Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 53-2 (2020) 1231–1236

KBERG: A MatLab toolbox for nonlinear kernel-based regularization and system identification

M. Mazzoleni *, **M. Scandella** *, **F. Previdi** *

* *Department of Management, Information and Production engineering University of Bergamo, via Galvani 2, 24044 Dalmine (BG), Italy (e-mail: mirko.mazzoleni@unibg.it).*

Abstract: We present KBERG, a MatLab package for nonlinear Kernel-BasEd ReGularization and system identification. The toolbox provides a complete environment for running experiments on simulated and experimental data from both static and dynamical systems. The whole identification procedure is supported: (i) data generation, (ii) excitation signals design; (iii) kernel-based estimation and (iv) evaluation of the results. One of the main differences of the proposed package with respect to existing frameworks lies in the possibility to separately define experiments, algorithms and test, then combining them as desired by the user. Once these three quantities are defined, the user can simply run all the computations with only a command, waiting for results to be analyzed. As additional noticeable feature, the toolbox fully supports the manifold regularization rationale, in addition to the standard Tikhonov one, and the possibility to compute different (but equivalent) types of solutions other than the standard one.

Keywords: Kernel methods; System Identification

1. INTRODUCTION

In the last years, kernel methods became one of the predominant approaches for time-domain system identification. Due to their flexibility and regularization properties, they quickly showed improved performance with respect to traditional Prediction Error Methods (PEM), see Pillonetto et al. (2014), both in linear and nonlinear settings. Their employment for the identification of dynamical models is not limited to time-domain, but extends also to frequency-domain data, Darwish et al. (2017).

Kernel methods are nonparametric approaches which aim to find the (possibly nonlinear) function that best matches input/output data. This function estimate is searched withing a functional space called Reproducing Kernel Hilbert Space (RKHS), see Aronszajn (1950). The *kernel function* (or simply kernel) determines the properties of the functions inside its corresponding RKHS. In the linear systems case, the unknown function to be estimated is the impulse response of the system, as reviewed in Pillonetto et al. (2014). In the nonlinear case, the aim is to learn the mapping from the regressors vector (with predefined exogenous and autoregressive orders) to the system output, as done in Pillonetto et al. (2011); Mazzoleni et al. (2020).

When dealing with kernel methods, the practitioner is involved with the following choices, adapted from Ljung et al. (2019):

(1) the choice of the regularization type
(2) the choice of the kernel function
(3) the choice of method for estimating the hyperparameters of the model.

One of the main reasons for such popularity and effectiveness of kernel methods is due to their regularized nature. In their standard formulation, this equals to a *Tikhonov-like* regularization term. By trading data fit and solution complexity in a *continuous way*, better results can be achieved than employing complexity criteria such as Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) for model order selection, see Pillonetto et al. (2011, 2014). Recently, *manifold regularization* has been added to the standard kernel formulation for nonlinear system identification in Formentin et al. (2019); Mazzoleni et al. (2018a,b). Manifold regularization relies on the concept of *regressors graph* and on the assumption that "nearby regressors should have a similar corresponding output" (*smoothness assumption*).

In addition to basic kernels inherited from machine learning (e.g. the Gaussian or polynomial ones), specific kernels for nonlinear system identification, that take into account the nature of dynamical systems, were proposed in Pillonetto et al. (2011); Pillonetto (2018).

The model defined by the chosen kernel function, the graph topology and properties (for the manifold regularization case only) and the regularization weights determines a set of hyperparameters to be determined from data prior to the computation of the estimated model. Common approaches are based on Generalized Cross Validation (GCV), the Stein's Unbiased Risk Estimator (SURE) and the Empirical Bayes (EB) methods, see Mu et al. (2018a,b). The EB estimate is available by relying to a Bayesian interpretation of the kernel-based learning problem.

10.1016/j.ifacol.2020.12.1340

This paper presents a software environment for performing and evaluating kernel-based methods for nonlinear system identification. Given the vast amount of possible choices for the setup of the problem and the number of simulations required for testing a new kernel-based approach, a tool that permits to simplify the iterations of development and testing of a new method is highly sought. With this in mind, we developed KBERG, an open-source MatLab toolbox for nonlinear *Kernel-BasEd ReGularization and system identification*, that supports the user throughout all the identification steps. Peculiar features of KBERG are: (i) the definition of a all-in-one environment for testing kernel-based nonlinear system identification approaches; (ii) the possibility to easily combine experiments (i.e. input/output data), algorithms and test evaluations; (iii) a full support for the manifold regularization rationale; (iv) freely configurable settings of the constraints on hyperparameters estimation; (v) full user-extendable functionalities. While the toolbox can be used also for the kernel-based estimation of *static system*, in this paper we will focus on the *dynamical systems* case.

The KBERG toolbox is available at the following link [1]. For computing alternative solutions, it requires YALMIP, see Löfberg (2004), equipped with a solver such as CPLEX.

The remainder of the paper is organized as follows. Section 2 reviews the formulation of kernel-based nonlinear system identification problems. Section 3 describes the main entities that compose the toolbox. Section 4 walks through a full example of nonlinear dynamical system estimate. Section 5 is then devoted to some concluding remarks.

2. KERNEL-BASED NONLINEAR SYSTEM IDENTIFICATION

Consider a mapping $f : \mathcal{X} \to \mathbb{R}$, $\mathcal{X} \subset \mathbb{R}^{d\times 1}$, such that

$$y_t = f(\mathbf{x}_t) + e_t, \tag{1}$$

where $\mathbf{x}_t \in \mathcal{X}$ and $y_t \in \mathbb{R}$ are, respectively, the system input regressor and output at time $t \in \mathbb{Z}_{\geq 0}$, and $e_t \sim \text{WN}\left(0, \beta^2\right)$ is an additive white noise. The regressor $\mathbf{x}_t$ could contain past samples of both input u_t and output y_t of the SISO dynamical system that generates the data. In this context, $f\left(\mathbf{x}_t\right)$ is the one-step ahead predictor $\hat{y}_{t|t-1}$ and e_t is the one-step ahead prediction error. Suppose that we have n observations of regressor-output data $\mathcal{D} = \{\mathbf{x}_t, y_t\}_{t=1}^{n}$. The aim is to obtain an estimate $\hat{f}$ of the unknown mapping f using $\mathcal{D}$.

Kernel methods look for this estimate by solving the variational problem

$$\hat{f} = \arg\min_{f\in\mathcal{H}} \sum_{t=1}^{n} \left(y_t - f\left(\mathbf{x}_t\right)\right)^2 + \tau \cdot \|f\|_{\mathcal{H}}^2 + \mu \cdot \mathbf{f}^\top \mathbf{M} \mathbf{f}, \tag{2}$$

where $\tau \in \mathbb{R}_{>0}$ and $\mu \in \mathbb{R}_{>0}$ are constant values (called *hyperparameters*) and $\mathcal{H}$ is a Reproducing Kernel Hilbert Space (RKHS) characterized by the *kernel function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The second element in (2) is the Tikhonov regularization term, while the third one is the manifold regularization component. Here, $\mathbf{f} = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)]^\top \in \mathbb{R}^{n\times 1}$ is the vector of noiseless function evaluations at measured regressors points $\mathbf{x}_t, t = 1, \dots n$, and $\mathbf{M} \in \mathbb{R}^{n\times n}$ is the graph-shift operator of the graph that connects the regressors (see Mateos et al. (2019)), such that the t-th component of the vector $\mathbf{f}$ represents the function value at $\mathbf{x}_t$ (t-th node of the regressors graph). A common choice for $\mathbf{M}$ is the graph Laplacian. We usually consider a *weighted* graph, such that a weight ω_{rs} is associated to two connected regressors $\mathbf{x}_r$ and $\mathbf{x}_s$, which value depends on some hyperparameters $\boldsymbol{\gamma} \in \mathbb{R}^{q_m\times 1}$.

The solution of (2) reads as $\hat{f}\left(\mathbf{x}\right) = \sum_{t=1}^{n} \hat{c}_t k_{\mathbf{x}_t}\left(\mathbf{x}\right)$, with $\hat{\mathbf{c}} = \left[\hat{c}_1, \dots, \hat{c}_n\right]^\top \in \mathbb{R}^{n\times 1}$ given by

$$\hat{\mathbf{c}} = \left(\mathbf{K} + \tau I_n + \mu \mathbf{M}\mathbf{K}\right)^{-1} \mathbf{y}, \tag{3}$$

where $\mathbf{y} \in \mathbb{R}^{n\times 1}$ contains the available output measurements $y_t, t = 1, \dots, n$, $\mathbf{K} \in \mathbb{R}^{n\times n}$ is a positive semidefinite matrix such that $\mathbf{K}_{rs} = k(\mathbf{x}_r, \mathbf{x}_s)$. In the following we indicate the hyperparameters vector of the method with $\boldsymbol{\theta} = \left[\boldsymbol{\psi}^\top \ \tau \ \mu \ \boldsymbol{\gamma}^\top\right]^\top \in \mathbb{R}^{q\times 1}$, where $\boldsymbol{\psi} \in \mathbb{R}^{q_k\times 1}$ are the hyperparameters of the kernel k.

Remark 1. In the following, we will refer to (3) as the *trivial solution*, in order to distinguish it from other solutions that minimize some norm of the vector $\hat{\mathbf{c}}$.

3. DESCRIPTION OF THE TOOLBOX

This section describes the main elements of the toolbox to: (i) define a dynamic model and simulate data from it using excitation signals; (ii) define the kernel function k and its hyperparameters; (iii) define the regressors graph; (iv) define the hyperparameters optimization properties; (iv) compute the solution to problem (2); (v) evaluate the test results of the estimated system model.

3.1 Main entities and relationships

The main element of the toolbox is the **Set**. A set is composed by the following three main entities:

(1) **Experiments**: they define the system used to simulate the identification data, the input signal, the process and output noises and the number of Monte Carlo simulations (each simulation corresponds to a different stochastic noise realization).
(2) **Algorithms**: they define the regressor exogenous and autoregressive orders, the kernel employed, the regressors graph settings, the regularization types and options for hyperparameters optimization.
(3) **Tests**: they define the test input signals used to generate test data, the metric for evaluating the estimated model performance, the noise on test data and if a simulation or a prediction is required.

A set can contain multiple experiments, algorithms and tests. An algorithm can run on zero or different experiments, and an experiment can be run by zero or different algorithms. Likewise, a test can evaluate zero or different experiments, and an experiment can be evaluated by zero or multiple tests. Therefore, two $n : n$ relations are present between the entities experiment-algorithm and experiment-test, as shown in Figure 1. These relations translate to the fact that each test is run on each model (estimated by each learning algorithm).

[1] `https://cal.unibg.it/publications/kberg-a-matlab-toolbox-for-nonlinear-kernel-based-regularization-and-system-identification/`

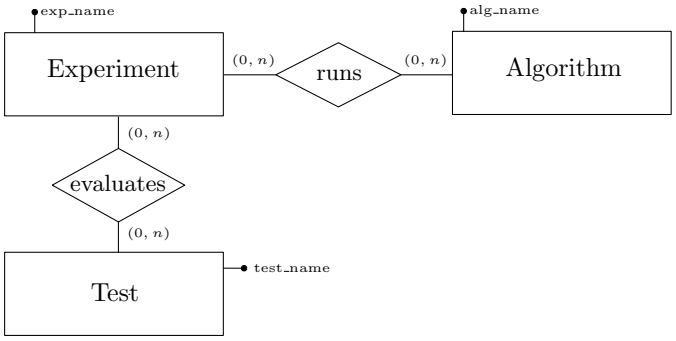

Fig. 1. Entities/Relations diagram of the main entities of the toolbox. The unique identifier of each entity is highlighted.

The philosophy of the toolbox is to *first define all the experiments, algorithms and tests*, and, subsequently, *run all the set with a single command*. The researcher can therefore focus its attention on other work while waiting for the computations to complete. When available, parallel computation can be leveraged.

The following results are then saved to disk in separate files: (i) the set configurations, i.e. system and algorithms names, input signal and noises settings, date of simulation, simulation noise seed and number of simulations; (ii) the simulation data; (iii) the estimated models for each algorithm, i.e. the vector $\hat{\mathbf{c}}$ in (3), the computation and optimization timings and the estimated hyperparameters $\boldsymbol{\theta}$; (iv) the performance results of each test applied to each model estimated by the algorithms.

The main entities cover all the following four system identification steps, as summarized in Table 1: (i) Data generation; (ii) Excitation signals design; (iii) Kernel-based estimation; (iv) Evaluation of the results.

Table 1. Mapping between the main toolbox entities and system identification steps.

Main entity	Identification steps covered
Experiment	(i) Data generation (ii) Excitation signals design
Algorithm	(iii) Kernel-based estimation
Test	(iv) Evaluation of the results

3.2 Data structures

The entities of the toolbox are implemented as the `struct` data-type in MatLab. In the following, we list the most important content of the structs used in the toolbox, along with the type of each element of the struct.

Let's first introduce the data structures of the three main entities: as it is possible to observe, they are composed of other structs that define, e.g., the input signal (`signal_struct`), the noises (`noise_struct`), the hyperparameters and tuning knobs of the method (`np_hp_struct`), and optimization settings (`np_op_struct`).

Experiment see `help mcs_exp_conf_struct`
- experiment_name: `string`
- system_type: `{'static', 'dynamic'}`
- sig_train: `signal_struct`
- ar_noise: `noise_struct`
- oe_noise: `noise_struct`
- seed: $\mathbb{N}$
- n_sim: $\mathbb{N}$

Algorithm see `help mcs_alg_struct`
- algorithm_name: `string`
- experiment_name: `string`
- hp: `np_hp_struct`
- op: `np_op_struct`

Test see `help mcs_test_struct`
- test_name: `string`
- experiment_name: `string`
- sig_test: `signal_struct`
- noise_type: `{'null', 'train', 'ar_train', 'oe_train', 'generic'}`

The `noise_type` attribute of the `mcs_test_struct` permits to define a test dataset that is either: (i) noiseless (`'null'`); (ii) with the same noise type as the identification dataset (`'train'`); (iii) with only the autoregressive or output-error part of the noise on identification data (`'ar_train', 'oe_train'`); (iv) with user-defined noise (`'generic'`, see `help noise_struct` to specify it).

Noise see `help noise_struct`
- noise: $\mathbb{R}$
- type: `{'snr', 'power'}`
- unit: `{'db', 'linear'}`

Signal see `help signal_struct`
- signal_type: `{'BLWN', 'sinewave', 'PRBS', 'multisine', 'ramp', 'step'}`
- [filter]: see MatLab `tf` command
- signal_hp: depends on signal type

The `signal_struct` defines the excitation input signal. It is possible to choose from pre-defined signal types and (optionally) filter them before their application to the system. Based on the signal type chosen, different signal hyperparameters have to be specified.

Hyperparameters see `help np_hp_struct`
- kernel_param: `kernel_struct`
- graph_param: `graph_struct`
- tau: $\mathbb{R}_{>0}$
- mu: $\mathbb{R}_{>0}$
- solution_type: `{'trivial', 'ln2', 'lnp'}`
- order_ar: $\mathbb{N}$
- order_ex: $\mathbb{N}$

The `np_hp_struct` permits to define the type and initial values of all the hyperparameters $\boldsymbol{\theta}$ and the tuning knobs of the algorithms. It is possible to set the kernel and graph properties, the regularization strengths, the type of the solution to be computed and the system orders for constructing the regressor vector.

Options see `help np_opt_struct`
- opt_index: `string`
- fmin_options: see MatLab `optimoptions` struct
- parallel_options: `opt_parallel_struct`

The `np_opt_struct` sets the user-defined objective function to be minimized, along with settings regarding the minimization algorithm (`optimoptions`) and parallel computations (`opt_parallel_struct`).

Kernel_params see `help kernel_struct`
- kernel_name: `string`
- kernel_hp: depends on the `kernel_name`

The `kernel_struct` defines the employed kernel and its hyperparameters $\boldsymbol{\psi}$. There are pre-defined "basic" kernels such as the constant, linear, polynomial and Gaussian ones. "Special" kernels are implemented, as the one in Pillonetto et al. (2011). It is possible to *create new kernels* by defining them or by *combine* existing kernels with standard operators such as $\{+, \cdot, \wedge\}$.

The settings of the regressors graph, used for the manifold regularization, are defined in the attribute **graph_params**, of type `graph_struct`. This is a composition of the structs `graph_edge_struct`, `graph_weights_struct` and `graph_manifold_struct`, that respectively define the settings for the graph edges, weights and the algorithm used to compute the regularization matrix $\mathbf{M}$.

Graph_edge see `help graph_edge_struct`
- edge_type: `{'all', 'quantity', 'range', 'range_tim', 'temporal'}`
- [edge_hp]: depends on the `edge_type` used.

Graph_weights see `help graph_weights_struct`
- weights_type: `{'Kernel', 'LLE'}`
- [weights_hp]: depends on the `weights_type` used.

Graph_manifold see `help graph_manifold_struct`
- manifold_type: `{'LEM', 'LEM_norm', 'LLE'}`

The `graph_edge_struct` defines how the regressors are connected. They can be: (i) all connected; (ii) connected with K other regressors; (iii) connected with regressors that are in a certain range defined by a radius; (iv) connected by following the method in Berry and Sauer (2016); (iv) connected by their temporal relation, see Formentin et al. (2019). The graph weights, defined by the `graph_weights_struct`, can be set by a custom function or using the Locally Linear Embedding (LLE) rationale in Roweis and Saul (2000). The regularization matrix is defined by the way the manifold is computed, as specified by `graph_manifold_struct`. Available algorithms are the Laplacian Eigenmaps (LEM), see Belkin and Niyogi (2003) and the LLE. For a comparison of different graph construction methods, see Mazzoleni et al. (2019).

The toolbox can be extended by adding the following custom objects: (i) signal types; (ii) kernels; (iii) graph edges and weights; (iv) static and dynamic systems; (v) solution types; (vi) hyperparameter estimation methods; (vii) performance indices for validation purposes.

4. A COMPLETE SYSTEM IDENTIFICATION EXAMPLE

This section shows a practical application of the KBERG toolbox for solving a system identification problem with kernel-based methods.

4.1 Data generation

We generate data from the following nonlinear dynamic system (named `placeholder_NP` and defined in the file `sys_dyn_placeholder_NP.m`)

$$y_{t+1} = u_t y_{t-1} + u_{t-2} y_t - 0.8 u_{t-3} + \eta_t \tag{4}$$

where autoregressive noise $\eta_t \sim \text{WGN}(0, 0.005)$ is present. The input signal u_t is a band-pass filtered white noise signal with zero mean and standard deviation of 0.1. The number of observed data is `n = 100`. The number of Monte Carlo simulations is `n_sim = 100`.

The following commands define the used system and the number of simulations:

```
%% system information
ex.experiment_name = 'example_exp';
ex.system_name = 'placeholder_NP';
ex.system_type = 'dynamic';
ex.seed = 12;  % for reproducibility
ex.n_sim = 100; % Monte Carlo simulations
```

4.2 Excitation signals design

After having selected the system, the next step is to design the input and noise signals. They are defined respectively by the data-types `signal_struct` and `noise_struct`:

```
%% Identification input signal
ex.sig_train.signal_type = 'BLWN'; % Filtered WN
ex.sig_train.lower_band = 0.1; % for filtering
ex.sig_train.upper_band = 0.8; % for filtering
ex.sig_train.mean = 0; % mean value
ex.sig_train.std = 0.1; % standard deviation
ex.sig_train.n = 100; % number of data
%% Noise signals
ex.ar_noise.unit = 'linear';
ex.ar_noise.type = 'power';
ex.ar_noise.value = 5e-3; % ar noise variance

ex.oe_noise.unit = 'linear';
ex.oe_noise.type = 'power';
ex.oe_noise.value = 0; % no output-error noise
```

An example of generated signals is shown in Figure 2.

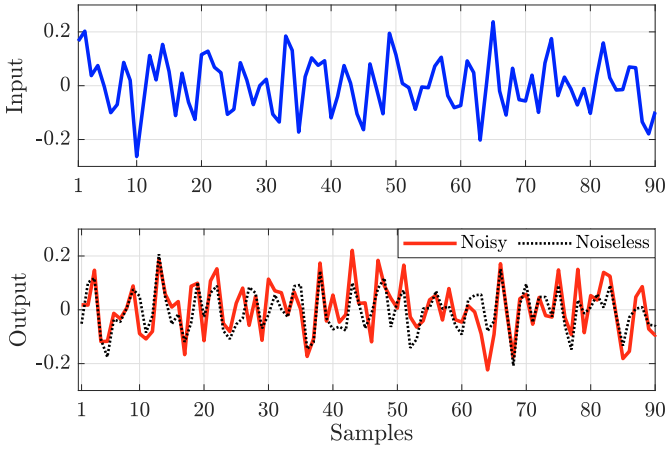

Fig. 2. Generated input and output identification data.

4.3 Kernel-based estimation

The model estimation is performed by defining the algorithms that have to be run on the previously specified

experiment. In this example, we defined two algorithms: (i) a standard Tikhonov-regularized kernel problem; (ii) a kernel problem with additional manifold regularization. Both algorithms employ the kernel in Pillonetto et al. (2011). For the first algorithm, hyperparameters are optimized with marginal likelihood optimization. For the second method, we employ the GCV estimator.

The first algorithm can be defined as follows (for sake of brevity, only the most important code is reported):

```
%% Associate the algorithm to the experiment
alg1.experiment_name = 'example_exp';
alg1.algorithm_name = 'example_tik';
%% System orders
alg1.hp.order_ex = 10; alg1.hp.order_ex = 10;
%% kernel parameters
alg1.hp.kernel_param.kernel_type = 'pillonetto';
alg1.hp.kernel_param.nl_beta.optimize = 'bounded'
alg1.hp.kernel_param.nl_beta.ini = 1;
alg1.hp.kernel_param.nl_beta.lb = 0;
alg1.hp.kernel_param.nl_beta.ub = inf;
alg1.hp.kernel_param.p.optimize = 'list';
alg1.hp.kernel_param.p.values = 1:9;
%% Regularization strenght parameters
alg1.hp.tau.optimize = 'bounded';
alg1.hp.tau.ini = 1; alg1.hp.tau.lb = 0;
alg1.hp.tau.ub = inf;
alg1.hp.mu = 0;  % only Tikhonov regularization
```

The above code snippet shows an important feature of the toolbox, i.e. the ability to deal with constraint on hyperparameters optimization. By specifying the property `optimize = 'bounded'` it is possible to constraint the hyperparameter value to a lower bound (`lb`) and an upper bound (`ub`). The keyword `optimize = 'list'` tests all values in the list `values` in a grid search fashion and retains the best one using identification data. The orders `order_ex` and `order_ar` are not optimized: if they have to be optimize, the keryword `optimize` has to be specified.

It is then possible to specify the type of solution that has to be computed, the method for estimating the hyperparameters and optimization options such as the optimization algorithm to use and its tolerances.

```
%% Type of solution
alg1.hp.solution_type = 'trivial';
%% Use marginal likelihood optimization
alg1.op.opt_index = 'marg';
%% Optimization options
alg1.op.fmin_options = optimoptions('fmincon');
alg1.op.fmin_options.ConstraintTolerance = 1e-8;
alg1.op.fmin_options.OptimalityTolerance = 1e-8;
alg1.op.fmin_options.MaxIterations = 1e4;
```

Regarding the second algorithm, most of the settings remain the same, so it is useful to copy the algorithm just defined:

```
alg2 = alg1;
alg2.algorithm_name = 'example_man';
```

We have now to define the graph properties. This can be performed with the following commands (suppose to use a Gaussian function for the edges weights):

```
%% Graph weights
alg2.hp.graph_param.weights_type = 'kernel';
alg2.hp.graph_param.gs_sigma.optimize = 'bounded'
alg2.hp.graph_param.gs_sigma.ini = 30;
alg2.hp.graph_param.gs_sigma.lb = eps;
alg2.hp.graph_param.gs_sigma.ub = inf;
alg2.hp.graph_param.gs_lambda = 1;
%% Graph edges
alg2.hp.graph_param.edge_type = 'temporal';
alg2.hp.graph_param.futu_dist.optimize='constrained'
alg2.hp.graph_param.futu_dist.to =
{'kernel_param', 'p'};
alg2.hp.graph_param.past_dist.optimize='constrained'
alg2.hp.graph_param.past_dist.to =
{'kernel_param', 'p'};
%% Manifold regularization
alg2.hp.mu = []; % reset previous settings of mu
alg2.hp.mu.optimize = 'bounded';
alg2.hp.mu.ini = 1; alg2.hp.mu.lb = 0;
alg2.hp.mu.ub = inf;
%% Manifold type
alg2.hp.graph_param.manifold_type = 'LEM';
%% Type of solution
alg2.hp.solution_type = 'trivial';
alg2.op.opt_index = 'gcv'; % use GCV for hyperparams
```

The first thing to notice is how, by defining `graph_param.weights_type = 'kernel'`, further hyperparameters need to be specified. In particular, `graph_param.gs_sigma` and `graph_param.gs_lambda` automatically define a Gaussian function for the edges weights. A similar approach can used with all the "basic kernels".

The previous code shows another possibility to constrain the hyperparameters: with the keyword `'constrained'`, and the value of an hyperparameter is constrained to be equal to the value of another hyperparameter. As an example, we have that both `graph_param.futu_dist` and `graph_param.past_dist` (defined by the `'temporal'` `edge_type`) are set equal to the value of `kernel_param.p`. Since we are referring to the nested hypeparameter `p`, it is necessary to use the syntax `{kernel_param, p}` for telling the software to which value the hyperparameters `graph_param.past_dist` and `graph_param.futu_dist` have to be constrained.

With this edge settings, a regressor at time t is connected to the t+`graph_param.futu_dist` regressors in the future and the $t-$`graph_param.past_dist` regressors in the past, see Formentin et al. (2019). The `kernel_param.p` tells the interaction order of the regressors at different time instants, see Pillonetto et al. (2011).

4.4 Evaluation of the results

The last step is to define the test signal to evaluate the estimated model performance. Suppose we want to evaluate the prediction performance on a white noise signal with $n = 500$ data, using various indicators such as the Root Mean Square Error (RMSE), its normalized version (NRMSE), or the Mean Absolute Error (MAE):

```
%% Associate the test to the experiment
test.experiment_name = 'example_exp';
test.test_name = 'example_test_wn';
%% Define test input and noise signals
test.sig_test.signal_type = 'BLWN';
test.sig_test.lower_band = 0.1;
test.sig_test.upper_band = 0.8;
test.sig_test.mean = 0;
test.sig_test.std = 0.1;
test.sig_test.n = 500; test.sig_test.seed = 23;
test.noise_type = 'null'; % no noise on data
%% Performance indicators to be computed
test.val_index = {'rmse', 'nrmse', 'mae'};
test.test_type = 'prediction';
```

The defined elements now need to be added to a set:

```
%% Add everything to the set
set_name = 'example_set';
mcs_exp_add(set_name, ex); % add experiment
mcs_alg_add(set_name, alg1); % add algorithm 1
mcs_alg_add(set_name, alg2); % add algorithm 2
mcs_test_add(set_name, test); % add test
%% Parallel properties
par_opt.parallel = true; % enable parallel
par_opt.verbose = 'detailed'; % show information
```

The set can now be *run with a single command*. Thus, the simulation effort is focused all in a single moment (in the beginning) and the researcher focus his/her time for other duties, optimizing the work.

```
%% Run the set
mcs_run_set(set_name, par_opt);
```

The estimation results are reported in Figure 3. It is possible to create boxplots that report the performance, on a specific test, of each of the algorithms defined.

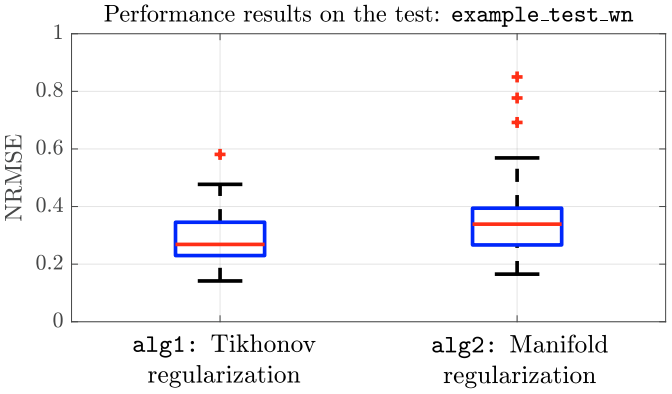

Fig. 3. Simulation results for the experiment `example_exp`, for each one of the defined algorithms `alg1` and `alg2`, on the test `'example_test_wn'`.

5. CONCLUSIONS

In this paper, we presented the MatLab toolbox KBERG, that permits to perform nonlinear nonparametric system identification using kernel methods. KBERG is a full-featured environment for performing simulations with dynamical systems, kernels and hyperparameters estimation methods. As peculiar characteristic, the toolbox fully supports the manifold regularization rationale and the possibility to compute alternative (but equivalent) solutions with respect to the trivial one. The software is very easy to extend with custom systems and kernels.

Future extension will regard the implementation of other methods for estimating the hyperparameters with respect to marginal likelihood or GCV, and the introduction of new kernels specifically developed for nonlinear system identification.

REFERENCES

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3), 337–404.

Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6), 1373–1396.

Berry, T. and Sauer, T. (2016). Consistent manifold representation for topological data analysis. *arXiv preprint arXiv:1606.02353*.

Darwish, M., Lataire, J., and Tóth, R. (2017). Bayesian frequency domain identification of lti systems with obfs kernels. *20th IFAC World Congress, Toulouse*, 50(1), 6238–6243. doi:10.1016/j.ifacol.2017.08.845.

Formentin, S., Mazzoleni, M., Scandella, M., and Previdi, F. (2019). Nonlinear system identification via data augmentation. *Systems & Control Letters*, 128, 56 – 63. doi:10.1016/j.sysconle.2019.04.004.

Ljung, L., Chen, T., and Mu, B. (2019). A shift in paradigm for system identification. *International Journal of Control*, 0(0), 1–8. doi:10.1080/00207179.2019.1578407.

Löfberg, J. (2004). Yalmip : A toolbox for modeling and optimization in matlab. In *In Proceedings of the CACSD Conference*. Taipei, Taiwan.

Mateos, G., Segarra, S., Marques, A.G., and Ribeiro, A. (2019). Connecting the dots: Identifying network structure via graph signal processing. *IEEE Signal Processing Magazine*, 36(3), 16–43. doi:10.1109/MSP.2018.2890143.

Mazzoleni, M., Scandella, M., and Previdi, F. (2019). A comparison of manifold regularization approaches for kernel-based system identification. *IFAC-PapersOnLine*, 52(29), 180 – 185. doi:https://doi.org/10.1016/j.ifacol.2019.12.641. 13th IFAC Workshop on Adaptive and Learning Control Systems ALCOS 2019.

Mazzoleni, M., Formentin, S., Scandella, M., and Previdi, F. (2018a). Semi-supervised learning of dynamical systems: a preliminary study. In *2018 European Control Conference (ECC)*, 2824–2829.

Mazzoleni, M., Scandella, M., Formentin, S., and Previdi, F. (2018b). Identification of nonlinear dynamical system with synthetic data: a preliminary investigation. *IFAC-PapersOnLine*, 51(15), 622 – 627. doi:10.1016/j.ifacol.2018.09.227. 18th IFAC Symposium on System Identification SYSID 2018.

Mazzoleni, M., Scandella, M., Formentin, S., and Previdi, F. (2020). Enhanced kernels for nonparametric identification of a class of nonlinear systems. In *18th European Control Conference (ECC20)*. IEEE.

Mu, B., Chen, T., and Ljung, L. (2018a). Asymptotic properties of generalized cross validation estimators for regularized system identification. *IFAC-PapersOnLine*, 51(15), 203 – 208. 18th IFAC Symposium on System Identification SYSID 2018.

Mu, B., Chen, T., and Ljung, L. (2018b). On asymptotic properties of hyperparameter estimators for kernel-based regularization methods. *Automatica*, 94, 381 – 395.

Pillonetto, G. (2018). System identification using kernel-based regularization: New insights on stability and consistency issues. *Automatica*, 93, 321 – 332.

Pillonetto, G., Dinuzzo, F., Chen, T., Nicolao, G.D., and Ljung, L. (2014). Kernel methods in system identification, machine learning and function estimation: A survey. *Automatica*, 50(3), 657 – 682. doi:10.1016/j.automatica.2014.01.001.

Pillonetto, G., Quang, M.H., and Chiuso, A. (2011). A new kernel-based approach for nonlinearsystem identification. *IEEE Transactions on Automatic Control*, 56(12), 2825–2840. doi:10.1109/TAC.2011.2131830.

Roweis, S.T. and Saul, L.K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500), 2323–2326.