tigercontrol Documentation

Release 0.0.1

alexjyu

Sep 25, 2019

Contents

For an introduction to TigerControl, start at the TigerControl GitHub page.

CHAPTER 1

tigercontrol package

1.1 Subpackages

1.1.1 tigercontrol.utils package

dataset_registry

unemployment([verbose])	Description: Checks if unemployment data exists, downloads if not.
uci_indoor([verbose])	Description: Checks if uci_indoor data exists, downloads if not.
sp500([verbose])	Description: Checks if S&P500 data exists, downloads if not.
crypto()	Description: Checks if cryptocurrency data exists, downloads if not.
enso(input_signals, include_month, ...)	Description: Transforms the ctrl_indices dataset into a format suitable for online learning.

tigercontrol.utils.unemployment

tigercontrol.utils.**unemployment**(*verbose=True*)

Description: Checks if unemployment data exists, downloads if not. Dataset credits: https://fred.stlouisfed.org/series/UNRATE, Federal Reserve Bank of St. Louis.

Parameters **verbose** (*boolean*) – Specifies if download progress should be printed

Returns Dataframe containing Unemployment data

tigercontrol.utils.uci_indoor

tigercontrol.utils.**uci_indoor**(*verbose=True*)

Description: Checks if uci_indoor data exists, downloads if not. Dataset credits: F. Zamora-Martínez, P. Romeu, P. Botella-Rocamora, J. Pardo, On-line learning of indoor temperature forecasting models towards energy efficiency, Energy and Buildings, Volume 83, November 2014, Pages 162-172, ISSN 0378-7788

Parameters **verbose** (*boolean*) – Specifies if download progress should be printed

Returns Dataframe containing uci_indoor data

tigercontrol.utils.sp500

tigercontrol.utils.**sp500**(*verbose=True*)

Description: Checks if S&P500 data exists, downloads if not.

Parameters **verbose** (*boolean*) – Specifies if download progress should be printed

Returns Dataframe containing S&P500 data

tigercontrol.utils.crypto

tigercontrol.utils.**crypto**()

Description: Checks if cryptocurrency data exists, downloads if not.

Parameters **None** –

Returns Dataframe containing cryptocurrency data

tigercontrol.utils.enso

tigercontrol.utils.**enso**(*input_signals*, *include_month*, *output_signals*, *history*, *timeline*)

Description: Transforms the ctrl_indices dataset into a format suitable for online learning.

Parameters

- **input_signals** (*list of strings*) – signals used for prediction
- **include_month** (*boolean*) – True if the month should be used as a feature, False otherwise
- **output_signals** (*list of strings*) – signals we are trying to predict
- **history** (*int*) – number of past observations used for prediction
- **timeline** (*int/list of ints*) – the forecasting timeline(s)

Returns Input Observations y (numpy.ndarray): Labels

Return type X (numpy.ndarray)

random

set_key([key])	Descripton: Fix global random key to ensure reproducibility of results.
generate_key()	Descripton: Generate random key.
get_global_key()	Descripton: Get current global random key.

tigercontrol.utils.set_key

tigercontrol.utils.**set_key**(*key=None*)
Descripton: Fix global random key to ensure reproducibility of results.

Parameters **key** (*int*) – key that determines reproducible output

tigercontrol.utils.generate_key

tigercontrol.utils.**generate_key**()
Descripton: Generate random key.

Returns Random random key

tigercontrol.utils.get_global_key

tigercontrol.utils.**get_global_key**()
Descripton: Get current global random key.

Returns Current global random key

1.1.2 tigercontrol.problems package

core

This is a core

Problem()	

tigercontrol.problems.Problem

class tigercontrol.problems.**Problem**

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory

Continued on next page

Table 4 – continued from previous page

help()	Description: prints information about this class and its methods
initialize(**kwargs)	Description: resets problem to time 0
step([action])	Description: run one timestep of the problem's dynamics.

Attributes

spec

custom

tigercontrol.problems.CustomProblem()	Description: class for implementing algorithms with enforced modularity
tigercontrol.problems.register_custom_problem(...)	Description: global custom problem method

tigercontrol.problems.CustomProblem

class tigercontrol.problems.**CustomProblem**

Description: class for implementing algorithms with enforced modularity

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.

tigercontrol.problems.register_custom_problem

tigercontrol.problems.**register_custom_problem**(*custom_problem_class*, *custom_problem_id*)

Description: global custom problem method

control

tigercontrol.problems.ControlProblem()	Description: class for online control tests
tigercontrol.problems.control.LDS_Control()	Description: Simulates a linear dynamical system.
tigercontrol.problems.control.LSTM_Control()	Description: Produces outputs from a randomly initialized recurrent neural network.
tigercontrol.problems.control.RNN_Control()	Description: Produces outputs from a randomly initialized recurrent neural network.

Continued on next page

Table 8 – continued from previous page

tigercontrol.problems.control.CartPole()	Description:
tigercontrol.problems.control.DoublePendulum()	Acrobot is a 2-link pendulum with only the second joint actuated.
tigercontrol.problems.control.Pendulum([g])	

tigercontrol.problems.ControlProblem

class tigercontrol.problems.**ControlProblem**

Description: class for online control tests

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: prints information about this class and its methods
initialize(**kwargs)	Description: resets problem to time 0
step([action])	Description: run one timestep of the problem's dynamics.

Attributes

spec

tigercontrol.problems.control.LDS_Control

class tigercontrol.problems.control.**LDS_Control**

Description: Simulates a linear dynamical system.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the hidden state of the system.

Continued on next page

Table 11 – continued from previous page

initialize(n, m, d[, noise])	Description: Randomly initialize the hidden dynamics of the system.
step(u)	Description: Moves the system dynamics one time-step forward.

Attributes

spec

tigercontrol.problems.control.LSTM_Control

class tigercontrol.problems.control.**LSTM_Control**

Description: Produces outputs from a randomly initialized recurrent neural network.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the hidden state of the RNN when computed on the last l inputs.
initialize(n, m[, h])	Description: Randomly initialize the RNN.
step(x)	Description: Takes an input and produces the next output of the RNN.

Attributes

spec

tigercontrol.problems.control.RNN_Control

class tigercontrol.problems.control.**RNN_Control**

Description: Produces outputs from a randomly initialized recurrent neural network.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.

Continued on next page

Table 15 – continued from previous page

close()	Description: closes the problem and returns used memory
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the hidden state of the RNN when computed on the last l inputs.
initialize(n, m[, h])	Description: Randomly initialize the RNN.
step(x)	Description: Takes an input and produces the next output of the RNN.

Attributes

spec

tigercontrol.problems.control.CartPole

class tigercontrol.problems.control.**CartPole**

Description: A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity.

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: prints information about this class and its methods
initialize()	Description: resets problem to time 0
render([mode])	
reset()	
step(action)	Description: run one timestep of the problem's dynamics.

Attributes

metadata
spec

tigercontrol.problems.control.DoublePendulum

class tigercontrol.problems.control.**DoublePendulum**
Acrobot is a 2-link pendulum with only the second joint actuated. Initially, both links point downwards. The goal is to swing the end-effector at a height at least the length of one link above the base. Both links can swing freely and can pass by each other, i.e., they don't collide when they have the same angle. **STATE:** The state

consists of the sin() and cos() of the two rotational joint angles and the joint angular velocities : [cos(theta1) sin(theta1) cos(theta2) sin(theta2) thetaDot1 thetaDot2]. For the first link, an angle of 0 corresponds to the link pointing downwards. The angle of the second link is relative to the angle of the first link. An angle of 0 corresponds to having the same angle between the two links. A state of [1, 0, 1, 0, ..., ...] means that both links point downwards. **ACTIONS:** The action is either applying +1, 0 or -1 torque on the joint between the two pendulum links.

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: prints information about this class and its methods
initialize()	Description: resets problem to time 0
render([mode])	
reset()	
step(a)	Description: run one timestep of the problem's dynamics.

Attributes

LINK_COM_POS_1	[m] position of the center of mass of link 1
LINK_COM_POS_2	[m] position of the center of mass of link 2
LINK_LENGTH_1	
LINK_LENGTH_2	
LINK_MASS_1	[kg] mass of link 1
LINK_MASS_2	[kg] mass of link 2
LINK_MOI	moments of inertia for both links
MAX_VEL_1	
MAX_VEL_2	
action_arrow	
actions_num	
domain_fig	
dt	
metadata	
spec	
torque_noise_max	

tigercontrol.problems.control.Pendulum

class tigercontrol.problems.control.**Pendulum**(*g=10.0*)

__init__(*g=10.0*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([g])	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: prints information about this class and its methods
initialize()	Description: resets problem to time 0
render([mode])	
reset()	
step(u)	Description: run one timestep of the problem's dynamics.

Attributes

metadata
spec

time_series

tigercontrol.problems.TimeSeriesProblem()	Description: class for online control tests
tigercontrol.problems.time_series.SP500()	Description: Outputs the daily opening price of the S&P 500 stock market index from January 3, 1986 to June 29, 2018.
tigercontrol.problems.time_series.UCI_Indoor()	Description: Outputs various weather metrics from a UCI dataset from 13/3/2012 to 11/4/2012
tigercontrol.problems.time_series.ENSO()	Description: Collection of monthly values of control indices useful for predicting La Nina/El Nino.
tigercontrol.problems.time_series.Crypto()	Description: Outputs the daily price of bitcoin from 2013-04-28 to 2018-02-10
tigercontrol.problems.time_series.Random()	Description: A random sequence of scalar values taken from an i.i.d.
tigercontrol.problems.time_series.ARMA()	Description: Simulates an autoregressive moving-average time-series.
tigercontrol.problems.time_series.Unemployment()	Description: Monthly unemployment rate since 1948.
tigercontrol.problems.time_series.LDS_TimeSeries()	Description: Simulates a linear dynamical system.
tigercontrol.problems.time_series.LSTM_TimeSeries()	Description: Produces outputs from a randomly initialized recurrent neural network.
tigercontrol.problems.time_series.RNN_TimeSeries()	Description: Produces outputs from a randomly initialized recurrent neural network.

tigercontrol.problems.TimeSeriesProblem

class tigercontrol.problems.**TimeSeriesProblem**

Description: class for online control tests

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: prints information about this class and its methods
initialize(**kwargs)	Description: resets problem to time 0
step([action])	Description: Run one timestep of the problem's dynamics.

Attributes

spec

tigercontrol.problems.time_series.SP500

class tigercontrol.problems.time_series.**SP500**

Description: Outputs the daily opening price of the S&P 500 stock market index from January 3, 1986 to June 29, 2018.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Not implemented
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the date corresponding to the last value of the S&P 500 that was returned :param None:
initialize()	Description: Check if data exists, else download, clean, and setup.
step()	Description: Moves time forward by one day and returns value of the stock index :param None:

Attributes

compatibles
spec

tigercontrol.problems.time_series.UCI_Indoor

class tigercontrol.problems.time_series.**UCI_Indoor**

Description: Outputs various weather metrics from a UCI dataset from 13/3/2012 to 11/4/2012

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Not implemented
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the date corresponding to the last value of the uci_indoor that was returned :param None:
initialize([pred_indices])	Description: Check if data exists, else download, clean, and setup.
step()	Description: Moves time forward by fifteen minutes and returns weather metrics :param None:

Attributes

compatibles
spec

tigercontrol.problems.time_series.ENSO

class tigercontrol.problems.time_series.**ENSO**

Description: Collection of monthly values of control indices useful for predicting La Nina/El Nino. More specifically, the user can choose any of pna, ea, wa, wp, eu, soi, esoi, nino12, nino34, nino4, oni of nino34 (useful for La Nino/El Nino identification) to be used as input and/or output in the problem instance.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Not implemented
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the timestep corresponding to the last (observation, label) pair returned.
initialize([input_signals, include_month, ...])	Description: Initializes the ctrl_indices dataset to a format suited to the online learning setting.
step()	Description: Moves time forward by one month and returns the corresponding observation and label.

Attributes

compatibles
spec

tigercontrol.problems.time_series.Crypto

class tigercontrol.problems.time_series.**Crypto**

Description: Outputs the daily price of bitcoin from 2013-04-28 to 2018-02-10

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Not implemented
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the date corresponding to the last price of bitcoin that was returned :param None:
initialize()	Description: Check if data exists, else download, clean, and setup.
step()	Description: Moves time forward by one day and returns price of the bitcoin :param None:

Attributes

compatibles
spec

tigercontrol.problems.time_series.Random

class tigercontrol.problems.time_series.**Random**

Description: A random sequence of scalar values taken from an i.i.d. normal distribution.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Not implemented
help()	Description: Prints information about this class and its methods.
hidden()	Not implemented
initialize()	Description: Randomly initialize the hidden dynamics of the system.

Continued on next page

Table 34 – continued from previous page

step()	Description: Moves the system dynamics one time-step forward.

Attributes

compatibles
spec

tigercontrol.problems.time_series.ARMA

class tigercontrol.problems.time_series.**ARMA**

Description: Simulates an autoregressive moving-average time-series.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the hidden state of the system.
initialize([p, q, noise_list, c, ...])	Description: Randomly initialize the hidden dynamics of the system.
step()	Description: Moves the system dynamics one time-step forward.

Attributes

compatibles
spec

tigercontrol.problems.time_series.Unemployment

class tigercontrol.problems.time_series.**Unemployment**

Description: Monthly unemployment rate since 1948.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Not implemented

Continued on next page

Table 38 – continued from previous page

help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the date corresponding to the last unemployment rate value :param None:
initialize()	Description: Check if data exists, else download, clean, and setup.
step()	Description: Moves time forward by one day and returns price of the bitcoin :param None:

Attributes

compatibles
spec

tigercontrol.problems.time_series.LDS_TimeSeries

class tigercontrol.problems.time_series.**LDS_TimeSeries**

Description: Simulates a linear dynamical system.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the hidden state of the system.
initialize(n, m, d[, noise])	Description: Randomly initialize the hidden dynamics of the system.
step()	Description: Moves the system dynamics one timestep forward.

Attributes

compatibles
spec

tigercontrol.problems.time_series.LSTM_TimeSeries

class tigercontrol.problems.time_series.**LSTM_TimeSeries**

Description: Produces outputs from a randomly initialized recurrent neural network.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the hidden state of the RNN when computed on the last l inputs.
initialize(n, m[, h])	Description: Randomly initialize the RNN.
step()	Description: Takes an input and produces the next output of the RNN.

Attributes

compatibles
spec

tigercontrol.problems.time_series.RNN_TimeSeries

class tigercontrol.problems.time_series.**RNN_TimeSeries**

Description: Produces outputs from a randomly initialized recurrent neural network.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
close()	Description: closes the problem and returns used memory
help()	Description: Prints information about this class and its methods.
hidden()	Description: Return the hidden state of the RNN when computed on the last l inputs.
initialize(n, m[, h])	Description: Randomly initialize the RNN.
step()	Description: Takes an input and produces the next output of the RNN.

Attributes

compatibles
spec

pybullet

tigercontrol.problems.pybullet. PyBulletProblem
tigercontrol.problems.pybullet. Simulator
tigercontrol.problems.pybullet.Ant
tigercontrol.problems.pybullet. CartPole
tigercontrol.problems.pybullet. CartPoleDouble
tigercontrol.problems.pybullet. CartPoleSwingup
tigercontrol.problems.pybullet. HalfCheetah
tigercontrol.problems.pybullet. Humanoid
tigercontrol.problems.pybullet.Kuka
tigercontrol.problems.pybullet. KukaDiverse
tigercontrol.problems.pybullet. Minitaur
tigercontrol.problems.pybullet. Obstacles

tigercontrol.problems.pybullet.PyBulletProblem

tigercontrol.problems.pybullet.Simulator

tigercontrol.problems.pybullet.Ant

tigercontrol.problems.pybullet.CartPole

tigercontrol.problems.pybullet.CartPoleDouble

tigercontrol.problems.pybullet.CartPoleSwingup

tigercontrol.problems.pybullet.HalfCheetah

tigercontrol.problems.pybullet.Humanoid

tigercontrol.problems.pybullet.Kuka

tigercontrol.problems.pybullet.KukaDiverse

tigercontrol.problems.pybullet.Minitaur

tigercontrol.problems.pybullet.Obstacles

1.1.3 tigercontrol.models package

core

Model

tigercontrol.models.Model

class tigercontrol.models.**Model**

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

help()
initialize(**kwargs)
predict([x])
update(**kwargs)

Attributes

spec

control

tigercontrol.models.control.ControlModel()	Description: class for implementing algorithms with enforced modularity
tigercontrol.models.control.KalmanFilter()	Description: Kalman Filter adjusts measurements of a signal based on prior states and knowledge of intrinsic equations of the system.
tigercontrol.models.control.ODEShootingMethod()	Description: Implements the shooting method to solve second order boundary value problems with conditions y(0) = a and y(L) = b.
tigercontrol.models.control.LQR()	Description: Computes optimal set of actions using the Linear Quadratic Regulator algorithm.
tigercontrol.models.control.MPPI()	Description: Implements Model Predictive Path Integral Control to compute optimal control sequence.
tigercontrol.models.control.CartPoleNN()	Description: Simple multi-layer perceptron policy, no internal state
tigercontrol.models.control.ILQR()	Description: Computes optimal set of actions using the Linear Quadratic Regulator algorithm.

tigercontrol.models.control.ControlModel

class tigercontrol.models.control.**ControlModel**
Description: class for implementing algorithms with enforced modularity

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
help()	
initialize(**kwargs)	
predict([x])	
update(**kwargs)	

Attributes

spec

tigercontrol.models.control.KalmanFilter

class tigercontrol.models.control.**KalmanFilter**

Description: Kalman Filter adjusts measurements of a signal based on prior states and knowledge of intrinsic equations of the system.

More precisely, we know that the signal at time t is a linear combination of its previous value plus a control signal u(t) and a process noise w(t - 1), i.e. x(t) = A x(t - 1) + B u(t) + w(t), and that the measurement at time t is a linear combination of the signal value and the measurement noise v(t), i.e. z(t) = H x(t) + v(t).

Based on these, the model can advance by itself in time using a 'time' update and/or incorporate and correct a measurement using a 'measurement' update:

a. Time Update (prediction) - Project state ahead: x(t) = A x(t - 1) + B u(t) - Project error covariance ahead: P(t) = A P(t - 1) A^T + Q

b. Measurement Update - Compute Kalman Gain: K(t) = P(t) H^T (H P(t) H^T + R)^{-1} - Update estimate based on measurement: x(t) = x(t) + K(t) (z(t) - H x(t)) - Update error covariance: P(t) = (I - K(t) H) P(t)

where we assume w(t) ~ N(0, Q) and v(t) ~ N(0, R).

The user must provide estimates for A, B, H, Q and R, as well as initial estimates for x(0) and P(0).

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
help()	Description:
initialize(x, A, B, H, P, Q, R)	Description:
predict(u, z[, n])	Description:
step(u, z[, n])	Description:
to_ndarray(x)	Description: If x is a scalar, transform it to a (1, 1) numpy.ndarray; otherwise, leave it unchanged.
update(**kwargs)	

Attributes

compatibles
spec

tigercontrol.models.control.ODEShootingMethod

class tigercontrol.models.control.**ODEShootingMethod**

Description: Implements the shooting method to solve second order boundary value problems with conditions y(0) = a and y(L) = b. Assumes that the second order BVP has been converted to a first order system of two equations.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
euler(f, a, z, t[, dt])	Description: Solve corresponding initial value problem.
help()	Description: Prints information about this class and its methods.
initialize(f, a, b, z1, z2, t)	Description: Initialize the dynamics of the model.
predict()	Description: Returns current solution estimation.
step([n])	Description: Updates internal parameters for n iterations and then returns current solution estimation.
update()	Description:

Attributes

compatibles
spec

tigercontrol.models.control.LQR

class tigercontrol.models.control.**LQR**

Description: Computes optimal set of actions using the Linear Quadratic Regulator algorithm.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
extend(x, T)	Description: If x is not in the correct form, convert it; otherwise, leave it unchanged.
help()	Description: Prints information about this class and its methods.

Continued on next page

Table 57 – continued from previous page

initialize(F, f, C, c, T, x)	Description: Initialize the dynamics of the model :param F: past value contribution coefficients :type F: float/numpy.ndarray :param f: bias coefficients :type f: float/numpy.ndarray :param C: quadratic cost coefficients :type C: float/numpy.ndarray :param c: linear cost coefficients :type c: float/numpy.ndarray :param T: number of timesteps :type T: postive int :param x: initial state :type x: float/numpy.ndarray
plan()	Description: Updates internal parameters and then returns the estimated optimal set of actions :param None:
predict([x])	
to_ndarray(x)	Description: If x is a scalar, transform it to a (1, 1) numpy.ndarray; otherwise, leave it unchanged.
update(**kwargs)	

Attributes

compatibles
spec

tigercontrol.models.control.MPPI

class tigercontrol.models.control.**MPPI**

Description: Implements Model Predictive Path Integral Control to compute optimal control sequence.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
compute_total_cost(k)	
help()	Description: Prints information about this class and its methods.
initialize(env, K, T, U[, lambda_, ...])	Description: Initialize the dynamics of the model.
plan([n])	Description: Updates internal parameters and then returns the estimated optimal set of actions :param n: Number of updates :type n: non-negative int
predict([x])	
update(**kwargs)	

Attributes

compatibles
spec

tigercontrol.models.control.CartPoleNN

class tigercontrol.models.control.**CartPoleNN**

Description: Simple multi-layer perceptron policy, no internal state

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
help()	
initialize(observation_space, action_space)	Description: initialize the NN :param observation_space: :param action_space:
predict(ob)	
update(**kwargs)	

Attributes

compatibles
spec

tigercontrol.models.control.ILQR

class tigercontrol.models.control.**ILQR**

Description: Computes optimal set of actions using the Linear Quadratic Regulator algorithm.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
help()	Description: Prints information about this class and its methods.
initialize(problem_dynamics, L, dim_x, dim_u)	Description: Initialize the dynamics of the model :param problem: problem instance *OR* dynamics of problem :type problem: instance/function :param L: loss function :type L: function :param dim_x: state_space dimension :type dim_x: int :param dim_u: action_space dimension :type dim_u: int
plan(x_0, T[, max_iterations, lamb, threshold])	
predict([x])	
update(**kwargs)	

Attributes

compatibles
spec

time_series

tigercontrol.models.time_series.TimeSeriesModel()	Description: class for implementing algorithms with enforced modularity
tigercontrol.models.time_series.AutoRegressor()	Description: Implements the equivalent of an AR(p) model - predicts a linear combination of the previous p observed values in a time-series
tigercontrol.models.time_series.LastValue()	Description: Predicts the last value in the time series, i.e.
tigercontrol.models.time_series.PredictZero()	Description: Predicts the next value in the time series to be 0, i.e.
tigercontrol.models.time_series.RNN()	Description: Produces outputs from a randomly initialized recurrent neural network.
tigercontrol.models.time_series.LSTM()	Description: Produces outputs from a randomly initialized LSTM neural network.
tigercontrol.models.time_series.LeastSquares()	Description: Implements online least squares.

tigercontrol.models.time_series.TimeSeriesModel

class tigercontrol.models.time_series.**TimeSeriesModel**

Description: class for implementing algorithms with enforced modularity

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
help()	
initialize(**kwargs)	
predict([x])	
update(**kwargs)	

Attributes

spec

tigercontrol.models.time_series.AutoRegressor

class tigercontrol.models.time_series.**AutoRegressor**

Description: Implements the equivalent of an AR(p) model - predicts a linear combination of the previous p observed values in a time-series

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
forecast(x[, timeline])	Description: Forecast values 'timeline' timesteps in the future :param x: Value at current time-step :type x: int/numpy.ndarray :param timeline: timeline for forecast :type timeline: int
help()	Description: Prints information about this class and its methods.
initialize([p, optimizer])	Description: Initializes autoregressive model parameters
predict(x)	Description: Predict next value given observation :param x: Observation :type x: int/numpy.ndarray
update(y)	Description: Updates parameters using the specified optimizer :param y: True value at current time-step :type y: int/numpy.ndarray

Attributes

compatibles
spec

tigercontrol.models.time_series.LastValue

class tigercontrol.models.time_series.**LastValue**

Description: Predicts the last value in the time series, i.e. x(t) = x(t-1)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
forecast(x[, timeline])	Description: Forecast values 'timeline' timesteps in the future :param x: Value at current time-step :type x: int/numpy.ndarray :param timeline: timeline for forecast :type timeline: int
help()	Description: Prints information about this class and its methods :param None:
initialize()	Description: Initialize the (non-existent) hidden dynamics of the model :param None:
predict(x)	Description: Takes input observation and returns next prediction value :param x: value at current time-step :type x: float/numpy.ndarray
update(y)	Description: Takes update rule and adjusts internal parameters :param y: true value :type y: float/np.ndarray

Attributes

compatibles
spec

tigercontrol.models.time_series.PredictZero

class tigercontrol.models.time_series.**PredictZero**

Description: Predicts the next value in the time series to be 0, i.e. x(t) = 0

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
forecast(x[, timeline])	Description: Forecast values 'timeline' timesteps in the future :param x: Value at current time-step :type x: int/numpy.ndarray :param timeline: timeline for forecast :type timeline: int
help()	Description: Prints information about this class and its methods :param None:
initialize()	Description: Initialize the (non-existent) hidden dynamics of the model :param None:
predict(x)	Description: Takes input observation and returns next prediction value :param x: value at current time-step :type x: float/numpy.ndarray
update([rule])	Description: Takes update rule and adjusts internal parameters :param rule: rule with which to alter parameters :type rule: function

Attributes

compatibles
spec

tigercontrol.models.time_series.RNN

class tigercontrol.models.time_series.**RNN**

Description: Produces outputs from a randomly initialized recurrent neural network.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.

Continued on next page

Table 74 – continued from previous page

forecast(x[, timeline])	Description: Forecast values 'timeline' timesteps in the future :param x: Value at current time-step :type x: float/numpy.ndarray :param timeline: timeline for forecast :type timeline: int
help()	Description: Prints information about this class and its methods.
initialize([n, m, l, h, optimizer, loss, lr])	Description: Randomly initialize the RNN.
predict(x[, timeline])	Description: Predict next value given observation :param x: Observation :type x: float/numpy.ndarray
to_ndarray(x)	Description: If x is a scalar, transform it to a (1, 1) numpy.ndarray; otherwise, leave it unchanged.
update(y)	Description: Updates parameters :param y: True value at current time-step :type y: int/numpy.ndarray

Attributes

compatibles
spec

tigercontrol.models.time_series.LSTM

class tigercontrol.models.time_series.**LSTM**

Description: Produces outputs from a randomly initialized LSTM neural network.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
forecast(x[, timeline])	Description: Forecast values 'timeline' timesteps in the future :param x: Value at current time-step :type x: int/numpy.ndarray :param timeline: timeline for forecast :type timeline: int
help()	Description: Prints information about this class and its methods.
initialize([n, m, l, h, optimizer])	Description: Randomly initialize the LSTM.
predict(x)	Description: Predict next value given observation :param x: Observation :type x: int/numpy.ndarray
to_ndarray(x)	Description: If x is a scalar, transform it to a (1, 1) numpy.ndarray; otherwise, leave it unchanged.
update(y)	Description: Updates parameters :param y: True value at current time-step :type y: int/numpy.ndarray

Attributes

compatibles
spec

tigercontrol.models.time_series.LeastSquares

class tigercontrol.models.time_series.**LeastSquares**

Description: Implements online least squares.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
help()	Description: Prints information about this class and its methods.
initialize(x, y[, reg])	Description: Initializes model parameters
predict([x])	
step(x, y)	Description: Predict next value given observation and update internal parameters based
update(**kwargs)	

Attributes

compatibles
spec

optimizers

tigercontrol.models.optimizers.Optimizer([...])	Description: Core class for model optimizers
tigercontrol.models.optimizers.Adagrad([...])	Description: Ordinary Gradient Descent optimizer.
tigercontrol.models.optimizers.Adam([pred, ...])	Description: Ordinary Gradient Descent optimizer.
tigercontrol.models.optimizers.ONS([pred, ...])	Online newton step algorithm.
tigercontrol.models.optimizers.SGD([pred, ...])	Description: Stochastic Gradient Descent optimizer.
tigercontrol.models.optimizers.OGD([pred, ...])	Description: Ordinary Gradient Descent optimizer.
tigercontrol.models.optimizers.mse(y_pred, ...)	Description: mean-square-error loss :param y_pred: value predicted by model :param y_true: ground truth value :param eps: some scalar
tigercontrol.models.optimizers.cross_entropy(...)	Description: cross entropy loss, y_pred is equivalent to logits and y_true to labels :param y_pred: value predicted by model :param y_true: ground truth value :param eps: some scalar

tigercontrol.models.optimizers.Optimizer

class tigercontrol.models.optimizers.**Optimizer**(*pred=None*, *loss=<function mse>*, *learning_rate=1.0*, *hyperparameters={}*)

Description: Core class for model optimizers

Parameters

- **pred** (*function*) – a prediction function implemented with jax.numpy
- **loss** (*function*) – specifies loss function to be used; defaults to MSE
- **learning_rate** (*float*) – learning rate. Default value 0.01
- **hyperparameters** (*dict*) – additional optimizer hyperparameters

Returns None

__init__(*pred=None*, *loss=<function mse>*, *learning_rate=1.0*, *hyperparameters={}*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([pred, loss, learning_rate, ...])	Initialize self.
gradient(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.
set_loss(new_loss)	Description: updates internal loss
set_predict(pred[, loss])	Description: Updates internally stored pred and loss functions :param pred: predict function, must take params and x as input :type pred: function :param loss: loss function.

tigercontrol.models.optimizers.Adagrad

class tigercontrol.models.optimizers.**Adagrad**(*pred=None*, *loss=<function mse>*, *learning_rate=1.0*, *hyperparameters={}*)

Description: Ordinary Gradient Descent optimizer. :param pred: a prediction function implemented with jax.numpy :type pred: function :param loss: specifies loss function to be used; defaults to MSE :type loss: function :param learning_rate: learning rate :type learning_rate: float

Returns None

__init__(*pred=None*, *loss=<function mse>*, *learning_rate=1.0*, *hyperparameters={}*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([pred, loss, learning_rate, ...])	Initialize self.
gradient(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.
set_loss(new_loss)	Description: updates internal loss

Continued on next page

Table 82 – continued from previous page

set_predict(pred[, loss])	Description: Updates internally stored pred and loss functions :param pred: predict function, must take params and x as input :type pred: function :param loss: loss function.
update(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.

tigercontrol.models.optimizers.Adam

class tigercontrol.models.optimizers.**Adam**(*pred=None*, *loss=<function mse>*, *learning_rate=1.0*, *hyperparameters={}*)

Description: Ordinary Gradient Descent optimizer. :param pred: a prediction function implemented with jax.numpy :type pred: function :param loss: specifies loss function to be used; defaults to MSE :type loss: function :param learning_rate: learning rate :type learning_rate: float

Returns None

__init__(*pred=None*, *loss=<function mse>*, *learning_rate=1.0*, *hyperparameters={}*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([pred, loss, learning_rate, ...])	Initialize self.
gradient(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.
set_loss(new_loss)	Description: updates internal loss
set_predict(pred[, loss])	Description: Updates internally stored pred and loss functions :param pred: predict function, must take params and x as input :type pred: function :param loss: loss function.
update(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.

tigercontrol.models.optimizers.ONS

class tigercontrol.models.optimizers.**ONS**(*pred=None*, *loss=<function mse>*, *learning_rate=1.0*, *hyperparameters={}*)

Online newton step algorithm.

__init__(*pred=None*, *loss=<function mse>*, *learning_rate=1.0*, *hyperparameters={}*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([pred, loss, learning_rate, ...])	Initialize self.
general_norm(x)	
gradient(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.
norm_project(y, A, c)	Project y using norm A on the convex set bounded by c.

Continued on next page

Table 84 – continued from previous page

set_loss(new_loss)	Description: updates internal loss
set_predict(pred[, loss])	Description: Updates internally stored pred and loss functions :param pred: predict function, must take params and x as input :type pred: function :param loss: loss function.
update(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.

tigercontrol.models.optimizers.SGD

class tigercontrol.models.optimizers.**SGD**(*pred=None*, *loss=<function mse>*, *learning_rate=0.0001*, *hyperparameters={}*)

Description: Stochastic Gradient Descent optimizer. :param pred: a prediction function implemented with jax.numpy :type pred: function :param loss: specifies loss function to be used; defaults to MSE :type loss: function :param learning_rate: learning rate :type learning_rate: float

Returns None

__init__(*pred=None*, *loss=<function mse>*, *learning_rate=0.0001*, *hyperparameters={}*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([pred, loss, learning_rate, ...])	Initialize self.
gradient(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.
set_loss(new_loss)	Description: updates internal loss
set_predict(pred[, loss])	Description: Updates internally stored pred and loss functions :param pred: predict function, must take params and x as input :type pred: function :param loss: loss function.
update(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.

tigercontrol.models.optimizers.OGD

class tigercontrol.models.optimizers.**OGD**(*pred=None*, *loss=<function mse>*, *learning_rate=1.0*, *hyperparameters={}*)

Description: Ordinary Gradient Descent optimizer. :param pred: a prediction function implemented with jax.numpy :type pred: function :param loss: specifies loss function to be used; defaults to MSE :type loss: function :param learning_rate: learning rate :type learning_rate: float

Returns None

__init__(*pred=None*, *loss=<function mse>*, *learning_rate=1.0*, *hyperparameters={}*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([pred, loss, learning_rate, ...])	Initialize self.

Continued on next page

Table 86 – continued from previous page

gradient(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.
set_loss(new_loss)	Description: updates internal loss
set_predict(pred[, loss])	Description: Updates internally stored pred and loss functions :param pred: predict function, must take params and x as input :type pred: function :param loss: loss function.
update(params, x, y[, loss])	Description: Updates parameters based on correct value, loss and learning rate.

tigercontrol.models.optimizers.mse

tigercontrol.models.optimizers.**mse**(*y_pred*, *y_true*)

Description: mean-square-error loss :param y_pred: value predicted by model :param y_true: ground truth value :param eps: some scalar

tigercontrol.models.optimizers.cross_entropy

tigercontrol.models.optimizers.**cross_entropy**(*y_pred*, *y_true*, *eps=1e-09*)

Description: cross entropy loss, y_pred is equivalent to logits and y_true to labels :param y_pred: value predicted by model :param y_true: ground truth value :param eps: some scalar

boosting

tigercontrol.models.boosting.SimpleBoost()	Description: Implements the equivalent of an AR(p) model - predicts a linear combination of the previous p observed values in a time-series

tigercontrol.models.boosting.SimpleBoost

class tigercontrol.models.boosting.**SimpleBoost**

Description: Implements the equivalent of an AR(p) model - predicts a linear combination of the previous p observed values in a time-series

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
help()	Description: Prints information about this class and its methods.
initialize(model_id, model_params[, N, ...])	Description: Initializes autoregressive model parameters :param model_id: id of weak learner model :type model_id: string :param model_params: dict of params to pass model :type model_params: dict :param N: default 3.

Continued on next page

Table 88 – continued from previous page

predict(x)	
to_ndarray(x)	Description: If x is a scalar, transform it to a (1, 1) numpy.ndarray; otherwise, leave it unchanged.
update(y)	

Attributes

compatibles

1.1.4 tigercontrol.experiments package

core

create_full_problem_to_models(problems_ids, ...)	Description: Associate all given problems to all given models.
run_experiment(problem, model[, metric, ...])	Description: Initializes the experiment instance.

tigercontrol.experiments.create_full_problem_to_models

tigercontrol.experiments.**create_full_problem_to_models**(*problems_ids*, *model_ids*)

Description: Associate all given problems to all given models.

Parameters

- **problem_ids** (*list*) – list of problem names
- **model_ids** (*list*) – list of model names

Returns association problem -> model

Return type full_problem_to_models (dict)

tigercontrol.experiments.run_experiment

tigercontrol.experiments.**run_experiment**(*problem*, *model*, *metric='mse'*, *key=0*, *timesteps=100*, *verbose=True*, *load_bar=True*)

Description: Initializes the experiment instance.

Parameters

- **problem** (*tuple*) – problem id and parameters to initialize the specific problem instance with
- **model** (*tuple*) – model id and parameters to initialize the specific model instance with
- **metric** (*string*) – metric we are interesting in computing for current experiment
- **key** (*int*) – for reproducibility
- **timesteps** (*int*) – number of time steps to run experiment for

Returns loss series for the specified metric over the entirety of the experiment time (float): time elapsed memory (float): memory used

Return type loss (list)

metrics

mse(y_pred, y_true)	Description: mean-square-error loss
cross_entropy(y_pred, y_true[, eps])	Description: cross entropy loss, y_pred is equivalent to logits and y_true to labels

tigercontrol.experiments.mse

tigercontrol.experiments.**mse**(*y_pred*, *y_true*)

Description: mean-square-error loss

Parameters

- **y_pred** – value predicted by model
- **y_true** – ground truth value
- **eps** – some scalar

tigercontrol.experiments.cross_entropy

tigercontrol.experiments.**cross_entropy**(*y_pred*, *y_true*, *eps=1e-09*)

Description: cross entropy loss, y_pred is equivalent to logits and y_true to labels

Parameters

- **y_pred** – value predicted by model
- **y_true** – ground truth value
- **eps** – some scalar

experiment

Experiment()	Description: Experiment class

tigercontrol.experiments.Experiment

class tigercontrol.experiments.**Experiment**

Description: Experiment class

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
add_model(model_id[, model_params, name])	Description: Add a new model to the experiment instance.
add_problem(problem_id[, problem_params, name])	Description: Add a new problem to the experiment instance.

Continued on next page

Table 93 – continued from previous page

avg_regret(loss)	
graph([problem_ids, metric, avg_regret, ...])	Description: Show a graph for the results of the experiments for specified metric.
help()	Description: Prints information about this class and its methods.
initialize([problems, models, ...])	Description: Initializes the experiment instance.
scoreboard([metric, n_digits, truncate_ids, ...])	Description: Show a scoreboard for the results of the experiments for specified metric.
to_csv(table_dict, save_as)	Save to csv file

new_experiment

NewExperiment()	Description: class for implementing algorithms with enforced modularity

tigercontrol.experiments.NewExperiment

class tigercontrol.experiments.**NewExperiment**

Description: class for implementing algorithms with enforced modularity

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
help()	Description: Prints information about this class and its methods.
initialize(problems, models[, ...])	Description: Initializes the new experiment instance.
run_all_experiments()	Descripton: Runs all experiments and returns results

precomputed

recompute([verbose, load_bar])	Description: Recomputes all the results.
load_prob_model_to_result([problem_ids, ...])	Description: Initializes the experiment instance.

tigercontrol.experiments.recompute

tigercontrol.experiments.**recompute**(*verbose=False*, *load_bar=False*)

Description: Recomputes all the results.

Parameters

- **verbose** (*boolean*) – Specifies whether to print what experiment is currently running.
- **load_bar** (*boolean*) – Specifies whether to show a loading bar while the experiments are running.

tigercontrol.experiments.load_prob_model_to_result

tigercontrol.experiments.**load_prob_model_to_result**(*problem_ids=['ARMA-v0', 'Crypto-v0', 'SP500-v0']*, *model_ids=['LastValue', 'AutoRegressor', 'RNN', 'LSTM']*, *problem_to_models=None*, *metrics='mse'*)

Description: Initializes the experiment instance.

Parameters

- **problem_ids** (*list*) – ids of problems to evaluate on
- **model_ids** (*list*) – ids of models to use
- **problem_to_models** (*dict*) – map of the form problem_id -> list of model_id. If None, then we assume that the user wants to test every model in model_to_params against every problem in problem_to_params
- **metrics** – metrics to load

CHAPTER 2

help

email alexjyu@google.com

2.1 Need further help

Please join the IRC channel

CHAPTER 3

License

Some license

CHAPTER 4

Contact

alexjyu@google.com

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

Symbols

A

C

D

E

G

I

K

L

M

N

O

P

R

S

T

U