F_UNCLE Documentation

Release 0.1

A. Fraser and S. Andrews

June 23, 2016

Contents

Contents:

CHAPTER 1

Documentation

The FUNCLE module

Functional UNcertainty Constrained by Law and Experiment

1.1 Authors

- Andrew Fraser (AF)
- Stephen A Andrews (SA)

1.2 Revision History

0.0 - Initial class creation (09-10-2015)

1.3 License

CHAPTER 2

Models

The pysics models used in the analysys

2.1 Isentrope

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

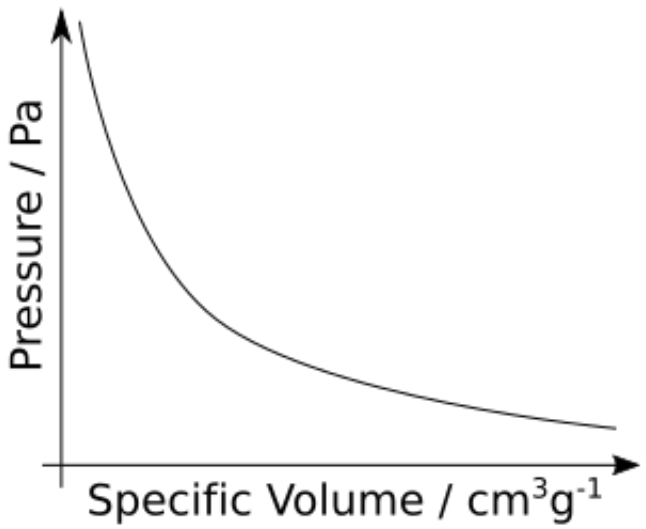

Fig. 2.1: The assumed shape of the equation of state isentrope

Options

Name	Type	Def	Min	Max	Units	Description
spline_N	(int)	50	7	None	''	Number of knots in the EOS spline
spline_min	(float)	0.1	0.0	None	'cm**3/g'	Minimum value of volume modeled by EOS
spline_max	(float)	1.0	0.0	None	'cm**3/g'	Maximum value of volume modeled by EOS
spline_end	(float)	4	0	None	''	Number of zero nodes at end of spline

__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'*

Returns None

plot(*axis=None*, *hardcopy=None*, *style='-k'*, **args*, ***kwargs*)

Plots the EOS

Overloads the *F_UNCLE.Utils.Struc.Struc.plot()* method to plot the eos over the range of volumes.

Parameters

- **axis** (*plt.Axes*) – The axis on which to plot the figure, if None, creates a new figure object on which to plot.
- **hard-copy** (*str*) – If a string, write the figure to the file specified
- **style** (*str*) – A `plt.Axis.plot()` format string for the eos

Returns A reference to the figure containing the plot

Return type (plt.Figure)

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)

2.2 BumpEOS

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

Model of an ideal isentrope with Gaussian bumps

This is treated as the *true* EOS

Options

Name	Type	Def	Min	Max	Units	Description
const_C	(float)	2.56e9	0.0	None	'Pa'	'Constant p = C/v**3'
bumps	(list)	[(0.4 0.1 0.25) (0.5 0.1 -0.3)]	None	None	''	'Gaussian bumps to the EOS'

__call__(*vol*)
Solve the EOS

Calculates the pressure for a given volume, is called the same way as the EOS model but uses underlying equation rather than the spline

Parameters **vol** (*np.ndarray*) – Specific volume

Returns **pr** – Pressure

Return type np.ndarray

__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(*order=1*)
Returns the nth order derivative

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

Returns **d1_fun** – Function object yielded first derivative of pressure w.r.t volume

Return type function

2.3 EOSModel

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

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

Options

Name	Type	Def	Min	Max	Units	Description
Spline_sigma	float	5e-3	0.0	None	'??'	'Multiplicative uncertainty of the prior (1/2%)
precondition	bool	False	None	None	''	Precondition flag

__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'*

_on_str()
Addeed information on the EOS

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

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 co-variance matrix of the spline

Returns Co-variance 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 coefficients

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 3

Experiments

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

3.1 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

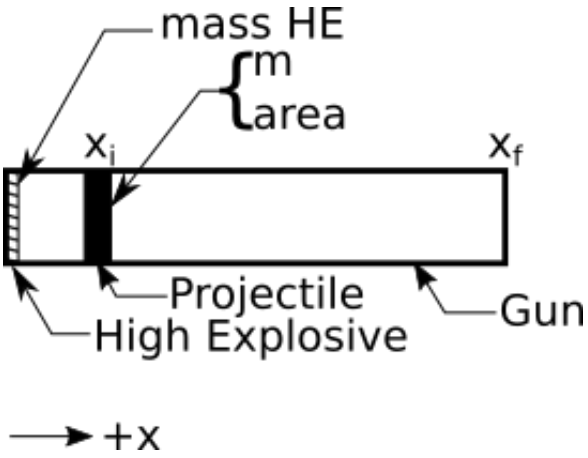

Fig. 3.1: variables defining the gun experiment

Options

Name	Type	Def	Min	Max	Units	Description
X_i	(float)	0.4	0.0	None	cm	Initial position of projectile
x_f	(float)	3.0	0.0	None	cm	Final/muzzle position of projectile
m	(float)	500.0	0.0	None	g	Mass of projectile
mass_he	(float)	4	0.0	None	g	The initial mass of high explosives used to drive the projectile
area	(float)	1.0	0.0	None	cm**2	Projectile cross section
sigma	(float)	1.0e0	0.0	None	??	Variance attributed to v measurements
t_min	(float)	1.0e-6	0.0	None	sec	Range of times for t2v spline
t_max	(float)	1.0e-2	0.0	None	sec	Range of times for t2v spline
n_t	(int)	250	0	None	''	Number of times for t2v spline

Attributes

eos
Isentrope

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

Methods

__call__(**args*, ***kwargs*)
Performs the simulation / experiment using the internal EOS

Parameters

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

Returns

Length 3, elements are:

0. (np.ndarray): Time, the independent variable
1. (tuple): length 2 for the two dependent variables
   (a) (np.ndarray): Velocity history of the simulation
   (b) (np.ndarray): Position history of the simulation
0. (Spline): A spline representing the velocity-time history

Return type (tuple)

__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 time-stamps

Parameters

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

Returns Spline of vel = f(time)

Return type (Spline)

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

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

Returns The force in dyes

Return type (float)

_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

Length 2 elements are:

0. (np.ndarray): time vector
1. (list): elements are:
   (a) (np.ndarray): position
   (b) (np.ndarray): velocity

Return type (tuple)

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

Error is *dep* less the *model_data*

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

get_sigma()
Returns the co-variance matrix

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

plot(*axis=None*, *hardcopy=None*, *level=0*, *data=None*, *style='-k'*, *err_style='-r'*, **args*, ***kwargs*)
Plots the gun experiment

Overloads *F_UNCLE.Utils.Struc.Struc.plot()*

Plot Levels

0.The velocity-time history

1.The position-time history

2.The velocity-position history

3.A 4 subplot figure with levels 1-3 as well as the EOS plotted

Parameters

- **axis** (*plt.Axis*) – Axis object on which to plot, if None, creates new figure
- **hardcopy** (*str*) – A writable location to save the file
- **level** (*int*) – A tag for which kind of plot should be generated
- **data** (*dict*) – A list of other data to be plotted for comparison

Returns A figure

Return type (plt.figure)

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 EOS model

Parameters **model** (Isentrope) – A new EOS model

Returns None

3.2 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

Attributes

eos

Isentrope

The products-of-detonation equation of state

Methods

__call__()

Performs the rate stick experiment

Returns

Length 3. Elements are

0. (np.ndarray): The independent variable, the *n* sensor positions

1. (tuple): The dependent variables, elements are:

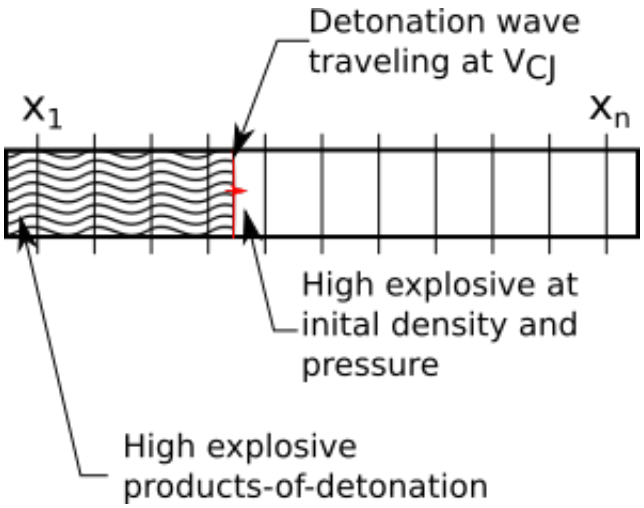

Fig. 3.2: The assumed geometry of the rate stick

(a) (None)

(b) (np.ndarray): The arrival *n* times at each sensor

0. (tuple): The other solution data

(a) the detonation velocity

(b) the specific volume at the CJ point

(c) the pressure at the CJ point

(d) a Rayleigh line function, see below

Return type (tuple)

Rayleigh Line Function

p = ray(v, vel, vol0, eos)

Args:

•v(np.ndarray): The specific volume

•vel(float): Detonation velocity

•vol_0(float): Specific volume ahead of the shock

•eos(Isentrope): An equation of state model

Return:

•p(float): The pressure along the Rayleigh line at v

__init__ (*eos*, *name='Rate Stick 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')

_get_cj_point (*eos*, *vol_0*)

Find CJ conditions using two nested line searches.

The CJ point is the location on the EOS where a Rayleigh line originating at the pre-detonation volume and pressure is tangent to the equation of state isentrope.

This method uses two nested line searches implemented by the `scipy.optimize.brentq()` algorithm to locate the velocity corresponding to this tangent Rayleigh line

Parameters

- **eos** (*Isentrope*) – The products of detonation equation of state
- **vol_0** (*float*) – The specific volume of the equation of state before the shock arrives

Returns

Length 3 elements are:

0. (float): The detonation velocity
1. (float): The specific volume at the CJ point
2. (float): The pressure at the CJ point
3. (function): A function defining the Rayleigh line which passes through the CJ point

Return type (tuple)

_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)

compare(*indep*, *dep*, *data*)

Compares the model instance to other data

The error is the difference in arrival times, dep less data.

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

get_sigma()

Returns the variance matrix

variance is

$$\Sigma_i = \sigma_t^2 + \frac{\sigma_x^2}{V_{CJ}}$$

Where

- σ_t is the error in time measurements
- σ_x is the error in sensor position
- V_{CJ} is the detonation velocity

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

plot(*axis=None*, *hardcopy=None*, *level=1*, *data=None*, *eos_style='-k'*, *ray_style=':k'*, *cj_style='ok'*, *data_style='-k'*)

Plots the EOS and Rayleigh line Plots the critical Rayleigh line corresponding to the detonation velocity tangent to the EOS.

Parameters

- **level** (*int*) – Specified what to plot 1. Plots the EOS with the Raylight line intersecting the CJ point 2. Plots the output from a simulation
- **eos_style** (*str*) – `Axis.plot()` format string for eos trend
- **ray_style** (*str*) – `Axis.plot()` format string for Rayleigh trend
- **cj_style** (*str*) – `Axis.plot()` format string for CJ point
- **data_style** (*str*) – `Axis.plot()` format string for data point
- **data** (*list*) – The output from a call to Stick

see *F_UNCLE.Utils.Struc.Struc.plot()*

shape()
Returns the shape of the object

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

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

Parameters **model** (`Isentrope`) – The EOS model

Returns None

3.3 Cylinder

Note: Cylinder has not been implemented yet

CHAPTER 4

Analysys

The methods used for the actual optimization

4.1 Bayesian

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

A calss for performing bayesian inference on a model given data

Attributes

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 response i to experiment DOF j

Options

Name	Type	Def	Min	Max	Units	Description
outer_atol	(float)	1E-6	0.0	1.0	•	Absolute tolerance on change in likelyhood for outer loop convergence
outer_rtol	(float)	1E-4	0.0	1.0	•	Relative tolerance on change in likelyhood for outer loop convergence
maxiter	(int)	6	1	100	•	Maximum iterations for convergence of the likelyhood
constrain	(bool)	True	None	None	•	Flag to constrain the optimization
precondition	(bool)	True	None	None	•	Flag to scale the problem
prior_weight	(float)	1.0	0.0	0.0	•	Weight of the prior when calculating the log likelihood
debug	(bool)	False	None	None	•	Flag to print debug information
verb	(bool)	True	None	None	•	Flag to print stats during optimization

Note: The options *outer_atol* and *prior_weight* are depricated and should be used for debugging purposes only

Methods

__call__()

Determines the best candidate EOS function for the models

Returns

length 2, elements are:

0. (PhysicsModel): The model which gives best agreement over the space
1. (list): is of solution history elements are:
   (a) (np.ndarray) Log likelyhood, (nx1) where n is number of iterations
   (b) (np.ndarray) model dof history (nxm) where n is iterations and m is the model dofs
   (c) (np.ndarray) model step history (nxm) where n is iterations and m is the model dofs

Return type (tuple)

__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*)
Get the constraints on the model

Note: This method is specific to the model type under consideration. This implementation is onlt for spline models of EOS

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

$$G * x \leq h$$

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

Since

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

$$G(c_f + x) \leq 0$$

is the same as

$$G * x \leq -G * c_f,$$

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, $f''rho + 2f'$ 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: (tuple): elements are

0. (np.ndarray): *p*, a nxn matrix where n is the model DOF

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

_get_sens(*sims*, *model*, *initial_data=None*)
Gets the sensitivity of the simulated experiment to the EOS

The sensitivity matrix is the attribute *self.sens_matrix* which is set by this method

Note: This method is specific to the model type under consideration. This implementation is onlt for spline models of EOS

Parameters

- **sims** (*list*) – List of tuples of simulation, experiment pairs
- **model** (PhysicsModel) – A valid physics model instance

Keyword Arguments **initial_data** (*list*) – The results of each simulation with the current best model. Each element in the list corresponds tho the output from a *__call__* to each element in the *self.simulations* list

Returns None

_get_sim_pq(*sims*, *model*, *initial_data*)
Gets the QP contribytions from the model

Note: This method is specific to the model type under consideration. This implementation is onlt for spline models of EOS

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

Returns

Elements are:

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

Return type (tuple)

_local_opt(*sims*, *model*, *initial_data*)
Soves the quadratic problem for minimization of the log likelyhood

Parameters

- **sims** (*list*) – The simulation/experiment pairs
- **model** (PhysicsModel) – The model being examined
- **initial_data** (*list*) – The initial data corresonding to simulations from sims

Returns The step direction for greates improvement in log lieklyhood

Return type (np.ndarray)

_on_str()
Print method for bayesian model

Parameters **None** –

Returns String describing the Bayesian object

Return type (str)

compare(*sims*, *model*)

Parameters

- **sims** (*list*) – List of tuples of simulation, experiment pairs
- **model** (PhysicsModel) – A valid physics model instance

Returns

List of lists for experiment comparison data

0. independent value
1. dependent value of interest

Return type (list)

fisher_decomposition(*fisher*, *tol=0.001*)
Performs a spectral decomposition on the fisher information matrix

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

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

Returns

Elements are:

0. (list): Eigenvalues greater than tol
1. (np.ndarray): nxm array.
   - n is number of eigenvalues greater than tol
   - m is model dof
2. (np.ndarray): nxm array
   - n is the number of eigenvalues greater than tol
   - m is an arbutary dimension of independent variable
3. (np.ndarray): vector of independent varible

Return type (tuple)

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 *self.model* given that model's prior

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=None*, *axis=None*, *hardcopy=None*)

Parameters

- **axis** (*plt.Axis*) – A valid `plt.Axis` object on which to plot. if none, generates a new figure
- **hist** (*list*) – Convergence history of log likelyhood
- **dof_hist** (*list*) – List of model DOFs at each iteration

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, m) dimensions of the problem

- n is the total degrees of freedom of all the model responses
- m is the degrees of freedom of the model

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 Each element in the list is the output from a __call__ to the corresponding element in the *self.simulations* list

Returns Log likelyhood of the simulation given the data

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

CHAPTER 5

Utilities

These are abstract classes which are used in the analysis

5.1 Struc

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

Abstract object to contain properties and warnings

Attributes

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

Methods

__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 children

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.
- **hard-copy** (*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 writable location

Returns None

5.2 PhysicsModel

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

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

Definitions

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 estimate 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

Attributes

prior
PhysicsModel

the prior

Methods

__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

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

Returns None

get_dof ()
Returns the model degrees of freedom

Note: Abstract Method: Must be overloaded to work with *F_UNCLE*

Parameters **None** –

Returns The model degrees of freedom

Return type (np.ndarray)

get_scaling ()
Returns a matrix to scale the model degrees of freedom

Note: Abstract Method: Must be overloaded to work with *F_UNCLE*

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

Returns a n x n matrix of scaling factors to make all dofs of the same order of magnitude.

Return type (np.ndarray)

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

Note: Abstract Method: Must be overloaded to work with *F_UNCLE*

Parameters

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

Returns A n x n diagonal matrix of the uncertainty in each dof. where n is the model degrees of freedom

Return type (np.ndarray)

set_dof(*dof_in*)
Sets the model degrees of freedom

Note: Abstract Method: Must be overloaded to work with *F_UNCLE*

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

Returns None

shape()
Returns the shape of the model dof space

Note: Abstract Method: Must be overloaded to work with *F_UNCLE*

Returns Dimensions

Return type (tuple)

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

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

5.3 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

Definitions

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

__call__ (**args*, ***kwargs*)
Runs the simulation.

Note: Abstract Method: Must be overloaded to work with *F_UNCLE*

The simulation should be structured so that the attributes and options of simulation should provide all the needed initial conditions are instantiated before calling the object.

Parameters

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

Returns

length 3 tuple with components

0. (np.ndarray): The single vector of the simulation independent variable
1. (list): A list of np.ndarray objects representing the various dependent variables for the problem. Element zero is the most important quantity. By default, comparisons to other data-sets are made to element zero. The length of each element of this list must be equal to the length of the independent variable vector.
2. (list): A list of other attributes of the simulation result. The composition of this list is problem dependent

Return type (tuple)

__init__ (*name='Experiment'*, **args*, ***kwargs*)
Instantiates the object.

Options can be set by passing them as keyword arguments

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

Note: Abstract Method: Must be overloaded to work with *F_UNCLE*

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 The error between the dependent variables and the model for each value of independent variable

Return type (np.ndarray)

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

Note: Abstract Method: Must be overloaded to work with *F_UNCLE*

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 `PhysicsModel.shape()`

Return type (np.ndarray)

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

Note: Abstract Method: Must be overloaded to work with *F_UNCLE*

Returns The number of independent variables in the experiment

Return type (int)

5.4 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 (np.ndarray)

`get_t`()
Gives the knot locations

Returns knot locations

Return type (np.ndarray)

set_c(*c_in*, *spline_end=None*)

Updates the new spline with updated coefficients

Sets the spline coefficients of this instance to the given values

Parameters **c_in** (*np.ndarray*) – The new set of spline coefficients

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

Returns None

CHAPTER 6

SciPy 2016

Script for re-generating the figures used in scipy2016

This script generates the figures used in ref 1

Dependencies

- python 2.7
- numpy
- matplotlib
- scipy
- cvxopt

Useage

from the command line:

```
$ python scipy2016.py
```

References

1. Fraser, A. M. and Andrews, S. A. 2016 “Functional Uncertanty Constrained by Law and Experiment.” Proceedings of the 15th Python in Science Conference LA-UR-23717

6.1 Figures

A trivial change to test git hub

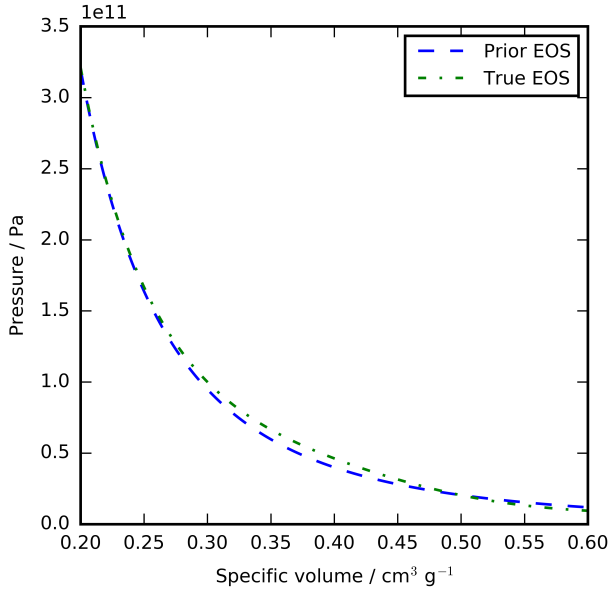

Fig. 6.1: Figure 0

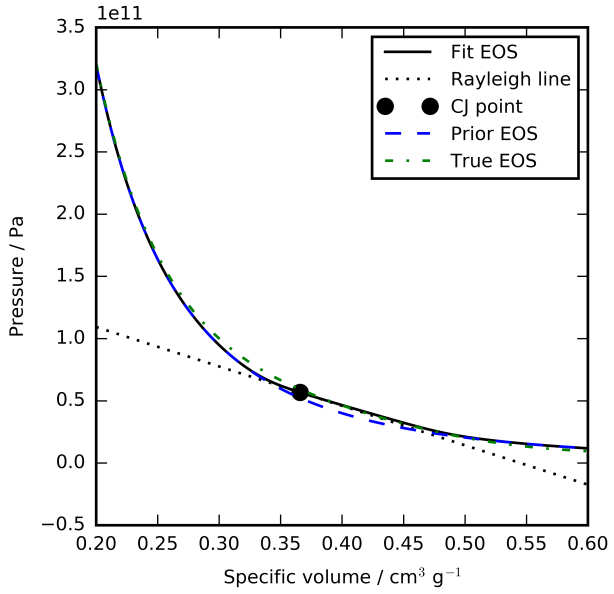

Fig. 6.2: Figure 1

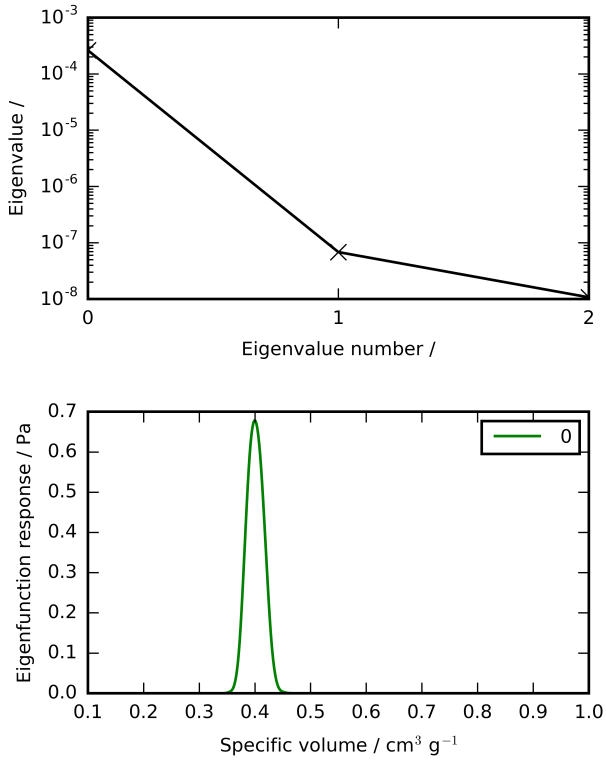

Fig. 6.3: Figure 2

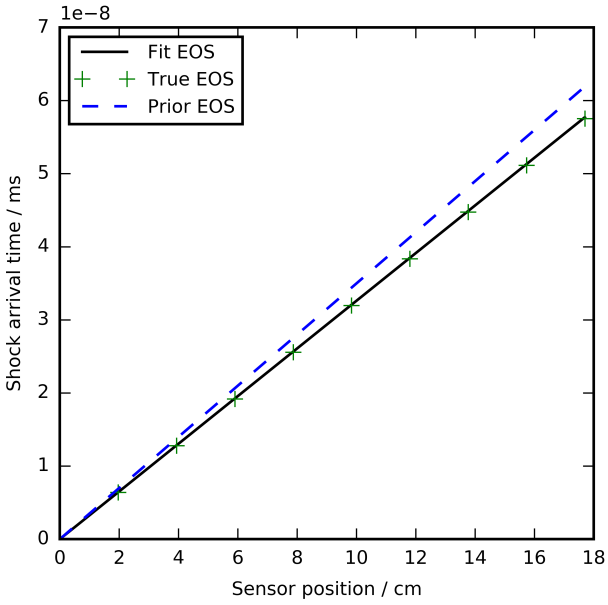

Fig. 6.4: Figure 3

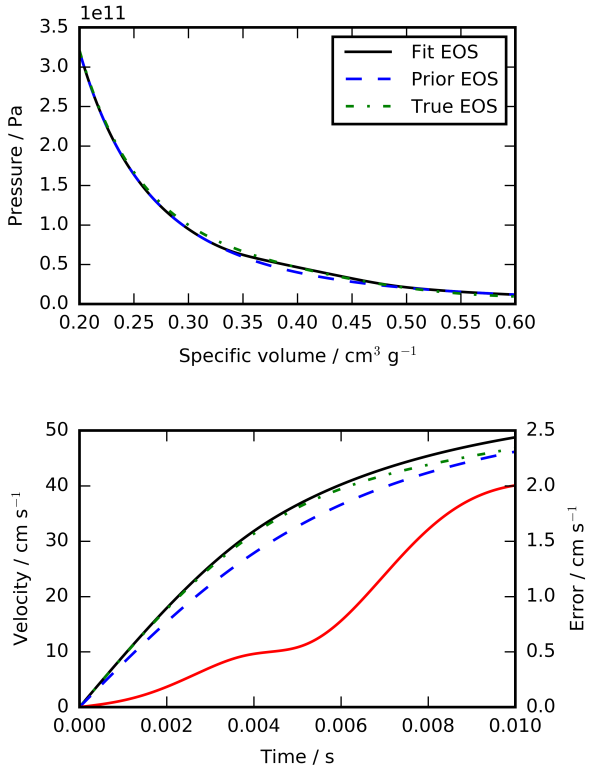

Fig. 6.5: Figure 4

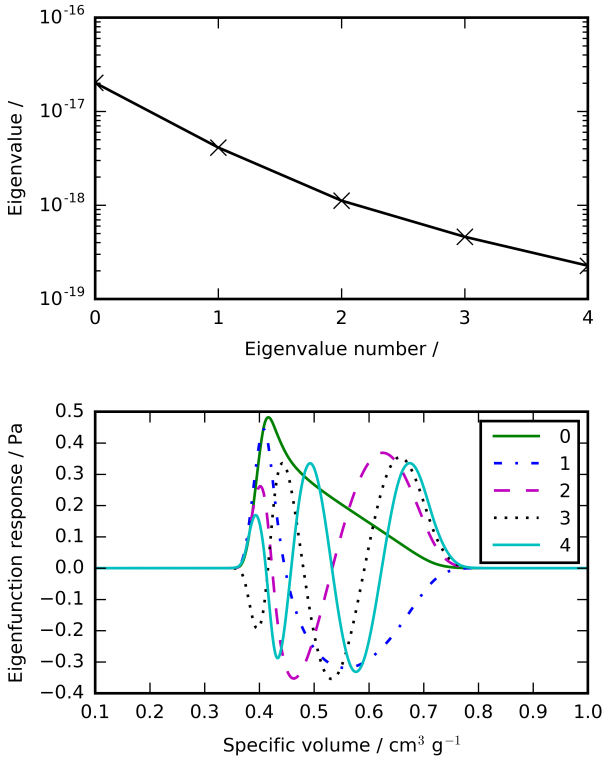

Fig. 6.6: Figure 5

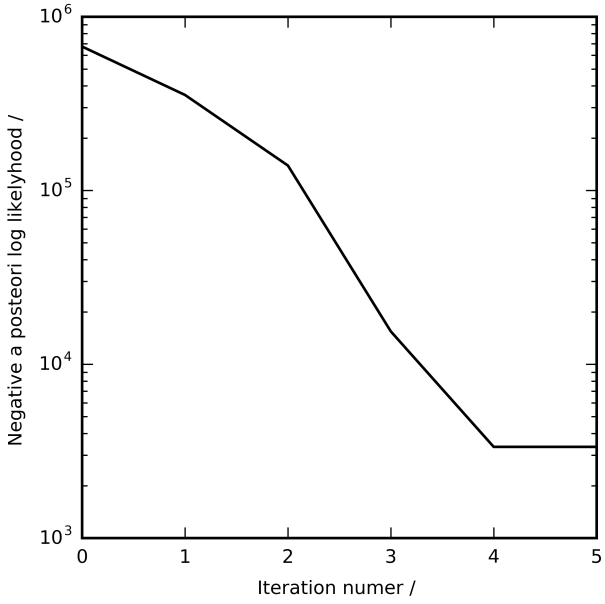

Fig. 6.7: Figure 6

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

f

Symbols

B

C

D

E

W