Kernel Predictive Linear Gaussian Models for Nonlinear Stochastic Dynamical Systems

David Wingate WINGATED@UMICH.EDU
Satinder Singh BAVEJA@UMICH.EDU
Computer Science and Engineering Department, 2260 Hayward Ave., University of Michigan, Ann Arbor, MI 48109 USA

Abstract

The recent Predictive Linear Gaussian model (or PLG) improves upon traditional linear dynamical system models by using a *predictive* representation of state, which makes consistent parameter estimation possible without any loss of modeling power and while using fewer parameters. In this paper we extend the PLG to model stochastic, nonlinear dynamical systems by using kernel methods. With a Gaussian kernel, the model admits closed form solutions to the state update equations due to conjugacy between the dynamics and the state representation. We also explore an efficient sigma-point approximation to the state updates, and show how all of the model parameters can be learned directly from data (and can be learned on-line with the Kernel Recursive Least-Squares algorithm). We empirically compare the model and its approximation to the original PLG and discuss their relative advantages.

1. Introduction

Model building is an important part of AI. Many agent-environment interactions can be modeled as dynamical systems, spanning things as diverse as the dynamics of a biped walking, the trajectory of a missile, or the shifting fish population in a lake. This paper builds on several recent advances in the machine learning community to motivate, derive and analyze a new nonlinear dynamical system model which combines predictive representations of state, kernel methods and statistical state updating.

In the machine learning community, models with *predictive* representations of state have recently attracted considerable interest. These models replace the traditional notion of state (which is typically a latent, unobservable quantity) by a set of (possibly action-conditional) predictions about the future. This is permissible because the most general definition of "state" is *any* sufficient statistic for history—in our case, a finite set of predictions about the future summarize our infinite past.

Most predictive models, including PSRs (Littman et al., 2001) and OOMs, have been defined for discrete observations, typically in the realm of discrete partially observable Markov decision processes. The Predictive Linear Gaussian (or PLG) model was introduced by Rudary et al. (2005) as an extension of PSRs to the case of continuous observations, and has several advantages when compared to traditional state-space models. First, the entire model is defined strictly in terms of statistics about observable quantities. This means that parameters of the model have definite meaning with respect to the observed data, which leads to *consistent* parameter estimation procedures: estimated parameters will asymptotically converge to their *true* value, which is a stronger guarantee than the guarantees which accompany the usual EM-style algorithms used to learn state-space model parameters. Second, the PLG estimation procedure works particularly well as the dimension of the system increases. Third, the PLG model strictly subsumes two popular linear dynamical system models: the celebrated Kalman filter (Kalman, 1960), and autoregressive time-series (or ARMA) models (Pandit & Wu, 1983). Finally, the PLG is just as compact as state-space models (meaning that only n predictions are necessary to model an n-dimensional LDS) and requires fewer parameters. There are, however, two drawbacks to the PLG model. First, it is only capable of modeling *linear* systems, and second, it is limited to cases where the observation is a single scalar.

The main contribution of this paper is a new model that extends the PLG to the nonlinear case by using kernels. We first present the general model, analyzing in depth the special case of the Gaussian kernel. We then present a computationally efficient approximation which is almost as accurate as the exact method. We discuss on-line and off-line parameter estimation, and conclude by empirically showing that both the model and its approximation are viable.

Appearing in *Proceedings of the 23^{rd} International Conference on Machine Learning*, Pittsburgh, PA, 2006. Copyright 2006 by the author(s)/owner(s).

1.1. Linear Dynamical Systems

A discrete time, linear dynamical system (LDS) is defined by a state update equation $x_{t+1} = Ax_t + \eta$, where $x_t \in \mathbb{R}^n$ is the state at time t, $A \in \mathbb{R}^{n\times n}$ is a transition matrix and $\eta \sim \mathcal{N}(0, Q)$ is mean-zero Gaussian noise (where $Q \in \mathbb{R}^{n\times n}$). Often, we are not able to observe the state directly. Many LDSs define a companion observation process, in which observations are linear functions of the true state: $y_t = Hx_t + \mathcal{N}(0, R)$, where $H \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{m\times m}$. There are generally no restrictions on H; in particular, it may collapse an n-dimensional state into a lower-dimensional (or even scalar) observation.

In some cases, the parameters of the LDS are not known, and must be estimated from data. Because the true state is never observed directly, these procedures typically involve some sort of expectation maximization (Ghahramani & Hinton, 1996) or maximum likelihood estimation, and suffer from several problems: they can get stuck in local maxima or minima, and are somewhat slow.

1.2. The Predictive Linear Gaussian Model

In the PLG model, we never refer to an unobservable or latent state x_t. Instead, we capture state as statistics about a random variable Z_t, which is defined as a vector of random variables predicting *future* observations. We associate each future observation at time $t+i$ with a random variable Y_{t+i} (all observations are scalars), and collect the next n of them into the vector $Z_t = [Y_{t+1} \cdots Y_{t+n}]^T$, as illustrated in Figure 1. These n variables are jointly Gaussian, with mean μ_t and covariance Σ_t. It is these two statistics that are used as the state of the system.

The system dynamics are defined by a special equation:

$$Y_{t+n+1} = \langle g, Z_t \rangle + \eta_{t+n+1}. \quad (1)$$

where $\langle\cdot\rangle$ denotes inner product, $g \in \mathbb{R}^n$ is a linear trend and $\eta_{t+n+1} \in \mathbb{R}$ is a noise term. The importance of modeling Y_{t+n+1} as a function of Z_t will be explained in the next section. The noise term is mean-zero with a fixed variance: $\eta_{t+n+1}|h_t \sim \mathcal{N}(0, \sigma_\eta^2)$, but is allowed to covary with the next n observations in a way that is independent of history: $\mathrm{Cov}[Z_t, \eta_{t+n+1}|h_t] = C$. The PLG's representational power comes from this special noise term: the fact that it covaries with future observations gives it the infinite memory of the LDS—an observation can have an effect far in the future through the chain of influence created by the correlation in the noise terms. The parameters of the PLG model are the trend g and the noise statistics C and σ_η^2.

Rudary et al. (2005) showed how the state of this system can be recursively updated in closed form, and that this model is equivalent in modeling power to the Kalman filter. They also provided a consistent estimation procedure to learn the parameters, as well an algorithm for converting parameters from an LDS to the equivalent PLG.

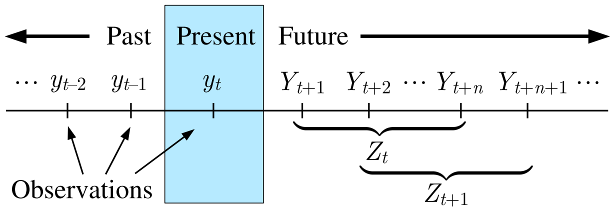

Figure 1. A timeline illustrating the random variables we use.

1.3. Updating State: Extend and Condition

We will now discuss our general strategy for updating state and modeling nonlinear dynamics; the next section discusses our specific model. As in the PLG, we restrict ourselves to scalar observations, but emphasize that this does not restrict the dimensionality of the underlying state space.

Modeling the system dynamics requires determining how to update the state of the system. The problem can be stated thus: given a state at time t, how can we incorporate an observation $Y_{t+1} = y_{t+1}$ to compute our state at time $t+1$? Our strategy is to *extend and condition*, as follows.

We begin with state extension. We assume that we have the state at time t, represented by μ_t and Σ_t. These statistics describe $Z_t \sim \mathcal{N}(\mu_t, \Sigma_t)$, which is an $n-$dimensional Gaussian describing the next n observations. We will extend this variable to include the variable Y_{t+n+1} (ensuring that it is still jointly Gaussian), creating a temporary $(n+1)$-dimensional Gaussian. *In the PLG, Y_{t+n+1} is a* linear *function of Z_t, but we will allow it to be a* nonlinear *function, which allows us to model nonlinear dynamics.* In order to extend Z_t to include the variable Y_{t+n+1}, we must compute three terms, which are $E_t = \mathrm{E}[Y_{t+n+1}]$, $C_t = \mathrm{Cov}[Y_{t+n+1}, Z_t]$ and $V_t = \mathrm{Var}[Y_{t+n+1}]$:

$$\begin{pmatrix} Z_t \\ Y_{t+n+1} \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_t \\ E_t \end{pmatrix}, \begin{pmatrix} \Sigma_t & C_t \\ C_t^T & V_t \end{pmatrix}\right].$$

We will then condition on the observation y_{t+1}, which will result in another $n-$dimensional Gaussian random variable describing $[Y_{t+2} \cdots Y_{t+n+1}]^T = Z_{t+1}$ (conditioning is done with standard techniques on multivariate Gaussians, for which it is well-known that the result is Gaussian). This results in $\mathrm{E}[Y_{t+2} \cdots Y_{t+n+1}] = \mathrm{E}[Z_{t+1}] = \mu_{t+1}$, along with $\mathrm{Cov}[Z_{t+1}] = \Sigma_{t+1}$, which are precisely the statistics representing our new state. Figure 1 illustrates this.

Rolling together the construction of the temporary Gaussian and the conditioning yields the complete state update:

$$\mu_{t+1} = \mu_{t+1}^- + K_t(y_{t+1} - e_1^T \mu_t) \quad (2)$$

$$\Sigma_{t+1} = (I - K_t e_1^T)\Sigma_{t+1}^- \quad (3)$$

where $\Sigma^-_{t+1} = I^-\Sigma_t I^{-T} + I^- C_t + C_t^T I^{-T} + e_n e_n^T V_t$, $K_t = \Sigma^-_{t+1} e_1 (e_1^T \Sigma_t e_1)^{-1}$, $\mu^-_{t+1} = I^- \mu_t + e_n E_t$,

$$I^- = \left(\begin{array}{c|c} \mathbf{0} & I_{n-1} \\ \hline \multicolumn{2}{c}{\mathbf{0}} \end{array} \right),$$

and e_i is the i-th column of the identity matrix. Note that Eqs. (2) and (3) have the same form as the Kalman filter.

2. The Kernel PLG Model

We will now extend the PLG model to handle nonlinear dynamics by allowing Y_{t+n+1} to be a nonlinear function of Z_t. We accomplish this by invoking the kernel trick. As discussed in the previous section, all that is needed to maintain state is the state extension and the observation conditioning, so the computation of E_t, C_t and V_t, combined with the construction and conditioning equations (2) and (3), constitutes the complete model.

Computing E_t, C_t and V_t in closed form for arbitrary kernels is impossible. Section 2.1 will therefore present the general model and analyze the special case where the kernels are Gaussian, which will make closed-form solutions possible by virtue of the conjugacy properties of Gaussians. Section 2.2 will then relax this assumption by presenting an approximation method which works for all kernels.

2.1. The KPLG Model

Our model is named the KPLG, or "Kernel Predictive Linear Gaussian" model, which defines the state extension as

$$Y_{t+n+1} = \sum_{j=1}^{J} \alpha_j K(\xi_j, Z_t) + \eta_{t+n+1}, \tag{4}$$

where $K()$ is our kernel. The $\xi_j \in \mathbb{R}^n$ are points that could come from a number of sources: they may come from training data, be derived analytically, or be randomly generated.

This is the most obvious way to kernelize the original PLG algorithm, because we have employed the standard technique of rewriting the linear trend g as a weighted combination of data points; this is possible by virtue of the Representer Theorem (Kimeldorf & Wahba, 1971):

$$\begin{aligned} Y_{t+n+1} &= \langle g, Z_t \rangle + \eta_{t+n+1} \\ &= \langle \sum_j \alpha_j \xi_j, Z_t \rangle + \eta_{t+n+1} \\ &= \sum_j \alpha_j \langle \xi_j, Z_t \rangle + \eta_{t+n+1} \\ &= \sum_j \alpha_j K(\xi_j, Z_t) + \eta_{t+n+1} \end{aligned}$$

Since this is a linear basis function model (with the kernels $K(\xi_j, \cdot)$ acting as the basis functions), we will refer to the ξ_j's as basis function centers. The model strictly generalizes the PLG, since using the linear kernel recovers it[1]. Note that η_{t+n+1} has the same properties as in the PLG.

With a Gaussian kernel, we can analytically derive expressions for E_t, C_t and V_t. Appendix A contains the lemmas and identities needed for their derivation, and a summary of what the terms mean (all summations are from 1 to J):

$$E_t = \sum_j \alpha_j K'_{tj} \tag{5}$$

$$C_t = \sum_j \alpha_j K'_{tj} (\mu'_{tj} - \mu_t)^T + C^T \tag{6}$$

$$\begin{aligned} V_t = {} & \sum_i \sum_j K^\dagger_{tij} \alpha_i \alpha_j - E_t^2 + \sigma^2_\eta \\ & + 2(\sum_j \alpha_j K'_{tj} (\mu'_{tj} - \mu_t)^T) \Sigma_t^{-1} C. \end{aligned} \tag{7}$$

The parameters of this model are therefore the ξ_j's, the α_j's, C, and σ^2_η. In the case of a Gaussian kernel, we allow an additional parameter ϕ_j (which is the covariance matrix of the Gaussian) and write the kernel as $K(\xi_j, Z_t; \phi_j)$. We use a fully normalized Gaussian for analytical purposes.

2.2. A Sigma-Point Approximation

With Gaussian kernels, the KPLG model is analytically tractable. While this is appealing, there are some computational liabilities. In particular, computing V_t is a $O(J^2)$ operation (where J is the number of basis functions; see the double summation of Eq. 7), which is prohibitively complex, especially since J typically scales exponentially with the dimension n. This motivates some sort of fast approximation. We would also like the approximation to relax the restriction to Gaussians, and free us to use arbitrary kernels. The following method accomplishes both goals (although exploring arbitrary kernels is left for future research).

Sigma-point approximations, or "unscented transformations" (Julier & Uhlmann, 1996), are a general method of propagating an arbitrary distribution through a nonlinear function. The method is conceptually simple, and should be thought of as a deterministic sampling approach. Suppose we are given a random variable $Y = f(Z, \eta)$ that is a nonlinear function of another random variable Z and a mean-zero Gaussian noise term η. Instead of recording the distribution information of Z in terms of a mean and covariance, we represent the same information with a small, carefully chosen number of *sigma points*. These points are selected so that they have the same mean and covariance as Z (in fact, they are the minimal such set), but the advantage is that they can be propagated *directly* through the function $f()$. We then compute the posterior statistics of the propagated points to approximate Y.

[1]This model is also closely related to an RBF network.

- Construct a RV relating predictions and noise:

$$P = \begin{pmatrix} Z_t \\ \eta_{t+n+1} \end{pmatrix} \sim N\left[\begin{pmatrix} \mu_t \\ 0 \end{pmatrix}, \begin{pmatrix} \Sigma_t & C \\ C^T & \sigma^2_\eta \end{pmatrix}\right]$$

- Ensure that $\mathrm{Cov}[P]$ is symmetric positive definite.
- Construct a set of $2(n+1)$ sigma points:

$$[z_t^{(2i-1)}, \eta_{t+n+1}^{(2i-1)}]^T = \mathrm{E}[P] + (\sqrt{(n+1)\mathrm{Cov}[P]})_i$$

$$[z_t^{(2i)}, \eta_{t+n+1}^{(2i)}]^T = \mathrm{E}[P] - (\sqrt{(n+1)\mathrm{Cov}[P]})_i$$

- Propagate each point: $y_{t+n+1}^{(i)} = f(z_t^{(i)}, \eta_{t+n+1}^{(i)})$
- Compute the empirical mean and covariance:

$$E_t = \frac{1}{2(n+1)} \sum_{i=1}^{2(n+1)} y_{t+n+1}^{(i)}$$

$$V_t = \frac{1}{2(n+1)} \sum_{i=1}^{2(n+1)} (y_{t+n+1}^{(i)} - \mathrm{E}[Y_{t+n+1}])^2$$

$$C_t = \frac{1}{2(n+1)} \sum_{i=1}^{2(n+1)} (z_t^{(i)} - \mu_t)(y_{t+n+1}^{(i)} - \mathrm{E}[Y_{t+n+1}])^T$$

Figure 2. The sigma-point approximation algorithm.

There are many advantages to sigma-point approximations. First, they are a good match for our needs: we only want first and second-order moments of the posterior (which they are designed to provide), and their strongest optimality guarantees are when Z is normally distributed (as it is in our case). They are provably accurate to at least a second order approximation of the dynamics for any distribution on Z and any nonlinearity, and are accurate to third order for a Gaussian distribution on Z and any nonlinearity, while fourth order terms can sometimes be corrected as well. They can flexibly incorporate noise and other constraints into $f()$. They are simple to implement because no analytical derivatives (such as Jacobians or Hessians) are required. They are also efficient: they require $2(n+1)J$ kernel evaluations at each timestep, which is far smaller than the $O(J^2)$ matrix operations required by the KPLG.

Sigma-point approximations should not be confused with particle filters. While they are similar in spirit, there are several important differences. Particle filters typically allow a multi-modal distribution over states, while sigma-point approximations require a Gaussian; it is the Gaussian assumption which gives the sigma-point approximation its strong theoretical guarantees with a small number of points. Also, where particle filters use random sampling, sigma-point approximations use deterministic sampling.

The algorithm is shown in Figure 2. If we let $f()$ be the state extension defined by the KPLG model (Eq. 4), then the final terms computed may be used in place of the analytical values of E_t, V_t, and C_t.

2.3. Complexity and Generalization

The KPLG model has high complexity: computing E_t and C_t is $O(Jn^3)$, but computing V_t is $O(J^2n^3)$ (and can be numerically unstable). There are other ways to estimate these terms, besides the sigma-point approximations. Nearest-neighbor style methods, such as the Fast Gauss Transform (Yang et al., 2003), are one possibility, and would also allow $O(Jn^3)$ computations, although these methods only work well for small n.

In the case of Gaussian kernels, the model can suffer from generalization problems. Because the Gaussians have local receptive fields, the state extension equation (Eq. 4) will return something close to zero for all states outside the training region. This implies a very strong, and somewhat strange, prior on the system dynamics. We believe that renormalized kernels (Hastie et al., 2001) will improve generalization, but this is left for future research.

2.4. Comparison to Nonlinear Autoregression

There is a significant difference between the KPLG model and an n-th order kernel autoregressive (KAR) model. The KAR model is $\mathrm{E}[Y_{t+1}] = \sum_j \alpha_j K(\xi_j, z_{t-n})$, which states that the next observation is predicted to be a nonlinear function of the *past* n observations z_{t-n}. It has a similar functional form to our predictive model: the same kernels, basis function centers, and coefficients are used, and it can be trained using similar methods as the KPLG (see Section 3). However, their differences are as important as their similarities. In particular, KAR assumes that n past observations constitute state, while the KPLG can summarize a potentially infinite amount of history into its predictions. These differences are what accounts for the empirical improvement of KPLG over KAR reported in Section 4.

3. Model Estimation

The KPLG model requires several parameters: the dimension n of the system, the basis function centers ξ_j and weights α_j, as well as the noise statistics C and σ^2_η. In the case of a Gaussian kernel, the covariance matrix ϕ_j is also required. The next two sections discuss off-line and on-line methods of estimating these parameters.

3.1. The Off-Line Case

In this scenario, we are interested in learning the parameters from training data. This data will be given as a set of

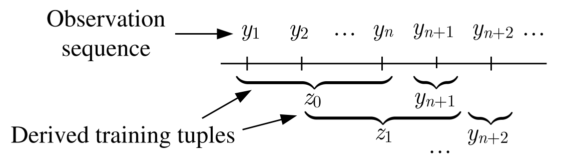

Figure 3. Extracting training pairs from a training trajectory.

trajectories from the system, with each trajectory consisting of at least $n+1$ sequential observations. We will slice these trajectories into training pairs (z_i, y_{i+n+1}) where $z_i \in \mathbb{R}^n$ is a vector of n successive observations (representing a noisy sample of some Z_t), and $y_{i+n+1} \in \mathbb{R}$ is the $(n+1)$-th observation (a sample of the corresponding Y_{t+n+1}, or the state extension). Each trajectory is sliced into all such pairs and collected into a set S. Figure 3 illustrates this process.

Model Order Selection. We must first estimate the order of the model, which includes the system dimension n and the number of basis functions J. For our experiments, we use cross-validation to select parameters from a set of likely candidates. However, there is nothing unusual about our model or estimation needs, meaning that many existing techniques are also suitable[2].

Finding Basis Function Parameters. Next, we must determine the basis function centers ξ_j and covariance matrices ϕ_j. We tested three methods: random selection, dictionary-based selection (explained in Section 3.2), and expectation maximization. For random selection, we set each ξ_j to be a random training sample z_i, we set $\phi_j = \sigma_\phi^2 I$, and we used cross-validation to select σ_ϕ^2. Expectation maximization (EM) is a well-known method for estimating mixture of Gaussian parameters. We will here summarize our experiments with EM by saying that it did not appear to offer any advantage over the other two methods, and since it was computationally more intensive, it was dropped. Again, many other methods are also suitable[3].

Estimating Coefficients. Given ξ_j and ϕ_j, finding the α_j's can be viewed as a simple kernel regression problem. It can be solved with a linear least squares approach, or more sophisticated methods such as support-vector regression (Shawe-Taylor & Cristianini, 2004). We chose regularized least-squares. We construct a regression matrix $A + \lambda I$, where $A_{ij} = K(\xi_j, z_i; \phi_j)$ and λ is the regularization coefficient. Let Y be a vector collecting all the y_{i+n+1}'s. Then, the optimal coefficients α are given by $\alpha = A^\dagger Y$, where $\dagger$ denotes the pseudo-inverse (giving a minimum-norm solution to an underconstrained system, and a least-squares solution to an overconstrained system).

3.2. The On-Line Case: KRLS

The previous section discussed the selection of the basis function centers ξ_j and their weights α_j as two separate problems. However, both steps may be combined into a single step by using the Kernel Recursive Least-Squares (KRLS) algorithm of Engel et al. (2004). As noted, finding the weights is a least-squares kernel regression problem, which KRLS is designed to solve. However, it does so in a *recursive* way: instead of presenting all of the training pairs simultaneously, they are presented one at a time, and the algorithm updates the resulting weights in a way that is independent of the total number of pairs used (in our case, it is equivalently independent of time).

Because of the nonlinear kernels involved, the KRLS algorithm is more complicated than the standard RLS algorithm. KRLS uses a *dictionary* to perform on-line sparsification of what would otherwise be prohibitively large matrices. The dictionary stores points (the z_i's) whose features are linearly independent of each other; points that are almost linearly dependent on the dictionary points are discarded (where "almost" is a tunable threshold). Thus, the dictionary holds a set of points which approximately linearize the feature space; equivalently, they can be thought of as points which are spaced evenly enough so that the features of new points will be a linear combination of them.

It is these dictionary points that we use as the basis function centers ξ_j, and the corresponding weights as the α_j's. This gives even coverage to the feature space, and can be controlled by only a single additional parameter. Note that KRLS is an instance of the KAR model (see Section 2.4).

3.3. Learning Noise Parameters

Either the off-line or the on-line techniques provide basis function centers, covariances and weights, allowing us to now estimate the noise parameters. For these, we can use sample statistics. Assume we have a set S of training pairs (z_i, y_{i+n+1}). In the off-line case, this may be the training set; in the on-line case, this set may be collected during training, or once the basis function parameters have been fixed. Let $\eta_i = y_{i+n+1} - \sum_j \alpha_j K(\xi_j, z_i; \phi_j)$. Then, the estimated noise term is $\widehat{\sigma_\eta^2} = \frac{1}{|S|-1} \sum_i (\eta_i)^2$. To estimate C, we run the algorithm on the training data (or run it on-line) with $C = 0$ and collect an estimate of μ_t at each t. We then compute $\mathrm{Cov}[Z_t \eta_{t+n+1}] = \mathrm{E}[(Z_t - \mu_t)(\eta_{t+n+1})]$, which is simply $\widehat{C_k} = \frac{1}{|S|-1} \sum_i (z_i - \mu_i)_k \eta_i$. Extending these estimators to be fully on-line is left as future work.

[2]These include growing and pruning methods, stacked generalization, regularized complexity criteria, or statistical tests such as Z tests (Bishop, 1995; Pandit & Wu, 1983).

[3]These include nonlinear gradient methods (such as Gauss-Newton or Marquardt-Levenburg), re-estimation methods (such as expectation maximization), adaptive k-means clustering, stochastic sequential estimation, or cross-validation (generalized, leave one out, or k-fold) (Hastie et al., 2001; Bishop, 1995).

	Problem	Best (10-CV)	Best (5-CV)	Best (direct)
LDSs	Rotation	KPLG-SP	KPLG-SP	KPLG/KPLG-SP
	Biped	KPLG/KAR	KPLG-SP	KPLG/KPLG-SP/KAR
	Peanut	KAR/PLG	KPLG-SP	KPLG/KPLG-SP
	NB3	KAR/PLG	KAR/PLG	KPLG-SP/KAR/PLG
	Spring	KAR	KAR	KAR
Time Series	M.G.	KAR	KAR	KAR
	Leuven	KAR	KAR	KAR
	Laser	KPLG	KPLG-SP	KAR

Figure 4. The best performing algorithms on the test problems.

4. Experiments

Our experiments were designed to assess the performance of the PLG, KPLG and KPLG-SP (the sigma-point approximation) algorithms across a variety of problems. For completeness, we also tested the KAR algorithm. We tested on five linear and nonlinear dynamical systems (the Rotation, Biped, Peanut, NB3, and Spring problems), where the underlying generative model was known. Since the models are limited to scalar observations, we also tested on three timeseries benchmarks (Santa Fe Laser, Mackey-Glass, and K.U. Leuven). The problems are described in Section 4.3.

We ran two types of experiments. The first type, which is the more important of the two, was a state estimation problem, in which the algorithms were run as explained in the text. This tested the algorithms' state update mechanisms and prediction performance. The second type was a far-horizon prediction test, where the algorithms predicted hundreds of steps into the future, without correcting state based on any observations. This tested modeling capacity and parameter estimation methods.

Parameters were selected by 10-fold cross-validation. Algorithms were judged on the mean-squared error (MSE) of their predictions. All data sets were normalized to be in $[0, 1]$. For the initial state, we set $\Sigma_0 = \widehat{\sigma_\eta^2} I + \sum_{i=1}^{n} (I^-)^i \widehat{C} + ((I^-)^i \widehat{C})^T$ and μ_0 to be the last n values of the training sequence, and then rolled it forward n timesteps (the test data was structured to be a continuation of the last sequence of training data). All algorithms were tested on $n = 2, 3, 4, 5, 6$, $\sigma_\phi^2 = 0.1, 0.4, 0.8, 1.2$, $\lambda = 0.00001, 0.001, 0.01$, and $\nu = 0.0001, 0.001, 0.01$ (the dictionary threshold).

4.1. State Estimation

For these experiments, we measured 1-step, 5-step and 10-step prediction MSEs (due to space limitations, only 1-step results are discussed). It is important to note that the measure of success is the difference between actual and predicted observations. This means that we are not attempting to estimate latent state, but we are allowed to *use* state to make our predictions. Basis function centers were selected with a dictionary, and α_j's were computed using regularized least-squares; this method appeared to be superior to using either EM or a random selection of basis functions.

The results are shown in Figure 4. Three columns are presented, the first of which shows the best algorithms on each problem when parameters are selected using cross-validation (all algorithms with an MSE within 5% of the lowest are considered equal). The results are mixed but encouraging. In particular, it seems that KPLG(-SP) generally performed well on the dynamical systems, where there really *is* an opportunity to leverage infinite memory via state. In contrast, KAR has performed well on the timeseries problems; in particular, it wins on the Mackey-Glass series, which really *is* an autoregressive model. This trend is more pronounced when only the best 5 out of the 10 cross-validation runs are used to select parameters, as shown in the second column. Here, we have eliminated outliers in the cross-validation runs, which has given us better parameters. We see in this case that KPLG-SP has won in four out of eight trials, and in the situations we expect it to.

The final column of Figure 4 shows best performers when tested *directly against the test set* (that is, without cross-validation). While it doesn't change the fundamental results, there are some noteworthy points: KAR does better on Laser, and KPLG is now competitive on the LDSs. These results should be taken with a grain of salt: there are enough parameters in the algorithms (and the test sequences are short enough) that they may be overfitting on the test data. However, the results show that all of the algorithms have the capacity to model the test data well. We also note that KAR still wins on the Spring problem. This is expected: Spring is deterministic with noiseless observations, so the uncertainty the KPLG(-SP) uses is unneeded.

Together, these results can be interpreted as preliminary evidence that each algorithm is winning when it is supposed to be, although it also appears that the test problems are not as discriminative as we would like. The results suggest three conclusions: first, that the nonlinear models are outperforming their linear counterparts; second, that the sigma-point approximation is competitive with the exact KPLG; and third, that our models are indeed capturing state, which results in an advantage over a simple autoregressive model, especially in noisy cases. Not reflected in these results is the fact that KAR seemed to give more consistent MSEs across parameter settings than KPLG(-SP); this is why we limited ourselves to only 5 out of 10 cross-validation sets, and indicates that further improvements to KPLG(-SP) are needed.

4.2. Long-Term Prediction

Here, each algorithm was asked to predict hundreds of timesteps into the future. This was done to assess the models' raw capacity, especially as compared to other methods.

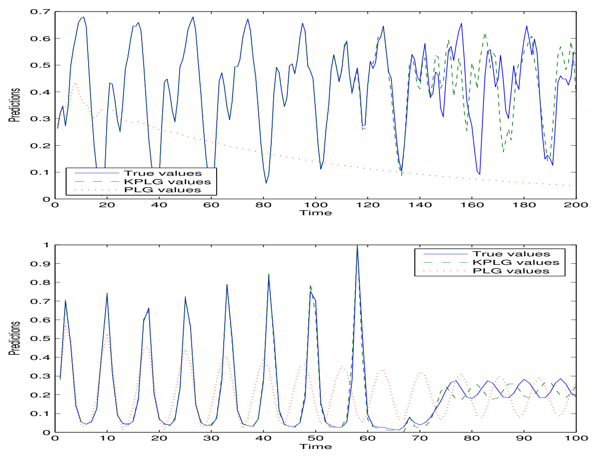

Figure 5. Top: the results of predicting the Mackey-Glass series. Bottom: the results of predicting the Santa Fe Laser series.

We trained KPLG(-SP) using the KRLS algorithm, incorporating the more sophisticated training method suggested by Engel et al. (2004). We set $\Sigma_t = 0$ for all t, making KPLG(-SP) and KAR equivalent; we merely wanted to compare PLG and KPLG. Figure 5 shows results on Laser and Mackey-Glass, both of which demonstrate a clear advantage of KPLG over PLG. The Laser result almost exactly reproduces the result obtained by Engel; as noted by him, the MSE incurred here (0.00120; equivalent to their NMSE of 0.026) would have been just enough to place first in the Santa Fe competition. This suggests that the model is capable of competing successfully with other methods.

4.3. Problem Descriptions

All problems except Laser were trained on 2000 sequential observations and tested on a 200 observation continuation. Laser had 1000 training and 100 testing observations.

Rotation, Peanut, Biped, NB3: Two-dimensional LDSs. Rotation is linear; the others are nonlinear. Observations and dynamics were noisy. NB3 had about an order of magnitude more noise than the other problems. **Spring:** A 2D system with a mass oscillating between damped springs with nonlinear forcing functions. Only the position of the mass was observed. Deterministic with noise-free observations. **Mackey-Glass:** A well-known timeseries generated from a delay differential equation. The series is deterministic but chaotic. Parameters were $a = 0.2, b = 0.1$, and $\tau = 30$, which are standard settings. **Santa Fe Laser:** Data from the Santa Fe timeseries competition. The series was recorded from a laser in a chaotic state, whose pulsations more or less follow the theoretical Lorenz model of a two-level system. **K. U. Leuven:** Competition data from the International Workshop on *Advanced Black-Box Techniques for Nonlinear Modeling*, K.U. Leuven Belgium, 1998.

5. Related Approaches

Here, we briefly survey other nonlinear methods that are similar in spirit and application to ours, focusing on nonlinear extensions to the Kalman filter. The first is the Extended Kalman filter (EKF), which updates state by linearizing the system dynamics, and propagating information through this first-order approximation. Unfortunately, it requires that analytical derivatives of the dynamics be available, and cannot capture discontinuities in the dynamics. The Unscented Kalman Filter (Wan & van der Merwe, 2000) improves on the EKF with a sigma-point approximation. It is the closest competitor to our method, except that it posits latent state and provides no parameter estimation methods; our method is also simpler because our observation and transition models are combined. Rudary and Singh (2004) proposed a nonlinear PSR based on "e-tests," but it is restricted to domains with discrete observations.

6. Conclusions and Future Research

Our focus has been an investigation of the viability of nonlinear predictive representations of state for continuous observation systems, as well as the advantages of our specific extension to the original PLG. Based on our empirical results, the broadest conclusion is that both the idea and our specific model are viable. While more work remains to be done, our algorithms have successfully modeled the real-world and synthetic problems presented here – while learning their parameters directly from data – and appear to provide competitive results to other methods. One of the advantages of the model is the straightforward method of parameter estimation. Only standard regressions and sample statistics are required, which is a direct consequence of the predictive nature of the state. This also lead us easily to an on-line version of the algorithm with KRLS.

We have not focused on raw empirical success, which leaves the door open for several obvious extensions. In particular, combining the strengths of KPLG and KAR into quasi-predictive models (which use history and predictions together) is an open and interesting avenue. It is also important to address the difficulties in parameter estimation and cross-validation, and to improve the algorithm's stability and generalization, but even with these problems, the algorithm is learning reasonable and competitive models.

An important practical conclusion is the success of the sigma-point approximations, which have provided results close to those of the KPLG for a fraction of the computational effort. We originally picked the Gaussian form of the kernels for analytical tractability, but the success of the approximations suggests that this is unnecessary. In addition to accuracy and speed, they provide freedom: future algorithms can use other kernels in more flexible models.

A. Derivation of and Key to Model Terms

Space precludes a full derivation of E_t, C_t and V_t for the KPLG model with a Gaussian kernel, so we here summarize the necessary identities and lemmas. An important lemma is a standard result about products of Gaussians: $K(a,b;A)K(b,c;B) = K(a,c;C)K(b,d;D)$ with $C = A+B$,$D = A^{-1}+B^{-1}$ and $d = A(A+B)^{-1}b + B(A+B)^{-1}c$ (where A, B, C and D are covariance matrices). Using it, we will compute one key identity by factoring terms out of the integrals needed when taking expectations and covariances; the other needed identities are easily computed by analogy. This first result states that the expected value of the kernel is the kernel of the expected value, with added variance. Let $K_{tj} = K(\xi_j, Z_t; \phi_j)$:

$$\begin{aligned} E[K_{tj}] &= \int K(\xi_j, Z_t; \phi_j) p(Z_t) dZ_t \\ &= \int K(\xi_j, Z_t; \phi_j) K(Z_t, \mu_t; \Sigma_t) dZ_t \\ &= K(\xi_j, \mu_t; \phi_j + \Sigma_t) \int K(Z_t, \mu'_{tj}; \Sigma'_{tj}) dZ_t \\ &= K(\xi_j, \mu_t; \phi_j + \Sigma_t) \end{aligned}$$

where $\Sigma'_{tj} = \Sigma_t^{-1} + \phi_j^{-1}$ and $\mu'_{tj} = \phi_j(\Sigma_t + \phi_j)^{-1}\mu_t + \Sigma_t(\Sigma_t + \phi_j)^{-1}\xi_j$. The last line follows because the integral is over an entire PDF with unit volume. As a corollary, it is easy to show that $E[K(\xi_j, Z_t; \phi_j)Z_t] = K(\xi_j, \mu_t; \phi_j + \Sigma_t)\mu'_{tj}$ because the integral in the penultimate line will become an expected value. The other identities needed for the derivation of E_t, C_t and V_t are:

$$\begin{aligned} E[K_{tj}] &= K'_{tj} \\ E[K_{tj}\eta_{t+n+1}] &= K'_{tj} C^T \Sigma_t^{-1} (\mu'_{tj} - \mu) \\ E[K_{tj}K_{ti}] &= K^{\dagger}_{tij} \end{aligned}$$

where

$$\begin{aligned} K'_{tj} &= K(\xi_j, \mu_t; \phi_j + \Sigma_t) \\ \mu'_{tj} &= \phi_j(\Sigma_t + \phi_j)^{-1}\mu_t + \Sigma_t(\Sigma_t + \phi_j)^{-1}\xi_j \\ K^{\dagger}_{tij} &= K(\mu^{\dagger}_{tij}, \mu_t; \Sigma_t + \Sigma^{\dagger}_{tij}) K(\xi_i, \xi_j; \phi_i + \phi_j) \\ \mu^{\dagger}_{tij} &= \phi_i(\phi_i + \phi_j)^{-1}\xi_j + \phi_j(\phi_i + \phi_j)^{-1}\xi_i \\ \Sigma^{\dagger}_{tij} &= \phi_i(\phi_i + \phi_j)^{-1}\phi_j. \end{aligned}$$

Acknowledgments

Special thanks to Matt Rudary for patient teaching and to Erik Talviti for talking through problems. This work is supported by the National Science Foundation under Grant Number IIS-0413004 and by an NSF Graduate Research Fellowship to David Wingate. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Engel, Y., Mannor, S., & Meir, R. (2004). The kernel recursive least squares algorithm. *IEEE Transactions on Signal Processing*, *52*, 2275–2285.

Ghahramani, Z., & Hinton, G. E. (1996). *Parameter estimation for linear dynamical systems* (Technical Report CRG-TR-96-2). Dept. of Computer Science, U. of Toronto.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. Springer.

Julier, S., & Uhlmann, J. K. (1996). *A general method for approximating nonlinear transformations of probability distributions* (Technical Report). University of Oxford.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problem. *Transactions of the ASME—Journal of Basic Engineering*, *82*, 35–45.

Kimeldorf, G. S., & Wahba, G. (1971). Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 82–95.

Littman, M. L., Sutton, R. S., & Singh, S. (2001). Predictive representations of state. *Advances in Neural Information Processing Systems 14*.

Pandit, S. M., & Wu, S.-M. (1983). *Time series and system analysis with applications*. John Wiley.

Rudary, M. R., & Singh, S. (2004). A nonlinear predictive state representation. *Advances in Neural Information Processing Systems 16*.

Rudary, M. R., Singh, S., & Wingate, D. (2005). Predictive linear-Gaussian models of stochastic dynamical systems. *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge University Press.

Wan, E. A., & van der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. *Proceedings of Symposium 2000 on Adaptive Systems for Signal Processing, Communication and Control*.

Yang, C., Duraiswami, R., Gumerov, N., & Davis, L. (2003). Improved fast Gauss transform and efficient kernel density estimation. *IEEE International Conference on Computer Vision* (pp. 464–471).