $\{\boldsymbol{u}_i\}$ of $\mathbf{T}_q$ as its columns and $\mathbf{S}_q$ is a diagonal matrix with the eigenvalues $\{s_i\}$ of $\mathbf{T}_q$ on its diagonal. Hence we can rewrite $\mathbf{B}_{k,q}^{\dagger}$ as

$$\begin{aligned} \mathbf{B}_{k,q}^{\dagger} =& \mathbf{V}_q \mathbf{U}_q \mathbf{S}_q \mathbf{U}_q' \mathbf{V}_q' \\ =& \mathbf{V}_q \mathbf{U}_q \mathbf{S}_q^{1/2} (\mathbf{V}_q \mathbf{U}_q \mathbf{S}_q^{1/2})' \\ \stackrel{\text{def}}{=}& \mathbf{X}_{k,q}^* (\mathbf{X}_{k,q}^*)'. \end{aligned} \tag{2.30}$$

Storing $\mathbf{X}_{k,q}^*$ of size $n \times q$ reduces the storage requirements of the approximation of $\mathbf{A}$. Note

that only CG iteration history was used to build $\mathbf{B}_{k,q}^{\dagger}$. Furthermore, since Lanczos is used to estimate the eigenvalue/eigenvector pairs of A, we note that the *Ritz vectors* $\{\mathbf{V}_q \boldsymbol{u}_i\}_{i=1}^{q}$ are estimates of the q eigenvectors $\{\boldsymbol{w}_i\}$ of $\mathbf{A}$ corresponding to $\{\lambda_i\}$.

Now that we have methods for approximating $\mathbf{A}$ and $\mathbf{A}^{-1}$ of (2.25), we discuss how these methods are introduced into the Kalman filters.

2.3.2 Conjugate Gradient in the Kalman Filter

If we consider the nonlinear Kalman filter method in Algorithm 8, we can apply CG twice: first to solve a linear system and second to calculate a covariance estimate. The first application comes from examining steps 3(a) and 3(b) of the algorithm where we notice that the expression

$$(\mathbf{K}_k \mathbf{C}_k^p \mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}(\boldsymbol{y}_k - \mathbf{K}_k \boldsymbol{x}_k^p)$$

appears. By letting $\mathbf{A} = \mathbf{K}_k \mathbf{C}_k^p \mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o}$ and $\boldsymbol{b} = \boldsymbol{y}_k - \mathbf{K}_k \boldsymbol{x}_k^p$, when we apply q iterations of CG to solving $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, we obtain $\boldsymbol{x}_q^{CG} \approx (\mathbf{K}_k \mathbf{C}_k^p \mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}(\boldsymbol{y}_k - \mathbf{K}_k \boldsymbol{x}_k^p)$. As above, we denote the approximation of $\mathbf{A}^{-1}$ as $\mathbf{B}_{k,q}$, where k indicates the time step in the Kalman filter and q indicates the stopping iteration of CG. Note that if the dimension of the observation space, m, is small, CG is not necessary to calculate this inverse.

CG is also applied to step 3(c) of Algorithm 8. If we use the CG/Lanczos relation and let $\mathbf{A} = \mathbf{C}_k^{est} \approx \mathbf{C}_k^p - \mathbf{C}_k^p \mathbf{K}_k' \mathbf{B}_{k,q} \mathbf{K}_k \mathbf{C}_k^p$, where $\mathbf{B}_{k,q} \approx (\mathbf{K}_k \mathbf{C}_k^p \mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}$ was obtained in the above application of CG, then we may obtain $\mathbf{B}_{k,q}^{\dagger}$, a q-rank, low-storage approximation of $\mathbf{A}$ as in (2.30). We choose $\boldsymbol{b}$ in (2.25) to be the random vector with entries 1 or -1, determined at random with equal probability, as this optimizes the accuracy of the covariance and inverse-covariance approximations [50]. In our tests, we also used $\boldsymbol{b} \sim N(\mathbf{0}, \mathbf{I})$, which works equally as well.

The pseudocode for the implementation of CG within the extended Kalman filter follows in Algorithm 13. We refer to this method as CG-KF. To ensure efficient computations, we assume that multiplication by linearizations $\mathbf{M}_k$ and $\mathbf{K}_k$, and their transposes, are efficient both computationally and in terms of storage. Note that the linear Kalman filter model is just a special case in which Step 2(a) is replaced with $\boldsymbol{x}_k^p = M_k \boldsymbol{x}_{k-1}^{est}$ since the linearizations of $\mathcal{M}$, $\mathcal{K}$ are simply the matrices $\mathbf{M}_k$, $\mathbf{K}_k$ in (2.1) and (2.2).

Algorithm 13 The Kalman Filter with Conjugate Gradient (CG-KF)

1. Select initial guesses for $\boldsymbol{x}_0^{est}$ and lower-storage initial covariance $\mathbf{B}_{0,0}^{\dagger} = \mathbf{C}_0^{est}$, and set $k = 1$.
2. Compute the evolution model estimate and covariance:
   (a) Compute $\boldsymbol{x}_k^p = \mathcal{M}(\boldsymbol{x}_{k-1}^{est})$.
   (b) Define $\mathbf{M}_k = \partial \mathcal{M}(\boldsymbol{x}_k^{est})/\partial \boldsymbol{x}$ and $\mathbf{C}_k^p = \mathbf{M}_k \mathbf{B}_{k-1,q}^{\dagger} \mathbf{M}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p}$.
3. Compute the Kalman filter and covariance estimates:
   (a) Apply CG to (2.25), with $\mathbf{A} = \mathbf{K}_k \mathbf{C}_k^p \mathbf{K}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^0}$ and $\boldsymbol{b} = (\boldsymbol{y}_k - \mathbf{K}_k \boldsymbol{x}_k^p)$ to obtain $\boldsymbol{x}_k^*$, and low-rank approximation $\mathbf{B}_{k,q}$ of $\mathbf{A}^{-1}$, where $\mathbf{K}_k = \partial \mathcal{K}(\boldsymbol{x}_k^p)/\partial \boldsymbol{x}$.
   (b) Compute approximate Kalman filter estimate $\boldsymbol{x}_k^{est} = \boldsymbol{x}_k^p + \mathbf{C}_k^p \mathbf{K}_k' \boldsymbol{x}_k^*$.
   (c) Apply CG to (2.25), with $\mathbf{A} = \mathbf{C}_k^p - \mathbf{C}_k^p \mathbf{K}_k' \mathbf{B}_{k,q} \mathbf{K}_k \mathbf{C}_k^p$ and $\boldsymbol{b}$ is a white noise random vector, to obtain low-rank approximation $\mathbf{B}_{k,q}^{\dagger}$ of $\mathbf{A}$.
4. Set $k = k + 1$ and return to Step 2.

2.3.3 Conjugate Gradient in the Variational Kalman Filter

Implementing CG within VKF is straightforward due to the existing quadratic minimization task in Step 3(a) of Algorithm 9. Specifically, our task is to minimize (2.23), or equivalently (2.25) with $\mathbf{A} = \mathbf{K}_k' \mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} \mathbf{K}_k + (\mathbf{C}_k^p)^{-1}$ and $\boldsymbol{b} = \mathbf{K}_k' \mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1} \boldsymbol{y}_k + (\mathbf{C}_k^p)^{-1} \boldsymbol{x}_k^p$, and for this we use CG. To ensure efficiency of the algorithm, we need multiplication by $\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}$ and $(\mathbf{C}_k^p)^{-1}$ to be efficient. To this end, we choose $\mathbf{C}_{\boldsymbol{\epsilon}_k^o}$ to be diagonal, and, using the form of the inverse Hessian

$\mathbf{C}_k^{est}$ found in (2.27), we use the matrix inversion lemma (2.4) to write

$$\begin{aligned}(\mathbf{C}_k^p)^{-1} &= (\mathbf{M}_k\mathbf{C}_{k-1}^{est}\mathbf{M}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p})^{-1} \\ &= ((\mathbf{M}_k\mathbf{X}_{k,q})(\mathbf{M}_k\mathbf{X}_{k,q})' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p})^{-1} \\ &= \mathbf{C}_{\boldsymbol{\epsilon}_k^p}^{-1} + \mathbf{C}_{\boldsymbol{\epsilon}_k^p}^{-1}\mathbf{M}_k\mathbf{X}_{k,q}(\mathbf{I} + \mathbf{X}_{k,q}'\mathbf{M}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^p}^{-1}\mathbf{M}_k\mathbf{X}_{k,q})^{-1}\mathbf{X}_{k,q}'\mathbf{M}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^p}^{-1}. \end{aligned} \tag{2.31}$$

Note that if q, the stopping iteration of CG, is small enough so that the $q \times q$ inverse in (2.31) is computed efficiently, then multiplication by $(\mathbf{C}_k^p)^{-1}$ will be efficient. If, on the other hand, q is too large for efficient computations with (2.31), we may return to optimization calculations, as was done in Algorithm 13 in Step 3(c). In that case, we would apply CG to (2.25) to obtain low-rank approximation $\mathbf{B}_{k,q}$ of $\mathbf{A}^{-1}$, with $\mathbf{A} = \mathbf{C}_k^p$ and $\boldsymbol{b}$ a white noise random vector.

We now present pseudocode for the implementation of CG within VKF in Algorithm 14; we refer to this method as CG-VKF. As before, in the case of a linear model, the linearizations of $\mathcal{M}$ and $\mathcal{K}$ are simply the matrices $\mathbf{M}_k$ and $\mathbf{K}_k$ of (2.1), (2.2). Furthermore, the nonlinear function in Step 2(a) becomes $\boldsymbol{x}_k^p = \mathbf{M}_k\boldsymbol{x}_{k-1}^{est}$.

The matrix-vector multiplies dominate the cost of implementation in both CG-KF and CG-VKF, with a cost of about $2n^2$ (or $2m^2$) flops in each CG iteration [98]. When the number of CG iterations, q, is small relative to n, then CG-KF and CG-VKF are cheaper to implement than KF and VKF. Additionally, in the implementation of CG, we store $n \times q$ elements for the covariance approximations, as opposed to the original $n \times n$ covariance matrices.

2.3.4 Conjugate Gradient in the Ensemble Kalman Filter

Next we discuss how to use CG within EnKF. As mentioned above, perturbations of the model from sampling the errors in the EnKF method of Algorithm 10, Steps 2(a) and 3(b) lead to

Algorithm 14 The Variational Kalman Filter with Conjugate Gradient (CG-VKF)

1. Select initial guesses for $\boldsymbol{x}_0^{est}$ and low-rank covariance approximation $\mathbf{B}_{0,0} = \mathbf{X}_0\mathbf{X}_0' \approx \mathbf{C}_0^{est}$, and set $k = 1$.
2. Compute the evolution model estimate and covariance:
   (a) Compute $\boldsymbol{x}_k^p = \mathcal{M}(\boldsymbol{x}_{k-1}^{est})$.
   (b) Define $\mathbf{M}_k = \partial\mathcal{M}(\boldsymbol{x}_k^{est})/\partial\boldsymbol{x}$ and $(\mathbf{C}_k^p)^{-1}$ using (2.31).
3. Compute the Kalman filter and covariance estimates:
   (a) Apply CG to the problem of minimizing

$$\ell(\boldsymbol{x}|\boldsymbol{y}_k) = \frac{1}{2}(\boldsymbol{y}_k - \mathbf{K}_k\boldsymbol{x})'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}(\boldsymbol{y}_k - \mathbf{K}_k\boldsymbol{x}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k^p)'(\mathbf{C}_k^p)^{-1}(\boldsymbol{x} - \boldsymbol{x}_k^p),$$

   which has the form of (2.25) with $\mathbf{A} = \mathbf{K}_k'(\mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1}$, $\boldsymbol{b} = \mathbf{K}_k'(\mathbf{C}_{\boldsymbol{\epsilon}_k^o})^{-1}\boldsymbol{y}_k + (\mathbf{C}_k^p)^{-1}\boldsymbol{x}_k^p$, and where $\mathbf{K}_k = \partial\mathcal{K}(\boldsymbol{x}_k^p)/\partial\boldsymbol{x}$, to obtain minimizer $\boldsymbol{x}_k^{est}$ and inverse Hessian, low-rank approximation $\mathbf{B}_{k,q} = \mathbf{X}_{k,q}\mathbf{X}_{k,q}'$ of $\mathbf{C}_k^{est}$.
4. Set $k = k + 1$ and return to Step 2.

the underestimation of the state covariance. The method of [90] skirts this issue by integrating forward the ensemble members $\boldsymbol{x}_{k-1,i}^{est}$ with the evolution operator, for $i = 1, \ldots, N$, as well as the state estimate, $\boldsymbol{x}_{k-1}^{est}$. The prediction error covariance matrix $\mathbf{C}_{\boldsymbol{\epsilon}_k^p}$ is then added to the ensemble empirical covariance to obtain $\mathbf{C}_k^p$, similar to that of Step 2(b) in Algorithms 7, 8 and 9. This removes the perturbations of the model that occur due to sampling the errors.

Step 3 of the EnKF algorithm may be reformulated using the variational approach, by which the minimizer $\boldsymbol{x}_k^{est}$ and approximate inverse Hessian $\mathbf{B}_{k,q} \approx \mathbf{C}_k^{est}$ are calculated from (2.23). To do so, we apply CG to (2.25) with $\mathbf{A} = \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1}$ and $\boldsymbol{b} = \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\boldsymbol{y}_k + (\mathbf{C}_k^p)^{-1}\boldsymbol{x}_k^p$. With the state and covariance estimates in hand, we obtain the new ensemble estimates by sampling from $\boldsymbol{x}_{k,i}^{est} \sim N(\boldsymbol{x}_k^{est}, \mathbf{B}_{k,q} \approx \mathbf{C}_k^{est})$ using the CG sampler.

The pseudocode for the resulting variational EnKF, which makes use of CG, is given in Algorithm 15, with the necessary discussion of the CG sampler provided below. We refer to this method as CG-EnKF, which is the ensemble-version of CG-VKF.

Algorithm 15 The Ensemble Kalman Filter with Conjugate Gradient (CG-EnKF)

1. Select initial guesses for $\boldsymbol{x}_0^{est}$ and $\mathbf{C}_0^{est}$. Sample initial ensemble members $\boldsymbol{x}_{0,i}^{est} \sim N(\boldsymbol{x}_0^{est}, \mathbf{C}_0^{est})$ for $i = 1, \ldots, N$ and set $k = 1$.

2. Integrate the ensemble forward in time and estimate the covariance:

   (a) Compute $\boldsymbol{x}_k^p = \mathcal{M}(\boldsymbol{x}_{k-1}^{est})$ and $\boldsymbol{x}_{k,i}^p = \mathcal{M}(\boldsymbol{x}_{k-1,i}^{est})$ for $i = 1, \ldots, N$.

   (b) Define the model covariance as

$$\mathbf{C}_k^p = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_{k,i}^p - \boldsymbol{x}_k^p)(\boldsymbol{x}_{k,i}^p - \boldsymbol{x}_k^p)' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p}.$$

3. Compute a new ensemble using the CG sampler:

   (a) Apply CG to (2.25) with $\mathbf{A} = \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\mathbf{K}_k + (\mathbf{C}_k^p)^{-1}$ and $\boldsymbol{b} = \mathbf{K}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^o}^{-1}\boldsymbol{y}_k + (\mathbf{C}_k^p)^{-1}\boldsymbol{x}_k^p$ to estimate the minimizer $\boldsymbol{x}_k^{est}$ and approximate inverse Hessian $\mathbf{B}_{k,q} \approx \mathbf{C}_k^{est}$, as well as compute new ensemble members $\boldsymbol{x}_{k,i}^{est} \sim N(\boldsymbol{x}_k^{est}, \mathbf{B}_{k,q})$, for $i = 1, \ldots, N$ using the CG sampler.

4. Set $k = k + 1$ and return to Step 2.

It remains to discuss the two computational limitations of this variational approach. The first is that computations with $(\mathbf{C}_k^p)^{-1}$ must remain efficient while adhering to the restriction that no large and dense matrices be stored. Since we desire efficient multiplication by $(\mathbf{C}_k^p)^{-1}$, we begin by noting that we can rewrite the covariance estimate as $\mathbf{C}_k^p = \mathbf{X}_k\mathbf{X}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p}$, where

$$\mathbf{X}_k = \left[(\boldsymbol{x}_{k,1}^p - \boldsymbol{x}_k^p), (\boldsymbol{x}_{k,2}^p - \boldsymbol{x}_k^p), \ldots, (\boldsymbol{x}_{k,N}^p - \boldsymbol{x}_k^p)\right]/\sqrt{N},$$

and $\boldsymbol{x}_{k,i}$ is the i^{th} ensemble member at the k^{th} time step of the filter. Then using the matrix inversion lemma (2.4) we can compute, similar to (2.31),

$$\begin{aligned}(\mathbf{C}_k^p)^{-1} &= (\mathbf{X}_k\mathbf{X}_k' + \mathbf{C}_{\boldsymbol{\epsilon}_k^p})^{-1} \\ &= \mathbf{C}_{\boldsymbol{\epsilon}_k^p}^{-1} - \mathbf{C}_{\boldsymbol{\epsilon}_k^p}^{-1}\mathbf{X}_k(\mathbf{I} + \mathbf{X}_k'\mathbf{C}_{\boldsymbol{\epsilon}_k^p}^{-1}\mathbf{X}_k)^{-1}\mathbf{X}_k\mathbf{C}_{\boldsymbol{\epsilon}_k^p}^{-1}. \end{aligned} \tag{2.32}$$

Assuming N is not too large, the inverse in (2.32) should be computationally feasible and assuming computations with $\mathbf{C}_{\boldsymbol{\epsilon}_k^p}^{-1}$ are efficient, then computations with (2.32) will be efficient.

In our examples, we assume $\mathbf{C}_{\boldsymbol{\epsilon}_k^p}$ is a diagonal matrix and use relatively small ensemble sizes.

Second, it must be efficient to obtain samples from $N(\boldsymbol{x}_k^{est}, \mathbf{B}_{k,q})$, where $\mathbf{B}_{k,q} \approx \mathbf{C}_k^{est}$. Gaussian sampling is typically computed through a Cholesky decomposition of the dense covariance matrix; however, we do not wish to store or use a dense covariance matrix in the computations for large-scale problems. To circumvent the issues associated with Gaussian sampling, we introduce the CG sampler for the Gaussian sampling in Step 3 of CG-EnKF, which can be computed from the CG iterations determined in Step 3.

The Conjugate Gradient Sampler

Parker and Fox show in [74] that while the CG algorithm computes the minimizer of (2.25), it can also approximately sample from $\boldsymbol{w}_{i,q} \sim N(\mathbf{0}, \mathbf{A}^{-1})$ with one additional line of code, where i represents the i^{th} ensemble member and q is the number of CG iterations. The sampler is initialized with $\boldsymbol{w}_{i,0} = \mathbf{0}$, for $i = 1, \ldots, N$. Recalling that $\mathbf{X}_{k,q} = \mathbf{P}_q \mathbf{D}_q^{-1/2}$ from (2.27), a CG sample can be written as $\boldsymbol{w}_{i,q} = \mathbf{P}_q \mathbf{D}_q^{-1/2} \boldsymbol{z}$, where $\boldsymbol{z} \sim N(\mathbf{0}, \mathbf{I})$, for $i = 1, \ldots, N$. Hence,

$$\boldsymbol{w}_{i,q} \sim N(\mathbf{0}, \mathbf{P}_q \mathbf{D}_q^{-1} \mathbf{P}_q'). \tag{2.33}$$

The matrix $\mathbf{P}_q \mathbf{D}_q^{-1} \mathbf{P}_q'$ is singular and the distribution of $\boldsymbol{w}_{i,q}$ is an intrinsic Gaussian [80]. However, at iteration $q = n$ of the CG sampler, in exact arithmetic and assuming n distinct eigenvalues, by (2.26) and (2.33), we have

$$\boldsymbol{w}_{i,n} \sim N(\mathbf{0}, \mathbf{A}^{-1}).$$

In the case of non-distinct eigenvalues, or when only an approximate minimizer of (2.25) is sought, CG terminates at iteration $q < n$ [67].

This allows us to compute the ensemble samples in Step 3(b) of CG-EnKF via

$$\boldsymbol{x}_{k,i}^{est} = \boldsymbol{x}_k^{est} + \boldsymbol{w}_{i,q},$$

where $\boldsymbol{w}_{i,q} \sim N(\mathbf{0}, \mathbf{P}_q\mathbf{D}_q^{-1}\mathbf{P}_q')$ and $\boldsymbol{x}_k^{est}$ is the approximate minimizer computed by CG in Step 3(a) of CG-EnKF. Therefore, ensemble samples will have the distribution $\boldsymbol{x}_{k,i}^{est} \sim N(\boldsymbol{x}_k^{est}, \mathbf{B}_{k,q})$, where $\mathbf{B}_{k,q} = \mathbf{P}_q\mathbf{D}_q^{-1}\mathbf{P}_q' \approx \mathbf{C}_k^{est}$. Pseudocode for the CG sampler is given in Algorithm 16, to be used concurrently for Steps 3(a) and 3(b) of Algorithm 15. Note that Algorithm 16 is simply Algorithm 11 with Step 3 added to compute the CG samples.

Algorithm 16 Conjugate Gradient Sampler

Given $\mathbf{A}$, $\boldsymbol{b}$, and $\boldsymbol{x}_0$, let $\boldsymbol{r}_0 = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_0$, $\boldsymbol{p}_0 = \boldsymbol{r}_0$, $d_0 = \boldsymbol{p}_0'\mathbf{A}\boldsymbol{p}_0$, $j = 1$, and $\boldsymbol{w}_{i,0} = \mathbf{0}$ for $i = 1, \ldots, N$. Set $j = 1$. Specify stopping tolerance ϵ and iterate:

1. $\gamma_{j-1} = \frac{\boldsymbol{r}_{j-1}'\boldsymbol{r}_{j-1}}{d_{j-1}}$.
2. $\boldsymbol{x}_j = \boldsymbol{x}_{j-1} + \gamma_{j-1}\boldsymbol{p}_{j-1}$.
3. $\boldsymbol{w}_{i,j} = \boldsymbol{w}_{i,j-1} + (z_i/\sqrt{d_{j-1}})\boldsymbol{p}_{j-1}$, where $z_i \sim N(0,1)$, for $i = 1, \ldots, N$.
4. $\boldsymbol{r}_j = \boldsymbol{b} - \mathbf{A}\boldsymbol{x}_j = \boldsymbol{r}_{j-1} - \gamma_{j-1}\mathbf{A}\boldsymbol{p}_{j-1}$.
5. $\beta_j = -\frac{\boldsymbol{r}_j'\boldsymbol{r}_j}{\boldsymbol{r}_{j-1}'\boldsymbol{r}_{j-1}}$.
6. $\boldsymbol{p}_j = \boldsymbol{r}_j - \beta_j\boldsymbol{p}_{j-1}$ and $d_j = \boldsymbol{p}_j'\mathbf{A}\boldsymbol{p}_j$.
7. Quit if $||\boldsymbol{r}_j|| < \epsilon$. Else set $j = j + 1$ and return to Step 1.

It is worthwhile to note that as long as the number of CG iterations, q, remains small compared to n, CG-EnKF is cheaper to implement than EnKF because the cost of implementing CG is dominated by the matrix-vector multiplies with $2n^2$ flops per CG iteration [98].

2.3.5 Analysis of the Approximations

We now discuss the accuracy of the CG covariance approximation, $\mathbf{B}_{k,q} = \mathbf{P}_q\mathbf{D}_q^{-1}\mathbf{P}_q'$. Recall that this approximation is used in CG-KF, CG-VKF and CG-EnKF. Thus the following

analysis extends to all presented implementations of CG within the filters.

If the eigenvalues of $\mathbf{A}$ are clustered into q distinct groups, CG returns the approximate solution at the q^{th} iteration in a *q-dimensional Krylov space*,

$$\mathcal{K}^q(\mathbf{A}, \boldsymbol{r}_0) = \text{span}(\boldsymbol{r}_0, \mathbf{A}\boldsymbol{r}_0, \mathbf{A}^2\boldsymbol{r}_0, \ldots, \mathbf{A}^{q-1}\boldsymbol{r}_0).$$

In [74], Remark 4, Parker and Fox show the Ritz vectors estimating the corresponding q eigenvectors of $\mathbf{A}$ are the eigenvectors of $\text{var}(\boldsymbol{w}_{i,q}|\boldsymbol{b}) = \mathbf{P}_q\mathbf{D}_q^{-1}\mathbf{P}_q'$ and when CG converges with residual $\boldsymbol{r}_q = \mathbf{0}$, then

$$\left[\mathbf{A}^{-1} - \mathbf{P}_q\mathbf{D}_q^{-1}\mathbf{P}_q'\right]\boldsymbol{v} = \mathbf{0},$$

for any $\boldsymbol{v} \in \mathcal{K}^q(\mathbf{A}, \boldsymbol{r}_0)$. At the convergence of CG, this Krylov space contains the eigenspaces corresponding to the extreme and well-separated eigenvalues of $\mathbf{A}$ [67, 75, 81, 89]. Therefore, $\mathbf{P}_q\mathbf{D}_q^{-1}\mathbf{P}_q'$ is the best q-rank approximation to $\mathbf{A}^{-1}$ in the eigenspaces corresponding to the extreme and well-separated eigenvalues of $\mathbf{A}$.

Regarding the Lanczos approximation, (2.28) demonstrates that the Lanczos algorithm is a Raleigh-Ritz process [75], where $\mathbf{T}_q$ is the minimizer of $\rho(\zeta|\mathbf{V}_q) = ||\mathbf{A}\mathbf{V}_q - \mathbf{V}_q\zeta||_2$, or equivalently,

$$\min_{\zeta \in \mathbb{R}^{n\times n}} \rho(\zeta|\mathbf{V}_q) = ||(\mathbf{A} - \mathbf{V}_q\mathbf{T}_q\mathbf{V}_q')\mathbf{V}_q||_2,$$

resulting in $\mathbf{B}_{k,q}^{\dagger} = \mathbf{V}_q\mathbf{T}_q\mathbf{V}_q'$ being the best q-rank approximation of $\mathbf{A}$ in range$(\mathbf{V}_q)$, the q-dimensional Krylov space also spanned by the CG conjugate directions. Therefore, in this Krylov subspace, $\mathbf{B}_{k,q}^{\dagger}$ is the optimal approximation of $\mathbf{A}$.

Furthermore, these results on the accuracy of the approximations have been proven by Parker and Fox [74, Theorem 3.1], given here for completion.

Theorem 5. *The covariance matrix of the CG sample* $\boldsymbol{w}_{i,q}$ *is*

$$\mathrm{var}(\boldsymbol{w}_{\mathrm{i,q}}|\boldsymbol{b}) = \mathbf{V}_{\mathrm{q}}\mathbf{T}_{\mathrm{q}}^{-1}\mathbf{V}_{\mathrm{q}},$$

and it has q nonzero eigenvalues $\{\frac{1}{s_i}\}$*, which are the Lanczos estimates of the eigenvalues of* A^{-1}*. The eigenvectors of* $\mathrm{var}(\boldsymbol{w}_{\mathrm{i,q}}|\boldsymbol{b})$ *are the Ritz vectors* $\mathbf{V}_q\boldsymbol{u}_i$ *which estimate the eigenvectors of A. When* $||\boldsymbol{r}_q|| = 0$*, then*

$$\mathrm{var}(\mathbf{A}\boldsymbol{w}_{\mathrm{i,q}}|\boldsymbol{b}) = \mathbf{V}_{\mathrm{q}}\mathbf{T}_{\mathrm{q}}\mathbf{V}_{\mathrm{q}},$$

and the q eigenvector/eigenvalue pairs of $\mathrm{var}(\mathbf{A}\boldsymbol{w}_{\mathrm{i,q}}|\boldsymbol{b})$ *with nonzero eigenvalues are the Lanczos Ritz pairs* $(s_i, \mathbf{V}_q\boldsymbol{u}_i)$*.*

2.4 Numerical Results

Here we perform tests in MATLAB with CG-KF, CG-VKF and CG-EnKF, comparing results to those obtained using the respective LBFGS techniques presented in [2,3,90] since the codes for these algorithms were readily available to us. Two synthetic test cases are used here. The Lorenz 95 system is a first-order, nonlinear, chaotic ODE system which shares similar features to weather models, and therefore is an appropriate test case. The second example is a two-dimensional heat equation that we use as a large-scale test case.

2.4.1 Lorenz 95

The Lorenz 95 test case was introduced in 1996 in [62] and analyzed two years later in [63]. The model is given by

$$\frac{\partial x_i}{\partial t} = (x_{i+1} - x_{i-2})x_{i-1} - x_i + 8, \quad i = 1, \ldots, 40, \tag{2.34}$$

with periodic state space variables where $x_{-1} = x_{39}$, $x_0 = x_{40}$ and $x_{41} = x_1$. While this test case has a rather small dimension of 40 variables, this system produces chaotic and unpredictable behavior, like that of realistic atmospheric and weather forecasting models, in addition to the periodic nature used in weather models [63].

The filters are applied to the problem of estimating the temporal state variables from data generated using the nonlinear model in (2.34). The data are generated using a fourth order Runge-Kutta (RK4) method with a time step of $\Delta t = 0.025$. When using (2.34) as a test case representing weather forecasting systems, [63] suggests the time step be three hours. Thus by generating $42,920$ time steps of data, we obtain 5365 days worth of 'truth' data, at three hour intervals. The initial state used is $x_{20} = 8 + 0.008$ and $x_i = 8$ for $i \neq 20$. Visualizing this data is difficult due to its dimension; however, a plot of the true values of x_1 is given in Figure 2.1.

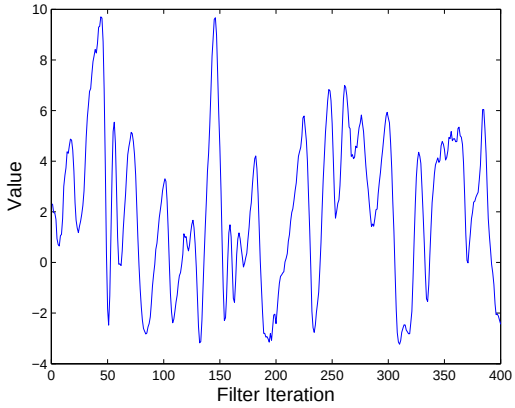

Figure 2.1: True values for variable x_1 in the Lorenz 95 model plotted in time.

From the true data we compute the observed data, such that, after a 365 day interval, the true data are observed at alternating time steps and at the last 3 grid points in each set of

five. This yields the observation matrix $\mathbf{K}_k = \mathbf{K}$ of size $m \times n$ with entries

$$[\mathbf{K}]_{rs} = \begin{cases} 1 & (r,s) \in \{(3j+i, 5j+i+2) \mid i = 1,2,3,\ j = 0,\ldots,7\}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.35}$$

For EKF and VKF, we use the system (2.19) and (2.20). Here $\mathcal{K}$ is the linear matrix given in (2.35) and we take $\boldsymbol{\epsilon}_k^p \sim N(\mathbf{0}, (0.05\sigma_{\text{clim}})^2\mathbf{I})$ and $\boldsymbol{\epsilon}_k^o \sim N(\mathbf{0}, (\sigma_{\text{clim}})^2\mathbf{I})$, where $\sigma_{\text{clim}} = 3.6414723$ is the standard deviation of the model state used in climatological simulations. For the ensemble filter, we use $\boldsymbol{\epsilon}_k^p \sim N(\mathbf{0}, (0.05\sigma_{\text{clim}})^2\mathbf{I})$ and $\boldsymbol{\epsilon}_k^o \sim N(\mathbf{0}, (0.15\sigma_{\text{clim}})^2\mathbf{I})$. Since this is a well-established test case, the tangent linear code is available to us from our collaborators for this model, as needed in EKF and VKF.

Finally, initial guesses for all methods are taken to be $\boldsymbol{x}_0^{est} = \mathbf{1}$ and $\mathbf{C}_0^{est} = \mathbf{I}$. Sensitivity to initial guesses was not examined here. In all implementations, CG is considered to have converged, and iterations are stopped, when $||\boldsymbol{r}_k|| < 10^{-6}$, with a maximum number of iterations set at 50. All tests are compared with LBFGS, using the same optimization settings above.

The root-mean-square error,

$$\sqrt{\frac{1}{N}\left|\left|\boldsymbol{x}_k - \boldsymbol{x}_k^{\text{true}}\right|\right|^2},$$

is computed to compare CG-KF and LBFGS-KF with EKF in the left-hand image in Figure 2.2, while the right-hand image compares CG-VKF and LBFGS-VKF with EKF. In both images, the root-mean-square error tends to be lower for the CG methods than the LBFGS methods. Also, CG-VKF has nearly the same root-mean-square error as EKF at each filter iteration past $k = 30$. Average computational times of five trials for these Kalman filters are compared in Table 2.1. The conjugate gradient-based methods are computationally faster than the LBFGS-based methods, though the EKF method remains the fastest. However, recall that in higher dimensional problems EKF becomes computationally infeasible.

The root-mean-square error of CG-EnKF is compared to the LBFGS-EnKF method of [90]

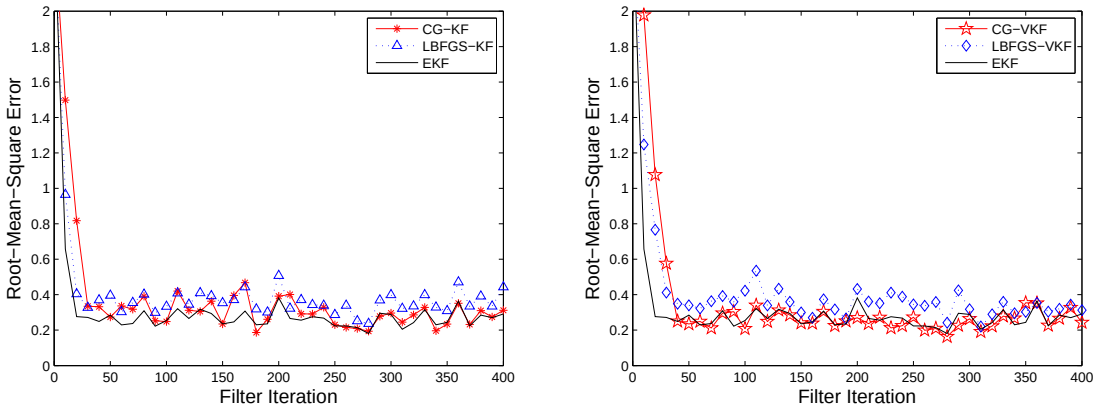

Figure 2.2: Root-mean-square error versus filter iteration to compare (left) the EKF, CG-KF and LBFGS-KF methods and (right) the EKF, CG-VKF and LBFGS-VKF methods, in the Lorenz 95 example.

Method	Average Time in Seconds (Standard Deviation)
CG-KF	9.6510 (0.0703)
LBFGS-KF	68.1245 (1.2075)
CG-VKF	13.4220 (0.3635)
LBFGS-VKF	48.6552 (2.2134)
EKF	3.3451 (0.0808)

Table 2.1: CPU average times and standard deviations of five trials for EKF and the CG-based and LBFGS-based KF and VKF methods in the Lorenz 95 test case.

as well as EKF in Figure 2.3 for $N = 20$ and 50. Recall that the ensemble methods are stochastic, thus the plotted errors are an average of 20 runs. EnKF typically performs poorly with smaller ensemble sizes, thus we compare the approximate EnKF methods with EKF. For both ensemble sizes, the root-mean-square error of the CG-based methods is lower than that of the LBFGS methods, after the first ten iterations. Average computational times of five trials for the ensemble Kalman filters are compared in Table 2.2. We see that CG-EnKF is about twice as fast, computationally, as LBFGS-EnKF.

Figure 2.4 demonstrates that as the ensemble size increases, CG-EnKF approaches CG-VKF. We use ensemble sizes $N = 10$, 20, and 50.

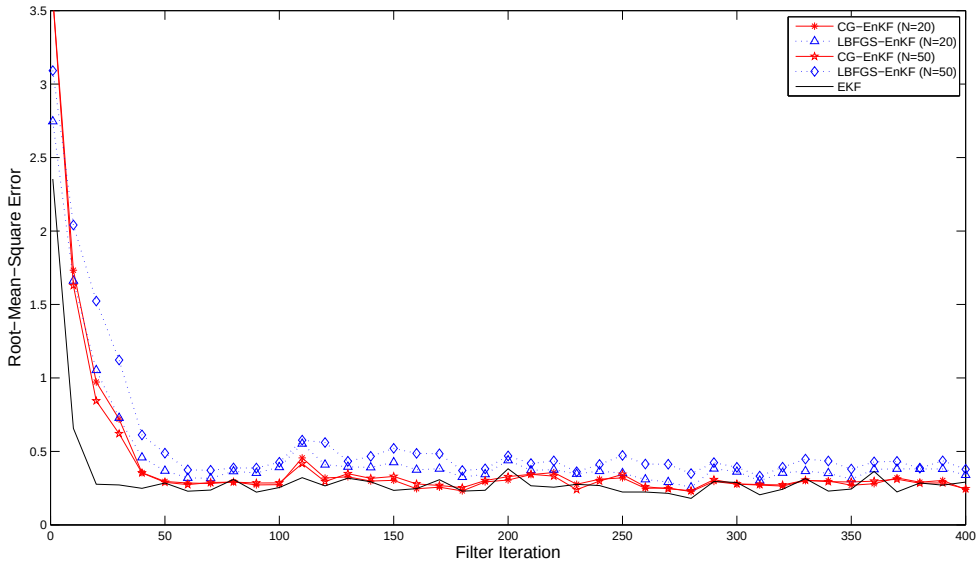

Figure 2.3: Root-mean-square error versus filter iteration to compare the CG-EnKF, LBFGS-EnKF and EKF methods for $N = 20$, 50 in the Lorenz 95 example.

Method	Average Time in Seconds (Standard Deviation)	
	$N = 20$	$N = 50$
CG-EnKF	14.2249 (0.4285)	30.5166 (0.6312)
LBFGS-EnKF	28.8748 (1.5203)	58.1476 (1.1062)

Table 2.2: CPU average times and standard deviations of five trials for the CG-based and LBFGS-based ensemble Kalman filter methods in the Lorenz 95 test case. The ensemble sizes are $N = 20$, 50.

2.4.2 Heat Equation

This example demonstrates the nature of the Kalman filter methods when the dimension is large. We consider the forced heat propagation equation

$$\frac{\partial x}{\partial t} = -\frac{\partial^2 x}{\partial u^2} - \frac{\partial^2 x}{\partial v^2} + \delta e^{-\frac{(u-2/9)^2+(v-2/9)^2}{\sigma^2}}, \tag{2.36}$$

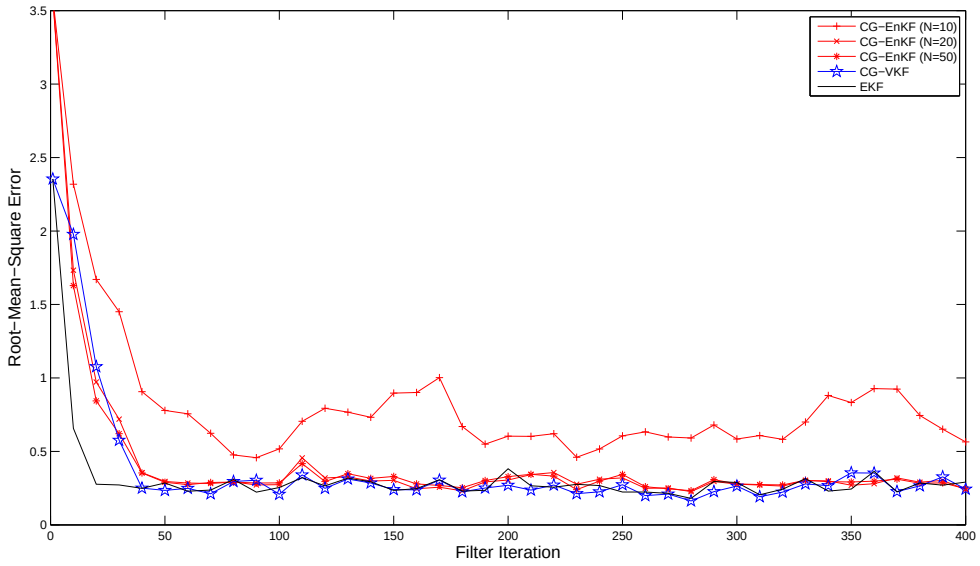

Figure 2.4: Root-mean-square error versus filter iteration to compare the ensemble sizes $N = 10, 20, 50$ for CG-EnKF with CG-VKF and EKF in the Lorenz 95 example.

where the temperature $x(u, v)$ is a function over the domain $\Omega = \{(u, v)|0 \leq u, v \leq 1\}$ and $\delta \geq 0$ is the magnitude of the external heat source. This yields a linear model to which we can apply KF; however, as the dimension increases, the calculations for KF become infeasible.

With discretization of (2.36), we can control the dimension of the uniform $S \times S$ grid, which leads to the linear forward model $\boldsymbol{x}_{k+1} = M\boldsymbol{x}_k + \boldsymbol{f}$, where $M = I - \Delta t L$. Here, $\boldsymbol{f}$ is the constant vector determined by evaluating the forcing term in (2.36), Δt is the time step, and L is the discrete negative-Laplacian with Dirichlet boundary conditions. The observation matrix $K_k = K$ is the full weighting matrix defined by

$$K = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix},$$

creating a weighted average of temperatures at the eight nearest neighboring points.

Data are generated using the stochastic equations

$$\boldsymbol{x}_{k+1} = \mathbf{M}\boldsymbol{x}_k + \boldsymbol{f} + \boldsymbol{\epsilon}_k^p,$$
$$\boldsymbol{y}_{k+1} = \mathbf{K}\boldsymbol{x}_{k+1} + \boldsymbol{\epsilon}_k^o,$$

where $\boldsymbol{\epsilon}_k^p \sim N(\mathbf{0}, (0.5\sigma_{\text{ev}})^2\mathbf{I})$ and $\boldsymbol{\epsilon}_k^o \sim N(\mathbf{0}, (0.8\sigma_{\text{ob}})^2\mathbf{I})$. Furthermore, we set $\delta = 0.75$ and choose σ_{ev} and σ_{obs} such that the signal-to-noise ratios $||\boldsymbol{x}_0||^2/S^2\sigma_{\text{ev}}^2$ and $||\mathbf{K}\boldsymbol{x}_0||^2/m^2\sigma_{\text{obs}}^2$ are both 50. For data generation we use the initial condition

$$[\boldsymbol{x}_0]_{ij} = e^{-\left((u_i-1/2)^2-(v_j-1/2)^2\right)},$$

where (u_i, v_j) is the ij^{th} grid point.

We use a biased model for filtering, i.e., we drop the forcing term by setting $\delta = 0$. Our model for implementation now reverts to (2.1) and (2.2), with $\boldsymbol{\epsilon}_k^p \sim N(\mathbf{0}, \sigma_{\text{ev}}^2\mathbf{I})$ and $\boldsymbol{\epsilon}_k^o \sim N(\mathbf{0}, \sigma_{\text{ob}}^2\mathbf{I})$. We take $\boldsymbol{x}_0^{est} = \mathbf{0}$ and $\mathbf{C}_0^{est} = \mathbf{0}$, though sensitivity to these initial conditions was not examined here.

For the first test, we choose $S = 2^5$, yielding a computational grid of 32×32, or a dimension of $S^2 = 1032$. The second test uses $S = 2^7$, which yields a computational grid of 128×128, or a dimension of $S^2 = 16384$. A stopping tolerance of $||\boldsymbol{r}_k|| < 10^{-6}$ was used in all CG implementations to indicate convergence, with a maximum number of iterations set to 40. We compare CG-KF and CG-VKF with the LBFGS-KF method of [2] and the LBFGS-VKF method of [3], using the same stopping tolerance and maximum number of iterations as CG.

Figure 2.5 shows the root-mean-square error for both grid sizes 32×32 and 128×128 comparing the CG-KF, LBFGS-KF, CG-VKF, LBFGS-VKF and KF methods. For $S = 32$, CG-VKF and LBFGS-VKF have near identical root-mean-square errors, until the filter iteration surpasses $k = 45$, otherwise, the CG-based methods tend to outperform the LBFGS-based methods in root-mean-square error. For the case with $S = 128$, KF is infeasible to calculate

Method	Average Time in Seconds (Standard Deviation)	
	$S = 32$	$S = 128$
CG-KF	5.1404 (0.5692)	34.8790 (0.8832)
LBFGS-KF	43.7063 (5.8862)	667.0230 (50.0171)
CG-VKF	6.7937 (0.9334)	90.3871 (18.6978)
LBFGS-VKF	13.2907 (1.6821)	120.5410 (15.3847)
KF	19.7148 (1.2362)	Computationally Infeasible

Table 2.3: CPU average times and standard deviations of five trials for KF and the CG-based and LBFGS-based KF and VKF methods in the heat equation test case. Average times are given for both grid sizes 32×32 and 128×128. KF on the 128×128 grid was not computationally feasible on the computer we used.

using a standard desktop computer and the CG-based methods tend to have lower root-mean-square error than the LBFGS-based methods. Table 2.3 compares the computational average and standard deviation of five trials for the approximate KF and VKF methods with KF. The CG-based Kalman filter methods are at least twice as fast as the LBFGS-based methods, as well as the KF method (when KF is computationally feasible).

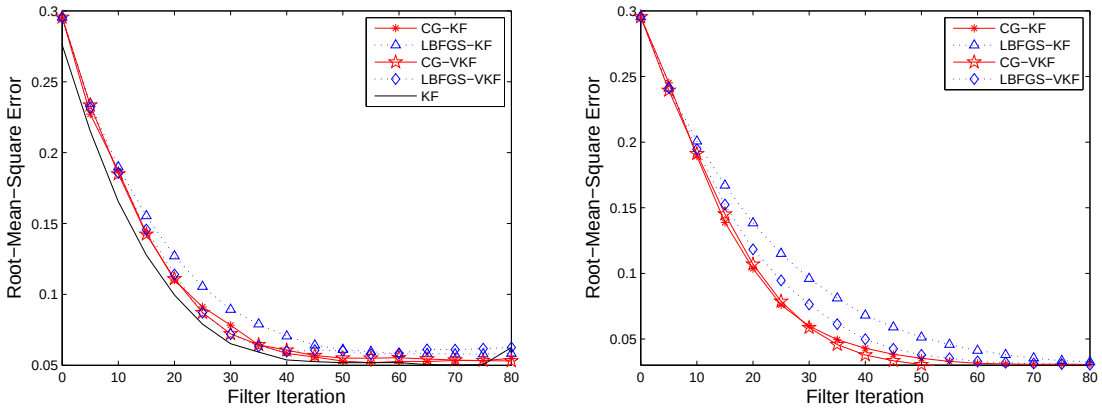

Figure 2.5: Root-mean-square error versus filter iteration to compare the KF, CG-KF, LBFGS-KF, CG-VKF and LBFGS-KF methods on a 32×32 computational grid (left) and 128×128 computational grid (right), in the heat equation example.

For the ensemble Kalman filter comparisons, we compare CG-EnKF with the LBFGS-EnKF method of [90] using an ensemble size of $N = 50$. EnKF typically performs poorly with smaller ensemble sizes, thus we compare the approximate EnKF methods with KF for $S = 32$. As

Method	Average Time in Seconds (Standard Deviation)	
	$S = 32$	$S = 128$
CG-EnKF	10.3096 (0.9512)	178.6400 (26.4002)
LBFGS-EnKF	21.3020 (1.2775)	446.9500 (24.8992)

Table 2.4: CPU average times and standard deviations of five trials for the CG-based and LBFGS-based ensemble Kalman filter methods in the heat equation test case. Average times are given for both grid sizes 32×32 and 128×128 and the ensemble size is $N = 50$.

before for $S = 128$, KF cannot be computed using the standard desktop computer. Figure 2.6 plots the root-mean-square error for both grid sizes for the ensemble filters. Recall that the ensemble methods are stochastic, thus the plotted errors are an average of 20 runs. For both grid sizes, CG-EnKF returns a smaller root-mean-square error than LBFGS-EnKF. Table 2.4 compares the average computational time and standard deviation for the approximate EnKF methods. For both grid sizes, CG-EnKF is at least twice as fast as LBFGS-EnKF.

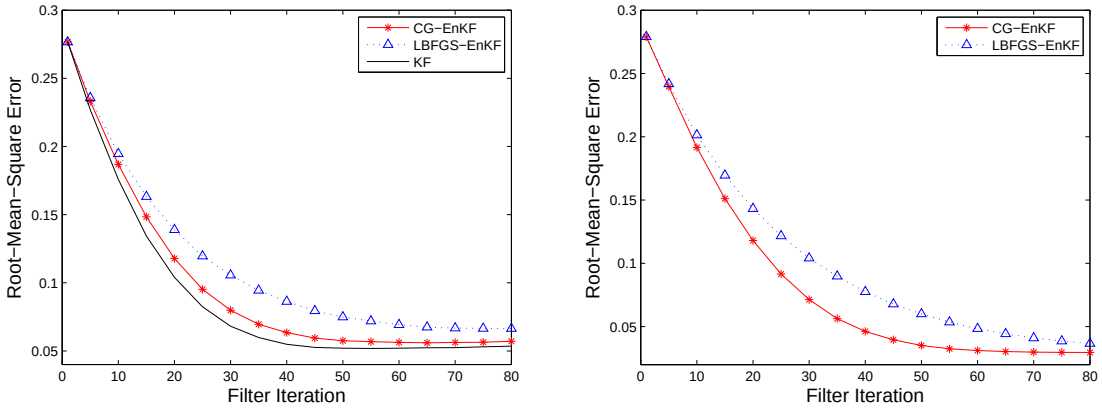

Figure 2.6: Root-mean-square error versus filter iteration to compare the KF, CG-EnKF and LBFGS-EnKF methods on a 32×32 computational grid (left) and CG-EnKF and LBFGS-EnKF methods on a 128×128 computational grid (right) for ensemble size $N = 50$, in the heat equation example.

Figure 2.7 demonstrates that as the ensemble size increases, CG-EnKF approaches CG-VKF. We use ensemble sizes $N = 10$, 20, and 50 on grid size 32×32.

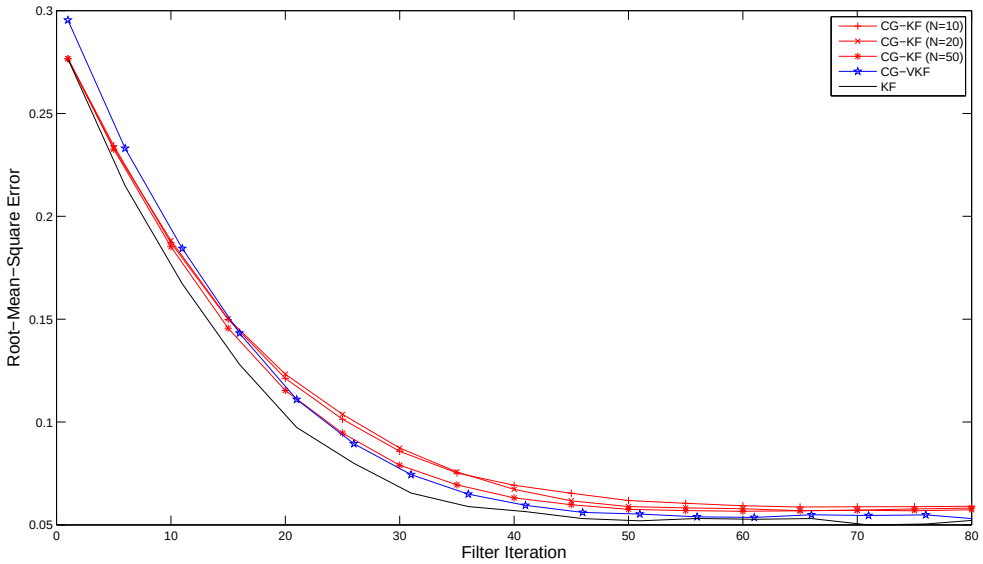

Figure 2.7: Root-mean-square error versus filter iteration to compare the ensemble sizes $N = 10, 20, 50$ for CG-EnKF with CG-VKF and KF using grid size 32×32, in the heat equation example.

2.5 Conclusions

In the Lorenz 95 test case, the CG-KF and CG-VKF methods tend to outperform LBFGS-KF and LBFGS-VKF when comparing root-mean-square error with the standard EKF. In EnKF for ensemble sizes of $N = 20$ and 50, CG-EnKF methods outperformed the LBFGS-EnKF methods in terms of root-mean-square error and were comparable to the standard EKF implementation. As ensemble size grows, the CG and LBFGS methods return more equivalent results as they approach EKF. Furthermore, as ensemble size increases, CG-EnKF approaches CG-VKF. The CG-based methods were at least twice as fast, computationally, as the LBFGS-based Kalman filter methods.

The CG approximation methods produced comparable root mean squared errors to KF in the heat example on the 32×32 grid, but were about two to eight times faster computationally than the respective LBFGS approximation methods, depending on the choice of Kalman filter.

On the 128×128 grid, KF was computationally infeasible but the CG methods outperformed the LBFGS methods in terms of root-mean-square error and computational time. CG-VKF tends to perform with root-mean-square error closer to KF than CG-KF, CG-EnKF, LBFGS-KF, LBFGS-VKF, and LBFGS-EnKF. CG-VKF also has the added benefit of being more computationally efficient than LBFGS-VKF due to only one CG optimization being required per iteration of the filter. The LBFGS methods converge slower than the CG methods.

We have seen that for large-scale examples, KF becomes computationally infeasible on standard desktop computers, and approximation methods are desirable. By integrating CG into the KF method, we are able to compute the quadratic minimization and find low-storage and low-rank approximations for the covariance and inverse-covariance matrices. For CG-VKF, we only require one optimization with CG per filter iteration, while for CG-KF, two CG optimizations are required per filter iteration for the covariance and inverse-covariance approximations. In the two test cases presented above, the CG methods are more computationally efficient to implement than KF and the LBFGS methods.

We also have seen that CG and its sampler may be implemented in EnKF for improvement in efficiency. The ensemble members computed are optimal in a certain Krylov subspace and implementation requires only one additional line of code to CG. CG-EnKF is computationally efficient and outperforms the LBFGS methods in the examples above, even in the cases where KF cannot be computed or standard EnKF performs poorly.

This method of incorporating CG and the CG sampler into EnKF yields a faster converging filter than when LBFGS is used. Moreover, with LBFGS, additional storage of the covariance approximation is required and an additional computation is required after the optimization iterations have terminated [90]. The CG approach is also very intuitive and requires only one additional line of inexpensive code within the CG iterations. Theory on the accuracy of the CG ensembles is discussed in [74] and to our knowledge, theory for the accuracy of the LBFGS covariance approximation is not yet developed.

Bibliography

[1] L. Armijo, *Minimization of Functions Having Lipschitz-Continuous First Partial Derivatives*, Pacific Journal of Mathematics **16** (1966), 1–3.

[2] H. Auvinen, Johnathan M. Bardsley, Heikki Haario, and T Kauranne, *Large-Scale Kalman Filtering Using the Limited Memory BFGS Method*, Electronic Transactions in Numerical Analysis **35** (2010), no. 9, 217–233.

[3] H. Auvinen, Johnathan M. Bardsley, Heikki Haario, and T. Kauranne, *The Variational Kalman Filter and an Efficient Implementation Using Limited Memory BFGS*, International Journal for Numerical Methods in Fluids **64** (2010), no. 3, 314–335.

[4] Owe Axelsson, *Iterative Solution Methods*, Cambridge University Press, New York, 1996.

[5] N. Ayache and O. Faugeras, *Maintaining a Representation of the Environment of a Mobile Robot*, IEEE Transactions of Robotics and Automation **5** (1989), no. 6, 804–819.

[6] Johnathan M. Bardsley, Albert Parker, Antti Solonen, and Marylesa Howard, *Krylov Space Approximate Kalman Filtering*, Numerical Linear Algebra with Applications (2013).

[7] Johnathan M. Bardsley, Antti Solonen, Albert Parker, Heikki Haario, and Marylesa Howard, *An Ensemble Kalman Filter Using the Conjugate Gradient Sampler*, International Journal for Uncertainty Quantification (2013).

[8] Dimitri P. Bertsekas, *Projected Newton Methods for Optimization Problems with Simple Constraints*, SIAM Journal of Control and Optimization **20** (1982), no. 2, 221–246.

[9] ———, *Nonlinear Programming*, Athena Scientific, Belmont, MA, 1995.

[10] S. Bonettini, R. Zanella, and L. Zanni, *A Scaled Gradient Projection Method for Constrained Image Deblurring*, Inverse Problems **25** (2009), no. 1, 015002.

[11] Léon Bottou, Olivier Bousquet, and Google Zürich, *The Tradeoffs of Large Scale Learning*, Advances in Neural Information Processing Systems **20**, 2008, pp. 161–168.

[12] Mark S. Boyce, Pierre R. Vernier, Scott E. Nielsen, and Fiona K.A. Schmiegelow, *Evaluatin Resource Selection Functions*, Ecological Modelling **157** (2002), 281–300.

[13] Leo Breiman and Philip Spector, *Submodel Selection and Evaluation in Regression. The X-Random Case*, International Statistical Review **60** (1992), no. 3, 291–319.

[14] Gerrit Burgers, Peter Jan van Leeuwen, and Geir Evensen, *Analysis Scheme in the Ensemble Kalman Filter*, Monthly Weather Review **126** (1998), 1719–1724.

[15] Christopher J.C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery **2** (1998), 121–167.

[16] Mark A. Cane, Alexey Kaplan, Robert N. Miller, Benyang Tang, Eric. C. Hackert, and Anthony J. Busalacchi, *Mapping Tropical Pacific Sea Level: Data Assimilation via Reduced State Kalman Filter*, Journal of Geophysical Research **101** (1996), 599–617.

[17] Bryan Catanzaro, Narayanan Sundaram, and Kurt Keutzer, *Fast Support Vector Machine Training and Classification on Graphics Processors*, Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 104–111.

[18] Olivier Chapelle, *Training a Support Vector Machine in the Primal*, Neural Computation **19** (2007), 1155–1178.

[19] R. Chatila and J. Laumond, *Position Referencing and Consistent World Modeling for Mobile Robots*, Proceedings of the IEEE International Conference on Robotics and Automation, 1985, pp. 138–145.

[20] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint, *LANCELOT: A FORTRAN Package for Large-Scale Nonlinear Optimization (Release A)*, Springer-Verlag, New York, 1992.

[21] Corinna Cortes and Vladimir Vapnik, *Support-Vector Networks*, Machine Learning **20** (1995), 273–297.

[22] Nello Cristianini and John Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press, Cambridge, United Kingdom, 2006.

[23] W. T. Crow and E. F. Wood, *The Assimilation of Remotely Sensed Soil Brightness Temperature Imagery into a Land Surface Model Using Ensemble Kalman Filtering: a Case Study Based on ESTAR Measurements During SGP97*, Advances in Water Resources **26** (2003), 137–149.

[24] Yu-Hong Dai and Roger Fletcher, *New Algorithms for Singly Linearly Constrained Quadratic Programs Subject to Lower and Upper Bounds*, Mathematical Programming **106** (2006), no. 3, 403–421.

[25] Dick P. Dee, *Simplification of the Kalman Filter for Meteorological Data Assimilation*, Quarterly Journal of the Royal Meteorological Society **117** (1991), 365–384.

[26] François-Zavier Le Dimet and Olivier Talagrand, *Variational Algorithms for Analysis and Assimilation of Meteorological Observations: Theoretical Aspects*, Tellus Series A **38** (1986), 97–110.

[27] M. W. M. Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F. Durrant-Whyte, and M. Csorba, *A Solution to the Simultaneous Localization and Map Building (SLAM) Problem*, IEEE Transactions on Robotics and Automation **17** (2001), 229–241.

[28] Jian-xiong Dong, Adam Krzyżak, and Ching Y. Suen, *A Fast SVM Training Algorithm*, Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines, 2002, pp. 53–67.

[29] Petros Drineas and Michael W. Mahoney, *On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning*, Journal of Machine Learning Research **6** (2005), 2153–2175.

[30] V. Echevin, P. D. Mey, and G. Evensen, *Horizontal and Vertical Structure of the Representer Functions for Sea Surface Measurements in a Coastal Circulation Model*, Journal of Physical Oceanography **30** (2000), 2627–2635.

[31] Bradley Efron, *Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation*, Journal of the American Statistical Association **78** (1983), no. 382, 316–331.

[32] Bradley Efron and Robert Tibshirani, *Improvements on Cross-Validation: The .632+ Bootstrap Method*, Journal of the American Statistical Association **92** (1997), no. 438, 548–560.

[33] Geir Evensen, *Sequential Data Assimilation with a Nonlinear Quasi-Geostrophic Model Using Monte Carlo Methods to Forecast Error Statistics*, Journal of Geophysical Research **99** (1994), 10143–10162.

[34] ———, *The Ensemble Kalman Filter: Theoretical Formulation and Practical Implementation*, Ocean Dynamics **53** (2003), no. 4, 343–367.

[35] ———, *Data Assimilation: The Ensemble Kalman Filter*, Springer, Berlin, 2007.

[36] Michael C. Ferris and Todd S. Munson, *Interior-Point Methods for Massive Support Vector Machines*, SIAM Journal of Optimization **13** (2003), no. 3, 783–804.

[37] Shai Fine and Katya Scheinberg, *Efficient SVM Training Using Low-Rank Kernel Representations*, Journal of Machine Learning Research **2** (2002), 243–264.

[38] Michael Fisher, *Development of a Simplified Kalman Filter*, Technical Report 260, Eurpoean Center for Medium Range Weather Forecasting Technical Memorandum, 1998.

[39] Michael Fisher and Erik Andersson, *Developments in 4D-var and Kalman Filtering*, Technical Report 347, Eurpoean Center for Medium Range Weather Forecasting Technical Memorandum, 2001.

[40] R. Fletcher, *Practical Methods of Optimization*, John Wiley and Sons, Inc. Great Britian, 1987.

[41] Vojtěch Franc and Soeren Sonnenburg, *Optimized Cutting Plane Algorithm for Support Vector Machines*, Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 320–327.

[42] K. F. Gauss, *Theory of Motion of the Heavenly Bodies*, Dover, New York, 1963.

[43] I. Yu Gejadze, François-Zavier Le Dimet, and V. Shutyaev, *On Analysis Error Covariances in Variational Data Assimilation*, SIAM Journal of Scientific Computing **30** (2008), 1847–1874.

[44] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, London, UK, 1981.

[45] Gene H. Golub and Charles F. Van Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, Baltimore, Maryland, 1996.

[46] R. Grønnevik and G. Evensen, *Application of Ensemble Based Techniques in Fish-Stock Assessment*, Sarsia **86** (2001), 517–526.

[47] Gudmundur Gudmundsson, *Time Series Analysis of Catch-At-Age Observations*, Journal of the Royal Statistical Society, Series C (Applied Statistics) **43** (1994), no. 1, 117–126.

[48] D. J. Hand, *Discrimination and Classification*, John Wiley & Sons, Chichester, 1981.

[49] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer-Verlag, New York, 2001.

[50] M. F. Hutchinson, *A Stochastic Estimator of the Trace of the Influence Matrice for Laplacian Smoothing Splines*, Communication in Statistics, Simulation and Computation **19** (1989), no. 2, 433–450.

[51] Thorsten Joachims, *Training Linear SVMs in Linear Time*, Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006, pp. 217–226.

[52] Rudolph E. Kalman, *A New Approach to Linear Filtering and Prediction Problems*, Transactions of the American Society of Mechanical Engineers – Journal of Basic Engineering **82** (1960), no. Series D, 35–45.

[53] C. T. Kelley, *Iterative Methods for Optimization*, SIAM, Philadelphia, 1999.

[54] Ron Kohavi, *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*, Proceedings of the 14th International Joint Conference on Artificial Intelligence, 1995, pp. 1137–1143.

[55] Sturla Kvamsdal and Leif Kristoffer Sandal, *The Ensemble Kalman Filter in Bioeconomics*, Technical Report 2012/5, Department of Finance & Management Science, Norwegian School of Economics, 2012.

[56] C. Lanczos, *Solutions of Linear Equations by Minimized Iterations*, Journal of Research of the National Bureau of Standards **49** (1920), 3.

[57] ———, *An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators*, Journal of Research of the National Bureau of Standards **45** (1950), 255–282.

[58] Changki Lee and Myung-Gil Jang, *Fast Training of Structured SVM Using Fixed-Threshold Sequential Minimal Optimization*, Electronics and Telecommunications Research Institute Journal **31** (2009), no. 2, 121–128.

[59] John J. Leonard and Hans Jacob S. Feder, *Experimental Analysis of Adaptive Concurrent Mapping and Localization Using Sonar*, The Sixth International Symposium on Experimental Robotics, 2000, pp. 297–306.

[60] Hong Li, Eugenia Kalnay, and Takemasa Miyoshi, *Simultaneous Estimation of Covariance Inflation and Observation Errors within an Ensemble Kalman Filter*, Quarterly Journal of the Royal Meteorological Society **135** (2009), 523–533.

[61] A. C. Lorenc, *Analysis Methods ofr Numerical Weather Prediction*, Quarterly Journal of the Royal Meteorological Society **112** (1986), 1177–1194.

[62] E. N. Lorenz, *Predictability: A Problem Partly Solved*, Proceedings of the Seminar on Predicability, Eurpoean Center on Medium Range Weather Forecasting **1** (1996), 1–18.

[63] E. N. Lorenz and K. A. Emanuel, *Optimal Sites for Supplementary Weather Observations: Simulation with a Small Model*, Journal of Atmospheric Science **55** (1998), 399–414.

[64] H. Madsen and R. Cañizares, *Comparison of Extended and Ensemble Kalman Filters for Data Assimilation in Coastal Area Modeling*, International Journal for Numerical Methods in Fluids **31** (1999), 961–981.

[65] Hongying Meng, John Shawe-Taylor, Sandor Szedmak, and Jason D. R. Farquhar, *Support Vector Machine to Synthesise Kernels*, Proceedings of the First international Conference on Deterministic and Statistical Methods in Machine Learning, 2005, pp. 242–255.

[66] Zhyong Meng and Fuqing Zhang, *Tests of an Ensemble Kalman Filter for Mesoscale and Regional-Scale Data Assimilation. Part II: Imperfect Model Experiments*, Monthly Weather Review **135** (2007), 1403–1423.

[67] Gérard Meurant, *The Lanczos and Conjugate Gradient Algorithms*, SIAM, Philadelphia, 2006.

[68] H. L. Mitchell, P. L. Houtemaker, and G. Pellerin, *Ensemble Size, and Model-Error Representation in an Ensemble Kalman Filter*, Monthly Weather Review **130** (2002), no. 2791–2808.

[69] Jorge J. Moré and Gerardo Toraldo, *On the Solution of Large Quadratic Programming Problems with Bound Constraints*, SIAM Journal of Optimization **1** (1991), no. 1, 93–113.

[70] Girish Nehate, *Solving Large Scale Support Vector Machine Problems Using Matrix Splitting and Decomposition Methods*, Ph.D. Thesis, Manhattan, KS, 2006.

[71] Jorge Nocedal and Stephen J. Wright, *Numerical Optimization*, Springer Series in Operations Research, Springer-Verlag, New York, 1999.

[72] E. Ott, B. R. Hunt, I. Szunyogh, A. V. Simin, E. J. Kostelich, M. Corazza, E. Kalnay, O. Pati, and J. A. Yorke, *A Local Ensemble Kalman Filter for Atmospheric Data Assimilation*, Tellus A **56** (2004), 415–428.

[73] J. H. Park and A. Kaneko, *Assimilation of Coastal Acoustic Tomography Data into a Barotropic Ocean Model*, Geophysical Research Letters **27** (2000), 3373–3376.

[74] Albert Parker and Colin Fox, *Sampling Gaussian Distributions in Krylov Spaces with Conjugate Gradients*, SIAM Journal on Scientific Computing **34** (2012), no. 3, B312–B334.

[75] Beresford N. Parlett, *Symmetric Eigenvalue Problem*, Prentice Hall, New Jersey, 1980.

[76] A. S. Paul and E. A. Wan, *Dual Kalman Filters for Autonomous Terrain Aided Navigation in Unknown Environments*, Proceedings of the IEEE International Joint Conference on Neural Networks, 2005, pp. 2784 –2789.

[77] John C. Platt, *Fast Training of Support Vector Machines Using Sequential Minimal Optimization*, Advances in Kernel Methods: Support Vector Learning, 1999, pp. 185–208.

[78] Alvin C. Rencher and G. Bruce Schaalje, *Linear Models in Statistics*, 2nd ed. John Wiley & Sons, Inc. New Jersey, 2008.

[79] John A. Richards and Xiuping Jia, *Remote Sensing Digital Analysis, An Introduction*, 4th ed. Springer-Verlag, Germany, 2006.

[80] H. Rue and L. Held, *Gaussian Markov Random Fields: Theory and Applications*, Chapman & Hall/CRC, New York, 2005.

[81] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press, Manchester, UK, 1992.

[82] William Sacher and Peter Bartello, *Sampling Errors in Ensemble Kalman Filtering. Part I: Theory*, Monthly Weather Review **136** (2008), 3035–3049.

[83] Katya Scheinberg, *An Efficient Implementation of an Active Set Method for SVMs*, Journal of Machine Learning Research **7** (2006), 2237–2257.

[84] Michael K. Schneider and Alan S. Willsky, *Krylov Subspace Algorithms for Space-Time Oceanography Data Assimilation*, In Proceedings of IEEE International Geoscience and Remote Sensing Symposium, 2000.

[85] ———, *Krylov Subspace Estimation*, SIAM Journal on Scientific Computing **22** (2000), no. 5, 1840–1864.

[86] ———, *A Krylov Subspace Method for Covariance Approximation and Simulation of Random Processes and Fields*, Multidimensional Systems and Signal Processing **14** (2003), 295–318.

[87] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter, *Pegasos: Primal Estimated Sub-Gradient Solver for SVM*, Mathematical Programming **127** (2011), no. 1, 3–30.

[88] A. Shilton, M. Palaniswami, Senior Member, D. Ralph, A. C. Tsoi, and Senior Member, *Incremental Training of Support Vector Machines*, IEEE Transactions on Neural Networks **16** (2005), 114–131.

[89] G. Sleijpen and A. V. D. Sluis, *Further Results on the Convergence Behavior of Conjugate Gradients and Ritz Values*, Linear Algebra and its Applications **246** (1996), 233–278.

[90] Antti Solonen, Heikki Haario, Janne Hakkarainen, Harri Auvinen, Idrissa Amour, and Tuomo Kauranne, *Variational Ensemble Kalman Filtering Using Limited Memory BFGS*, Electronic Transactions on Numerical Analysis **39** (2012), 271–285.

[91] H. W. Sorenson, *Least Squares Estimation: From Gauss to Kalman*, IEEE Spectrum **7** (1970), no. 7, 63–68.

[92] Xiangjun Tian, Zhenghui Xie, and Aiguo Dai, *An Ensemble-Based Explicit Four-Dimensional Variational Assimilation Method*, Journal of Geophysical Research **113** (2008), no. D21124, 1–13.

[93] J. Tshimanga, S. Gratton, A. T. Weaver, and A. Sartenaer, *Limited-Memory Preconditioners, with Application to Incremental Four-Dimensional Variational Data Assimilation*, Quarterly Journal of the Royal Meteorological Society **134** (2008), 751–769.

[94] Vladimir Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.

[95] ———, *Statistical Learning Theory*, John Wiley & Sons, Inc. New York, 1998.

[96] Fabrice Veersé, *Variable-Storage Quasi-Newton Operators for Modeling Error Covariances*, Proceedings of the 3rd WMO International Symposium on Assimilation of Observations in Meteorology and Oceanography, 1999, pp. 7–11.

[97] Fabrice Veersé, D. Auroux, and M. Fisher, *Limited-Memory BFGS Diagonal Preconditioners for a Data Assimilation Problem in Meteorology*, Optimization and Engineering **1** (2000), 323–339.

[98] D. Watkins, *Fundamentals of Matrix Computations*, 2nd ed. Wiley, New York, 2002.

[99] Greg Welch and Gary Bishop, *An Introduction to the Kalman Filter*, Technical Report TR 95-041, Department of Computer Science, University of North Carolina at Chapel Hill, 1995.

[100] Jeffrey S. Whitaker and Thomas M. Hamill, *Ensemble Data Assimilation without Perturbed Observations*, Montlhy Weather Review **130** (2002), 1913–1924.