NLoed

Release 0.0.1

Nathan Braniff

Jan 22, 2021

CONTENTS:

CHAPTER

ONE

NLOED'S MODEL CLASS

class nloed.model.**Model**(*observ_struct*, *input_names*, *param_names*, *options={}*)

Bases: `object`

The NLoed Model class implements a mathematical model and provides useful functions for model building.

This class encodes casadi symbolic structures connecting model inputs and parameters to the observation variables. The class also encodes the assumed distirbution for each model observation variable.

Upon construction, the class populates a series of casadi function attributes for computing various model predictions (i.e. mean, variance, sensitivity), logliklhoods, data sampling, and the fisher information. These function attributes are used to implement the class's public user-callable functions; fit(), predict(), evaluate() and sample(). The function attributes are also used by NLoed's Design class when a Model instance is passsed for experimental design.

symbolics_boolean

A boolean indicating if Casadi SX (True) or MX (false) symbolics should be used.

Type bool

num_observ

An integer indicating the number of observation variables in the model.

Type integer

num_input

An integer indicating the number of input variables accepted by the model.

Type integer

num_param

An integer indicating the number of parameters accepted by the model.

Type integer

input_name_list

A list of the input variable names, in the order passed to the model constructor.

Type list of strings

param_name_list

A list of the parameter names, in the order passed to the model constructor.

Type list of strings

observ_name_list

A list of the observation variable names, in the order passed to the model constructor.

Type list of strings

`loglik`

This function attribute consists of a dictionary in which the keys are the model's observation variable names and the values are casadi functions computing the loglikelihood of a single observation of the variable at the passed observation value with the passed input and parameter settings.

Call Structure: Model.loglik[obs_name](obs_value, inputs, parameters)

Type dictionary of functions

`fisher_info_matrix`

This function attribute consists of a dictionary in which the keys are the model's observation variable names and the values are casadi functions computing the fisher information matrix for a single observation of the specified observation variable at the passed input and parameter settings. Model.model_mean(inputs, parameters)

Call Structure: Model.fisher_info_matrix[obs_name](inputs, parameters)

Type dictionary of functions

`model_mean`

This function attribute consists of a dictionary in which the keys are the model's observation variable names and the values are casadi functions computing the expected mean observation value at the passed input and parameter settings.

Call Structure: Model.model_mean[obs_name](inputs, parameters)

Type dictionary of functions

`model_variance`

This function attribute consists of a dictionary in which the keys are the model's observation variable names and the values are casadi functions computing the expected variance of the observation variable at the passed input and parameter settings.

Call Structure: Model.model_variance[obs_name](inputs, parameters)

Type dictionary of functions

`model_sensitivity`

This function attribute consists of a dictionary in which the keys are the model's observation variable names and the values are casadi functions computing the parameteric sensitivity of the expected mean observation of the specified variable at the passed input and parameter settings.

Call Structure: Model.model_variance[obs_name](inputs, parameters)

Type dictionary of functions

`observation_sampler`

This function attribute consists of a dictionary in which the keys are the model's observation variable names and the values are casadi-based functions generating random realizations of the observation variable value from its expected distribution, conditioned on the passed input and parameter settings.

Call Structure: Model.model_variance[obs_name](inputs, parameters)

Type dictionary of functions

`observation_percentile`

This function attribute consists of a dictionary in which the keys are the model's observation variable names and the values are casadi-based functions computing the requested percentile of the observation variable's distribution conditioned on the passed input and parameter settings.

Call Structure: Model.model_variance[obs_name](percentile, inputs, parameters)

Type dictionary of functions

distribution_dict = {'Bernoulli': ['Probability'], 'Binomial': ['Probability'], 'Exp

__init__(*observ_struct*, *input_names*, *param_names*, *options={}*)
The class constructor for NLoed's Model class.

This function accepts the users Casadi functions, distribution lables, and input and parmeter names, and generate an NLoed model object for the specified model. During instantiation the Casadi functions are used to generate a variety of function attributes for each model observation variable; including functions for computing model predictions, loglikelihood, fisher information and data sampling. These function attributes are used both by the NLoed Model instance's public function and if/when the Model instance is passed to the NLoed Design class for optimizing an experimental design.

Parameters

- **observ_struct** (*list of tuples*) – A list of tuples, one tuple for each of the models observation variables. The first element of the tuple is a Casadi function mapping the model inputs and parameters to the given observation's sampling statistics. The second element of each tuple is a string indicating the type of parameteric distribution assigned to that observation variable. Observation names are extracted from the string label of the Casadi function for each obervation variable.
- **input_names** (*list of strings*) – A list of strings specifying the names of the model inputs in the same order that the inputs are expected by the Casadi functions also passed by the user.
- **param_names** (*list of strings*) – A list of strings specifying the names of the model parameters in the same order that the parameters are expected by the Casadi functions also passed by the user.
- **options** (*dict, optional*) – A dictionary of user-defined options, possible key-value pairs include:

  "ScalarSymbolics" – Purpose: Determines whether SX or MX Casadi symbolics are used within the NLoed Model class, True implies scalar symbolics via SX., Type: boolean, Default Value: True, Possible Values: True or False

fit(*datasets*, *start_param=None*, *options={}*)
A function for fitting the NLoed model to a dataset contained in a dataframe.

This function fits the model to a dataset using maximum likelihood. This function can also return marginal confidence intervals as well as plots of liklihood profiles and projections of profiles traces and confidence contours.

Parameters

- **datasets** (*dataframe OR list of dataframes*) – A dataframe containing the dataset to be fit. OR a list of dataframes, each containing a dataset replicate of a given design OR a list of lists of dataframes, where each index in the outer list corresponds to a unique design and each inner index coresponds to a replicate of the given design
- **start_param** (*array-like, optional*) – An array of starting parameter values where the local fitting optimization should be started
- **options** (*dict, optional*) – A dictionary of user-defined options, possible key-value pairs include:

  "Confidence" – Purpose: Determines confidnece diagnostics to be returned or plotted, Type: string, Default Value: "None", Possible Values: "None" = No intervals returned, "Intervals" = Marginal intervals returned, "Profiles" = Same as "Intervals" but trace projections are plotted using Matplotlib, "Contours" = Same as "Profiles" but confidence contour projections are also plotted.

"ConfidenceLevel" – Purpose: Sets the confidence level for the marginal intervals, traces, profiles and contours, Type: float, Default Value: 0.95, Possible Values: 0<1.

'RadialNumber' – Purpose: Determines the number of radial searches performed out from the fit parameter value used to find the confidence contour projections, Type: integer Default Value: 30 Possible Values: >1, but sufficient density is needed for good interpolation

"SampleNumber" – Purpose: Type: integer Default Value: 10 Possible Values: >1

"Tolerance" – Purpose: Type: float Default Value: 0.001 Possible Values: >0

"InitialStep" – Purpose: Type: float Default Value: 0.01 Possible Values: >0

"MaxSteps" – Purpose: Type: integer Default Value: 2000 Possible Values: >1

"SearchFactor" – Purpose: Type: float Default Value: 5.0 Possible Values: >0

"InitParamBounds" – Purpose: Type: array-like Default Value: False Possible Values:

"InitSearchNumber" – Purpose: Type: integer Default Value: 3 Possible Values: >0

"Verbose" – Purpose: Type: boolean Default Value: True Possible Values: True or False

Returns A dataframe containing the fit parameters, and if requested, confidence interval information. If a list of datasets was provided, each row of the dataframe corresponds to the dataset index in the passed list. If a list of lists of dataframes was provided (designs by replicates), a list of dataframes will be returned with the same length as the outer index of the passed list of lists.

Return type dataframe OR list of dataframes

`sample` (*designs*, *param*, *design_replicates=1*, *options={}*)

A function for generating simulated data from the NLoed model for a given design passed as a dataframe.

This function generates datasets using the NLoed model and a provided design (or a list of designs) via simulation and Numpy/Scipy's random number generation. This simulation is done at a user-provided set of parameter values. The number of replicates of the design that are simulated is optional but defaults to one.

Parameters

- **`designs`** (*`dataframe OR list of dataframes`*) – A dataframe containing the design to be simulated, OR a list of dataframes containing multiple designs to be simulated
- **`param`** (*`array-like, floats`*) – The parameter values used to generate the data
- **`design_replicates`** (*`integer, optional`*) – An integer indicating the number of dataset replicates to be generated for each design. The default is 1 per design passed.
- **`options`** (*`dict, optional`*) – A dictionary of user-defined options, possible key-value pairs include:

  "Verbose" – Purpose: Determines the amount of print feedback provided while the function executes Type: bool, Default Value: True, Possible Values: True or False

Returns A dataframe containing a simulation of the design is returned by default OR, if design_replicates was set to >1, a list of dataframes containg simulated replicates is returned for the given design OR, if a list of dataframes containing a set of design was passed, a list of lists of dataframes is returned, the outer index corresponds to the design list, and the inner index corresponds to the number of replicates requested.

Return type dataframe OR list of dataframes

predict (*input_struct*, *param*, *covariance_matrix=None*, *options={}*)
A function for generating prediction information from the NLoed model.

This function can be used to compute predictions from an NLoed model instance given the user-provided input conditions and paramater values. Prediction information includes the predicted mean response of the model, confidence intervals for the mean response under parameter uncertainty, confidence intervals for the observation distribution, and sensitivity analysis of the mean response. The returned intervals can be computed in a number of ways; exactly, with a normal (local and deterministic) approximation, or using Monte Carlo simulation.

Parameters

- **input_struct** (*dataframe*) – A dataframe specifying the combination of inputs values and observations at which predictions are desired.
- **param** (*array-like, floats*) – The parameter vector values at which the predictions are to be made.
- **covariance_matrix** (*array-like, floats, optional*) – A symetric matrix specifying the parameter's normal covariance matrix, required if parameter uncertainty is to be included. The prior mean set by the values passed via the param argument.
- **options** (*dictionary, optional*) – A dictionary of user-defined options, possible key-value pairs include:

  ”Method” – Purpose: Selects the method used to compute the mean and prediction and observation intervals. Type: string Default Value: “Delta” Possible Values: “Exact”=Compute predictions and intervals exactly; this option is only available if no parameter uncertainty information is NOT provided . “Delta”=Use a normal, local apporximation to compute the prediction and intervals, “MonteCarlo”=Use Monte Carlo simulation of the model to compute the prediction and intervals under any parameter or observation uncertainty.

  ”PredictionInterval” – Purpose: A boolean to indicat if prediction intervals are to be returned, true if yes. Type: bool Default Value: False Possible Values: True or False

  ”ObservationInterval” – Purpose: A boolean to indicat if observation intervals are to be returned, true if yes. Type: bool Default Value: False Possible Values: True or False

  ”Sensitivity” – Purpose: A boolean to indicat if observation intervals are to be returned, true if yes. Type: bool Default Value: False Possible Values: True or False

  ”PredictionSampleNumber” – Purpose: An integer indicating the number of parameter vectors to be sampled from the prior in order to compute the prediction (and observation0 intervals using Monte Carlo simulation. Type: integer Default Value: 10000 Possible Values:

  ”ObservationSampleNumber” – Purpose: An integer indicating the number of observation values to be sampled from the observation distribution in order to compute the observation interval using Monte Carlo simulation. Type: integer Default Value: 10 Possible Values:

  ”ConfidenceLevel” – Purpose: A float specifying the confidence level desired for any intervals computed. Type: float Default Value: 0.95 Possible Values: <1, >0

Returns

A dataframe containing the requested prediction quntities computed at the input and parameter settings passed.

Return type dataframe

evaluate (*designs*, *param*, *options={}*)

A function for evaluating the peformance metrics of an experimental design applied to the NLoed Model.

This function can be used to produce evaluation metrics regarding the expected parameter fitting accurach a given experimental design will achieve when used with the given NLoed Model instance.

Evaluation metrics inclide the expected parameter covariance, bias and mean squared error as well as the Fisher information matrix. These metrics can be computed either asymptoticall (i.e. via the Fisher information matrix; in this case the bias is zero and the covariance and MSE are equal) or using Monte Carlo simulation and fitting (in which case the bias, covariance and MSE may be unique). By default this function only returns the parameter covariance for the design, computed asymptoticall.

All evaluation is performed at the candidate parameter vector passed, and as such these metrics are only local appoximations valid near the candidate point.

Parameters

- **`designs`** (*`dataframe OR list of dataframes`*) – A dataframe specifying the experimental design to be evaluated, OR a list of dataframes describing a set ofexperimental designs to be evaluated.
- **`param`** (*`array-like, floats`*) – The parameter vector values at which the predictions are to be made.
- **`options`** (*`dictionary, optional`*) – A dictionary of user-defined options, possible key-value pairs include:

  ”Method” – Purpose: A string indicating which computational method, asymptotic or Monte Carlo, is used to compute the evaluation metrics. Type: string Default Value: “Asymptotic Possible Values: “Asymptotic”=Uses a first-order approximation based onvthe FIM matrix to compute the parameter covariance; bias is zero. “MonteCarlo”= Uses repeated data simulation and fitting to compute the evaluation metrics.

  ”Covariance” – Purpose: A boolean value indicating if the parameter covariance matrix should be computed and returned, true implies yes. Type: boolean Default Value: True Possible Values: True or False

  ”Bias” – Purpose: A boolean value indicating if the parameter bias vector should be computed and returned, true implies yes. Type: boolean Default Value: False Possible Values: True or False

  ”MSE” – Purpose: A boolean value indicating if the parameter mean squated error vector should be computed and returned, true implies yes. Type: boolean Default Value: False Possible Values: True or False

  ”FIM” – Purpose: A boolean value indicating if the Fisher information matrix should be computed and returned, true implies yes. Type: boolean Default Value: False Possible Values: True or False

  ”SampleNumber” – Purpose: An integer indicating the number of simulations of the experimental design, and subsequent fittings, that should be performed if the Monte Carlo method is used to compute the evaluation metrics. Type: integer Default Value: 1000 Possible Values: >0, must be large enough for a statistically stable estimate

Returns A dataframe containg the design evaluation metrics, OR if a list of design dataframes was passed, a list of dataframes is returned each containing the design evaluation metrics for its corresponding design.

Return type dataframe

`__get_distribution_functions`(*observ_model*, *observ_distribution*)

A private function that automatically generates function attributes for the provided observation variable information.

This private function is used during the NLoed Model class construction.

This function accepts the observation name, distribution type and observation model and constructs casadi functions to compute the logliklihood, fisher information, prediction mean/sensitivity/variance and also a numpy/casadi function to return random samples from the the model

Parameters

- **`observ_model`** – A symbolic Casadi function mapping the model input and parameter vectors to the observation distribution statistics.
- **`observ_distribution`** (*`string`*) – A string specifying the observation distribution type, must be one of the supported distributions.

Returns

A list of Casadi(-based) functions in the following order; loglikelihood, fisher information, observation mean , observation variance, mean sensitivity, observation sampler, observation percentile

Return type list of functions

`__confidence_intervals`(*mle_params*, *loglik_func*, *options*)

A private helper function for computing parameter confidence intervals.

This private function is used in some calls to the public Model.fit() function and is triggered by a user call after instantiation.

This function computes marginal parameter confidence intervals for the model around the MLE estimate using profile likelihoods.

Parameters

- **`mle_params`** (*`array-like, floats`*) – A vector specifying the MLE parameter estimates generated and passed during a call to Model.fit().
- **`loglik_func`** – A Casadi function for computing the total data log-likelihood for the current dataset being handled within the calling Model.fit() call. The only argument is a putative parameter vector.
- **`options`** (*`dictionary`*) – A dictionary of user-defined options encoded as key-value pairs. This dictionary is passed through from a call to Model.fit(), see the fit() functions documentation for possible key-value pairs.

Returns

A list of lists, the outer index corresponds to each parameter coordinate, the innder list contains both the upper and lower bounds for each parameter.

Return type list of lists

`__profile_plot`(*mle_params*, *loglik_func*, *figure*, *options*)

A private helper function for plotting both the profile likelihoods and the projections of the profiles traces.

This private function is used in some calls to the public Model.fit() function and is triggered by a user call after instantiation.

This function generates a square grid of Matplotlib sub-plots, with a row and column for each parameter. The logelikelihood profiles of each parameter are plotted on sub-plots along the diagonal. These diganol plots also include a horizantal reference line indicating loglikleihood value at which the perscribed confidence level boundaries occur. Projections of the profile likelihood traces for each pair of parameters value are plotted on the lower triangular sub-plots. This function also returns lists containing the computed intervals, traces and profiles.

Parameters

- **mle_params** (*array-like, floats*) – A vector specifying the MLE parameter estimates generated and passed during a call to Model.fit().
- **loglik_func** (*function*) – A Casadi function for computing the total data log-likelihood for the current dataset being handled within the calling Model.fit() call. The only argument is a putative parameter vector.
- **figure** (*integer*) – The figure object handle on which the plot is generated.
- **options** (*dictionary*) – A dictionary of user-defined options encoded as key-value pairs. This dictionary is passed through from a call to Model.fit(), see the fit() functions documentation for possible key-value pairs.

Returns

A multi-level list of lists is returned, the outer list contains three elements; 1 - a list of lists where the outer index corresponds to each parameter and the inner list contains the upper and lower confidence interval bounds for each parameter, 2 - a list of list of lists where the outer index corresponds to each parameter and the middle index corresponds to points along each profile trace and the inner list contains the value of the parameter vector at that point in the parameter trace. 3 - a ;ist of lists where the outer index corresponds to each parameter value and the inner index corresponds to logliklihood ratio values for the given parameter at points along the likelihood profile. Note this return is passed up from a call to __profile_trace()

Return type nested lists

__profile_trace(*mle_params*, *loglik_func*, *options*)

A private helper function for computing the likelihood profile and trace for each parameter.

This private function is used in some calls to the public Model.fit() function and is triggered by a user call after instantiation.

This function accepts the MLE parameter estimate and a Casadi function for the total data log-likelihood of the target dataset being processed by the current Model.fit() call and uses this to organize computation of a profile likelihood and the corresponding parameter vector trace along the computed profile, for each parameter. In the course of the this computation, this function also computes the likelihood confidence intervals for each parameter as well. This function organizes the profile computation for each parameter and in both increasing and decreasing directions, however it relies on lower level helper functions (__loglik_search and __profile_setup) to do the actual computing.

Parameters

- **mle_params** (*array-like, floats*) – A vector specifying the MLE parameter estimates generated and passed during a call to Model.fit().
- **loglik_func** (*function*) – A Casadi function for computing the total data log-likelihood for the current dataset being handled within the initating Model.fit() call. The only argument is a putative parameter vector.
- **options** (*dictionary*) – A dictionary of user-defined options encoded as key-value pairs. This dictionary is passed through from a call to Model.fit(), see the fit() functions documentation for possible key-value pairs.

Returns

A multi-level list of lists is returned, the outer list contains three elements; 1 - a list of lists where the outer index corresponds to each parameter and the inner list contains the upper and lower confidence interval bounds for each parameter, 2 - a list of list of lists where the outer index corresponds to each parameter and the middle index corresponds to points along each profile trace and the inner list contains the value of the parameter vector at that point in the parameter trace. 3 - a ;ist of lists where the outer index corresponds to each parameter value and the inner index corresponds to logliklihood ratio values for the given parameter at points along the likelihood profile.

Return type nested lists

`__contour_plot` (*mle_params*, *loglik_func*, *figure*, *options*)

A private helper function for plotting the likelihood confidence contour plots during calls to Model.fit()

This private function is used in some calls to the public Model.fit() function and is triggered by a user call after instantiation.

This function plots the projections of the profile likelihood-based confidence volume in a 2d plane for each pair of parameters using Matplotlib. This creates marginal confidence contours for each pair of parameters which are added to the trace projection plots generated by __profile_plot() in the lower-triangualr sub-plots of the passed figure.

Parameters

- **`mle_params`** (*`array-like, floats`*) – A vector specifying the MLE parameter estimates generated and passed during a call to Model.fit().
- **`loglik_func`** (*`function`*) – A Casadi function for computing the total data log-likelihood for the current dataset being handled within the calling Model.fit() call. The only argument is a putative parameter vector.
- **`figure`** (*`integer`*) – The figure object handle on which the plot is generated.
- **`options`** (*`dictionary`*) – A dictionary of user-defined options encoded as key-value pairs. This dictionary is passed through from a call to Model.fit(), see the fit() functions documentation for possible key-value pairs.

`__contour_trace` (*mle_params*, *loglik_func*, *coordinates*, *options*)

A private helper function for computing the profile likelihood confidence contour projections for a specified pair of parameters.

This private function is used in some calls to the public Model.fit() function and is triggered by a user call after instantiation.

This function computes a projection boundary of the profile likelihood confidence volume in a 2D plane for the specified pair of parameters. To do this a series of vectors radiating out from the MLE in the target parameter pair plane are searched. Along these vectors the marginal (non-target-pair) parameters are re-optimized iteratively. The vectors are extended until the conditionally optimized value of the likelihood falls below the selected confidence threshold. The 2D coordinates of these boundary points are recorded and their values are used to interpolate the projected confidence boundary via a periodic spline. This function sets up the radial search and performs the interpolation but relies on lower level helper functions (__profile_setup and __loglik_search) to compute the boundary points.

Parameters

- **`mle_params`** (*`array-like, floats`*) – A vector specifying the MLE parameter estimates generated and passed during a call to Model.fit().
- **`loglik_func`** (*`function`*) – A Casadi function for computing the total data log-likelihood for the current dataset being handled within the calling Model.fit() call. The only argument is a putative parameter vector.

- **coordinates** (*array-like, integers*) – A pair of parameter indices specifying the specific parameter pair for which a two 2D contour is to be computed in parameter space.
- **options** (*dictionary*) – A dictionary of user-defined options encoded as key-value pairs. This dictionary is passed through from a call to Model.fit(), see the fit() functions documentation for possible key-value pairs.

Returns A list of lists is returned, where the outer list contains two entries; a list of x coorindates followed by a list of y coordinates. The coordinates fall on the contour projection in the 2D target parameter pair plane. The number of of x and y coordinates is equal and is set by the "RadialNumber" key in Model.fit()'s options dictionary.

Return type list of lists

__profile_setup(*mle_params*, *loglik_func*, *fixed_params*, *direction*, *options*)

A private helper function for constructing profile likelihood optimization solvers.

This private function is used in some calls to the public Model.fit() function and is triggered by a user call after instantiation.

This function creates a Casadi optimization solver object for optimizing the model's likelihood (on the target dataset of the iniating Model.fit() call) w.r.t. the marginal parameter values while holding a specific subset of parameters fixed at specified values.

There can be a single fixed parameter (for the common profile likelihood computations), or a set (currently we only ever need a pair, used for computing confidence contour projections in 2D). The fixed parameter values are specified by a radius and direction vector which makes it easy to perform radial searches required for the confidence contour trace (the optimizer is iteratively run with different radia values in order to find the boundary points). The direction vector is hard-coded into the resulting solver object after a call to __profile_setup() but the radius can be adjust (hence moving the 'fixed' parameters) when the solver object is actually run for optimization. This alterable radius value enables the actual profile likelihood search.

The solver object actually optimizes the loglikelihood ratio (LLR) gap, the likelihood ratio gap is the negative difference between the chi-squared boundary (specified by the user in the initiating call to Model.fit()) and the loglikelihood ratio at the current marginal parameter vector being optimized. This is just a rezeroing of the scale and is equivlant to optimizing the loglikelihood but makes some of the numerics and output more convenient to manage.

Note that limited testing has been done on single parameter models however the resulting return object for a 1D model should behave like a regular solver but actually just returns the LLR gap value at the radius-specified fixed values of the lone parameter.

Parameters

- **mle_params** (*array-like, floats*) – A vector specifying the MLE parameter estimates generated and passed during a call to Model.fit().
- **loglik_func** (*function*) – A Casadi function for computing the total data loglikelihood for the current dataset being handled within the calling Model.fit() call. The only argument is a putative parameter vector.
- **fixed_params** (*array-like, booleans*) – A boolean vector that is the same length as the model's parameter vector. True values imply the cooresponding parameter (by position) should remain fixed in the solver object by the provideddirection and radius, False values indicate the corresponding parameter is marginal and should optimized.
- **direction** (*array-like, floats*) – A vector that is the same length as the parameter vector specifying a direction in parameter space. Coordinate specified as True (non-marginal) in the fixed_params argument are used as the direction (along which the solver's run-time radius argument specifies the value of the fixed parameters)

- **options** (*dictionary*) – A dictionary of user-defined options encoded as key-value pairs. This dictionary is passed through from a call to Model.fit(), see the fit() functions documentation for possible key-value pairs.

Returns A Casadi Ipopt solver object is returned, which can optimize the loglikelihood ratio gap w.r.t. the marginal parameter values, conditioned on the fixed parmeter value which are specified by a radius (set at call time) and direction (set in __profile_setup() above when the solver is created).

Return type casadi solver

__loglik_search(*profile_loglik_solver*, *marginal_param*, *options*, *forward=True*)

A private helper function for performing a bisection search for profile likelihood boundary points.

This private function is used in some calls to the public Model.fit() function and is triggered by a user call after instantiation.

This function performs a root finding algorithm using the solver object (profile_loglik_solver) generated by __profile_setup(). Currently the function uses a bisection search. The search is performed along the direction vector specified when the solve object is created by __profile_setup(), with the given set of fixed and marginal parameters specified in that call. The search dimensions is the radius argument of the solver object, which specified when combined with the direction vector hard-coded in the solver object to specify its magnuted, resulst in a specific set of fixed parmaeter values at which the conditional likelihood optimization is performed. The radius at which a root is found indicates a boundary point in the confidence volume at the confidence level specified by the use in their initating call to Model.fit().

Note a passed version used Halley's method; a higher order extension of Newton's method for finding roots. This was possible using Casadi's ability to differentiate optimization solutions and was faster, but proved to be numerically unstable.

Parameters

- **profile_loglik_solver** – The Casadi Ipopt solver object functions for finding the conditionally optimized loglikelihood ratio gap at a given radius from MLE. This object is generated with a call to __profile_setup().
- **marginal_param** (*array-like, floats*) – The starting values (usually the MLE) for the marginal parameters that are passed to the Ipopt solver.
- **options** (*dictionary*) – A dictionary of user-defined options encoded as key-value pairs. This dictionary is passed through from a call to Model.fit(), see the fit() functions documentation for possible key-value pairs.
- **forward** (*boolean*) – A boolean indicating the directino of the search, if True the search is done in the forward (positive) radius direction (relative to direction specidied in solver object), if False the search is performed starting with a negative radius.

Returns The return object is a three element list, the first entry is contains the radius value at which the root was found, the second entry contains a list of the optimzed marginal parameter values at the root, and the third entry contains the residual value of the LLR gap at the root (this should be near zero).

Return type list

__create_grid(*input_candidate_list*)

A private helper function to create a grid of candidate parameter points

This fucntion creates a grid of candidate parameter points by permuting a lits of candidate points. This is used for an a crude initial parmeter search prior to maximum likelihood optimization by the Model.fit function. This function uses recursion to permute the candidate grid.

Parameters **`input_candidate_list`** (*`list of array-likes`*) – A list, the same length as the model's parameter vector, each element is an array-like vector of candidate parameter values for the given parameter.

Returns The return is a list, the same length as the number of permutations of the candidate paramter points passed, each element of the list contains a Numpy parameter vector with a unique permutation of the candidate points.

Return type list of arrays

`__progress_bar` (*iteration*, *total*, *prefix*='')

A private helper function to print a progress bar in a looped process

This fucntion prints or update a progress bar on the commandline output indicating, the current progress through a loop.

Parameters

- **`iteration`** (*`integer`*) – Current iteration in the process
- **`total`** (*`integer`*) – Total number of iterations in the process
- **`prefix`** (*`string`*) – A prefix string to name the process

CHAPTER

TWO

NLOED'S DESIGN CLASS

class nloed.design.**Design**(*models*, *parameters*, *objective*, *discrete_inputs=None*, *continuous_inputs=None*, *observ_groups=None*, *fixed_design=None*, *options={}*)

Bases: `object`

The NLoed Design class formulates a design optimization problem and generates an optimized relaxed experimental design.

This class accepts a set of NLoed model instances as well as the users specifications of their experimental requirments. This information is used to formulate the experimental design problem as a non-linear programming problem that is passable to the IPOPT solver via Casadi's interface.

During construction, optimization problem is structured and passed to IPOPT. The problem solution is then extracted and stored within the model object. This solution represents a relaxed solution to the design problem and has real valued weights representing the number of samples taken in various input conditions. The classes user-callable function can be used to round the relaxed solution to an exact design with a specified sample size.

num_models

The number of models for which the design was optimized, generally 1.

Type integer

input_dim

The dimensions of the model input, this must be consistent across all of the models passed.

Type integer

observ_dim

The number of observation variables that each model has, all models must have the same observation variables.

Type integer

input_name_list

A list of names for the model inputs, these must be shared by all of the passed models.

Type list of strings

observ_name_list

Alist of names for the model observation variables, these must match across all models passed.

Type list of strings

model_list

A list of the NLoed Model instances passed to the design constructor.

Type list of models

`param_list`
A list of the nominal parameter values at which the design is to be optimized.

Type list of array-like

`objective_list`
A list of the objectives which are to be optimized (as a weighted combination).

Type list of strings

`approximate_design`
A dataframe containing the optimized relaxed design returned by IPOPT.

Type dataframe

`__init__`(*models*, *parameters*, *objective*, *discrete_inputs=None*, *continuous_inputs=None*, *observ_groups=None*, *fixed_design=None*, *options={}*)
The class constructor for NLoed's Design class.

This function accepts an NLoed Model instance as well as experimental specifications such as the nominal parameter values, the objective, input constraints and format, observation groupings and past design information passed by the user. This information specifies the objectives constraints the user has for their desired experiment. During construction a Casadi symbolic structure is created for the overall design objective. This symbol is used to call IPOPT via Casadi's optimization interface. After IPOPT optimization is complete, the constructor function parses the returned solution into a dataframe which is stored as a class attribute.

Parameters

- **`models`** (*`list of models`*) – A list of NLoed model isntances for which an optimize design is desired.
- **`parameters`** (*`list of array-like`*) – A list of the nominal parameter values at which the design should be optimized.
- **`objective`** (*`list of strings`*) – A list of string names for the objectives to be used for design optimization. Multiple objectives are combined as a weighted combination.
- **`discrete_inputs`** (*`dictionary, conditionally optional`*) – A dictionary indicating which model inputs, if any, should be handled discretly. Discret input values are fixed within the optimizatio call, according to the discretization chosen. Inputs are selected based on adjusting the sampling weights over the discrete grid of values. The exact method used for discretization can also be specified in this structure.

  NOTE– need examples
- **`continuous_inputs`** (*`dictionary, conditionally optional`*) – A dictionary indicating which model inputs, if any, should be handled continuously. Continuous inputs are treated as optimization variables within the optimization call, and can be adjusted as real values within their boundaries. The user needs to specify the number of unique possible continuous levels that each input can take within the overall design. The continuous bounds and number of unique levels for each continuous input can also be specified in this structure.

  NOTE– need examples
- **`observ_groups`** (*`list of list of strings`*) – A list of lists, the outer list consists of an entry for each grouping of observation variables that must be sampled together. Each inner list contains the string names of the observation variables that are sampled together in each group.
- **`fixed_design`** (*`dictionary`*) – A dictionary specifiying the structure of any fixed aspects of the experimental design, or past experimental designs on which the new design

should be conditioned. This dictionary specifies both the fixed design structure and its expected sampleing weight relative to the new design.

NOTE– need examples

- **options** (*dict, optional*) – A dictionary of user-defined options, possible key-value pairs include:

  ”LockWeights” – Purpose: Type: , Default Value: , Possible Values:

  ”NumGridPoints” – Purpose: Type: , Default Value: , Possible Values:

round(*sample_size*, *options={}*)

A function

This function

Parameters

- **sample_size** (*integer*) –
- **options** (*dictionary*) –

Return:

relaxed(*options={}*)

A function to return the relaxed design dataframe, containing the real-valued sampling weights.

Parameters **()** (*options*) –

Return:

power(*sample_size*, *options={}*)

A function to

This function

Parameters

- **sample_size** (*integer*) –
- **options** (*dictionary*) –

Return:

__optim_settup(*fim_list*, *continuous_symbol_list*, *continuous_lowerbounds*, *continuous_upperbounds*, *continuous_init*, *weight_symbol_list*, *weight_sum*, *weight_init*, *options*)

A function to

This function

Parameters

- **fim_list** –
- **()** (*weight_init*) –
- **()** –
- **()** –
- **()** –
- **()** –
- **()** –
- **()** –

- **options** –

Return:

__weighting_settup (*discrete_input_names*, *discrete_input_grid*, *continuous_input_names*, *continuous_symbol_archetypes*, *fixed_design*, *options*)

A function to

This function

Parameters

- **()** (*options*) –
- **()** –
- **()** –
- **()** –
- **()** –
- **()** –

Return:

__discrete_settup (*discrete_inputs*, *options*)

A function to

This function

Parameters

- **()** (*options*) –
- **()** –

Return:

__continuous_settup (*continuous_inputs*, *options*)

A function to

This function

Parameters

- **()** (*options*) –
- **()** –

Return:

__create_grid (*input_names*, *input_candidates*, *constraints*)

A function to

This function

Parameters

- **()** (*constraints*) –
- **()** –
- **()** –

Return:

__sort_inputs(*newrow*, *rows*, *rowpntr=0*, *colpntr=0*)

A function to

This function

Args:

Return:

CHAPTER THREE

INDEX

- genindex

PYTHON MODULE INDEX

INDEX