Linear Dynamical Systems

Release 1.0.1

Kapa Kudaibergenov

Aug 13, 2021

CONTENTS:

CHAPTER

ONE

LDS

1.1 LDS package

1.1.1 Subpackages

LDS.LDS package

Subpackages

LDS.LDS.ds package

Submodules

LDS.LDS.ds.dynamical_system module

Originally the class comes from inputlds.py file.

class LDS.LDS.ds.dynamical_system.**DynamicalSystem**(*matrix_a*, *matrix_b*, *matrix_c*, *matrix_d*, ***kwargs*)

Bases: `object`

Creates LDS.

Inits DynamicalSystem with four matrix args and adds possibility of additional keywords in arguments.

G - matrix_a
w - process noise
F_dash - matrix_c
v - sensor noise.

If a matrix_a is a number, transforms it into float and makes d-state vector equal to 1. If a matrix_a is square y x y, set d equal to y. If a matrix_b is a number, transform it into float and set n-input vector equal to 1. matrix_b can't take place in case of single numbered matrix_a. If matrix_b is a matrix, number of its columns is assigned to n. If matrix_c is a number, transform it into float and set m-observation vector equal to 1. matrix_c can be a number only if matrix_a is a number too. If matrix_c is a matrix, number of its rows is assigned to m. matrix_d can't be not zero number if matrix_b is a matrix. Number of columns of matrix_d must be equal to n.

Parameters

- **matrix_a** – Evolution, system, transfer or state matrix (G matrix). Shape nxn.
- **matrix_b** – Control matrix.
- **noise. Shape nx1. Shape of covariance matrix nxn.%** (*%Processing*) –
- **matrix_c** – First derivative of the observation direction(aka design matrix F(nxm)). Shape mxn.
- **matrix_d** – Feedthrough matrix.
- **noise or observational error. Shape mx1. Shape of covariance matrix mxm.%** (*%Sensor*) –

Optional arguments

- **process_noise** – Processing noise w.
- **observation_noise** – Observation noise v.
- **timevarying_multiplier_b**
- **corrupt_probability**

Raises

- **KeyError** – in case of no additional keywords.
- **Exits in case of wrong format of a matrix.** –
- **Exits in case of not square matrix_a.** –
- **Exits in case of having any matrix_b, but matrix_a is a number.** –
- **Exits if number of rows of matrix_b isn't equal to d.** –
- **Exits if matrix_c is a number, but matrix_a is not.** –
- **Exits if number of columns of matrix_c is not equal to d.** –
- **Exits if matrix_b is a matrix, but matrix_d is not zero number.** –
- **Exits if number of columns of matrix_d is not equal to n-input vector.** –

check_input (*operator*)

Checks variable type of matrices A,B,C,D.

Parameters **operator** – Number or a matrix.

Returns 1

Raises **TypeError** – This error occurs if the argument is none of possible formats.

solve (*h_zero*, *inputs*, *t_t*, ***kwargs*)

Finds outputs of LDS. The function is used in filters to find the error of prediction.

t_t must be an integer greater than 1. Length of h_zero array must be equal to self.d(number of arrays in matrix A) if matrix_a is matrix If self.n-input vector is 1(matrix_b is a number), self.inputs will be transformed to a columns with t_t size. If matrix_b is matrix, inputs must have n x t_t size. If self.process_noise has Gaussian distribution, we create it with size d x t_t. If it isn't of Gaussian, we create matrix of zeros. If self.observation_noise has Gaussian distribution, we create it with size m x t_t. If it isn't of Gaussian, we create matrix of zeros. If it's wasn't given in init, we put earlies_event_time to zero.

Parameters

- **h_zero** – 1x2 array.
- **inputs** – Array of zeros of t_t size.
- **t_t** – Time horizon.

Optional arguments **earliest_event_time**

Raises

- **Exits if t_t is 1 or a float.** –
- **Exits if matrix_a is a number, but h_zero can't be transformed into float.** –
- **Exits if length of h_zero isn't equal to d(if matrix_a is matrix)** –
- **Exits if self.n==1, but inputs don't have a size of t_t.** –
- **Exits if matrix_b is a matrix, but inputs don't have n x t_t size.** –

Module contents

LDS.LDS.filters package

Submodules

LDS.LDS.filters.filtering_abc_class module

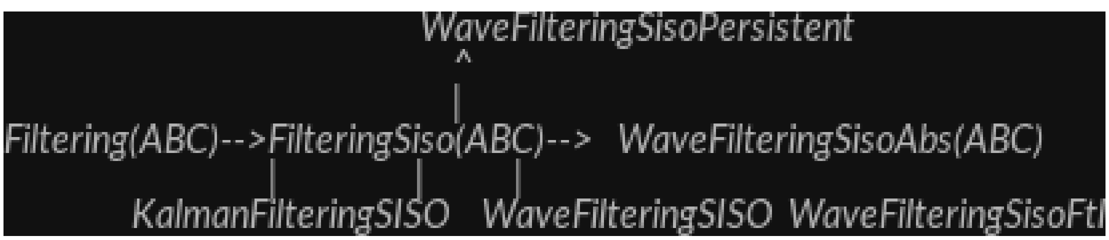
This script implements ABC class.

class `LDS.LDS.filters.filtering_abc_class.`**`Filtering`**(*sys*, *t_t*)

Bases: `abc.ABC`

Abstract class for creation of filters.

Hierarchy tree ((ABC)):



Initializing a basic filter.

Parameters

- **sys** – LDS. DynamicalSystem object.
- **t_t** – Time horizon.

LDS.LDS.filters.filtering_siso module

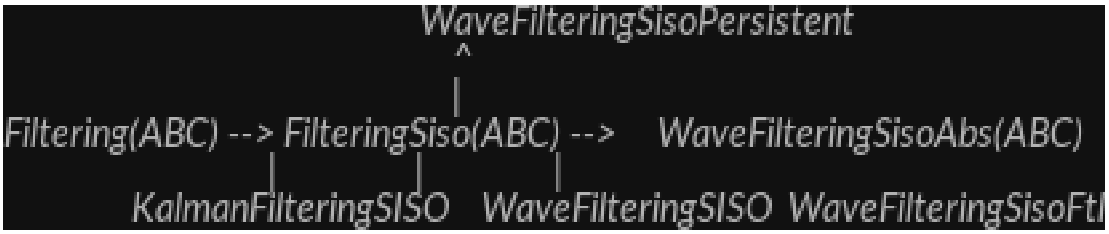
This script implements ABC class.

class `LDS.LDS.filters.filtering_siso.`**`FilteringSiso`**(*sys*, *t_t*)

Bases: *LDS.LDS.filters.filtering_abc_class.Filtering*

Abstract class. Specifically written to separate Kalman filter and auto-regression from spectral and persistent filters.

Hierarchy tree ((ABC)):



Inherits init method of Filtering.

Parameters

- **sys** – LDS. DynamicalSystem object.
- **t_t** – Time horizon.

abstract predict()

Creates empty lists for prediction and error of filters.

Returns

tuple containing:

- y_pred_full : Output prediction.
- pred_error : Prediction error.

Return type (tuple)

LDS.LDS.filters.kalman_em module

LDS.LDS.filters.kalman_filtering_siso module

Implements Kalman filter prediction. Originates from function Kalman_filtering_SISO from onlinelds.py.

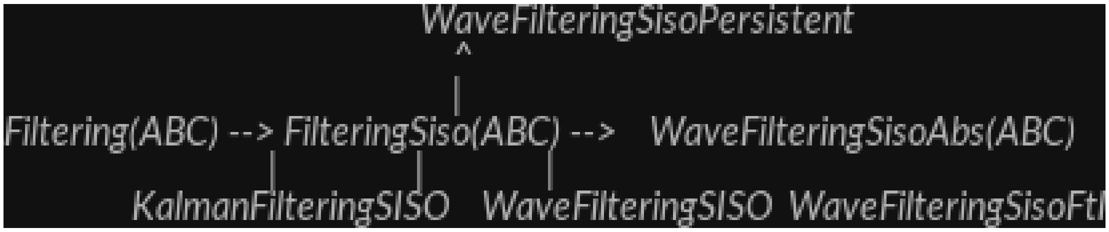
class `LDS.LDS.filters.kalman_filtering_siso.`**`KalmanFilteringSISO`**(*sys*, *G*, *f_dash*, *proc_noise_std*, *obs_noise_std*, *t_t*, *Y*)

Bases: *LDS.LDS.filters.filtering_siso.FilteringSiso*

Calculates Kalman filter parameters. Finds the prediction for Kalman and auto-regression. Uses abstract superclass FilteringSiso.

Fig. 1.1 shows an example of predictions of the Kalman, spectral and persistent filters. G and F' are random normally distributed matrices.

Hierarchy tree ((ABC)):



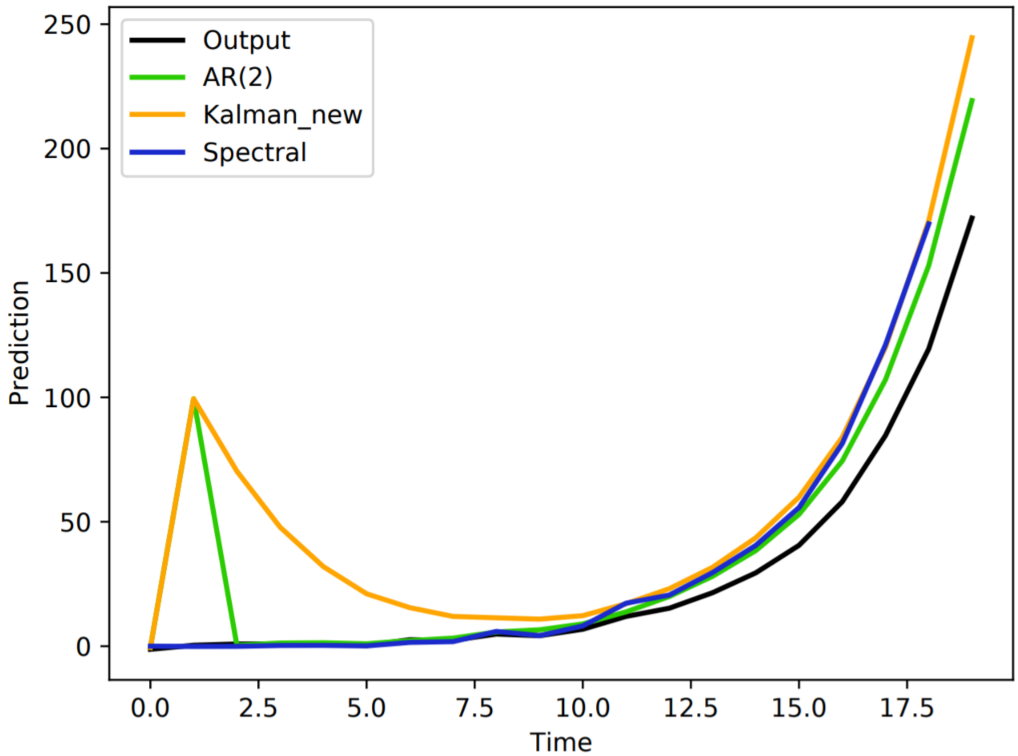


Fig. 1.1: The prediction of the Kalman filter, persistent filter and spectral filter compared to the real outputs over N = 100 iterations.

Inherits init method of FilteringSiso. Assignment variable names to LDS matrices. Calls to method "parameters".

Parameters

- **sys** – LDS. DynamicalSystem object.
- **G** – State transition matrix. Shape nxn.
- **f_dash** – Observation direction. Shape mxn.
- **proc_noise_std** – Standard deviation of processing noise.
- **obs_noise_std** – Standard deviation of observation noise.
- **t_t** – Time horizon.
- **Y** – Observations.

parameters()

Finds Kalman filter's parameters:

Parameters

- **n** – Input vector. Shape of processing noise.
- **m** – Observation vector. Shape of observational error.
- **W** – Processing noise covariance.
- **V** – Observation noise covariance.
- **matrix_c** – Covariance matrix of state.
- **R** – Covariance matrix of observation noise.
- **Q** – Covariance matrix of processing noise.
- **matrix_a** – Kalman filter parameter. Not the same as matrix_a in DynamicalSystem class.
- **Z** – Kalman filter parameter.

Raises: #Not raises yet Q can't be zero.

predict(*s*, *error_AR1_data*, *error_kalman_data*)

Calculates output predictions and errors for auto-regression and Kalman filter.

Parameters

- **s** – Auto-regression depth.
- **error_AR1_data** – Auto-regression error. 2-norm.
- **error_kalman_data** – Kalman error. 2-norm.

Returns

tuple containing:

- y_pred_full : Output prediction.
- error_AR1_data : Auto-regression error. 2-norm.
- error_kalman_data : Kalman error. 2-norm.

Return type (tuple)

predict_kalman(*s*, *error_AR1_data*, *error_kalman_data*)

Calculates output predictions and errors for auto-regression and Kalman filter.

Parameters

- **s** – Auto-regression depth.
- **error_AR1_data** – Auto-regression error. 2-norm.
- **error_kalman_data** – Kalman error. 2-norm.

Returns

tuple containing:

- y_pred_full : Output prediction.
- error_AR1_data : Auto-regression error. 2-norm.
- error_kalman_data : Kalman error. 2-norm.

Return type (tuple)

LDS.LDS.filters.wave_filtering_siso module

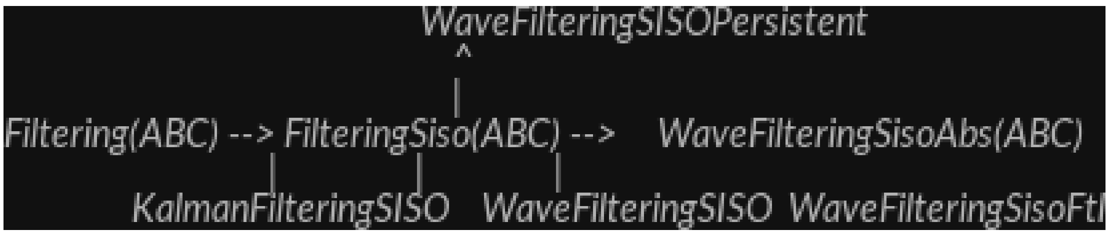
Originates from function wave_filtering_SISO from onlineLDS.py. The related work is "Learning Linear Dynamical Systems via Spectral Filtering" by E.Hazan, K.Singh and C.Zhang.

class `LDS.LDS.filters.wave_filtering_siso.`**`WaveFilteringSISO`**(*sys*, *t_t*, *k*, *eta*, *r_m*)

Bases: *`LDS.LDS.filters.wave_filtering_siso_abs.WaveFilteringSisoAbs`*

Implements spectral filter. Fig. 1.2 shows an example of its real-time prediction.

Hierarchy tree ((ABC)):



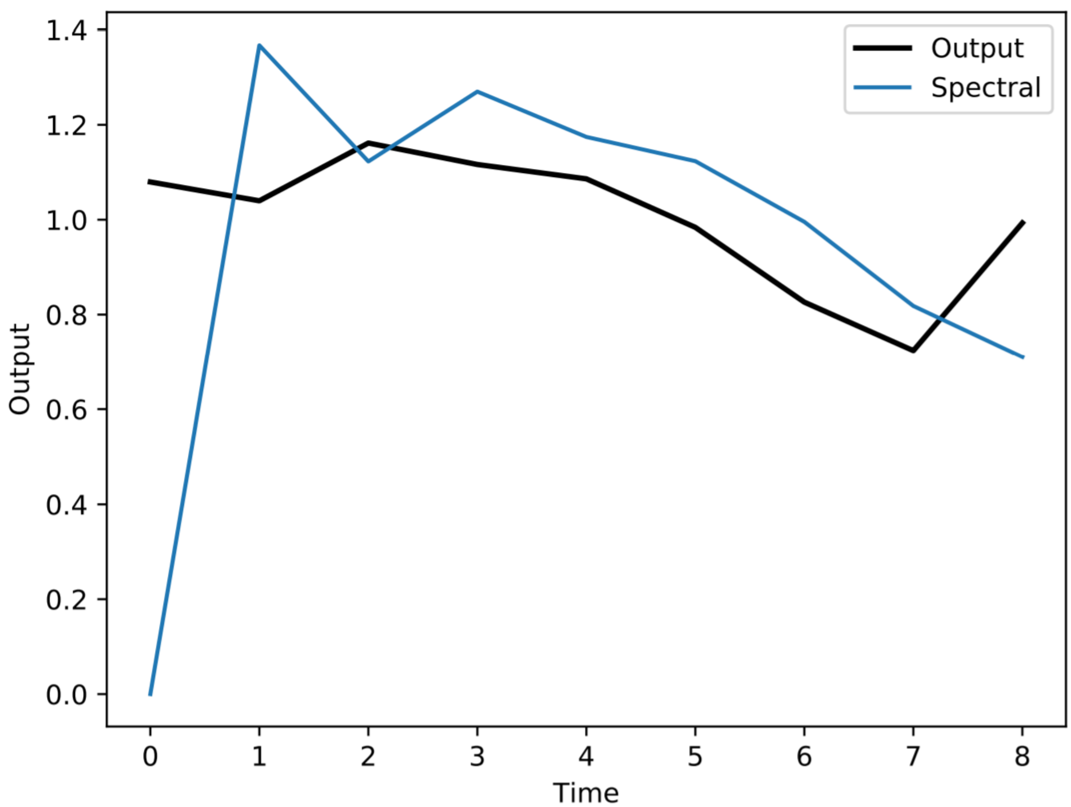


Fig. 1.2: The prediction of the spectral filter compared to the real outputs over N = 100 iterations.

Inits all the attributes of its superclass(see WaveFilteringSisoAbs) and adds Learning rate and Radius parameter. Goes through all the methods and gets the predictions.

Parameters

- **sys** – LDS. DynamicalSystem object.
- **t_t** – Time horizon.
- **k** – Number of wave-filters for a spectral filter.
- **eta** – Learning rate.
- **r_m** – Radius parameter.

predict()

Calculation of output predictions and prediction errors.

Returns

tuple containing:

- y_pred_full : Prediction values.
- **M** [Matrix specifying a linear map] from featurized inputs to predictions.
- pred_error : Spectral filter error.

Return type (tuple)

LDS.LDS.filters.wave_filtering_siso_abs module

In FilteringSiso we separated KalmanFilteringSISO. By doing this we dedicated the written below class to be the abstract class for spectral and persistent filters.

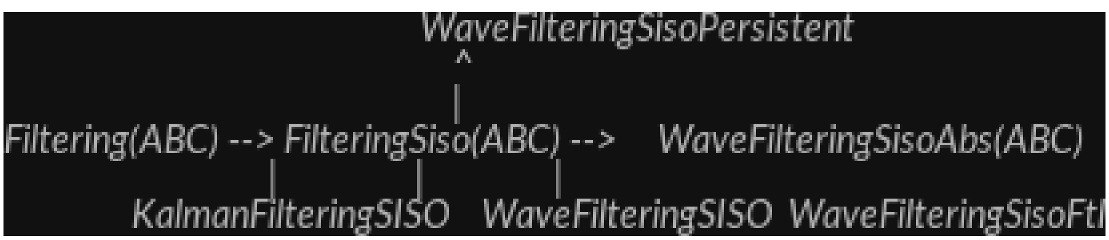
class LDS.LDS.filters.wave_filtering_siso_abs.**WaveFilteringSisoAbs**(*sys*, *t_t*, *k*)

Bases: *LDS.LDS.filters.filtering_siso.FilteringSiso*

Abstract class for creation of persistent and spectral filters. The subclass WaveFilteringSISO is spectral filter only for symmetric transition matrix. The related work is "Learning Linear Dynamical Systems via Spectral Filtering" by E.Hazan, K.Singh and C.Zhang.

WaveFilteringSisoFtl is the class for general case prediction. The related work is "Spectral Filtering for General Linear Dynamical Systems" by E.Hazan, K.Singh, H.Lee and C.Zhang.

Hierarchy tree ((ABC)):



Inherits FilteringSiso method.

Parameters

- **sys** – LDS. DynamicalSystem object.
- **t_t** – Time horizon.
- **k** – Number of wave-filters for a spectral filter.

abstract predict()

Abstract method for calculating output predictions and errors.

Returns

tuple containing:

- y_pred_full : Output prediction.
- **M** [Matrix specifying a linear map from featurized inputs to predictions.] Siso filter parameter.
- pred_error : Spectral filter prediction error.
- pred_error_persistent : Persistent filter error.

Return type (tuple)

var_calc()

Initializes spectral filter's parameters:

Parameters

- **n** – Input vector. Shape of processing noise.
- **m** – Observation vector. Shape of observational error.
- **k_dash** – Siso filter parameter.
- **H** – Hankel matrix.
- **M** – Matrix specifying a linear map from featurized inputs to predictions. Siso filter parameter.

LDS.LDS.filters.wave_filtering_siso_ftl module

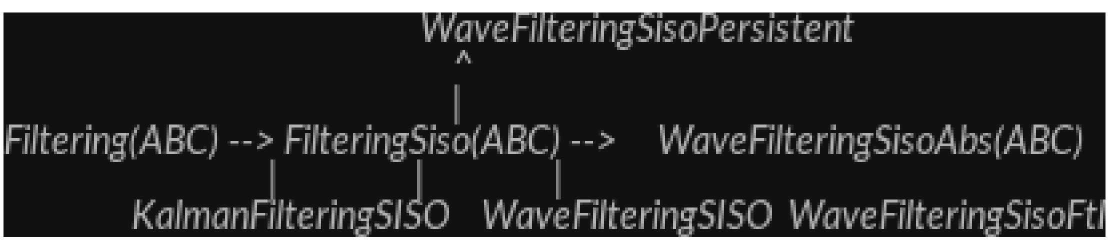
Implements spectral filtering class with follow-the-leader algorithm. Originates from function wave_filtering_SISO_ftl from onlinelds.py. The related work is "Spectral Filtering for General Linear Dynamical Systems" by E.Hazan, K.Singh, H.Lee and C.Zhang.

class LDS.LDS.filters.wave_filtering_siso_ftl.**WaveFilteringSisoFtl**(*sys*, *t_t*, *k*)

Bases: *LDS.LDS.filters.wave_filtering_siso_abs.WaveFilteringSisoAbs*

Spectral filter with follow-the-leader algorithm.

Hierarchy tree ((ABC)):



Inherits all the attributes of its superclass(see WaveFilteringSisoAbs). With initialization goes through all the methods and gets the predictions.

Parameters

- **sys** – LDS. DynamicalSystem object.
- **t_t** – Time horizon.
- **k** – Number of wave-filters for a spectral filter.

Variables initialized with var_calc():

Variables

- **n** – Input vector. Shape of processing noise.
- **m** – Observation vector. Shape of observational error.
- **k_dash** – Siso filter parameter.
- **H** – Hankel matrix.
- **M** – Matrix specifying a linear map from featurized inputs to predictions. Siso filter parameter.

Uses method args4ftl_calc to create an array with m and k_dash.

`args4ftl_calc``()`

Parameters calculation.

Creates a 5-element array with m on the zero position and k_dash on the first position. All others are zeros.

- self.m : Observation vector. Shape of observational error.
- self.k_dash : Siso filter parameter.

`predict``()`

Prediction step.

Returns

tuple containing:

- y_pred_full : Output prediction.
- **M** [Matrix specifying a linear map from featurized inputs] to predictions. Siso filter parameter.
- pred_error_persistent : Persistent filter prediction error.

Return type (tuple)

LDS.LDS.filters.wave_filtering_siso_ftl_persistent module

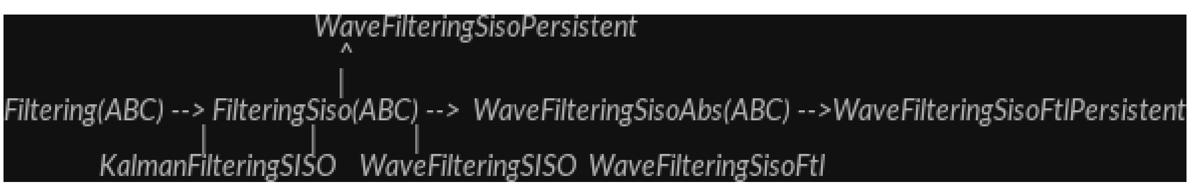
Implements persistent filter with follow-the-leader algorithm. Originates from function wave_filtering_SISO_ftl from onlinelds.py. The related work is "Spectral Filtering for General Linear Dynamical Systems" by E.Hazan, K.Singh, H.Lee and C.Zhang.

class `LDS.LDS.filters.wave_filtering_siso_ftl_persistent.`**`WaveFilteringSisoFtlPersistent`**(*sys*, *t_t*, *k*)

Bases: *LDS.LDS.filters.wave_filtering_siso_abs.WaveFilteringSisoAbs*

Persistent filter with follow-the-leader algorithm.

Hierarchy tree ((ABC)):



Inherits all the attributes of its superclass(see WaveFilteringSisoAbs). With initialization goes through all the methods and gets the predictions.

Parameters

- **sys** – LDS. DynamicalSystem object.
- **t_t** – Time horizon.
- **k** – Number of wave-filters for a spectral filter.

Variables initialized with var_calc():

Variables

- **n** – Input vector. Shape of processing noise.
- **m** – Observation vector. Shape of observational error.
- **k_dash** – Siso filter parameter.
- **H** – Hankel matrix.
- **M** – Matrix specifying a linear map from featurized inputs to predictions. Siso filter parameter.

Uses method args4ftl_calc to create an array with m and k_dash.

args4ftl_calc()

Parameters calculation.

Creates a 5-element array with m on the zero position and k_dash on the first position. All others are zeros.

- self.m : Observation vector. Shape of observational error.
- self.k_dash : Siso filter parameter.

predict()

Prediction step.

Returns

tuple containing:

- y_pred_full : Output prediction.
- **M** [Matrix specifying a linear map from featurized inputs] to predictions. Siso filter parameter.
- pred_error_persistent : Persistent filter prediction error.

Return type (tuple)

LDS.LDS.filters.wave_filtering_siso_persistent module

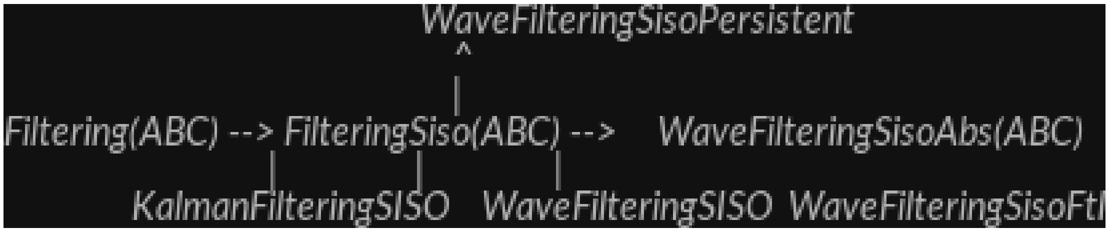
Originates from function wave_filtering_SISO from onlineLDS.py. The related work is "Learning Linear Dynamical Systems via Spectral Filtering" by E.Hazan, K.Singh and C.Zhang.

class LDS.LDS.filters.wave_filtering_siso_persistent.**WaveFilteringSISOPersistent**(*sys*, *t_t*, *k*, *eta*, *r_m*)

Bases: *LDS.LDS.filters.wave_filtering_siso_abs.WaveFilteringSisoAbs*

Implements persistent filter. Fig. 1.3 shows comparison of filter's errors. Fig. 1.4 shows comparison of spectral and persistent filters.

Hierarchy tree ((ABC)):



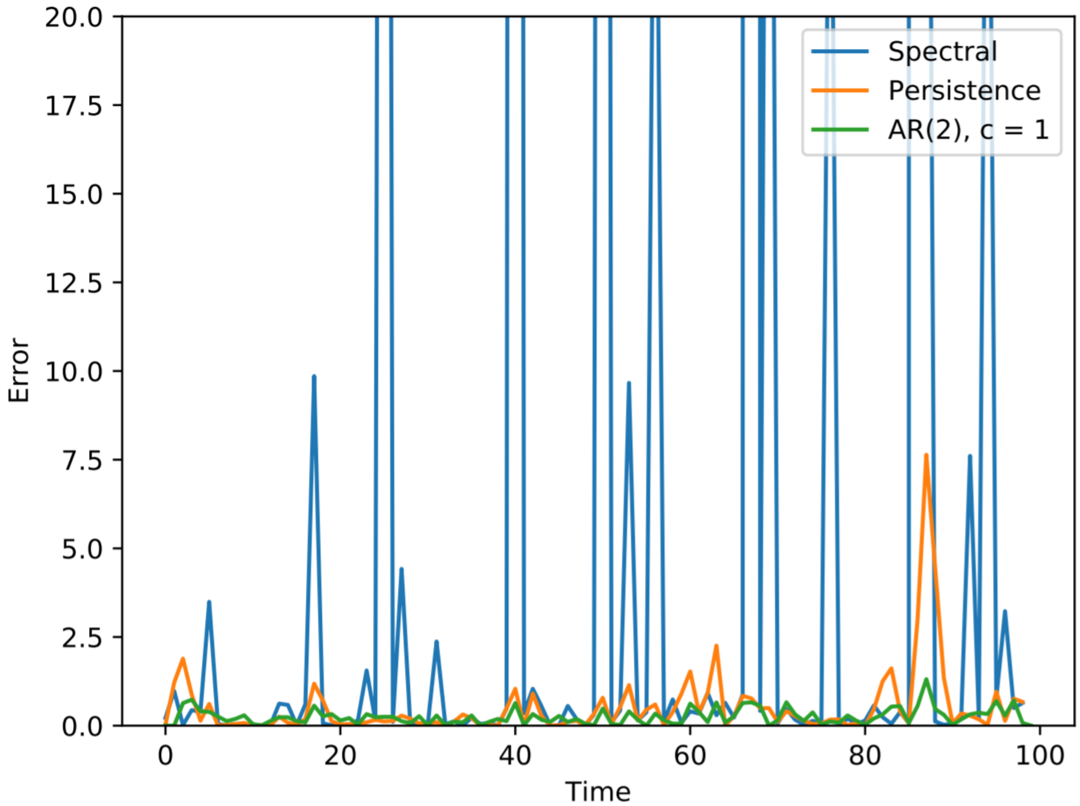


Fig. 1.3: The errors of the persistent, spectral filters and AR(2) on the first T=100 elements of the time series.

Inits all the attributes of its superclass(see WaveFilteringSisoAbs) and adds Learning rate and Radius parameter. Goes through all the methods and gets the predictions.

Parameters

- **sys** – LDS. DynamicalSystem object.
- **t_t** – Time horizon.
- **k** – Number of wave-filters for a spectral filter.
- **eta** – Learning rate.
- **r_m** – Radius parameter.

predict ()

Calculation of output predictions and prediction errors.

Returns

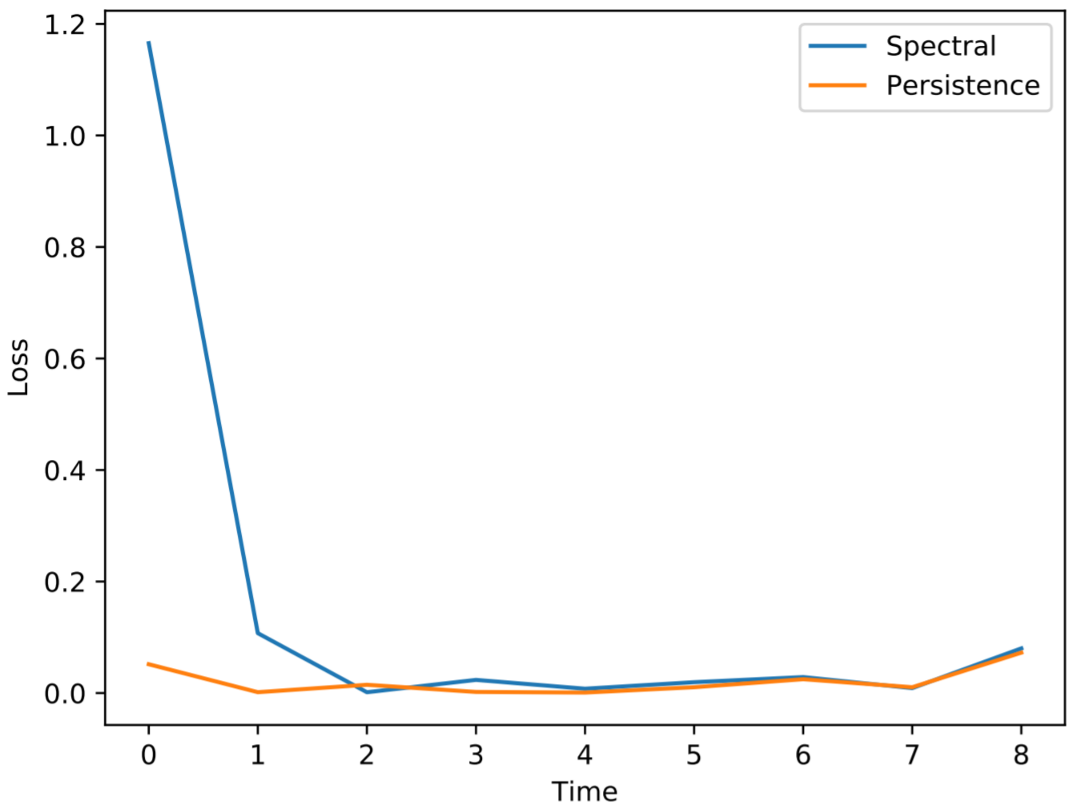


Fig. 1.4: The loss of the persistent filter compared to the spectral filter over N=100 iterations.

tuple containing:

- y_pred_full: Prediction values.
- **M: Matrix specifying a linear map** from featurized inputs to predictions.
- pred_error_persistent : Persistent filter error.

Return type (tuple)

Module contents

LDS.LDS.h_m package

Submodules

LDS.LDS.h_m.hankel module

Implements Hankel matrix.

class LDS.LDS.h_m.hankel.**Hankel**(*t_t*)

Bases: object

Class originated from onlinelds.py, which was the first version of the algorithm. Creates Hankel matrix.

Inits Hankel class with t_t argument. Stores Hankel matrix, its eigenvalues and normalized eigenvectors.

Parameters **t_t** – integer, size of Hankel matrix

Module contents

LDS.LDS.matlab_options package

Submodules

LDS.LDS.matlab_options.matlab_class_options module

Analogy of Matlab options

class LDS.LDS.matlab_options.matlab_class_options.**ClassOptions**

Bases: object

Mimics 'options' from Matlab

Module contents

LDS.LDS.online_lds package

Submodules

LDS.LDS.online_lds.cost_ftl module

`LDS.LDS.online_lds.cost_ftl.`**`cost_ftl`**(*M_flat*, **args*)
from onlinelds.py

LDS.LDS.online_lds.gradient_ftl module

`LDS.LDS.online_lds.gradient_ftl.`**`gradient_ftl`**(*M_flat*, **args*)
from onlinelds.py

LDS.LDS.online_lds.print_verbose module

`LDS.LDS.online_lds.print_verbose.`**`print_verbose`**(*a*, *verbose*)
from onlinelds.py

Module contents

LDS.LDS.ts package

Submodules

LDS.LDS.ts.time_series module

Implements time series from inputlds.py

`class` `LDS.LDS.ts.time_series.`**`TimeSeries`**(*matlabfile*, *varname*)
Bases: `object`

Class originated from inputlds.py, which was the first version of the algorithm.

Inits TimeSeries.

Parameters

- **`matlabfile`** – the matlab file './OARIMA_code_data/data/setting6.mat'
- **`varname`** – uses 'seq_d0'. 1 x 35701 double vector.

Raises **`HDF5ExtError`** – can't open given matlabfile.

`logratio`()
Replaces the time series by a log-ratio of subsequent element therein.

`solve`(*h_zero=[]*, *inputs=[]*, *t_t=100*, ***kwargs*)
This just truncates the series loaded in the constructor.

Raises **`Exits if time horizon isn't an integer.`** –

Module contents

Module contents

1.1.2 Submodules

1.1.3 LDS.OnlineLDS_library module

`LDS.OnlineLDS_library.`**`A_trans_calc`**`(`*A_trans*, *grad*`)`
begin{gather} Regret(w_{1:T}) = sum_{t=0}^{T}l(heta_t)-min_{L in S} sum_{t=0}^{T}l(y_{t},f_{t}(L)), end{gather} MATLAB: A_trans = A_trans - A_trans * grad' * grad * A_trans/(1 + grad * A_trans * grad'); we have to convert data[] from 1D vector to a numpy matrix (2D) to apply the transpose OR data[].reshape(-1,1) can be also used to mimick the transpose.

Parameters

- **`A_trans`** – np.eye(mk) * epsilon
- **`grad`** – Gradient, the return of the function grad_calc.

Returns:

`LDS.OnlineLDS_library.`**`arima_ogd`**`(`*data*, *options*`)`
from arima_ogd.m Used by example.py. ARIMA Online Newton Step algorithm. The function was originally written in MATLAB by Liu, C.; Hoi, S. C. H.; Zhao, P.; and Sun, J. It's described in their work "Online arima algorithms for time series prediction."

Parameters

- **`data`** – Array of 10000 elements.
- **`options`** – Instance of ClassOptions class.

Returns:

`LDS.OnlineLDS_library.`**`arima_ons`**`(`*data*, *options*`)`
Originates from arima_ons.m. ARIMA Online Newton Step algorithm. Used by example.py. The function was originally written in MATLAB by Liu, C.; Hoi, S. C. H.; Zhao, P.; and Sun, J. It's described in their work "Online arima algorithms for time series prediction."

Parameters

- **`data`** – Array of 10000 elements.
- **`options`** – Instance of ClassOptions class.

Returns:

`LDS.OnlineLDS_library.`**`close_all_figs`**`()`
Closes all the figures. Originally the function comes from experiments.py file.

`LDS.OnlineLDS_library.`**`cost_ar`**`(`*theta*, **args*`)`
Loss function of auto-regression. After the prediction is made, the true observation is revealed to the algorithm, and a loss associated with the prediction is computed. Here we consider the quadratic loss for simplicity. Originally the function comes from onlinelds.py file.

Parameters

- **`theta`** – auto-regressive parameters.
- **`args[0]`** – observation at time t
- **`args[1]`** – past s observations (most to least recent: t-1 to t-1-s)

Returns Quadratic loss function of auto-regression.

LDS.OnlineLDS_library.**diff_calc**(*w*, *data*, *mk*, *i*)

Auxiliary function to implement ARIMA in python. Others functions use it in their iterations. MATLAB: diff = w*data(i-mk:i-1)'-data(i); remember! MATLAB_data(1) == Python_data[0] we have to convert data[] from 1D vector to a numpy matrix (2D) to apply the transpose OR data[].reshape(-1,1) can be also used to mimick the transpose

Parameters

- **w** – Uniform distribution array with options.mk number of columns.
- **data** – Array of 10000 elements.
- **mk** – Integer number. Here we used 10.
- **i** – Iterative number. In range from mk till data - 1.

Returns:

LDS.OnlineLDS_library.**error_stat**(*error_spec_data*, *error_persist_data*)

if have_spectral_persistent:

Returns Mean error of spectral filtering error_spec_std: Std of spectral filtering error error_persist_mean: Mean error of last-value prediction error_persist_std: Std of last-value prediction error

Return type error_spec_mean

LDS.OnlineLDS_library.**grad_calc**(*data*, *i*, *mk*, *diff*)

MATLAB: grad = 2*data(i-mk:i-1)*diff Used by function arima_ons.

Parameters

- **data** – Array of 10000 elements.
- **i** – Iterative number. In range from mk till data - 1.
- **mk** – Integer number. Here we used 10.
- **diff** – Result of diff_calc function

Returns Gradient.

LDS.OnlineLDS_library.**gradient_ar**(*theta*, **args*)

Gradient function of auto-regression. We use the general scheme of on-line gradient decent algorithms, where the update goes against the direction of the gradient of the current loss. In addition, it is useful to restrict the state to a bounded domain. Originally the function comes from onlinelds.py file.

Parameters

- **theta** – s parameters.
- **args[0]** – Observation.
- **args[1]** – Past s observations.

Returns Gradient function of auto-regression.

LDS.OnlineLDS_library.**heatmap**(*data*, *row_labels*, *col_labels*, *ax=None*, *cbar_kw={}*, *cbarlabel=''*, ***kwargs*)

The function is taken from pyplot documentation. Create a heatmap from a numpy array and two lists of labels. Used by testNoiseImpact to implement Fig. 1.5 and Fig. 1.6. Originally the function comes from experiments.py file.

Parameters

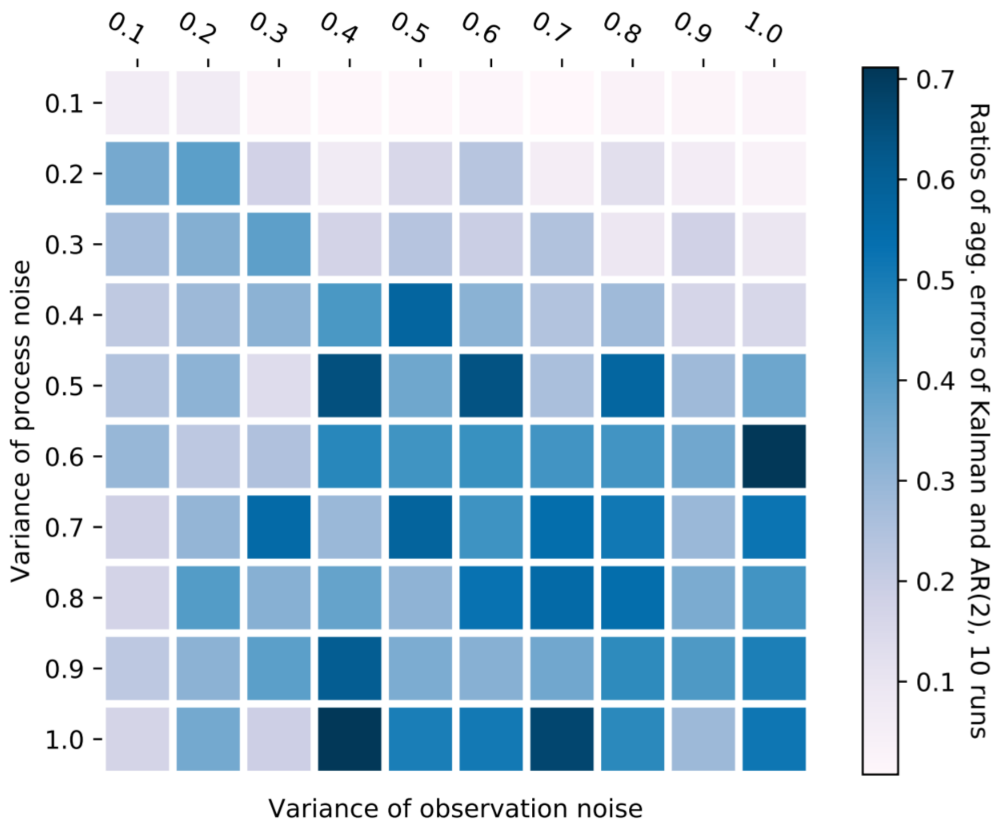


Fig. 1.5: The ratio of the errors of Kalman filter and AR(2) on Example 7 from Marecek's paper indicated by colours as a function of w, v of process and observation noise, on the vertical and horizontal axes, resp. Origin is the top-left corner.

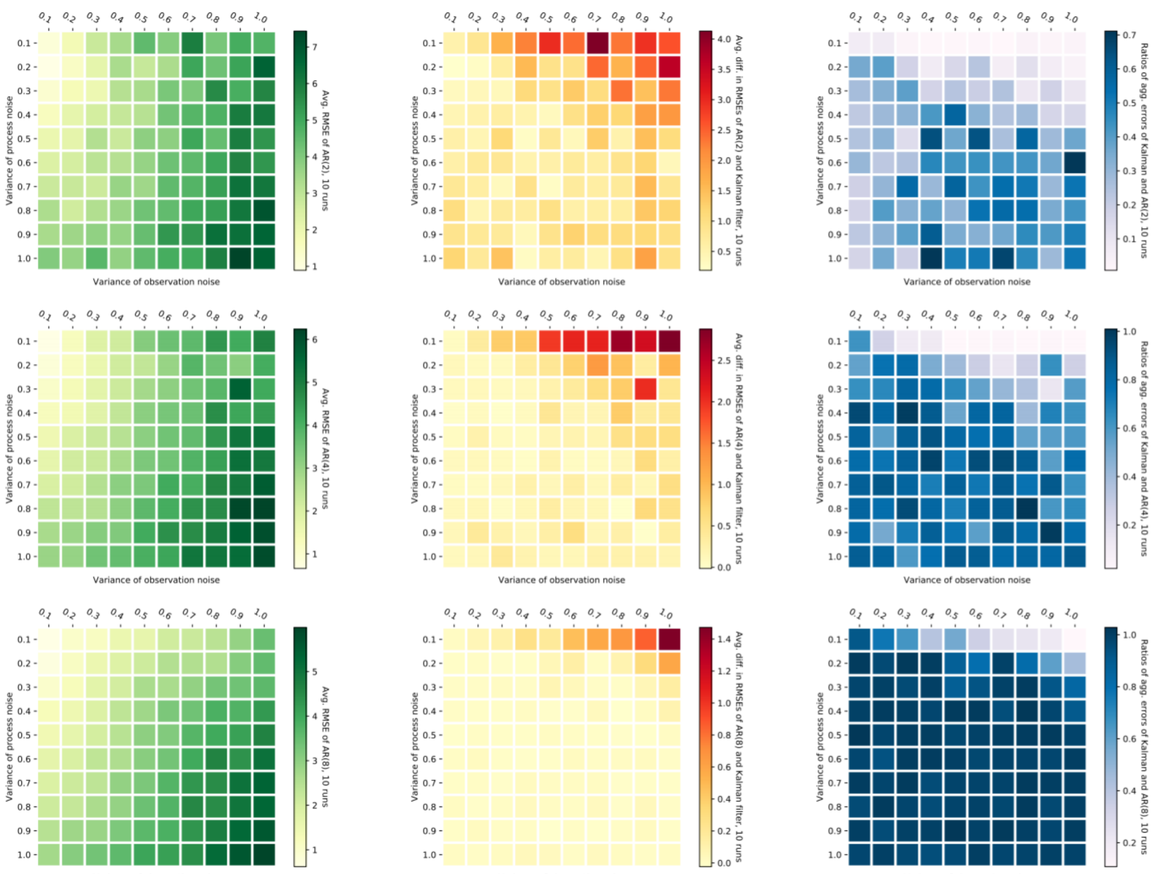


Fig. 1.6: The effect of varying the magnitude of noise in Example 7 on AR(2) (top), AR(4) (middle), and AR(8) (bottom). Left: average RMSE of predictions of AR(s+ 1) as a function of the variance of the process noise (vertical axis) and observation noise (horizontal axis). Center: The differences in average RMSE of Kalman filters and AR(s + 1) as a function of the variance of the process noise (vertical axis) and observation noise (horizontal axis). Throughout averages are taken over 10 runs. Right: The ratio (70) of the errors of Kalman filters and AR(s + 1) as a function of the variance of the process noise (vertical axis) and observation noise (horizontal axis). Throughout, notice the origin is in the topleft corner.

- **data** – A 2D numpy array of shape (N,M)
- **row_labels** – A list or array of length N with the labels for the rows
- **col_labels** – A list or array of length M with the labels for the columns

Optional arguments

- **ax** – A matplotlib.axes.Axes instance to which the heatmap is plotted. If not provided, use current axes or create a new one.
- **cbar_kw** – A dictionary with arguments to `matplotlib.Figure.colorbar()`.
- **cbarlabel** – The label for the colorbar

All other arguments are directly passed on to the imshow call.

`LDS.OnlineLDS_library.`**`lab`**(*s*, *eta_zero*)

Gives a label to auto-regression outputs and labels in seq0,seq1,seq2 pdfs.

Returns auto-regression label. Example: "AR(2), c = 2500".

Return type lab1

`LDS.OnlineLDS_library.`**`p3_for_test_identification2`**(*ylim*, *have_spectral_persistent*, *Tlim*, *error_spec*, *sequence_label*, *error_spec_mean*, *error_spec_std*, *alphaValue*, *error_persist*, *error_persist_mean*, *error_persist_std*, *error_AR1_mean*, *error_AR1_std*, *have_kalman*, *error_Kalman_mean*, *error_Kalman_std*, *p_p*)

Plots Fig. 1.10, Fig. 1.11 after getting all the errors data. In Fig. 1.10, we compare the prediction error for 4 methods: the standard baseline last-value prediction $\hat{y}_{t+1} := y_t$, also known as persistence prediction, the spectral filtering of \cite{hazan2017online}, Kalman filter, and AR(2).

We first continue the Example \ref{HazanEx} form the main body of the paper, with a system given by (\ref{eq:experem1_system_hazan}) and $v = w = 0.5$. Figure \ref{fig1}(right) shows a sample observations trajectory of the system, together with forecast for the four methods. Figure \ref{fig1}(left) show the mean and standard deviations of the errors for the first 500 time steps. Figure \ref{fig1brief} in the main text is the restriction of this Figure \ref{fig1}(left) to the first 20 steps. Similarly to Figure \ref{fig1brief}, we observe that the AR(2) predictions are better than the spectral and persistence methods, and worse than the Kalman filter, since only two first terms are considered.

`LDS.OnlineLDS_library.`**`plot_p1`**(*ymin*, *ymax*, *inputs*, *sequence_label*, *have_spectral_persistent*, *predicted_spectral*, *predicted_ar*, *sys*, *p_p*)

Plots seq0, seq1, seq2, logratio pdf files.

Parameters

- **ymin** – Minimal value of y-axis.
- **ymax** – Maximal value of y-axis.
- **inputs** – Input to the system matrix.
- **sequence_label** – Plot's label.
- **have_spectral_persistent** – True if we want to build spectral and persistent filters.
- **predicted_spectral** – Predicted values of spectral filter. If have_spectral_persistent is False, it's equal to an empty list.

- **predicted_ar** – Predicted values of auto-regression.
- **sys** – Linear Dynamical System created with DynamicalSystem class.
- **p_p** – PDF file, to which are export the plots.

LDS.OnlineLDS_library.**plot_p2**(*have_spectral_persistent*, *error_spec*, *error_persist*, *error_ar*, *lab*, *p_p*)

Plots seq0, seq1, seq2, logratio pdf files.

Parameters

- **have_spectral_persistent** – True if we want to build spectral and persistent filters.
- **error_spec** – Spectral filter error.
- **error_persist** – Persistent filter error.
- **error_ar** – Auto-regression error.
- **lab** – Auto-regression plot label.
- **p_p** – PDF file, to which are export the plots.

LDS.OnlineLDS_library.**plot_p3**(*ymin*, *ymax*, *have_spectral_persistent*, *error_spec_mean*, *error_spec_std*, *error_persist_mean*, *error_persist_std*, *error_ar_mean*, *error_ar_std*, *t_t*, *p_p*)

Plots seq0, seq1, seq2, logratio pdf files.

Parameters

- **ymin** – Minimal value of y-axis.
- **ymax** – Maximal value of y-axis.
- **have_spectral_persistent** – True if we want to build spectral and persistent filters.
- **error_spec_mean** – Mean error of spectral filtering.
- **error_spec_std** – Std of spectral filtering error.
- **error_persist_mean** – Mean error of last-value prediction.
- **error_persist_std** – Std of last-value prediction error.
- **error_ar_mean** – Mean error of auto-regression.
- **error_ar_std** – Std of auto-regression error.
- **p_p** – PDF file, to which are export the plots.

LDS.OnlineLDS_library.**pre_comp_filter_params**(*G*, *f_dash*, *proc_noise_std*, *obs_noise_std*, *t_t*)

Kalman filter auxiliary recursive parameters calculation.

LDS.OnlineLDS_library.**prediction**(*t_t*, *f_dash*, *G*, *matrix_a*, *sys*, *s*, *Z*, *Y*)

Auto-regression prediction values. Finds the formula for Figure 1(AR(s+1)): The unrolling of the forecast f_{t+1}. The remainder term goes to zero exponentially fast with s, by Lemma

LDS.OnlineLDS_library.**prediction_kalman**(*t_t*, *f_dash*, *G*, *matrix_a*, *sys*, *Z*, *Y*)

Kalman filter prediction values

LDS.OnlineLDS_library.**testImpactOfS**(*t_t=200*, *no_runs=100*, *sMax=15*)

Creates file './outputs/impacts.pdf', which stores plots of average error of auto-regression as a function of regression depth s. In the main paper we present it again with Example 7 and Fig. 1.7 . Increasing s decreases the error, until the error approaches that of the Kalman filter. For a given value of the observation noise, the convergence w.r.t s is slower for smaller process noise. Originally the function comes from experiments.py file.

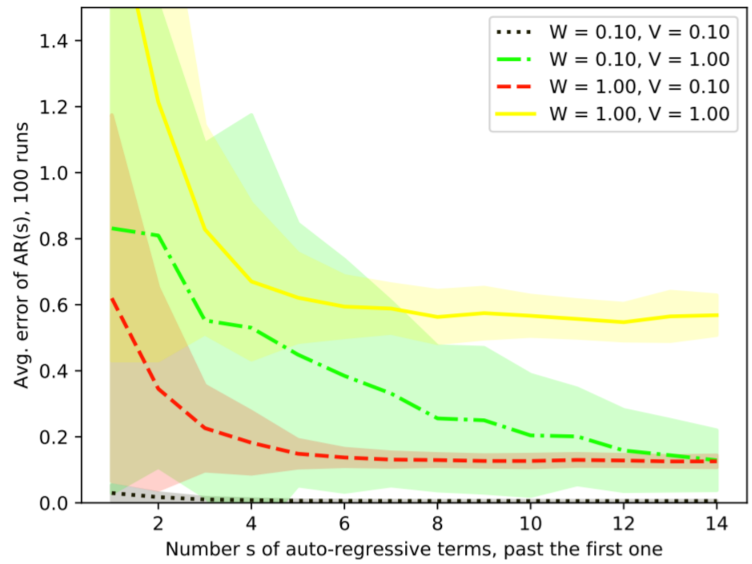


Fig. 1.7: The error of AR(s + 1) as a function of s + 1, in terms of the mean and standard deviation over N = 100 runs on Example 7, for 4 choices of w, v of process and observation noise, respectively.

Parameters

- **t_t** – Time horizon.
- **no_runs** – Number of runs.
- **sMax** – Number of auto-regressive terms.

Raises **Exits if sMax > t_t.** –

`LDS.OnlineLDS_library.`**testNoiseImpact**(*t_t=50*, *no_runs=10*, *discretisation=10*)

Produces './outputs/noise.pdf'. Plots heatmap of process noise variance vs observation noise variance based on relative error between any two predictive algorithms. LaTeX shows the example of the ratio of the errors of Kalman filter and AR(2)(see Fig. 1.5). Originally the function comes from experiments.py file.

Plots RMSE of AR Fig. 1.6 (left): average RMSE of predictions of AR(s+ 1) as a function of the variance of the process noise (vertical axis) and observation noise (horizontal axis).

Plots Fig. 1.6 (center): The differences in average RMSE of Kalman filters and AR(s + 1) as a function of the variance of the process noise (vertical axis) and observation noise (horizontal axis).

Plots Fig. 1.6 (right): The ratio (70) of the errors of Kalman filters and AR(s + 1) as a function of the variance of the process noise (vertical axis) and observation noise (horizontal axis).

Parameters

- **t_t** – Time horizon.
- **no_runs** – Number of runs.
- **discretisation** – Number of trajectories.

`LDS.OnlineLDS_library.`**testSeqD0**(*no_runs=100*)

Makes several initiations of test_identification function so as to plot "logratio.pdf" and "seq0.pdf", "seq1.pdf", "seq2.pdf". Originally the function comes from experiments.py file.

Parameters **no_runs** – Number of runs.

LDS.OnlineLDS_library.**test_AR**()

Function implements Algorithm 1(On-line Gradient Descent). Originally the function comes from experiments.py file.

LDS.OnlineLDS_library.**test_arima_ogd**(*i*, *mk*, *lrate*, *data*)

Used to test arima_ogd function for i=10 case. The test cases are based on MATLAB: The test numbers were taken from the output of MATLAB function, the random array w is fixed.

Parameters

- **i** – Iterative number. In range from mk till data - 1.
- **mk** – Integer number. Here we used 10.
- **lrate** – Learning rate. Assigned 1 in example.py.
- **data** – Array of 10000 elements.

Raises:

LDS.OnlineLDS_library.**test_arima_ons**(*i*, *mk*, *lrate*, *data*, *A_trans_in*)

to test arima_ons function the test casees are based on MATLAB: the test numbers were taken from the output of MATLAB function the random array w is fixed

Parameters

- **i** –
- **mk** –
- **lrate** –
- **data** –

Returns

LDS.OnlineLDS_library.**test_identification**(*sys*, *filename_stub='test'*, *no_runs=2*, *t_t=100*, *k=5*, *eta_zeros=None*, *ymin=None*, *ymax=None*, *sequence_label=None*, *have_spectral_persistent=True*)

Implements here On-line Gradient Descent Algorithm 1 by the use of cost_ar and gradient_ar functions. Data found is used by plot_p1,plot_p2, plot_p3 functions which create “seq0”, “logration” pdfs. Implements Example 8 from Experiments section of Marecek’s paper. Originally the function comes from experiments.py file. Plots Fig. 1.8, Fig. 1.9.

Parameters

- **sys** – LDS.
- **filename_stub** – Name of the output file.
- **no_runs** – Number of runs.
- **t_t** – Time horizon.
- **k** – Number of filters.
- **eta_zeros** – Learning rate.
- **y_min** – Minimal value of y-axis.
- **y_max** – Maximal value of y-axis.
- **sequence_label** –
- **have_spectral_persistent** – False if there’s no need to plot spectral and persistent filters. Default value - True.

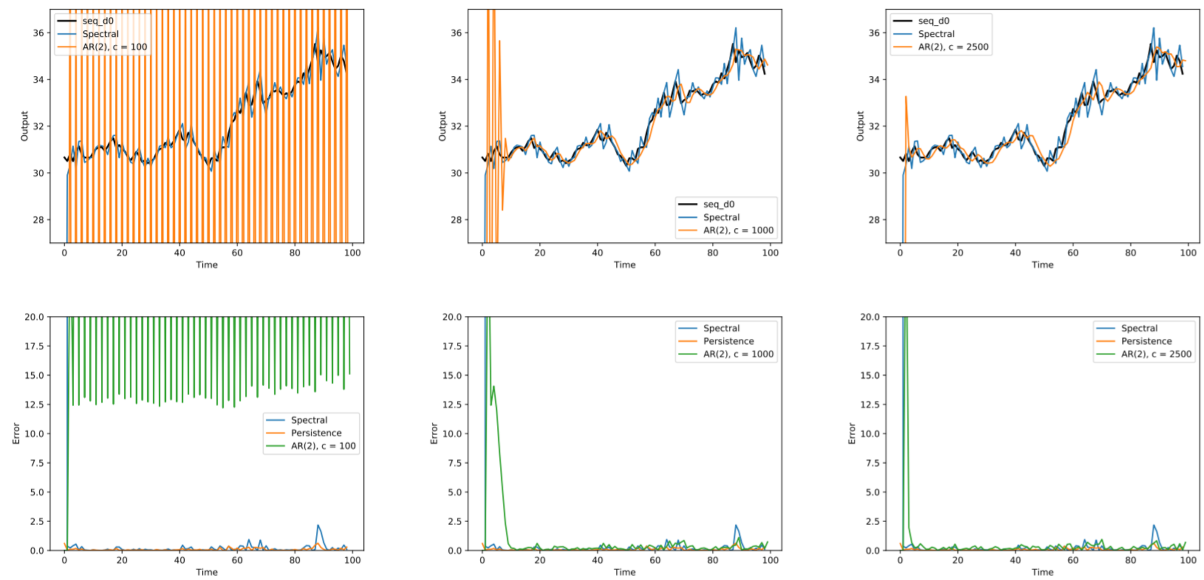


Fig. 1.8: An illustration of the impact of constants in the learning rate on Example 8 from Marecek's paper, the well-known time series. Top: The forecasts for three different values of c. Bottom: The error for three different values of c.

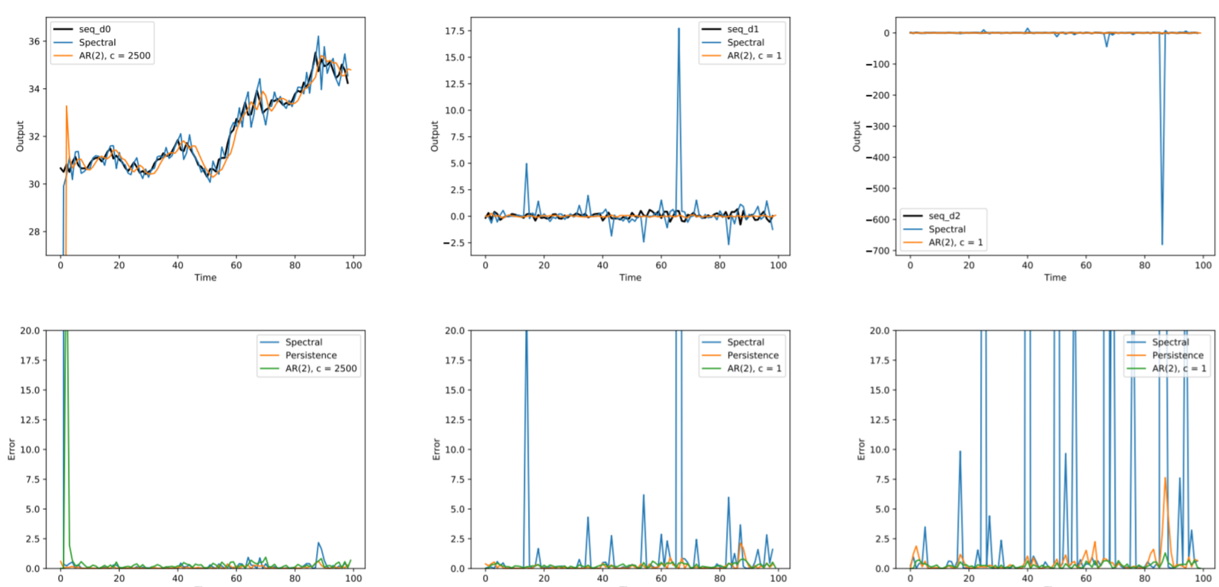


Fig. 1.9: Illustrations on Example 8, the well-known time series. Top: the predictions of AR(2) compared with the predictions of the spectral filter of Hazan, Singh, and Zhang (2017) and the trivial last-value prediction on the first T = 100 elements of series d0 (left), d1 (center), and d2 (right). Bottom: the corresponding errors.

Raises **Exits if k > t_t.** –

`LDS.OnlineLDS_library.`**`test_identification2`**(*t_t=100, no_runs=10, s_choices=[15, 3, 1], have_kalman=False, have_spectral_persistent=True, G=array([[0.47818304, 0.63191781], [0.71975662, 0.51588563]]), f_dash=array([[0.77427218, 0.8161933]]), sequence_label=''*)

Implements Example 7 from Experiments section of the paper. Creates './outputs/AR.pdf'.Finds all the filters' errors and uses function p3_for_test_identification2 for plotting them. Plots Fig. 1.10, Fig. 1.11. Plots Figure 2,5 of the main paper. Originally the function comes from experiments.py file.

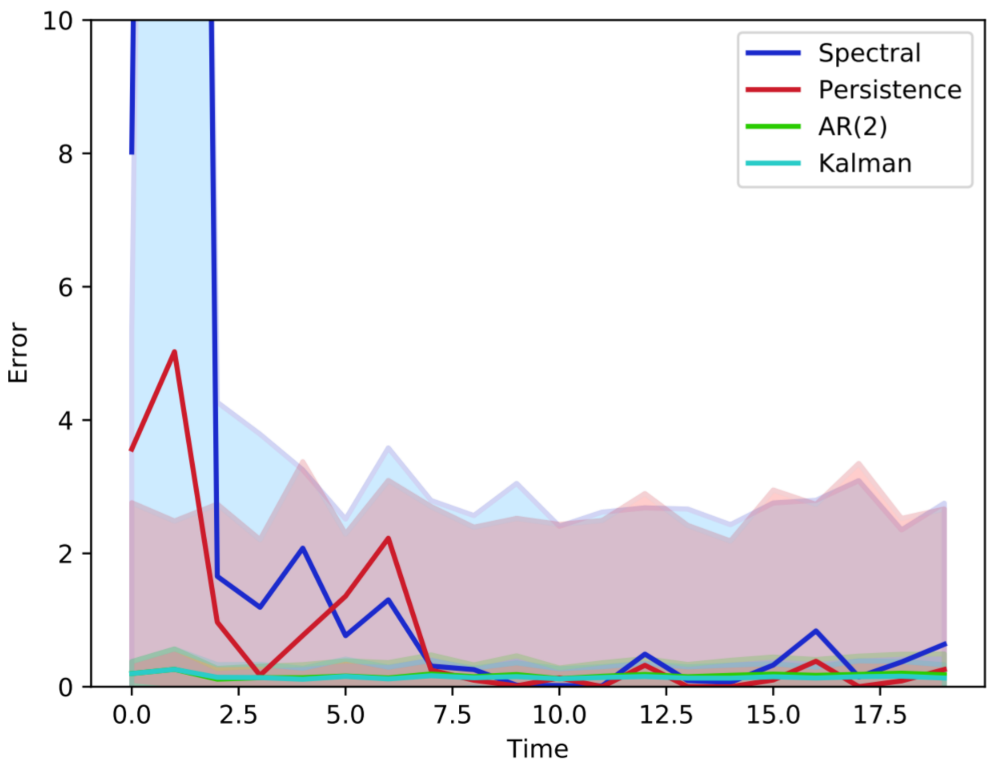


Fig. 1.10: The error of AR(2) compared against Kalman filter, last-value prediction, and spectral filtering in terms of the mean and standard deviation over N = 100 runs on Example 7.

Parameters

- **t_t** – Time horizon.
- **no_runs** – Number of runs.
- **s_choices** –
- **have_kalman** – False if there's no need to plot kalman filter. Default value - True.
- **have_spectral_persistent** – False if there's no need to plot spectral and persistent filters. Default value - True.
- **G** – State matrix.

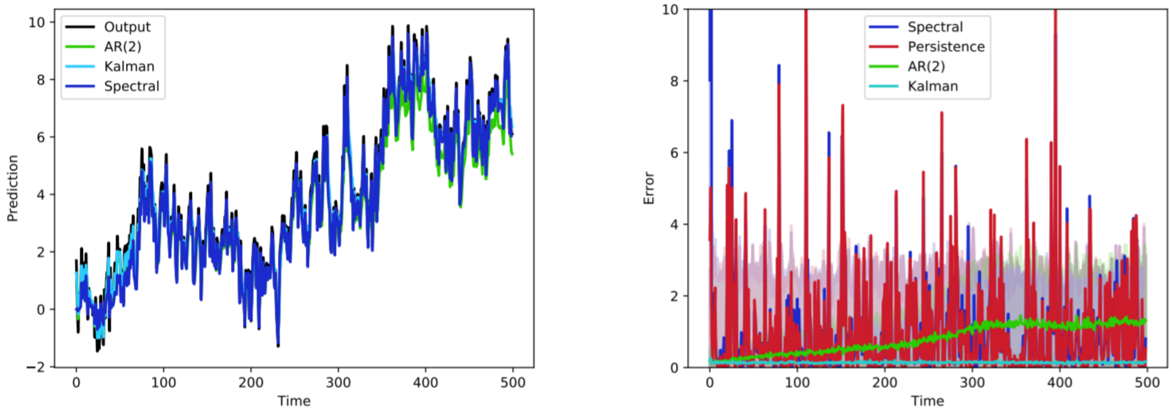


Fig. 1.11: Left: sample outputs and predictions with AR(2), compared against Kalman filter, last-value prediction, and spectral filtering of Hazan, Singh, and Zhang (2017). Right: Same as Fig. 1.10, over longer time period.

- **f_dash** – first derivative of the observation direction.
- **sequence_label** –

Raises Exits if number of runs is less than 2. –

`LDS.OnlineLDS_library.`**`w_calc`**(*w*, *data*, *mk*, *i*, *diff*, *lrate*)

Auxiliary function to implement ARIMA in python. Others functions use it in their iterations. MATLAB: w = w - data(i-mk:i-1)*2*diff/sqrt(i-mk)*lrate;

Parameters

- **w** – Uniform distribution array with options.mk number of columns.
- **data** – Array of 10000 elements.
- **mk** – Integer number. Here we used 10.
- **i** – Iterative number. In range from mk till data - 1.
- **diff** – Result of diff_calc function
- **lrate** – Learning rate. Assigned 1 in example.py.

Returns:

`LDS.OnlineLDS_library.`**`w_calc_arima_ons`**(*w*, *lrate*, *grad*, *A_trans*)

MATLAB: w = w - lrate * grad * A_trans Calculation of the weight with Gradient Descent algorithm.

Parameters

- **w** – Uniform distribution array with options.mk number of columns.
- **lrate** – Learning rate. Assigned 1 in example.py.
- **grad** – Gradient, the return of the function grad_calc.
- **A_trans** – Return of the function A_trans_calc.

Returns Weight after an iteration of the gradient descent algorithm.

1.1.4 LDS.example module

1.1.5 LDS.main module

1.1.6 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

INDEX

V

W