F_UNCLE Documentation

Release 0.0

A. Fraser and S. Andrews

Jun 09, 2016

CONTENTS

Contents:

CHAPTER

ONE

DOCUMENTATION

The FUNCLE module

Functional UNcertainty Constrained by Law and Experiment

CHAPTER

TWO

UTILITIES

These are abstract classes which are used in the analysis

2.1 Struc

class F_UNCLE.Utils.Struc.**Struc**(*name*, *def_opts=None*, *informs=None*, *warns=None*, **args*, ***kwargs*)

Abstract object to contain properties and warnings

name
str – The name of the object

def_opts
dict – Default options and bounds

informs
dict – Important user information prompts

warns
dict – Optional warnings

options
dict – The options as set by the user

__init__(*name*, *def_opts=None*, *informs=None*, *warns=None*, **args*, ***kwargs*)

Instantiates the structure

Parameters

- **name** (*str*) – Name of the structure
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Keyword Arguments

- **def_opts** (*dict*) – Dictionary of the default options for the structure Formatted as follows:

```
{option_name(str):
 [type(Type), default(num), lower_bound(num),
  upper_bound(num), unit(str), note(str)]
}
```

- **informs** (*dict*) – Dictionary of the default informs for the structure
- **warns** (*dict*) – Dictionary of the warnings for the structure

Returns None

__str__ (*inner=False*, **args*, ***kwargs*)
Returns a string representation of the object

Parameters

- **inner** (*bool*) – Flag if the string is being called within another string function
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns A string describing the object

Return type (str)

__weakref__
list of weak references to the object (if defined)

_on_str (**args*, ***kwargs*)
Print methods of childeren

Parameters

- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns A string representing the object

Return type (str)

get_inform (*err_id*)
Returns an inform corresponding to the error code

Parameters **err_id** (*int*) – Error ID number

Returns String containing the error message

Return type (str)

get_option (*name*)
Returns the option corresponding the the given name

Parameters **name** (*str*) – Name of the option

Returns Value of the option corresponding to 'name'

get_warn (*warn_id*)
Returns an inform corresponding to the warning code

Parameters **warn_id** (*int*) – Warning ID number

Returns String containing the warning message

Return type (str)

plot (*axis=None*, *hardcopy=None*)
Returns creates

Parameters

- **axis** (*plt.Axes*) – The axis on which to plot the figure, if None, creates a new figure object on which to plot.
- **hardcopy** (*bool*) – If a string, write the figure to the file specified

Returns A reference to the figure containing the plot

Return type (plt.Figure)

set_option(*name*, *value*)
Sets the option corresponding to the given name to a specified value.

Enforces the following checks

1.name is valid

2.value is of correct type

3.value is within bounds

4.**Not Implemented** value has correct units

Parameters

- **name** (*str*) – Name of the option to set
- **value** – Value of the option to set

Returns None

write_to_file(*filename*)
Writes the object to a file

Parameters **filename** (*string*) – A path to a writeable location

2.2 PhysicsModel

class F_UNCLE.Utils.PhysicsModel.**PhysicsModel**(*prior*, *name='Abstract Physics Model'*, **args*, ***kwargs*)
Abstract class for a pysics model

A physics model is computer code that represents how some physical process responds to changes in the regime.

DOF A physics model has degrees of freedom, dof, which represent how many parameters the model has which can be adjusted to affect its response

Prior A physics model has a prior, which represents the best esimate of the model's degrees of freedom. This prior is used by Bayesian methods

..note:

```
**all** abstract methods must be overloaded for a physics model to
work in the `F_UNCLE` framework
```

prior
PhysicsModel – the prior

__init__(*prior*, *name='Abstract Physics Model'*, **args*, ***kwargs*)

Parameters **prior** – Can be either a PhysicsModel' object or a function or a vector which defines the prior

Keyword Arguments **name** (*str*) – A name for the model

_on_update_prior(*prior*)
Instance specific prior update

get_dof(*dof_in*)
ABSTRACT Sets the model's degrees of freedom

Parameters **dof_in** (*Iterable*) – The new values for *all* model degrees of freedom

Returns None

get_scale()
ABSTRACT Returns a matrix to scale the model degrees of fredom

Scaling the model dofs may be necessary where there are large changes in the magnitude of the dofs

Returns

Return type (np.ndarray)

get_sigma(**args*, ***kwargs*)
Abstract Gets the covariance matrix of the model

Parameters

- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

set_dof()
ABSTRACT Gets the model degrees of freedom

Parameters **None** –

Returns The iterable defining the model degrees of freedom

Return type (Iterable)

shape()
ABSTRACT Returns the degrees of freedom of the experiment

Returns Dimensions

Return type (tuple)

update_prior(*prior*)
Updates the prior for the pyhsics model

Parameters **prior** (PhysicsModel) – The prior

2.3 Experiment

class F_UNCLE.Utils.Experiment.**Experiment**(*name='Experiment'*, **args*, ***kwargs*)
Abstract class for experiments

A child of the Struc class. This abstract class contains methods common to all Experiment objects. This class can be used to model two different cases

Simulation Makes use of a single model or set of models internal to the object to simulate some physical process

Experiment Can be of two types

1. A "computational experiment" where a simulation is performed using a nominal *true* model
2. A representation of a real experiment using tabulated values

In order for an Experiment to work with the F_UNCLE framework, it must implement **all** the inherited methods from *Experiment*, regardless if it is a Simulation or Experiment

Attributes

None

Methods

compare(*indep*, *dep*, *model_data*)
Compares a set of experimental data to the model

Parameters

- **indep** (*list*) – The list of independent variables for comparison
- **dep** (*list*) – The list or array of dependent variables for comparison
- **model_data** (*tuple*) – Complete output of a *__call__* to an *Experiment* object which *dep* is compared to at every point in *indep*

Returns

(np.ndarray): The error between the dependent variables and the model for each value of independent variable

get_sigma(**args*, ***kwargs*)
ABSTRACT Gets the co-variance matrix of the experiment

Parameters

- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns A nxn array of the co-variance matrix of the simulation. Where n is the length of the independent variable vector, given by *shape()*

Return type (np.ndarray)

shape(**args*, ***kwargs*)
ABSTRACT Gives the length of the independent variable vector

Returns The number of independent variables in the experiment

Return type (int)

2.4 Spline

class F_UNCLE.Models.Isentrope.**Spline**(*x, y, w=None, bbox=[None, None], k=3, ext=0, check_finite=False*)
Overloaded scipy spline to work as a PhysicsModel

Child of the Scipy IU spline class which provides access to details to the knots which are treated as degrees of freedom

get_basis(*indep_vect*, *spline_end=None*)
Returns the matrix of basis functions of the spline

Parameters **indep_vect** (*np.ndarray*) – A vector of the independent variables over which the basis function should be calculated

Keyword Arguments **spline_end** (*int*) – The number of fixed nodes at the end of the spline

Returns

The n x m matrix of basis functions where the n rows are the response over the independent variable vector to a unit step in the m'th spline coefficient

Return type (np.ndarray)

get_c(*spline_end=None*)
Return the coefficients for the basis functions

Keyword Arguments **spline_end** (*int*) – The number of fixed nodes at the end of the spline

Returns basis function spline coefficients

Return type (numpy.ndarray)

get_t()
Gives the knot locations

Returns knot locations

Return type (numpy.ndarray)

set_c(*c_in*, *spline_end=None*)
Updates the new spline with updated coefficients

Sets the spline coefficeints of this instance to the given values

Parameters **c_in** (*numpy.ndarray*) – The new set of spline coefficeints

Keyword Arguments **spline_end** (*int*) – The number of fixed nodes at the end of the spline

Returns None

CHAPTER

THREE

MODELS

The pysics models used in the analysys

3.1 pyIsentrope

class F_UNCLE.Models.Isentrope.**Isentrope**(*name='Isentrope'*, **args*, ***kwargs*)

Abstract class for an isentrope

The equation of state for an isentropic expansion of high explosive is modeled by this class. The experiments for which this model is used occur at such short timescales that the process can be considered adiabatic

Units

Isentropes are assumed to be in CGS units

Diagram

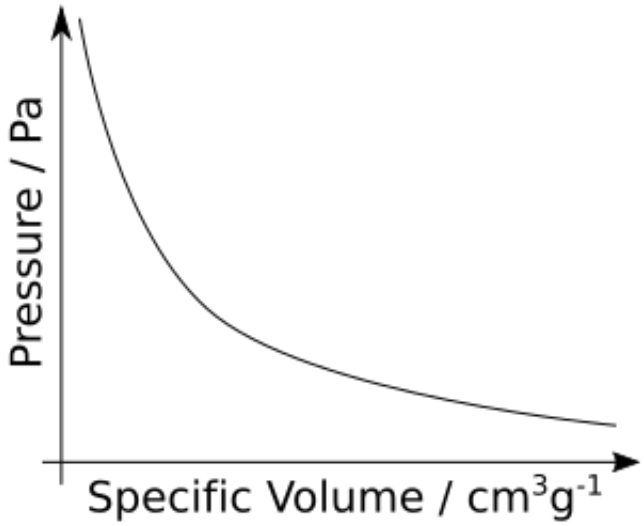

Fig. 3.1: The assumed shape of the equation of state isentrope

__init__(*name='Isentrope'*, **args*, ***kwargs*)

Parameters

- ***args** – Variable length argument list.

- ****kwargs** – Arbitrary keyword arguments.

Keyword Arguments **name** (*str*) – Name if the isentrope *Def 'Isentrope'*

plot(*axis=None*, **args*, ***kwargs*)
Plots the EOS

shape()
Overloaded class to get isentrope DOF's

Overloads `F_UNCLE.Utils.PhysModel.PhysModel.shape()`

Returns (n,1) where n is the number of dofs

Return type (tuple)

3.2 Spline

class `F_UNCLE.Models.Isentrope.`**Spline**(*x*, *y*, *w=None*, *bbox=[None, None]*, *k=3*, *ext=0*, *check_finite=False*)
Overloaded scipy spline to work as a PhysicsModel

Child of the Scipy IU spline class which provides access to details to the knots which are treated as degrees of freedom

get_basis(*indep_vect*, *spline_end=None*)
Returns the matrix of basis functions of the spline

Parameters **indep_vect** (*np.ndarray*) – A vector of the independent variables over which the basis function should be calculated

Keyword Arguments **spline_end** (*int*) – The number of fixed nodes at the end of the spline

Returns

The n x m matrix of basis functions where the n rows are the response over the independent variable vector to a unit step in the m'th spline coefficient

Return type (np.ndarray)

get_c(*spline_end=None*)
Return the coefficients for the basis functions

Keyword Arguments **spline_end** (*int*) – The number of fixed nodes at the end of the spline

Returns basis function spline coefficients

Return type (numpy.ndarray)

get_t()
Gives the knot locations

Returns knot locations

Return type (numpy.ndarray)

set_c(*c_in*, *spline_end=None*)
Updates the new spline with updated coefficients

Sets the spline coefficeints of this instance to the given values

Parameters **c_in** (*numpy.ndarray*) – The new set of spline coefficeints

Keyword Arguments **spline_end** (*int*) – The number of fixed nodes at the end of the spline

Returns None

3.3 BumpEOS

class F_UNCLE.Models.Isentrope.**EOSBump**(*name='Bump EOS'*, **args*, ***kwargs*)

Model of an ideal isentrope with gausian bumps

This is treated as the *true* EOS

__call__(*vs*)

Solve the EOS

Calculates the pressure for a given volume, replicates the EOS model but uses underlying equation rather than the spline

Parameters **vs** (*float*) – Specific volume

Returns **pr** – Pressure

Return type float

__init__(*name='Bump EOS'*, **args*, ***kwargs*)

Instantiate the bump EOS

Parameters

- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Keyword Arguments **name** (*str*) – Name if the isentrope *Def 'Bump EOS'*

derivative(*n=1*)

Returns the nth order derrivative

Keyword Arguments **n** (*int*) – The order of the derrivative. *Def 1*

Retrun

d1_fun(function): Function object yeilded first derrivative of pressure w.r.t volume

3.4 EOSModel

class F_UNCLE.Models.Isentrope.**EOSModel**(*p_fun*, *name='Equation of State Spline'*, **args*, ***kwargs*)

Spline based EOS model

Multiply inhereited structure from both *Isentrope* and *Spline*

__init__(*p_fun*, *name='Equation of State Spline'*, **args*, ***kwargs*)

Instantiates the object

Parameters

- **p_fun** (*function*) – A function defining the prior EOS
- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Keyword Arguments **name** (*str*) – Name of the isentrope *Def 'Equation of State Spline'*

get_dof(**args*, ***kwargs*)
Returns the spline coefficients as the model degrees of fredom

Returns The degrees of freedom of the model

Return type (np.ndarray)

get_scaling()
Returns a scaling matrix to make the dofs of the same scale

The scaling matrix is a diagonal matrix with off diagonal terms zero the terms along the diagonal are the prior DOFs times the variance in the DOF values.

Returns A nxn matrix where n is the number of model DOFs.

Return type (np.ndarray)

get_sigma()
Returns the covariance matrix of the spline

Returns

Covariance matrix for the eos shape is (nxn) where n is the dof of the model

Return type (np.ndarray)

set_dof(*c_in*, **args*, ***kwargs*)
Sets the spline coeffecients

Parameters **c_in** (*Iterable*) – The knot positions of the spline

update_prior(*prior*, **args*, ***kwargs*)
Updated the prior

Parameters **prior** (EOSModel) – A function which defines the prior EOS shape

CHAPTER

FOUR

EXPERIMENTS

The F_UNCLE project currently does not use any *true* experimental data

4.1 Gun

class F_UNCLE.Experiments.GunModel.**Gun**(*eos*, *name='Gun Toy Computational Experiment'*, **args*, ***kwargs*)

A toy physics model representing a gun type experiment

The problem integrates the differential equation for a mass being accelerated down the barrel of a gun by an the expanding products- of-detonation of a high explosive. The gun has finite dimensions and the integration lasts beyond when the projectile exits the gun.

Units

This model is based on the CGS units system

Diagram

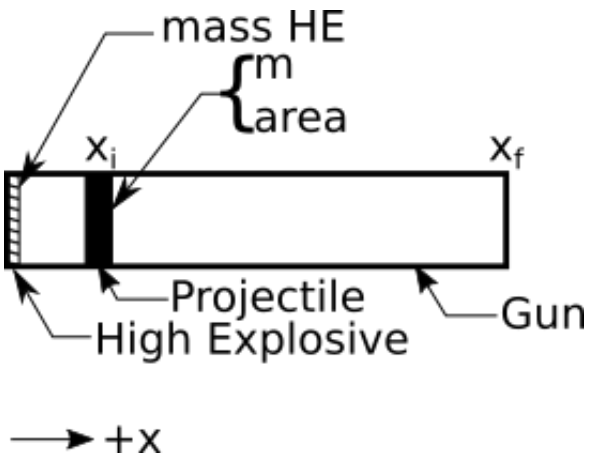

Fig. 4.1: variables defining the gun experiment

eos

Isentrope – A model of the products-of-detonation equation of state

__call__(**args*, ***kwargs*)

Performs the simulation / experiment using the internal EOS

Args:

Returns

Time, the independent variable (tuple): length 2 for the two depdendent variables

[0] (np.ndarray): Velocity history of the simulation [1] (np.ndarray): Position history of the simulation

(Spline): A spline representing the velocity-time history

Return type (np.ndarray)

__init__(*eos*, *name='Gun Toy Computational Experiment'*, **args*, ***kwargs*)
Instantiate the Experiment object

Parameters **eos** (Isentrope) – The equation of state model used in the toy computational experiment

Keyword Arguments **name** (*str*) – A name. (Default = 'Gun Toy Computational Experiment')

_fit_t2v(*vel*, *time*)
Fits a cubic spline to the velocity-time history

This allows simulations and experiments to be compared at the experimental timestamps

Parameters

- **vel** (*np.ndarray*) – Velocity history
- **time** (*np.ndarray*) – Time history

Return (Spline): Spline of vel = f(time)

_get_force(*posn*)
Calculates the force on the prjectile

The force is the pressure of the HE gas acting on the projectile. The pressure is given by the EOS model

Parameters **posn** (*float*) – The scalar position

Retun: (float): The force in dynes

_on_str(**args*, ***kwargs*)
Print method of the gun model

Parameters

- ***args** – Variable length argument list.
- ****kwargs** – Arbitrary keyword arguments.

Returns A string representing the object

Return type (str)

_shoot()
Run a simulation and return the results: t, [x,v]

Solves the ODE

$$F(x, v, t) = \frac{d}{dt}(x, v)$$

Parameters **None** –

Returns

time vector (list): elements are

- [0] -> np.ndarray: position
- [1] -> np.ndarray: velocity

Return type (np.ndarray)

compare(*indep*, *dep*, *model_data*)
Compares a set of experimental data to the model

see *F_UNCLE.Utils.Experiment.Experiment.compare()*

get_sigma()
Returns the covariance matrix

see *F_UNCLE.Utils.Experiment.Experiment.get_sigma()*

plot(*axis=None*, *level=0*, *data=None*, **args*, ***kwargs*)
Plots the gun experiment

shape()
Returns the degrees of freedom of the model

see *F_UNCLE.Utils.Experiment.Experiment.shape()*

update(*model=None*)
Update the analysis with a new model

4.2 Stick

class F_UNCLE.Experiments.Stick.**Stick**(*eos*, *name='Rate Stick Computational Experiment'*, **args*, ***kwargs*)
A toy physics model representing a rate stick

Units

Units are based on CGS system

Diagram

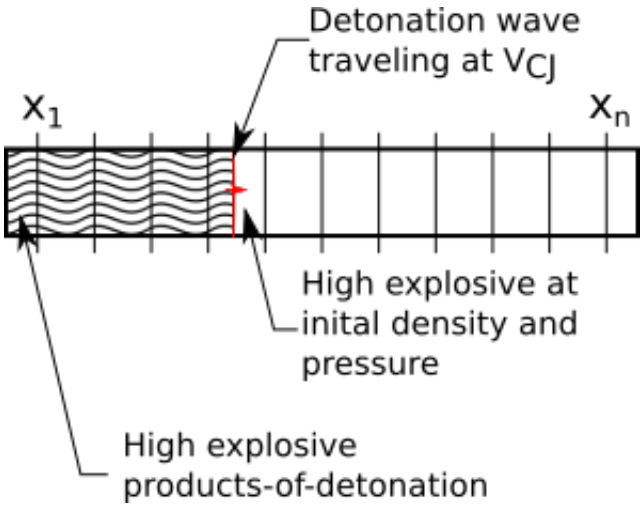

Fig. 4.2: The assumed geometry of the rate stick

const
dict – A dictionary of conversion factors

compare(*indep*, *dep*, *data*)
Compares the model instance to other data

Parameters

- **indep** (*np.ndarray*) – The sensor positions of the other model
- **dep** (*np.ndarray*) – The data from the other model
- **data** (*tuple*) – Summary of data for comparisson

Return:

get_sigma()
Returns the variance matrix

shape()
Returns the shape of the object

update(*model=None*)
Update the analysis with a new model

CHAPTER

FIVE

ANALYSYS

The methods used for the actual optimization

5.1 Bayesian

pyBayesian

An object to extract properties of the bayesian analysis of experiments

5.1.1 Authors

- Stephen Andrews (SA)
- Andrew M. Fraiser (AMF)

5.1.2 Revisions

0 -> Initial class creation (03-16-2016)

class F_UNCLE.Opt.Bayesian.**Bayesian**(*simulations*, *model*, *name='Bayesian'*, **args*, ***kwargs*)

A calss for performing bayesian inference on a model given data

simulations

list – Each element is a tupple with the following elemnts [0] A simulation [1] Experimental results

model

PhysicsModel – The model under consideration

sens_matrix

nump.ndarray – The (nxm) sensitivity matrix n - model degrees of freedom m - total experiment DOF [i,j] - sensitivity of model DOF i

to experiment DOF j

__call__()

Determines the best candidate EOS function for the models

Returns The isentrope which gives best agreement over the space (list): The history of candiate prior DOF's

Return type (Isentrope)

__init__(*simulations*, *model*, *name='Bayesian'*, **args*, ***kwargs*)

Instantiates the Bayesian analysis

Parameters

- **sim_exp** (Experiment) – The simulated experimental data
- **true_exp** (Experiment) – The true experimental data
- **prior** (Struc) – The prior for the physics model

Keyword Arguments **name** (*str*) – Name for the analysis.('Bayesian')

Returns None

_get_constraints(*model*)

EOS MODEL - get the constraints on the model

Parameters **model** (PhysicsModel) – The physics model subject to physical constraints

Returns ():

Return type ()

Method

Calculate constraint matrix G and vector h. The constraint enforced by *cvxopt.solvers.qp* is

$$G * x \leq_c omponent_wiseh$$

Equivalent to $max(G * x - h) \leq 0$

Since

$$c_{f_{new}} = c_f + x,$$

$$G(c_f + x) \leq_c omponent_wise0$$

is the same as

$$G * x \leq_c omponent_wise - G * c_f,$$

and $h = -G * c_f$

Here are the constraints for $p(v)$:

p" positive for all v p' negative for v_max p positive for v_max

For cubic splines between knots, f" is constant and f' is affine. Consequently, f''*rho + 2*f' is affine between knots and it is sufficient to check eq:star at the knots.

_get_model_PQ(*model*)

Gets the quadratic optimizaiton matrix contributions from the prior

Parameters **model** (PhysicsModel) – A physics model with degrees of freedom

Retrun: (np.ndarray): *P*, a nxn matrix where n is the model DOF (np.ndarray): *q*, a nx1 matrix where n is the model DOF

_get_sens(*sims*, *model*, *initial_data*)

Gets the sensitivity of the simulated experiment to the EOS

Parameters **initial_data** (*list*) – The results of each simulation with the curent best model

_get_sim_PQ(*sims*, *model*, *initial_data*)

Gets the QP contribytions from the model

Parameters

- **sims** (*list*) – A list of tuples of experiments each tuple contains [0] the simulation [1] the corresponding experiment
- **model** (PhysicsModel) – A physics model with degrees of freedom
- **initial_data** (*list*) – A list of the inital results from the simulations in the same order as in the *sim* list

Retrun: (np.ndarray): *P*, a nxn matrix where n is the model DOF (np.ndarray): *q*, a nx1 matrix where n is the model DOF

_local_opt(*sims*, *model*, *initial_data*)

_on_str()
Print method for bayesian model

fisher_decomposition(*fisher*, *tol=0.001*)

Parameters **fisher** (*np.ndarray*) – A nxn array where n is model dof

Keyword Arguments **tol** (*float*) – Eigen values less than tol are ignored

Returns

Eigenvalues greater than tol (np.ndarray): nxm array.

n is number of eigenvalues greater than tol m is model dof

(np.ndarray): nxm array: n is the number of eigenvalues greater than tol m is an arbutary dimension of independent variable

(np.ndarray): vector of independent varible

Return type (list)

get_fisher_matrix(*simid=0*, *sens_calc=True*)
Returns the fisher information matrix of the simulation

Keyword Arguments

- **simid** (*int*) – The index of the simulation to be investigated *Default 0*
- **sens_calc** (*bool*) – Flag to recalcualte sensitivities *Default True*

Returns

The fisher information matrix, a nxn matrix where *n* is the degrees of freedom of the model.

Return type (np.ndarray)

model_log_like()
Gets the log likelyhood of the model given the prior

Parameters **None** –

Returns Log likelyhood of the model

Return type (float)

$$\log(p(f|y))_{model} = -\frac{1}{2}(f-\mu_f)\Sigma_f^{-1}(f-\mu_f)$$

plot_convergence(*hist*, *dof_hist*)

plot_fisher_data(*fisher_data*, *filename=None*)

Parameters **fisher_dat** (*tuple*) – Data from the fisher_decomposition function *see docscring for definition*

Keyword Arguments **filename** (*str or None*) – If none, do not make a hardcopy, otherwise save to the file specified

plot_sens_matrix(*initial_data*)
Prints the sensitivity matrix

shape()
Gets the dimenstions of the problem

Returns The n x m dimensions of the problem

Return type (tuple)

sim_log_like(*initial_data*)
Gets the log likelyhood of the simulations given the data

Parameters **initial_data** (*list*) – A list of the initial data for the simulations

Returns Log likelyhood of the prior

Return type (float)

$$\log(p(f|y))_{model} = -\frac{1}{2}(y_k - \mu_k(f))\Sigma_k^{-1}(y_k - \mu_k(f))$$

update(*simulations=None*, *model=None*)
Updates the properties of the bayesian analtsis

Keyword Arguments

- **simulations** (*Experiment*) – The tupples of simulations and experiments (Default None)
- **model** (*PhysicsModel*) – The physics model used in the simulaitons (Default None)

Returns None

class F_UNCLE.Opt.Bayesian.**TestBayesian**(*methodName='runTest'*)
Test class for the bayesian object

setUp()
Setup script for each test

test_bad_instantiaion()
Tets impropper instantiation raises the correct errors

test_fisher_matrix()
Tests if the fisher information matrix can be generated correctly

test_gun_case_sens()

test_instantiation()
Test that the object can instantiate correctly

test_mlt_case_PQ_mod()
Tests the P and Q matrix generation for a multiple case

test_model_pq()
Tests the pq matrix generation by the model

test_mult_case_sens()
Test of sens matrix generation for mult models

test_singel_case_PQ_mod()
Tests the P and Q matrix generation for a single case

test_stick_case_sens()
Test of the sensitivity of the stick model to the eos

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

f

Symbols

A

B

C

D

U

W