Some manifold learning considerations towards explicit model predictive control

Robert J. Lovelett, Felix Dietrich, Seungjoon Lee, Ioannis G. Kevrekidis

Department of Chemical and Biomolecular Engineering, Johns Hopkins University, Baltimore, MD 21218

Abstract

Model predictive control (MPC) is among the most successful approaches for process control and has become a de facto standard across the process industries. There remain, however, applications for which MPC becomes difficult or impractical due to the demand that an optimization problem is solved at each time step. In this work, we present a link between explicit MPC formulations and recent advances in data mining, and especially manifold learning, to enable facilitated prediction of the entire MPC control policy even when the function mapping from the system state to the control policy is complicated. We use a carefully designed similarity measure, informed by both control policies and system state variables, to “learn” an intrinsic parametrization of the MPC controller using a diffusion maps algorithm. We then use function approximation algorithms (i.e., regression or interpolation) to project points from state space to the intrinsic space, and then from the intrinsic space to policy space. With our similarity measure, the function from intrinsic space to the control policy may often be approximated using simple (and therefore fast) techniques, such as polynomial regression or modest-sized artificial neural networks. The manifold learning approach is amenable to alternative parametrizations for (observations of) the state space, and will discover nonlinear relationships among the state variables that can result in a lower dimensional representation. We demonstrate our approach by effectively stabilizing and controlling an open-loop unstable nonisothermal continuous stirred tank reactor subject to step changes in the reference trajectory and white noise disturbances.

Key words: Model Predictive Control, Data Mining, Diffusion Maps, Deep Learning

1 Introduction

Model predictive control (MPC) is a de facto standard method for control across the process industries [28]. In MPC (also known as receding horizon control), the control action is calculated on-line by solving a receding horizon optimal control problem at each time step to determine the subsequent control action to take [10,22,21]. MPC for constrained linear systems has been well-established for some time now [22], and that success has led to a significant effort to extend MPC to systems that are harder to control due to stochasticity [21], nonlinearity [23], decentralization [5], or other challenges. Many of the theoretical challenges associated with these more complex MPC problems have been overcome [23,12], enabling practitioners to use MPC in new applications that are even more demanding. Because, however, MPC entails solving an optimization problem at every sampled time step, it will always be limited by the computational time it takes to solve that problem. Due to limited computational resources, deploying MPC remains challenging for strongly nonlinear, high dimensional, and stiff systems [23]. Furthermore, given recent interest in MPC for distributed or mobile systems [31,5], avoiding high computational overhead is even more important for applications where computational resources are limited at the point of control action. These concerns motivate methods for reducing (and sometimes nearly eliminating) on-line computation in MPC.

To address the demand for excessive computational resources, there has been significant research into “fast” MPC over the past two decades. Broadly speaking, two approaches have been investigated for fast MPC: (1) suboptimal MPC, where a simpler optimization problem is solved that is equivalent (or approximately equivalent) to the solution of the full problem [37,36], and (2) explicit MPC, where an explicit control law is found that approximates the implicit MPC controller but does not...
require on-line optimization [3]. In this work, we focus on the latter, explicit MPC.

For constrained linear systems, an explicit solution can be found by solving a single multiparametric quadratic programming problem off-line [3,4,25]. The “parameters” are the system states and the solution to this optimization problem is piecewise affine; with this solution in hand, the controller only needs to first follow a look-up table (to determine the relevant polytopic region of state-space in which the system currently lies) and then perform an affine computation. Unfortunately, for high dimensional systems, the number of polytopes increases exponentially, which can make even the look-up operation too slow for some applications [4]. Multiparametric programming has also been applied to nonlinear MPC, but finding exact solutions to multiparametric nonlinear programming problems is not always feasible, and even when it is feasible, the solution is not necessarily piecewise affine. Therefore, approximation methods are typically used instead [8].

Although the multiparametric programming approach is dominant in the literature, another approach to explicit MPC is interpolation or function approximation. In this framework, a large number of control policies are computed off-line and the on-line control law is constructed by interpolation (or regression) from states to the optimal policies that were computed off-line. This method has been most successful using artificial neural networks (ANNs) as the interpolation functions [1]. Recently, this method was refined for constrained linear MPC problems using deep neural nets combined with Dykstra’s projection to ensure constraint satisfaction [4].

In this work, we present an alternative framework for explicit MPC based on finding an appropriate function between the state space and the control policy that approximates the MPC controller. We build upon recent advances in nonlinear data mining, especially manifold learning, to show how we can find an intrinsic parametrization of the MPC problem that respects similarities in control policy space. By working in this intrinsic space, we can design effective interpolating or approximating functions as our explicit controllers. Specifically, we use state feedback to project the system’s position in state space onto a latent manifold, with a parameterization informed by the control policy space and by the state space; and then, using our position in the latent space, we can estimate the entire optimal control policy. We also demonstrate how we can, in some cases, approximate the “inverse problem” of finding the system state given the optimal control policy. Solving the inverse problem could be valuable in contexts outside traditional MPC, where system feedback can be easily observed, but measuring the system state is challenging, as in some biological systems. In order to effectively interpolate between the state space, latent manifold, and control policy, we apply four tools for learning the transformations: polynomial regression (PR), artificial neural networks (ANNs), Gaussian process (GP) regression, and radial basis function (RBF) interpolation.

The remainder of this paper is organized as follows: in Section 2, we discuss our approach to MPC using manifold learning and function approximation in more detail; in Section 3, we present an illustrative application of our approach to a nonisothermal continuous stirred tank reactor (CSTR) control problem; and in Section 4 we mention possible applications of this framework and provide some directions for future research.

2 Theory

2.1 Model Predictive Control Background

First, consider the discrete time nonlinear system written in state space form as a difference equation:

\[
x_{k+1} = f(x_k, u_k) \\
y_k = h(x_k, u_k)
\]

(1)

where \( x \in \mathcal{X} \) is the system state vector, \( u \in \mathcal{U} \) is the input vector, \( y \in \mathbb{R}^{n_y} \) is the output vector (for a system with \( n_y \) outputs), and \( k \in \mathbb{N} \) is the sample index. For system 1, the MPC controller is defined as the controller that minimizes the cost [22]:

\[
V(y, k, u) = \sum_{i=k}^{k+N-1} l(y_i, u_i) + F(y_{k+N})
\]

(2)

where \( u = \{u_k, u_{k+1}, ..., u_{k+N-1}\} \), \( l(y_i, u_i) \) is the stage cost (such as a normed difference between a reference trajectory, \( r_k \), and the predicted trajectory) and \( F(y_{k+N}) \) is a terminal cost. For \( i > k \) is found by evolving the model (Equation 1) in time using the policy \( u \) and initial condition \( x_k \). There are often constraints on the range of inputs, range of states, or rate of change of inputs; refer to [22] for a full discussion of constrained MPC. Here, we will assume that we can deploy an optimization algorithm to find the optimal control policy, \( u^* \), that minimizes Equation 2. Because we are solving the optimization problem off-line, this optimization algorithm need not be highly efficient. In this paper, we will also assume that the reference trajectory is constant over the entire prediction horizon, \( r_k = r_0 \), and could be piecewise constant during process operation; we allow ourselves this strong assumption since the focus is on showcasing the data-driven features of the approach.

Footnote: We use upper case, **Boldface** notation to refer to matrices, lower case, *boldface* notation to refer to a vector of points in time (e.g., an optimal control policy), and nonboldface notation to refer to vector variables at a single time step, as well as for scalars.
The solution to the optimization problem defines a feedback control law. If many optimization problems are solved off-line, then it is possible, subject to some weak smoothness conditions, to interpolate between the current state, \( x_k \), and the optimal control policy, \( u^* \). The interpolation function is a surrogate for the implicitly defined MPC controller that allows for near instantaneous calculation of the optimal policy. For first order (i.e., single state), single-input, single-output systems, the problem is trivial; but for more challenging problems, interpolation may be hindered by the “curse of dimensionality,” where function training and evaluation time can increase exponentially with dimensionality. In this work, we discuss recent advances in machine learning and nonlinear data mining that allow us to use tools for function approximation in high-dimensional (but effectively low-dimensional) spaces, thus enabling explicit computation of the MPC control law.

2.2 Manifold Learning of MPC Systems

Manifold learning algorithms comprise a class of unsupervised machine learning techniques that attempt to find a low-dimensional manifold on which high-dimensional data points are embedded. The earliest manifold learning algorithm to be developed was principal component analysis (PCA), first presented by Pearson in 1901 \cite{27}. PCA and related techniques, however, are limited in that they can only find a linear embedding of the data. Since 2000, a number of related nonlinear manifold learning techniques have been presented \cite{33,30,2,6,7,35}. These methods all share the attribute that they can find nonlinear embeddings; therefore, for example, if 2D data points were all to lie on a smooth curve, PCA would find variation in 2 axes, whereas these manifold learning techniques would (correctly) discover that the data can be represented using only a single variable.

Our key observation is that the optimal control policy, \( u^* \), which has extrinsic dimensionality of \((N \times \text{dim}(u))\) (where \(N\) is the number of steps in the control horizon and \(\text{dim}(u)\) is the number of manipulated inputs) must lie on a manifold with intrinsic dimensionality limited by:

\[
\text{dim}_{i}(u^*) \leq \text{dim}(x) + \text{dim}(r_0) \quad (3)
\]

where \(\text{dim}()\) is the extrinsic dimensionality and \(\text{dim}_i()\) is the intrinsic dimensionality of an underlying manifold/vector. This limit results from observing that \(u^*\) is a function of the current system state, \(x\), and the current reference trajectory, \(r_0\) (the latter, in our case, having dimensionality equal to the number of controlled output variables).

Because of this property, we will refer to the augmented state vector, \(x^* \in \chi^*\), as the concatenation of the state variables and the variables parametrizing the reference trajectory. The intrinsic dimension may be lower than this maximum if, for example, the system quickly relaxes to a slow manifold (i.e., it is singularly perturbed), if the state space realization is nonminimal (i.e., containing redundant information), or if the control policy may be a function of the difference between a state variable and a reference variable (as could be the case for linear control of linear systems).

For many systems, however, even though \(u^*\) lies on a manifold of equivalent or lower dimension than \(\chi^*\), the augmented state space may be very poorly parametrized for predicting \(u^*\). In other words, a function \(c(x^*) : \chi^* \rightarrow U\) is likely to be quite complicated, requiring many basis functions to represent, and therefore challenging to learn and expensive to evaluate. We therefore seek an effective reparametrization of \(\chi^*\) that respects similarities in policy space, potentially enabling us to find simpler representations of the relationship between (augmented) system states and optimal control policies.

In this work, we use the diffusion maps (DMAPS) algorithm for manifold learning \cite{6,7}, which is reviewed in Appendix A. For the MPC problem, we seek to reparametrize the augmented state space in such a way that we can predict the control policy based on knowledge of the system state, as well as the state variables based on knowledge of the control policy. We will therefore use an “informed” metric (c.f., \cite{13,19}) using positional data in both augmented state space and the accompanying control policy space to build the weight matrix in Equation A.1:

\[
d_{ij} = \left\| z_i - z_j \right\|^2 \frac{\varepsilon^2}{\varepsilon^2} + \left\| f(z_i) - f(z_j) \right\|^2 \frac{\xi^4}{\xi^4} \quad (4)
\]

where \(\|\cdot\|\) is the Euclidean norm, \(\varepsilon\) is the median distance between points \(z_i\) and \(z_j\), and \(\xi\) is the median distance between points \(f(z_i)\) and \(f(z_j)\). For our application, \(z\) is the augmented state vector \(x^*\) and \(f(z)\) is the corresponding optimal control policy \(u^*\). The informed metric therefore finds a parametrization of the augmented state space that respects similarities in policy space—when \(\| u_i^* - u_j^* \|\) is large, it will spread these points far apart in the intrinsic space. The \(\xi^4\) in the denominator of the second term (versus \(\varepsilon^2\) in the first term) is to assure that the parameterization will favor similarities in policy space over similarities in augmented state space. In this way, the intrinsic parametrization will organize the augmented state space using as few variables as possible while hopefully making prediction of the high-dimensional control policy “simpler”.

We will call the manifold discovered by this procedure “the control manifold”, \(\mathcal{M}_c\); the non-redundant DMAPS eigenvectors \(\phi\), provide its parametrization (see Appendix A). We note that \(\mathcal{M}_c\) has dimensionality less
than or equal to that of \( \mathcal{X}^* \), and typically much less than that of the control policies.

Because the parametrization of \( \mathcal{M}_c \) respects similarities in policy space, we expect that predicting \( \mathbf{u}^* \) given \( \phi \) should be a “simple” task. If \( \| \mathbf{u}^*_i - \mathbf{u}^*_j \| \) is small, then \( \| \phi_i - \phi_j \| \) should be small as well (unless \( \| x^*_i - x^*_j \| \) is very large). To demonstrate, consider the function \( q = f(p) : \mathbb{R}^2 \to \mathbb{R} \). Here, \( q = f(p) = 10 \sin \sqrt{p_1^2 + p_2^2} + p_2 \). Data \( (p) \) sampled from a regular grid in \( \mathbb{R}^2 \) and colored by function value \( (q) \) are shown in Figure 1a. If we did not \textit{a priori} know the function, we may resort to complicated and/or computationally expensive methods to approximate it in \((p_1, p_2)\) space. However, in the reorganized space discovered via DMAPS and shown in Figure 1b, the values \( q \) are a simple function of the single DMAPS coordinate \( \phi_1 \); in fact (by visual inspection) the relationship appears nearly linear for this case.

2.3 Function Approximation

The above discussion implies that it is possible to link \( x^* \) to \( \mathbf{u}^* \) through \( \mathcal{M}_c \), but does not discuss how to find our position on \( \mathcal{M}_c \) in \( \phi \) coordinates, nor, how, given \( \phi \), to predict the policy, \( \mathbf{u}^* \). Additionally, we would like to solve the inverse problem, which requires predicting \( \phi \) from \( \mathbf{u}^* \), and then predicting \( x^* \) from \( \phi \), with the important caveat that the problem is not in general invertible, primarily when the control policy pushes against constraints.

The DMAPS metric from Equation 4 was designed to make the task of predicting \( \mathbf{u}^* \) from \( \phi \) (or vice versa) simple in comparison with predicting \( \mathbf{u}^* \) from \( x^* \), since the former function is defined over the lower-dimensional, intrinsic space.

In any case, we use four methods for function approximation, each of which has advantages and disadvantages: polynomial regression (PR), artificial neural networks (ANNs), Gaussian process (GP) regression, and radial basis function (RBF) interpolation. In principle, any of these methods may be applicable to learn any of the functions of interest (\( x^* \to \phi \), \( \phi \to \mathbf{u}^* \), and their inverses), though in practice PR is unlikely to effectively capture the mapping between \( x^* \) and \( \phi \), which is typically quite complicated.

**Polynomial Regression** First, we examine polynomial regression, a classical linear technique. In some cases, the mapping between \( \phi \) and \( \mathbf{u} \) is sufficiently simple that this classical approach works quite well (see Figure 1, where a complicated function in the original space becomes nearly linear in the intrinsic space). We use the ordinary least squares estimator, \( \Theta = (X^TX)^{-1}X^TY \) to find the coefficients of the polynomial regression estimator, where \( X \) is a feature matrix and \( Y \) is an output matrix. For this problem, rows of \( X = [\phi \ \phi^2 \ldots \ \phi^n] \) and rows of \( Y = \mathbf{u}^* \). Then, the output at new values of \( \phi \) can be predicted using \( \hat{Y} = \Theta \hat{X} \) [11].

**Artificial Neural Networks** Neural networks have become popular in many applications due to their versatility and ability to represent arbitrary continuous functions, and are widely considered the workhorse method for “deep learning” [24,20]. Artificial neural networks have also been widely used in control for decades, mostly for empirical approximation of state equations [14], but also for explicit MPC problems [1,4]. The main disadvantage of ANNs is that they require significant computational time for training, and require manual design of network topology and choice of activation function. More complex networks, with more nodes and hidden layers, require more training time and are susceptible to overfitting.

**Gaussian Process Regression** GP regression, as a non-parametric Bayesian modeling technique, provides the conditional distribution of an output as a function of its observed inputs. In order to deal with higher dimensional input spaces, we introduce the automatic relevance determination (ARD) weight in the covariance kernel, which employs an individual lengthscale hyperparameter for each input dimension [29]. In this paper, we employ a Matérn kernel for the covariance:

\[
\kappa(x_i, x_j) = \left( 1 + \sqrt{3}d(x_i, x_j) \right) \exp \left( -\sqrt{3}d(x_i, x_j) \right). \tag{5}
\]

GP regression typically requires some training time, in that hyperparameters should be optimized; but there are usually far fewer adjustable parameters than in ANNs.
Radial Basis Function Interpolation  For a final function approximation tool we use RBF interpolation, which has the key advantage that it does not require any training (except, perhaps, a regularization parameter), but can still represent highly complicated mappings. The possible downside, however, is that it is comparably expensive to evaluate. In RBF interpolation, the function is approximated at unsampled (new) data points using the position of the new points relative to all sampled data points [11]. Many kernel choices are possible to define the RBFs, and in our case, we use the multiquadric RBF:

$$f(r) = \sqrt{1 + \alpha r^2}$$  \hspace{1cm} (6)

where \(\alpha\) is a tuning parameter. The interpolation function is then:

$$\Psi(x) = \sum_{i=0}^{N} [w_i f(\|x - x_i\|)]$$  \hspace{1cm} (7)

where the weights \(w_i\) are fit via (regularized) ordinary least squares.

3 Results and Discussion

To demonstrate our approach, we present a tutorial example of our DMAPS-enabled explicit MPC formulation for controlling a jacketed nonisothermal continuous stirred tank reactor (CSTR). The CSTR performs a single reaction, \(A \rightarrow B\), which is exothermic with reaction rate temperature dependence having the Arrhenius form. We will control the concentration of species \(B\) using the temperature of the cooling water as the manipulated variable. The cooling water temperature is constrained both with minimum and maximum absolute values and with a maximum stepwise rate of change. All of the data presented are expressed in dimensionless variables. For detailed descriptions of the model equations and the MPC parameters, refer to Appendix B and Appendix C, respectively.

The nonisothermal CSTR is challenging to control partly because a large range of state space is unstable at steady state. Figure 2 shows the bifurcation diagram of the open loop system with cooling water temperature as the bifurcation parameter. The system contains two saddle-node bifurcations and requires feedback stabilization to operate at conversions between approximately 0.2 and 0.8. If using MPC, this means that short sampling intervals are required to ensure that feedback is frequent enough to stabilize the system. Because the sampling interval is short, that likewise means that the control horizon must be long, so that significant dynamics are included in the cost function. These challenges make this system an appropriate candidate to test our formulation of explicit MPC control.

One of the key advantages of the manifold learning approach is that it will automatically detect equivalent or redundant descriptions of the state space in a purely data-driven manner. For example, we know from theory that our process model is intrinsically second order and can be described using the physical variables of \(C_A\) and \(T\) (see Appendix B). For more complicated processes, however, we may not recognize the intrinsic state space and instead model our process using whichever measured variables are at hand to represent it. For a purely ANN-driven approach, using such alternative (and possibly redundant) parametrizations may require redesigning the network topology for the new problem, and certainly
Fig. 4. (a-h) 750 randomly chosen training points plotted in the intrinsic coordinates, $\phi$, learned from $X^*_{\alpha}$ and $U^*$. Each plot is colored by one step of the control policy, in ascending order (i.e., (a) is colored by $u^*_k$, (b) by $u^*_k+1$, etc). (i) The same 750 points plotted as augmented states, $X^*_{\alpha}$, and colored by $u^*_k$. Requires retraining the network. The manifold learning approach, however, will always detect an intrinsic 2D space, regardless of how state space was parametrized.

To illustrate, we consider three alternative parametrizations of state space: (1) $x_{\alpha} = \begin{bmatrix} C_a T_r \end{bmatrix}$, the original variables used for modeling the system, (2) $x_{\beta} = \begin{bmatrix} (kC_a) \left( \frac{4}{3} (T_0 - T_r) - \frac{\Delta H}{\rho C_p} C_A \right) \end{bmatrix}$, the reaction rate and heating rate (see Appendix B) and (3) $x_{\gamma} = \begin{bmatrix} x_{\alpha} \ x_{\beta} \end{bmatrix}$, the concatenation of the other two parametrizations. By concatenating the reference variable to each of these, we define three augmented state vectors, $X^*_{\alpha}$, $X^*_{\beta}$ and $X^*_{\gamma}$, any one of which is a complete description of the system state and sufficient (in principle) to predict the control policy.

To discover $\mathcal{M}_c$, the control manifold, we need to sample the augmented state space and compute the control policies off-line. We sampled 200 points randomly in augmented state space, and evolved the system in time for 20 time steps using a nonlinear model predictive controller with time horizon of 20 time steps. Taken together, we have 4000 samples, and we collected all of the control policies in matrix $U^* \in \mathbb{R}^{4000 \times 20}$ and augmented state variables in matrices $X^*_{\alpha} \in \mathbb{R}^{4000 \times 3}$, $X^*_{\beta} \in \mathbb{R}^{4000 \times 3}$, and $X^*_{\gamma} \in \mathbb{R}^{4000 \times 5}$. To test our tools for function approximation, we randomly partitioned the data into 3000 points for training and 1000 points for testing.

We apply DMAPS three times using the informed metric from Equation 4 to the downsampled policy matrix $U^*$ with each of the augmented state matrices $X^*_{\alpha}$, $X^*_{\beta}$, and $X^*_{\gamma}$. The results for each case are shown in Figure 3, which shows augmented state variables projected into the intrinsic space and colored by $u^*_k$. These results demonstrate that DMAPS can find an effective, possibly lower dimensional, intrinsic manifold regardless of which state variables are made explicit, from which the control policy can be predicted. Using $x^*_{\alpha}$ (for example) as the augmented state variables, Figure 4 shows that given the position in DMAPS coordinates, we can predict the entire control policy—not just the first step, $u^*_k$—though for brevity, only $u^*_k$ to $u^*_k+7$ are plotted. The 20-dimensional policy space can be effec-
Fig. 5. (a-e) Predicted vs. actual control policy with different explicit formulations of the MPC controller for the first five steps of the control policy using the test data. \( \phi \) was estimated from \( x_\alpha \) using RBFs, an ANN model, and GP regression, and \( u^* \) was estimated from \( \hat{\phi} \) using cubic PR, an ANN model, and RBFs. (g-i) Predicted vs. actual augmented state variables based on control policy information for the test data. \( \phi \) was estimated from \( u^* \) using cubic PR and an ANN model, and \( x_\alpha \) was estimated from \( \hat{\phi} \) using RBFs, and ANN model, and GP regression. Only control policies where \( u^*_k \neq 2.5 \) and \( u^*_k \neq -2.5 \) (i.e., control policies far from the boundary constraints) are shown.

Now, using \( x_\alpha^* \) as inputs, we design explicit feedback control laws, i.e., functions \( c(x_\alpha^*) : X_\alpha^* \to \mathcal{U} \). This task is divided into two stages: first, estimate the intrinsic variables \( \phi \) given the augmented state variables \( x_\alpha^* \), and second, predict \( u^*_k \) given \( \phi \). We emphasize that because we transformed to the intrinsic variables, we can also easily predict \( [u^*_{k+1} \ u^*_{k+2} \ ... \ u^*_{k+19}] \).

For estimating \( \phi \) from \( x_\alpha^* \), we use three methods presented in Section 2: ANNs, GPs, and RBFs. We write the neural network model as:

\[
\hat{\phi}_{\text{ANN}} = p(x_\alpha^*; W)
\]

where \( p \) is an artificial neural network with three hidden layers of 50 nodes each, three input nodes corresponding to each element in \( x_\alpha^* \), three output nodes corresponding to each element in \( \phi \), and \( W \) is the weight matrix for the neural network. Rectified linear activation functions (which have become a de facto standard in deep learning [20]) were used for each of the nodes in the hidden layers.
\[ f(x) = \max(0, x) \] (9)

To facilitate the use of the activation function in Equation 9 (which will never predict a negative output), all of the data variables were linearly rescaled from 0 to 1, and scaled back to their original ranges for presentation. Equation 8 was built using pyTorch and trained using the Adam optimizer (a similar algorithm to stochastic gradient descent) with learning rate of $1 \times 10^{-4}$ [17,26]. Alternatively, we predict $\phi$ from $x_\alpha^*$ using GP regression. Using the Matérn covariance function in Equation 5, we optimize hyperparameters for prediction over our training data set by minimizing negative log marginal likelihood [29] to obtain $\hat{\phi}_{GP}$. And finally, we predict $\phi$ from $x_\alpha^*$ via RBF interpolation. We use the multiquadric basis functions in Equation 6 and find the weights via regularized linear regression to find $\hat{\phi}_{RBF}$.

With intrinsic variables $\phi$ in hand using one of the three methods above, we now predict $u^*$. We again use three methods: cubic PR (higher order polynomial regression was tested, with no improvement), an ANN model, and RBF interpolation. The ANN model also uses the rectified linear activation function from Equation 9 (with inputs and outputs appropriately rescaled), four hidden layers of 50 nodes each, three input layers for $\phi$, and 20 output neurons for $u^*$ (the additional hidden layer compared to the first ANN was chosen because we predict 20 outputs, rather than only 3). The RBF interpolation procedure is the same as above, using the multiquadric basis functions given in Equation 6.

Using three techniques for predicting $\phi$ and three techniques for predicting $u^*$, we have designed nine explicit MPC controllers. Their prediction accuracy is shown for the first five steps of the control policy in Figure 5a-e. Generally speaking, the controllers that use the ANN or RBF models for predicting $u^*$ perform better than the PR models, though at the cost of more expensive function evaluation, and for the ANN models, more training time. For the first stage, predicting $\phi$ from $x_\alpha^*$, each of the methods appears to work about as well in terms of performance. The ANN model has the advantage of faster evaluation time, but slowest training time; RBFs and GPs are both more complex functions than the ANN model, and therefore somewhat more expensive to evaluate.

We also solve (where feasible) the inverse problem: predicting the augmented state space from the control policy, by following the same procedure as above in reverse. We find PR and ANN models to predict $\phi$ from $u^*$, and then find ANN, GP, and RBF models to predict $x_\alpha^*$ from $\phi$. For this second task, we note that when the control policy pushes against a constraint (defined here as $u^*_k = -2.5$ or $u^*_k = 2.5$) predicting the augmented state accurately becomes infeasible because the transformation is not invertible. Figure 5f-h shows our state predictions based on observations of control policies for test points that are not against constraints.

Finally, we test the control laws we developed on-line using the full simulated CSTR model. Figure 6 shows the performance of our DMAPS enabled, explicit nonlinear model predictive controller. We use state observations, $x_\alpha^*_{k-1}$, to predict $\hat{\phi}_k$ using RBF interpolation, and then PR or ANN models to predict $\hat{\phi}_k$ at each time step. The reference trajectory is open-loop unstable (see Figure 2), and white Gaussian noise is added to each state as a disturbance. Still, the controller can maintain the process near the reference value, as well as effectively track set point changes. The ANN model has visibly better performance in terms of reference tracking, which is unsurprising given that it better predicted the test policies in Figure 5, but even the very simple PR controller can effectively track set point changes and reject disturbances in an unstable, highly nonlinear process. Furthermore, the PR controller appears (visually) less aggressively tuned, possibly due to the smoothness of the polynomial prediction model or modest overfitting of the ANN. We do not enforce the constraints on the rate of change of the control policy, though we do enforce the boundary constraint. The rate constraints, however, are somewhat effectively “learned” by our explicit controller, with only few instances that violate them, and even then only slightly.

4 Conclusions and Future Directions

In this paper, we have developed and demonstrated a data-driven approach to designing explicit model predictive controllers. We uncover intrinsic, low-dimensional structure in the high-dimensional control policies and state vectors that provides a link between state space
and policy space. Our manifold learning method is agnostic to the particular parametrization of state space, and equally valid regardless of which state variables are available (as long as we have sufficiently many). Furthermore, because the similarity measure used by DMAPS is designed to favor similarities in policy space, predicting the entire control policy becomes about as easy as predicting just its first step. Like other approaches to explicit MPC, by developing functions between the state space and control action, we avoid the need for on-line optimization. Although our tutorial demonstration was for a single-input, single-output control problem, our framework naturally generalizes to multiple-input, multiple-output systems—in fact, we showed that we can easily generate multiple outputs in that we can predict the full time series of control actions.

Here, we assumed that we have access to the full system state, \( x \), either via direct measurement or from a state estimator, at each sampling point. Inspired by delay embedding theorems, such as that of Takens [32], we expect that we do not need full state feedback to apply our methodology. Much like a state estimator synthesizes information from histories of measurements to “observe” the system state, we could directly “observe” the control policy using this information. In future work, we will investigate how to build a purely data-driven “policy observer,” where the MPC policy is predicted from only measured data. We anticipate that, as long as the usual state observability conditions are satisfied, the control policy will be equally observable.

Alternatively, there may be situations where feedback is infrequent, but actuation is comparatively fast. Here, we could use our framework to design an explicit controller that takes control action in between state observations. Our procedure is especially well-suited for such systems because we can easily estimate the full, \( N \)-step control policy, and not only the first step like other explicit controllers that use function approximation. Finally, even if not accurate enough to implement, our data-inferred control policy can conceivably be used for “smarter” initialization of the optimization algorithm in regular MPC.

We also demonstrated how, given observations of the control policy, we can predict the system state. From the perspective of process control, where the objective is developing a feedback control law, the inverse problem may not appear to be of interest. Yet, we can imagine a system in which the “feedback” is easy to observe, while the system state is not. For example, we may consider an “expert human” controller, or a neural network controller. Using this approach, we could design a state observer that uses control observations to determine that state, instead of output measurements like in conventional state observers.

**Acknowledgements**

This work is supported by the National Science Foundation (NSF) and the Defense Advance Research Projects Agency (DARPA). The assistance of Dr. Mahdi Kooshkbaghi with ANN training is gratefully acknowledged.

**A Diffusion Maps**

DMAPS is an algorithm for manifold learning [6,7]. We assume that the data points \( X \in \mathbb{R}^N \) all lie on a smooth low dimensional manifold, \( \mathcal{M} \), such that \( \dim(\mathcal{M}) \ll N \). The algorithm works by finding data-driven approximations for eigenfunctions of the Laplace-Beltrami (i.e., diffusion) operator on \( \mathcal{M} \). One can show that these eigenfunctions provide an effective minimal parameterization of \( \mathcal{M} \) [16].

To approximate the eigenfunctions, we find eigenvectors of the appropriately normalized graph Laplacian, which is constructed as follows. Given \( m \) observations of the data, define a weight matrix, \( W \in \mathbb{R}^{m \times m} \):

\[
W_{ij} = \exp\left(-d_{ij}^2\right), i, j = 1, ..., m \tag{A.1}
\]

where \( d_{ij} \) is a metric representing distance between data points \( i \) and \( j \) that is often the Euclidean distance, though in this work we use an input-output informed metric defined in Equation 4. To account for nonuniform sampling, the weight matrix is normalized with \( P_{ii} = \sum_{k=1}^{m} W_{ik} \) using:

\[
\tilde{W} = P^{-\alpha}W P^{-\alpha} \tag{A.2}
\]

where for isotropic DMAPS, \( \alpha = 1 \). Next, we define a diagonal matrix from the row sums of the weights:

\[
D_{ii} = \sum_j \tilde{W}_{ij} \tag{A.3}
\]

Finally, we construct the Markov transition matrix:

\[
A = D^{-1}\tilde{W}, \tag{A.4}
\]

which is the desired graph Laplacian. It has been shown that in the limit as \( \varepsilon \to 0 \) and \( m \to \infty \), \( A \) converges to the Laplace-Beltrami operator on \( \mathcal{M} \) [7]. Because \( A \) is a Markov matrix, its eigenvalues \( \lambda \) are real-valued and vary between 0 and 1 and its eigenvectors \( \Phi \), stacked column-wise by convention) are real-valued; the trivial eigenvector, \( \phi_0 = 0 \) has eigenvalue \( \lambda_0 = 1 \). We discard \( \phi_0 \) and sort the remaining eigenvectors \( \{\phi_i\}, i = 1, ..., m \) in order of descending eigenvalue. The first several eigenvectors provide an effective parameterization of the manifold, which we call the DMAPS embedding of the data.
Some of the DMAPS eigenvectors may simply be harmonics that provide no new information about $M$. These can be safely discarded either by visual inspection or (more systematically) by using local linear regression to test whether a new eigenvector can be predicted using information from the previous eigenvectors. When using local linear regression, Dsilva et al. define a relative leave-one-out cross validation residual for each eigenvector as [9]:

$$ R_k = \sqrt{\frac{\sum_{i=1}^{n} (\phi_k(i) - \hat{\alpha}_k(i) + \hat{\beta}_k(i)^T \Phi_{k-1}(i)))^2}{\sum_{i=1}^{n} (\phi_k(i))^2}} $$  \hspace{1cm} (A.5)

where $\hat{\alpha}$ and $\hat{\beta}$ are coefficients from regression and $\Phi_{k-1}$ is a matrix containing the first $k-1$ eigenvectors. If $R_k \ll 1$, then $\phi_k$ can be predicted from the previous eigenvectors and therefore provides only redundant information.

### B Nonisothermal CSTR Model

The MPC controller for the CSTR is based on a mechanistic model of a constant density reactor (c.f. [23]). The model equations are:

$$ C_A = \frac{q}{V} (C_{A0} - C_A) - k C_A $$

$$ T_r = \frac{q}{V} (T_0 - T_r) - \frac{\Delta H}{\rho C_p} C_A + \frac{UA}{\rho C_p V} (T_c - T_r) $$

$$ C_B = C_{A0} - C_A $$

Equation (B.1)

$C_i$ for $i \in \{A, B\}$ are the species concentrations; $C_{A0}$ is the concentration of species $A$ at the inlet (concentrations of species $B$ is zero at the inlet). $T_0$ is the inlet temperature, and $T_c$ is the cooling water temperature. $T_r$ is used here as the manipulated variable, $q$ is the flow rate, $V$ is the reactor volume, $C_p$ is the heat capacity, $\rho$ is the density, $U$ is a heat transfer coefficient, $A$ is the heat transfer area, and $\Delta H$ is the reaction enthalpy (note that $\Delta H$ is a negative number as the reaction is exothermic). The reaction rate constants, $k$, are given as an Arrhenius expression:

$$ k = k_0 e^{\frac{-E}{RT}} $$  \hspace{1cm} (B.2)

$k_0$ is the pre-exponential factor, $E$ is the activation energy, $R$ is the gas constant. Parameter values are given in Table B.

To improve numerical performance, we nondimensionalized the concentration variables using $C_i = \frac{C_i}{C_{i0}}$ for $i \in \{A, B\}$ and temperature variables using $T_i = \frac{T_i - 300 K}{100 K}$ for $i \in \{r, c\}$. All of the results shown in Section 3 use the nondimensionalized variables.

### C MPC Controller

To build the datasets, $U^*$, $X_0^*$, $X_3^*$, and $X_4^*$, we designed a fully nonlinear MPC controller. The system was discretized with a sampling time of 0.05 s. The control policy was obtained from the current system state by solving the following optimization problem:

$$ \min_u \quad V(x, k, u) = \sum_{i=k}^{k+10} (C_{B,i} - r_0)^2 $$

s.t.

$$ u_i \leq 2.0, \quad i = k, k+1, \ldots, k+19 $$

$$ u_i \geq -2.0, \quad i = k, k+1, \ldots, k+19 $$

$$ |u_i - u_{i+1}| \leq 0.5, \quad i = k, k+1, \ldots, k+19 $$

which indicates a control horizon of 20 time steps (1.0 s continuous time), constraints on the maximum and minimum values of $u$, and constraints on the rate of change of $u$. In Equation C.1, $x = [C_A \ T_r]$, and $u_k = T_{r,k}$.

The optimization problem was solved using a sequential quadratic programming algorithm as implemented in the SciPy numerical computing package [15,18].

The system was initialized at random states uniformly sampled from $C_A \in [0.1, 0.9]$ and $T \in [0.0, 0.55]$ using random constant references $r_0 \in [0.1, 0.9]$. The MPC controller was used to evolve the system in time for 20 time steps, and all of the data points from every time step were collected to discover the control manifold and design the explicit MPC controller.

### References

[1] Bernt M. Åkesson and Hannu T. Toivonen. A neural network model predictive controller. Journal of Process Control, 16(9):937–946, 2006.

[2] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. Neural Computation, 15(6):1373–1396, 2003.

[3] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. Automatica, 38(1):3–20, 2002.
[4] Steven Chen, Kelsey Saulnier, Nikolay Atanasov, Daniel D. Lee, Vijay Kumar, George J. Pappas, and Manfred Morari. Approximating Explicit Model Predictive Control Using Constrained Neural Networks. In American Control Conference, pages 1520–1527, 2018.

[5] Panagiotis D. Christofides, Riccardo Scattolini, David Muñoz de la Peña, and Jinfeng Liu. Distributed model predictive control: A tutorial review and future research directions. Computers and Chemical Engineering, 51:21–41, 2013.

[6] R R Coifman, S Lafon, a B Lee, M Maggioni, B Nadler, F Warner, and S W Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: diffusion maps. Proceedings of the National Academy of Sciences of the United States of America, 102(21):7426–31, 2005.

[7] Ronald R. Coifman and Stephane Lafon. Diffusion maps. Applied and Computational Harmonic Analysis, 21(1):5–30, 2006.

[8] Luis F. Domínguez, Diogo A. Narciso, and Efstratios N. Pistikopoulos. Recent advances in multiparametric nonlinear programming. Computers and Chemical Engineering, 34(5):707–716, 2010.

[9] Carmelina J. Dsilva, Ronen Talmon, Ronald R. Coifman, and Ioannis G. Kevrekidis. Parsimonious representation of nonlinear dynamical systems through manifold learning: A chemotaxis case study. Applied and Computational Harmonic Analysis, pages 1–15, 2015.

[10] Carlos E. García, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice–A survey. Automatica, 25(3):335–348, 1989.

[11] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[12] Michael A. Henson and Dale E. Seborg. Feedback Linearizing Control. In Dale E. Seborg and Michael A. Henson, editors, Nonlinear Process Control. 1997.

[13] Alexander Holiday, Mahdi Kooshkbaghi, Juan M Bellorín, C. William Gear, Antonios Zagaris, and Ioannis G. Kevrekidis. Manifold learning for parameter reduction. 2018.

[14] K J Hunt, D Sbarbaro, R ˙Zbikowski, and P J Gawthrop. Neural networks for control systemsA survey. Automatica, 28(6):1083–1112, 1992.

[15] Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open source scientific tools for python.

[16] Peter W Jones, Mauro Maggioni, and Raanan Schul. Manifold parametrizations by eigenfunctions of the Laplacian and heat kernels. Proceedings of the National Academy of Sciences of the United States of America, 105(6):1803–1808, 2008.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. pages 1–15, 2014.

[18] Dieter Kraft. A Software Package for Sequential Quadratic Programming. Technical report, Germany Aerospace Center – Institute for Flight Mechanics, Cologne, Germany, 1988.

[19] Stephane Lafon. Diffusion Maps and Geometric Harmonics. PhD thesis, Yale University, 2004.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. Nature, 521(7553):436–444, 2015.

[21] David Q. Mayne. Model predictive control: Recent developments and future promise. Automatica, 50(12):2967–2986, 2014.

[22] David Q. Mayne, James B. Rawlings, Christopher V. Rao, and Pierre O.M. Scokaert. Constrained model predictive control: Stability and optimality. Automatica, 36(6):789–814, 2000.

[23] Edward S Meadows and James B Rawlings. Model Predictive Control. In Michael A. Henson and Dale E. Seborg, editors, Nonlinear Process Control, chapter 5, pages 233–310. Prentice Hall, Englewood Cliffs, NJ, 1996.

[24] Michael A. Nielsen. Neural Networks and Deep Learning. Determination Press, 2015.

[25] Richard Oberdieck, Nikolaos A. Diangelakis, and Efstratios N. Pistikopoulos. Explicit model predictive control: A connected-graph approach. Automatica, 76:103–112, 2017.

[26] Adam Paszke, Gregory Chanan, Zeming Lin, Sam Gross, Edward Yang, Luca Antiga, and Zachary Devito. Automatic differentiation in PyTorch. Advances in Neural Information Processing Systems 30, (Nips):1–4, 2017.

[27] Karl Pearson. On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2(11):559–572, 1901.

[28] S. Joe Qin and Thomas A. Badgwell. A survey of industrial model predictive control technology. Control Engineering Practice, 11:737–764, 2003.

[29] Carl Edward Rasmussen and Christopher K I Williams. Gaussian Processes for Machine Learning. The MIT Press, 2005.

[30] Sam T Roweis and Lawrence K Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. Science, 290(December):2323–2326, 2000.

[31] Riccardo Scattolini. Architectures for distributed and hierarchical Model Predictive Control - A review. Journal of Process Control, 19(5):723–731, 2009.

[32] Floris Takens. Detecting strange attractors in turbulence. In Dynamical systems and turbulence, Warwick 1980, pages 366–381. Springer, 1981.

[33] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. Science, 290(December):2319–2323, 2000.

[34] Petter Tøndel, Tor Arne Johansen, and Alberto Bemporad. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. Automatica, 39(3):489–497, 2003.

[35] L J P Van Der Maaten and G E Hinton. Visualizing high-dimensional data using t-sne. Journal of Machine Learning Research, 9:2579–2605, 2008.

[36] Inga J. Wolf and Wolfgang Marquardt. Fast NMPC schemes for regulatory and economic NMPC: A review. Journal of Process Control, 44:162–183, 2016.

[37] Melanie N. Zeilinger, Davide M. Raimondo, Alexander Domahidi, Manfred Morari, and Colin N. Jones. On real-time robust model predictive control. Automatica, 50(3):683–694, 2014.