FEAST

Release 3.1

Chandler Kemp and Clay Bell

Nov 30, 2020

CONTENTS:

CHAPTER

ONE

FEAST MODULES

1.1 DetectionModules

1.1.1 abstract_detection_method

This module contains an abstract class that all DetectionMethods should inherit.

1.1.1.1 DetectionMethod

`class DetectionMethod`(*time*, *detection_variables=None*, *op_envelope=None*, *ophrs=None*)

DetectionMethod is an abstract super class that defines the form required for all detection methods

Parameters

- **`time`** – a Time object
- **`detection_variables`** – list of variable names to be used in the detection calculations (eg. ['wind speed'])
- **`op_envelope`** – operating envelope specifications for the detection method
- **`ophrs`** – a dict specifying operating hours for the DetectionMethod

1.1.2 comp_survey

This module defines the component level survey based detection class, CompSurvey.

1.1.2.1 CompSurvey

`class CompSurvey`(*time*, *dispatch_object*, *survey_interval*, *survey_speed*, *labor*, *site_queue*, *detection_probability_points*, *detection_probabilities*, *ophrs*, *comp_survey_index=0*, *site_survey_index=0*, *op_env_wait_time=7*, ***kwargs*)

This class specifies a component level, survey based detection method. A component level method identifies the specific component that is the source of the emission at the time of detection. Examples of components include connectors, valves, open ended lines, etc. A survey based method inspects emissions at specific moments in time (as opposed to a monitor method that continuously monitors for emissions). The class has three essential attributes:

1. An operating envelope function to determine if conditions satisfy requirements for the method to be deployed
2. A probability of detection surface function to determine which emissions are detected

3. The ability to call a follow up action

Parameters

- **`time`** – a Time object
- **`dispatch_object`** – an object to dispatch for follow-up actions (typically a Repair method)
- **`survey_interval`** – Time between surveys (float–days)
- **`survey_speed`** – Speed of surveys (float–components/hr)
- **`labor`** – Cost of surveys (float–$/hr)
- **`site_queue`** – Sites to survey (list of ints)
- **`detection_probability_points`** – Set of conditions at which the probability was measured (array)
- **`detection_probabilities`** – Set of probabilities that were measured (array)
- **`comp_survey_index`** – Index of the component to be surveyed next (int)
- **`site_survey_index`** – Index of the site to be surveyed next (or currently under survey) (int)
- **`op_env_wait_time`** – Time to wait if operating envelope conditions fail part way through a site before moving to the next site (float–days)
- **`ophrs`** – range of times of day when the method performs survey (dict–hours). eg: {'begin': 8, 'end': 17}

1.1.3 ldar_program

This module defines the LDARProgram class.

1.1.3.1 LDARProgram

`class LDARProgram`(*time*, *gas_field*, *tech_dict*)

An LDAR program contains one or more detection methods and one or more repair methods. Each LDAR program records the find and repair costs associated with all detection and repair methods in the program. The LDAR program deploys runs the action methods of each detection and repair method contained in the program. The detection and repair methods determine their own behavior at each time step.

Parameters

- **`time`** – a Time object
- **`gas_field`** – a GasField object
- **`tech_dict`** – a dict containing all of the detection methods to be employed by the LDAR program. The dict must have the form {"name": DetectionMethod}. All of the relationships between detection methods and between detection methods and repair methods must be defined by the dispatch_objects specified for each method.

1.1.4 repair

This module defines the Repair class. Repair may be called by detection objects as follow up actions.

1.1.4.1 Repair

class Repair(*repair_delay=0*, *name=None*)

Defines a repair process. A repair process determines when emissions are ended by an LDAR program and the associated costs.

Parameters **repair_delay** – The time between when an emission is passed to Repair and when it is removed from the simulation (float–days)

1.1.5 site_monitor

The site_monitor module defines the site monitor detection class, SiteMonitor.

1.1.5.1 SiteMonitor

class SiteMonitor(*time*, *dispatch_object*, *time_to_detect_points*, *time_to_detect_days*, *ophrs=None*, *capital=0*, *site_queue=None*, ***kwargs*)

This class specifies a site level continuous monitoring method. A site monitor continuously observes emissions from an entire site and determines when an action should be dispatched at the site. The method has three essential characteristics:

1. A list of the sites where the method applies
2. A time-to-detect surface specified as a list of conditions and associated mean detection times
3. The ability to dispatch a follow up action

Parameters

- **time** – a Time object
- **dispatch_object** – the object to dispatch for follow up actions
- **time_to_detect_points** – The conditions at which the time to detection was measured. (NxM array, where N is the number of distinct conditions and M is the number of variables (up to two)).
- **time_to_detect_days** – The list of probabilities of detection associated with every point in detection_probability_points (array of shape N, where N is the number of conditions with an associated probability of detection).
- **ophrs** – The times of day when the SiteMonitor is operational. Should be a dict:

  {'begin': hour integer, 'end': hour integer}
- **capital** – The total cost of installing the site monitor system in the simulation (float–$)
- **site_queue** – A list of sites where the site monitor system is installed

1.1.6 site_survey

The site_survey module defines the site level level survey based detection class, SiteSurvey.

1.1.6.1 SiteSurvey

class SiteSurvey(*time*, *dispatch_object*, *sites_per_day*, *site_cost*, *detection_probability_points*, *detection_probabilities*, *op_envelope=None*, *ophrs=None*, *site_queue=None*, *survey_interval=None*, ***kwargs*)

SiteSurvey specifies a site level, survey based detection method. A site level detection method is sensitive to the total emissions from a site. If emissions are detected, the site is identified as the source of emissions rather than a component on the site. Survey based detection methods search for emissions at a specific moment in time (as opposed to monitor detection methods that continuously scan sites for new emissions). The class has three essential attributes:

1. An operating envelope function to determine if the detection method can be applied
2. A probability of detection surface function to determine which emissions are detected
3. The ability to dispatch a follow up action

Parameters

- **time** – a Time object
- **dispatch_object** – the object that SiteSurvey will pass flagged site indexes to (DetectionMethod or Repair)
- **sites_per_day** – the number of sites that the method can survey in one day (int)
- **site_cost** – the cost per site of the detection method ($/site–float)
- **detection_probability_points** – The conditions at which the detection probability was measured. (NxM array, where N is the number of distinct conditions and M is the number of variables (up to two)).
- **detection_probabilities** – The list of probabilities of detection associated with every point in detection_probability_points (array of shape N, where N is the number of conditions with an associated probability of detection).
- **op_envelope** – The set of conditions underwhich the SiteSurvey may operate. The op_envelope must be passed as a dict with the following form–

  {'parameter name': {'class': int, 'min': list of minimum conditions, 'max': list of maximum conditions}}

  Unique minima can be defined for every site in a list if the op_envelope 'class' is site specific. Multiple minima can be defined in a list for a single site if multiple ranges should be considered.
- **ophrs** – The times of day when the SiteSurvey can be deployed. Should be a dict:

  {'begin': hour integer, 'end': hour integer}
- **site_queue** – an ordered list of sites to be surveyed. An LDAR program may update this list.
- **survey_interval** – The time between surveys (int–days)

1.2 EmissionSimModules

1.2.1 emission_class_functions

A class for storing emission properties and functions for modifying emission proporeties throughout a simulation are defined in this module.

1.2.1.1 Emission

class Emission (*flux=()*, *reparable=True*, *site_index=()*, *comp_index=()*, *start_time=0*, *end_time=inf*, *repair_cost=()*, *emission_id=None*)

Stores all properties of all emissions that exist at a particular instant in a simulation.

Parameters

- **flux** – An array of emission rates (array of floats–gram/second)
- **reparable** – An array of True/False values to indicate whether or not an emission is reparable
- **site_index** – An array indicating the index of the site that contains every emission
- **comp_index** – An array indicating the index of the component that is the source of each emission
- **start_time** – An array specifying the time when every emission begins
- **end_time** – An array specifying the time when every emission will end (days)
- **emission_id** –
- **repair_cost** – An array storing the cost of repairing every emission ($)

1.2.1.2 bootstrap_emission_maker

bootstrap_emission_maker (*n_em_in*, *comp_name*, *site*, *time*, *start_time=None*, *reparable=True*)

Create leaks using a bootstrap method.

Parameters

- **n_em_in** – number of leaks to generate
- **comp_name** – key to a Component object in site.comp_dict
- **site** – a Site object
- **time** – a Time object
- **start_time** – the times at which each emission begins
- **reparable** – Specifies whether emissions should be reparable or not (boolean)

1.2.1.3 comp_indexes_fcn

`comp_indexes_fcn` (*site*, *comp_name*, *n_inds*)

Returns an array of indexes to associate with new emissions

Parameters

- **`site`** – a EmissionSimModules.simulation_classes.Site object
- **`comp_name`** – name of a component contained in Site.comp_dict
- **`n_inds`** – Integer of indexes to generate

Returns An array of indexes in the range specified for the relevant component

1.2.1.4 emission_objects_generator

`emission_objects_generator` (*dist_type*, *emission_data_path*, *custom_emission_maker=None*)

emission_objects_generator is a parent function that will be called to initialize gas fields

Parameters

- **`dist_type`** – Type of leak distribution to be used
- **`leak_data_path`** – Path to a leak data file

1.2.1.5 permitted_emission

`permitted_emission` (*n_emit*, *sizes*, *duration*, *time*, *site*, *comp_name*, *start_time*)

Creates an emission object specifying new permitted emissions

Parameters

- **`n_emit`** – number of emissions to create
- **`sizes`** – a list of leak sizes from which to specify the emission rate
- **`duration`** – a float defining the duration of the emission
- **`time`** – a Time object
- **`site`** – a Site object
- **`comp_name`** – Name of the component to be considered from within site.comp_dict
- **`start_times`** – array of times at which emissions start

Returns an Emission object

1.2.2 infrastructure_classes

This module stores component, gasfield and site classes to represent infrastructure in a simulation

1.2.2.1 Component

class Component(*repair_cost_path=None, emission_data_path=None, base_reparable=None, custom_emission_maker=None, emission_production_rate=0, emission_per_comp=None, episodic_emission_sizes=[0], episodic_emission_per_day=0, episodic_emission_duration=0, vent_sizes=[0], vent_period=inf, vent_starts=array([], dtype=float64), vent_duration=0, name='default', null_repair_rate=None, dist_type='bootstrap'*)

A class to store parameters defining a component (for example, name, leak production rate, leak size distribution, etc)

Parameters

- **repair_cost_path** – path to a repair cost data file
- **emission_data_path** – path to an emission data file
- **base_reparable** – Defines whether emissions generated are reparable with a boolean true/false
- **custom_emission_maker** – Optional custom defined function for creating new emissions
- **emission_production_rate** – The rate at which new emissions are created (emissions per day per component)
- **emission_per_comp** – The number of emissions expected per component (must be less than 1) If emission_per_comp is left as None, then emission_per_comp is set equal to the emissions per component recorded in the file at emission_data_path.
- **episodic_emission_sizes** – A list of emission sizes to draw from for episodic emissions (g/s)
- **episodic_emission_per_day** – The average frequency at which episodic emissions occur (1/days)
- **episodic_emission_duration** – The duration of episodic emissions (days)
- **vent_sizes** – A list of emission sizes for periodic emissions (g/s)
- **vent_period** – The time between emissions (days)
- **vent_duration** – the time that a periodic vent persits (days)
- **vent_starts** – the time at which the first periodic vent occurs at each component in the simulation
- **name** – A name for the instance of Component
- **null_repair_rate** – the rate at which fugitive emissions are repaired. If None, a steady state assumption is enforced based on emission_production_rate and emission_per_comp.
- **dist_type** – The type of distribution to be used in determining emission rates for new emissions

1.2.2.2 GasField

class GasField(*time=None*, *sites=None*, *emissions=None*, *met_data_path=None*)

GasField accommodates all data that defines a gas field at the beginning of a simulation.

Parameters

- **time** – A FEAST time object
- **sites** – a dict of sites like this: {'name': {'number': n_sites, 'parameters': site_object}}
- **emissions** – A FEAST emission object to be used during the simulations
- **met_data_path** – A path to a met data file

1.2.2.3 Site

class Site(*name='default'*, *comp_dict=None*, *prod_dat=None*)

A class to store the number and type of components associated with a site.

Parameters

- **name** – The name of the site object (a string)
- **comp_dict** – A dict of components at the site, for example: {'name': {'number': 650, 'parameters': Component()}}
- **prod_dat** –

1.2.3 result_classes

result_classes defines classes that are used to store event counts and continuous variable data for saving.

1.2.3.1 ResultAggregate

class ResultAggregate(*units=None*, *time_value=None*)

A super class designed to store aggregate results during a simulation. Time and value pairs are stored in a list.

1.2.3.2 ResultContinuous

class ResultContinuous(***kwargs*)

Designed to store continuous rates that endure between consecutive time recordings as opposed to discrete variables that occur at a specific time. For example, emission rate can be recorded as a continuous data type.

1.2.3.3 ResultDiscrete

class ResultDiscrete(***kwargs*)

Designed to store discrete values associated with specific times, as opposed to continuous rates that persist between consecutive data points. For example, the number of sites surveyed can be recorded as a discrete data type.

1.2.4 simulation_classes

simulation_classes stores the classes used to represent time, results and financial settings in simulations.

1.2.4.1 Scenario

class Scenario(*time*, *gas_field*, *ldar_program_dict*)

A class to store all data specifying a scenario and the methods to run and save a realization

Parameters

- **time** – Time object
- **gas_field** – GasField object
- **ldar_program_dict** – dict of detection methods and associated data

1.2.4.2 Time

class Time(*delta_t=1*, *end_time=365*, *current_time=0*)

Instances of the time class store all time related information during a simulation

Parameters

- **delta_t** – length of one timestep (days)
- **end_time** – length of the simulation (days)
- **current_time** – current time in a simulation (days)

1.3 input_data_classes

This module defines all classes used to store input data.

1.3.1 DataFile

class DataFile(*notes='No notes provided'*, *raw_file_name=None*, *data_prep_file=None*)

DataFile is an abstract super class that all data file types inherit from FEAST.

Parameters

- **notes** – A string containing notes on the object created
- **raw_file_name** – path (or list of paths) to a raw input file(s)
- **data_prep_file** – path to the file used to process input data and create the DataFile object

1.3.2 LeakData

class LeakData(*notes='No notes provided'*, *raw_file_name=None*, *data_prep_file=None*, *leak_sizes=None*)

LeakData is designed to store all leak size data from a reference. It accommodates multiple detection methods within a single instance.

Creates a LeakData object

Parameters

- **notes** – a string containing notes on the object created
- **raw_file_name** – path to a raw input file
- **data_prep_file** – path to a script used to build the object from the raw data file
- **leak_sizes** – list of leak sizes. If leaks were detected using multiple methods, leak_sizes must be a dict with one key for each detection method

1.3.3 ProductionData

class ProductionData(*site_prod=None*, ***kwargs*)

Stores an array of production rates that may be associated with gas production sites

Parameters

- **prod_dat** – an array of production rates
- **kwargs** – pass through

1.3.4 RepairData

class RepairData(*notes='No notes provided'*, *raw_file_name=None*)

RepairData is designed to store the costs of repairing leaks from a particular reference andd associated notes.

Parameters

- **notes** – A string containing notes on the object created
- **raw_file_name** – path to a raw input file

1.4 MEET_1_importer

1.4.1 gas_comp_to_dict

gas_comp_to_dict(*gc*, *delta_t*)

converts a gas_comp DataFrame to a dict specifying the start time, top time, emission rate, location, emission type and emission ID for every emitter in gas_comp. If an emission changes its emission rate, it is recorded as stopping and restarting at the time of the change. LDAR programs will treat the emission correctly due to the emission ID that persists after the change in emission rate.

Parameters

- **gc** – a gas_comp DataFrame as created by the function load_gas_comp_file
- **delta_t** – The time resolution of the gas_comp file (seconds)

Returns the dict containing emitter data from the gas_comp file

1.4.2 gascomp_reader

gascomp_reader(*path_to_gas_comp*, *feast_delta_t*, *duration*, *comps_per_site*, *rep_cost_path*, *met_data_path=None*, *met_start_hr=None*)

Read a MEET 1.0 GasComp result file and return a feast Time object and GasField object specifying all of the emissions to occur in a FEAST simulation

Parameters

- **path_to_gas_comp** – path to a gas_comp file
- **feast_delta_t** – time resolution of the FEAST simulation (days). FEAST will represent emissions with the precision of the original MEET simulation, but will invoke LDAR events as though they occur instantaneously with the time resolution specified here.
- **duration** – duration of the simulation to run in FEAST (days)
- **comps_per_site** – A dict of form {loc: number of components}. This will specify the number of components to be inspected for leaks at every site. Must have a key for every location ID in the gas_comp file.
- **rep_cost_path** – Path to a repair cost data file. The FEAST will randomly assign these repair costs to leaks simulated in MEET.
- **met_data_path** – Path to a meteorological data file (if left as None, FEAST will run without met data)
- **met_start_hr** – FEAST will rotate the meteorological data to begin at the hour specified by an integer here.

Return time_obj a FEAST Time object to specify a simulation

Return gas_field a FEAST GasField object

1.4.3 gc_dat_to_gas_field

gc_dat_to_gas_field(*feast_delta_t*, *duration*, *comps_per_site*, *rep_cost_path*, *met_data_path=None*, *met_start_hr=None*, *size=0*, *loc=0*, *start=0*, *stop=0*, *emtype=0*, *em_id=None*)

Ports data from a data_dict returned by the funcation gas_comp_to_dict into a FEAST GasField

Parameters

- **feast_delta_t** – FEAST time resolution (days)
- **duration** – FEAST simulation duration (days)
- **comps_per_site** – A dict of form {loc: number of components}. This will specify the number of components to be inspected for leaks at every site
- **rep_cost_path** – Path to a repair cost data file. The FEAST will randomly assign these repair costs to leaks simulated in MEET
- **met_data_path** – Path to a meteorological data file (if left as None, FEAST will run without met data)
- **met_start_hr** – FEAST will rotate the meteorological data to begin at the hour specified by an integer here.
- **size** – an array of emission sizes (g/s)

- **`loc`** – an array of site ids
- **`start`** – an array of emission start times (days)
- **`stop`** – an array of emission stop times (days)
- **`emtype`** – an array of emission types (reparable or no)
- **`em_id`** – an array of emission IDs

Returns a FEAST Time object

Returns a FEAST GasField object

1.4.4 load_gas_comp_file

`load_gas_comp_file` (*path_to_gas_comp*)

Loads a gas_comp.csv file generated by MEET 1.0

Parameters **`path_to_gas_comp`** – a path (str) to a gas_comp file

Returns a DataFrame of the gas_comp file

Returns the time resolution of meet simulation (seconds)

1.5 ResultsProcessing

1.5.1 plotting_functions

1.5.1.1 abatement_cost_plotter

`abatement_cost_plotter` (*directory*, *gwp=34*)

Generates a box plot of the cost of abatement gwp defaults to 34, which is the value provided in the IPCC 5th assessment report including climate-carbon feedbacks (see Table 8.7, page 714 in Chapter 8 of Climate Change 2013: The Physical Science Basis.)

Parameters

- **`directory`** – A directory containing one or more realizations of a scenario
- **`gwp`** – global warming potential of methane

Returns

1.5.1.2 plot_fixer

`plot_fixer` (*fig=None*, *ax=None*, *fsize=18*, *color=(0, 0, 0)*, *tight_layout=True*, *line_width=4*, *fontweight='bold'*)

1.5.1.3 summary_plotter

summary_plotter(*directory*, *n_wells=None*, *ylabel=None*)

The NPV for each realization stored in ‘directory is calculated and displayed in a stacked bar chart. Each component of the NPV is displayed separately in the chart.

Parameters

- **directory** – path to a directory containing results files
- **n_wells** – if set to a number, then the NPV will be reported on a per well basis
- **ylabel** – yaxis label

1.5.1.4 time_series

time_series(*results_file*, *line_width=6*)

Display a time series of emissions from each detection method in a results file

Parameters

- **results_file** – path to a results file
- **line_width** – width at which to plot lines

1.5.2 results_analysis_functions

1.5.2.1 npv_calculator

npv_calculator(*filepath*, *discount_rate*, *gas_price*)

Calculates the net present value (NPV) of each LDAR program in the results file

Parameters

- **filepath** – path to a results file
- **discount_rate** – The discount rate of future cash flows (should be between 0 and 1)
- **gas_price** – The value to assign to mitigated gas losses ($/gram)

Returns null_npv NPV of each LDAR program compared to a scenario with only the Null LDAR program [k$/well]

1.5.2.2 results_analysis

results_analysis(*directory*, *discount_rate*, *gas_price*)

Process many realizations of a single scenario stored in a directory

Parameters **directory** – A directory of results files all generated under the same scenario

Returns null_npv array of null-NPV of each LDAR program in each realization [k$/well] emissions_timeseries Array of emissions in each LDAR program in each realization at each time step costs Array of costs associated with each LDAR program (no discounting, all costs summed) techs list of detection program names

PYTHON MODULE INDEX