conrad Documentation

Release 0.0.3a

Baris Ungun, Anqi Fu, Stephen Boyd

Aug 07, 2017

Contents

convex optimization in radiation therapy

CHAPTER 1

Contents:

Tutorial

Tutorial

Case

Case

Medicine

Dose Constraints

Prescription

Define `Prescription` and methods for parsing prescription data from python objects as well as JSON- or YAML-formatted files.

Parsing methods expect the following formats.

YAML:

```
- name : PTV
  label : 1
  is_target: Yes
  dose : 35.
  constraints:
  - "D90 >= 32.3Gy"
  - "D1 <= 1.1rx"

- name : OAR1
  label : 2
  is_target: No
  dose :
  constraints:
  - "D95 <= 20Gy"
  - "V30 Gy <= 20%"
```

Python `list` of `dict` (JSON approximately the same):

```
[{
        "name" : "PTV",
        "label" : 1,
        "is_target" : True,
        "dose" : 35.,
        "constraints" : ["D1 <= 1.1rx", "D90 >= 32.3Gy"]
}, {
        "name" : "OAR1",
        "label" : 2,
        "is_target" : False,
        "dose" : None,
        "constraints" : ["D95 <= 20Gy"]
}]
```

JSON verus Python syntax differences:

- `true`/`false` instead of `True`/`False`
- `null` instead of `None`

class `prescription.`**Prescription**(*prescription_data=None*)

Class for specifying structures with dose targets and constraints.

constraint_dict

dict – Dictionary of `ConstraintList` objects, keyed by structure labels.

structure_dict

dict – Diciontionary of `Structure` objects, keyed by structure labels.

rx_list

list – List of dictionaries representation of prescription.

add_structure_to_dictionaries(*structure*)

Add a new structure to internal representation of prescription.

Parameters **structure** (`Structure`) – Structure added to *Prescription.structure_dict*. An corresponding, empty constraint list is added to *Prescription.constraint_dict*.

Returns None

Raises `TypeError` – If `structure` not a `Structure`.

constraints_by_label
Dictionary of constraints in prescription, by structure label.

dict
Dictionary of structures in prescription, by label.

digest (*prescription_data*)
Populate *Prescription*'s structures and dose constraints.

Specifically, for each entry in `prescription_data`, construct a `Structure` to capture structure data (e.g., name, label), as well as a corresponding but separate `ConstraintList` object to capture any dose constraints specified for the structure.

Add each such structure to *Prescription.structure_dict*, and each such constraint list to *Prescription.constraint_dict*. Build or copy a "list of dictionaries" representation of the prescription data, assign to *Prescription.rx_list*.

Parameters **prescription_data** – Input to be parsed for structure and dose constraint data. Accepted formats include `str` specifying a valid path to a suitably-formatted JSON or YAML file, or a suitably-formatted *list* of *dict* objects.

Returns None

Raises `TypeError` – If input not of type *list* or a `str` specfying a valid path to file that can be loaded with the `json.load()` or `yaml.safe_load()` methods.

list
List of structures in prescription

report (*anatomy*)
Reports whether `anatomy` fulfills all prescribed constraints.

Parameters **anatomy** (`Antomy`) – Container of structures to compare against prescribed constraints.

Returns Dictionary keyed by structure label, with data on each dose constraint associated with that structure in this *Prescription*. Reported data includes the constraint, whether it was satisfied, and the actual dose achieved at the percentile/threshold specified by the constraint.

Return type *dict*

Raises `TypeError` – If `anatomy` not an `Anatomy`.

report_string (*anatomy*)
Reports whether `anatomy` fulfills all prescribed constraints.

Parameters **anatomy** (`Anatomy`) – Container of structures to compare against prescribed constraints.

Returns Stringified version of output from `Presription.report`.

Return type `str`

Anatomy

Define *Anatomy*, container for treatment planning structures.

class `anatomy.`**Anatomy** (*structures=None*)
Iterable container class for treatment planning structures.

Provides simple syntax via overloaded operators, including addition, retrieval, and removal of structures from anatomy:

```
anatomy = Anatomy()

# target structure with label = 4
s1 = Structure(4, 'target', True)

# non-target structure with label = 12
s2 = Structure(12, 'non-target', False)

# non-target structure with label = 7
s3 = Structure(7, 'non-target 2', False)

anatomy += s1
anatomy += s2
anatomy += s3

# remove structure s3 by name
anatomy -= 'non-target 2'

# remove structure s2 by label
anatomy -= 12

# retrieve structure s1 by name
anatomy[4]
anatomy['target']
```

calculate_doses(*beam_intensities*)
Calculate voxel doses to each structure in *Anatomy*.

Parameters **beam_intensities** – Beam intensities to provide to each structure's *Structure.calculate_dose* method.

Returns None

clear_constraints()
Clear all constraints from all structures in *Anatomy*.

Parameters **None** –

Returns None

dose_summary_data(*percentiles=[2, 98]*)
Collimate dose summaries from each structure in *Anatomy*.

Parameters **percentiles** (*list*) – List of percentiles to include in dose summary queries.

Returns Dictionary of dose summaries obtained by calling *Structure.summary* for each structure.

Return type dict

dose_summary_string
Collimate dose summary strings from each structure in *Anatomy*.

Parameters **None** –

Returns Dictionary of dose summaries obtained by calling *Structure.summary_string* for each structure.

Return type dict

is_empty
True if *Anatomy* contains no structures.

label_order
Ranked list of labels of structures in *Anatomy*.

Raises ValueError – If input to setter contains labels for structures not contained in anatomy, or if the length of the input list does not match *Anatomy.n_structures*.

labels
List of labels of structures in *Anatomy*.

list
List of structures in *Anatomy*.

n_structures
Number of structures in *Anatomy*.

plannable
True if all structures plannable and at least one is a target.

plotting_data(*constraints_only=False*, *maxlength=None*)
Dictionary of matplotlib-compatible plotting data for all structures.

Parameters

- **constraints_only** (bool, optional) – If True, return only the constraints associated with each structure.
- **maxlength** (int, optional) – If specified, re-sample each structure's DVH plotting data to have a maximum series length of maxlength.

propagate_doses(*voxel_doses*)
Assign pre-calculated voxel doses to each structure in *Anatomy*

Parameters **voxel_doses** (dict) – Dictionary mapping structure labels to voxel dose subvectors.

Returns None

satisfies_prescription(*constraint_dict*, *satisfaction_tol=0.0*)
Check whether anatomy satisfies supplied constraints.

:param dict: Dictionary of ConstraintList objects :param keyed by structure labels.:

Returns True if each structure in

Return type int

size
Total number of voxels in all structures in *Anatomy*.

structures
Dictionary of structures in anatomy, keyed by label.

Setter method accepts any iterable collection of Structure objects.

Raises

- TypeError – If input to setter is not iterable.
- ValueError – If input to setter contains elements of a type other than Structure.

Define *Structure*, building block of Anatomy.

structure.**W_UNDER_DEFAULT**
float – Default objective weight for underdose penalty on target structures.

structure.**W_OVER_DEFAULT**
float – Default objective weight for underdose penalty on non-target structures.

structure.**W_NONTARG_DEFAULT**
float – Default objective weight for overdose penalty on non-target structures.

class structure.**Structure**(*label*, *name*, *is_target*, *size=None*, ***options*)
Structure manages the dose information (including the dose influence matrix, dose calculations and dose volume histogram), as well as optimization objective information—including dose constraints—for a set of voxels (volume elements) in the patient volume to be treated as a logically homogeneous unit with respect to the optimization process.

There are usually three types of structures:

- **Anatomical structures, such as a kidney or the spinal** cord, termed organs-at-risk (OARs),
- **Clinically delineated structures, such as a tumor or a target** volume, and,
- **Tissues grouped together by virtue of not being explicitly** delineated by a clinician, typically lumped together under the catch-all category "body".

Healthy tissue structures, including OARs and "body", are treated as non-target, are prescribed zero dose, and only subject to an overdose penalty during optimization.

Target tissue structures are prescribed a non-zero dose, and subject to both an underdose and an overdose penalty.

label
(int or str): Label, applied to each voxel in the structure, usually generated during CT contouring step in the clinical workflow for treatment planning.

name
str – Clinical or anatomical name.

is_target
bool – True if structure is a target.

dvh
DVH – Dose volume histogram.

constraints
ConstraintList – Mutable collection of dose constraints to be applied to structure during optimization.

A
Alias for *Structure.A_full*.

A_full
Full dose matrix (dimensions = voxels x beams).

Setter method will perform two additional tasks:

- **If *Structure.size* is not set, set it based on** number of rows in A_full.
- **Trigger *Structure.A_mean* to be calculated from** *Structure.A_full*.

Raises

- TypeError – If A_full is not a matrix in np.ndarray, sp.csc_matrix, or sp.csr_matrix formats.
- ValueError – If *Structure.size* is set, and the number of rows in A_full does not match *Structure.size*.

A_mean
Mean dose matrix (dimensions = 1 x beams).

Setter expects a one dimensional np.ndarray representing the mean dose matrix for the structure. If this optional argument is not provided, the method will attempt to calculate the mean dose from *Structure.A_full*.

Raises

- TypeError – If A_mean provided and not of type np.ndarray, *or* if mean dose matrix is to be calculated from *Structure.A_full*, but full dose matrix is not a conrad-recognized matrix type.
- ValueError – If A_mean is not dimensioned as a row or column vector, or number of beams implied by A_mean conflicts with number of beams implied by *Structure.A_full*.

assign_dose(*voxel_doses*)
Alias for *Structure.assign_y()*.

assign_y(*y*)
Assign dose vector to structure.

Parameters **y** – Vector-like input of voxel doses.

Returns None

Raises ValueError – if structure size is known and incompatible with length of y.

boost
Adjustment factor from precription dose to optimization dose.

calc_y(*x*)
Calculate voxel doses as: attr:*Structure.y* = *Structure.A* * x.

Parameters **x** – Vector-like input of beam intensities.

Returns None

calculate_dose(*beam_intensities*)
Alias for *Structure.calc_y()*.

collapsable
True if optimization can be performed with mean dose only.

constraints_string
String of structure header and constraints

dose
Dose level targeted in structure's optimization objective.

The dose has two components: the precribed dose, *Structure.dose_rx*, and a multiplicative adjustment factor, *Structure.boost*.

Once the structure's dose has been initialized, setting *Structure.dose* will change the adjustment factor. This is to distinguish (and allow for differences) between the dose level prescribed to a structure by a clinician and the dose level request to a numerical optimization algorithm that yields a desirable distribution, since the latter may require some offset relative to the former. To change the reference dose level, use the *Structure.dose_rx* setter.

Setter is no-op for non-target structures, since zero dose is prescribed always.

Raises

- TypeError – If requested dose does not have units of DeliveredDose.

- `ValueError` – If zero dose is requested to a target structure.

dose_rx
Prescribed dose level.

Setting this field sets *`Structure.dose`* to the requested value and *`Structure.boost`* to `1`.

dose_unit
One times the `DeliveredDose` unit of the structure dose.

max_dose
Maximum dose to structure's voxels.

mean_dose
Average dose to structure's voxels.

min_dose
Minimum dose to structure's voxels.

objective_string
String of structure header and objectives

plannable
True if structure's attached data is sufficient for optimization.

Minimum requirements:

- Structure size determined, and
- Dose matrix assigned, *or*
- Structure collapsable and mean dose matrix assigned.

plotting_data(*constraints_only=False*, *maxlength=None*)
Dictionary of `matplotlib`-compatible plotting data.

Data includes DVH curve, constraints, and prescribed dose.

Parameters

- **constraints_only** (`bool`, optional) – If `True`, return only the constraints associated with the structure.
- **maxlength** (`int`, optional) – If specified, re-sample the structure's DVH plotting data to have a maximum series length of `maxlength`.

reset_matrices()
Reset structure's dose and mean dose matrices to `None`

satisfies(*constraint*, *satisfaction_tol=0.0*)
Test whether structure's voxel doses satisfy `constraint`.

Parameters **constraint** (`Constraint`) – Dose constraint to test against structure's voxel doses.

Returns `True` if structure's voxel doses conform to the queried constraint.

Return type `bool`

Raises

- `TypeError` – If `constraint` not of type `Constraint`.
- `ValueError` – If *`Structure.dvh`* not initialized or not populated with dose data.

set_constraint(*constr_id*, *threshold=None*, *relop=None*, *dose=None*)
Modify threshold, relop, and dose of an existing constraint.

Parameters

- **constr_id** (str) – Key to a constraint in *Structure.constraints*.
- **threshold** (*optional*) – Percentile threshold to assign if queried constraint is of type PercentileConstraint, no-op otherwise. Must be compatible with the setter method for PercentileConstraint.percentile.
- **relop** (*optional*) – Inequality constraint sense. Must be compatible with the setter method for Constraint.relop.
- **dose** (*optional*) – Constraint dose. Must be compatible with setter method for Constraint.dose.

Returns None

Raises ValueError – If constr_id is not the key to a constraint in the Constraintlist located at *Structure.constraints*.

size
Structure size (i.e., number of voxels in structure).

Raises ValueError – If size not an int.

summary (*percentiles=[2, 25, 75, 98]*)
Dictionary summarizing dose statistics.

Parameters **percentiles** (list, optional) – Percentile levels at which to query the structure dose. If not provided, will query doses at default percentile levels of 2%, 25%, 75% and 98%.

Returns Dictionary of doses at requested percentiles, plus mean, minimum and maximum voxel doses.

Return type dict

summary_string
String of structure header and dose summary

voxel_weights
Voxel weights, or relative volumes of voxels.

The voxel weights are the 1 vector if the structure volume is regularly discretized, and some other set of integer values if voxels are clustered.

Raises ValueError – If *Structure.voxel_weights* setter called before *Structure.size* is defined, or if length of input does not match *Structure.size*, or if any of the provided weights are negative.

y
Vector of structure's voxel doses.

y_mean
Value of structure's mean voxel dose.

Physics

Define *DoseFrame* and *Physics* classes for treatment planning.

class physics.**DoseFrame**(*voxels=None*, *beams=None*, *data=None*, *voxel_labels=None*, *beam_labels=None*, *voxel_weights=None*, *beam_weights=None*, *frame_name=None*, ***options*)
Describe a reference frame (voxels x beams) for dosing physics.

A *DoseFrame* provides a description of the mathematical basis of the dosing physics, which usually consists of a matrix in $\mathbf{R}^{\text{voxels}\times\text{beams}}$, mapping the space of beam intensities, $\mathbf{R}^{\text{beams}}$ to the space of doses delivered to each voxel, $\mathbf{R}^{\text{voxels}}$.

For a given plan, we may require conversions between several related representations of the dose matrix. For instance, the beams may in fact be beamlets that can be coalesced into apertures, or—in order to accelerate the treatment plan optimization—may be clustered or sampled. Similarly, voxels may be clustered or sampled. For voxels, there is also a geometric frame, with `X * Y * Z` voxels, where the tuple (`X`, `Y`, `Z`) gives the dimensions of a regularly discretized grid, the so-called dose grid used in Monte Carlo simulations or ray tracing calculations. Since many of the voxels in this rectangular volume necessarily lie outside of the patient volume, there is only some number of voxels `m < X * Y * Z` that are actually relevant to treatment planning.

Accordingly, each *DoseFrame* is intended to capture one such configuration of beams and voxels, with corresponding data on labels and/or weights attached to the configuration. Voxel labels allow each voxel to be mapped to an anatomical or clinical structure used in planning. The concept of beam labels is defined to allow beams to be gathered in logical groups (e.g. beamlets in fluence maps, or apertures in arcs) that may be constrained jointly or treated as a unit in some other way in an optimization context. Voxel and beam weights are defined for accounting purposes: if a *DoseFrame* represents a set of clustered voxels or beams, the associated weights give the number of unitary voxels or beams in each cluster, so that optimization objective terms can be weighted appropriately.

beam_labels
Vector of labels mapping beams to beam groups.

Setter will also use dimension of input vector to set beam dimensions (*DoseFrame.beams*) if not already assigned at call time.

Raises `ValueError` – If provided vector dimensions inconsistent with known frame dimensions.

beam_lookup_by_label (*label*)
Get indices of beam labeled `label` in this *DoseFrame*.

beam_weights
Vector of weights assigned to each (meta-)beam.

Setter will also use dimension of input vector to set voxel dimensions (*DoseFrame.beams*) if not already assigned at call time.

Raises `ValueError` – If provided vector dimensions inconsistent with known frame dimensions.

beams
Number of beams in dose frame.

If *DoseFrame.beam_weights* has not been assigned at call time, the setter will initialize it to the 1 vector.

Raises `ValueError` – If *DoseFrame.beams* already determined. Beam dimension is a write-once property.

dose_matrix
Dose matrix.

Setter will also use dimensions of input matrix to set any dimensions (*DoseFrame.voxels* or *DoseFrame.beams*) that are not already assigned at call time.

Raises

- `TypeError` – If input to setter is not a sparse or dense matrix type recognized by `conrad`.

- `ValueError` – If provided matrix dimensions inconsistent with known frame dimensions.

static indices_by_label (*label_vector*, *label*, *vector_name*)
Retrieve indices of vector entries corresponding to a given value.

Parameters

- **label_vector** – Vector of values to search for entries corresponding
- **label** – Value to find.
- **vector_name** (`str`) – Name of vector, for use in exception messages.

Returns Vector of indices at which the entries of `label_vector` are equal to `label`.

Return type `ndarray`

Raises

- `ValueError` – If `label_vector` is `None`.
- `KeyError` – If `label` not found in `label_vector`.

plannable
True if both dose matrix and voxel label data loaded.

This can be achieved by having a contiguous matrix and a vector of voxel labels indicating the identity of each row of the matrix, or a dictionary of submatrices that map label keys to submatrix values.

shape
Frame dimensions, $\{\mathbf{R}^{\text{voxels}} \times \mathbf{R}^{\text{beams}}\}$.

voxel_labels
Vector of labels mapping voxels to structures.

Setter will also use dimension of input vector to set voxel dimensions (*DoseFrame.voxels*) if not already assigned at call time.

Raises `ValueError` – If provided vector dimensions inconsistent with known frame dimensions.

voxel_lookup_by_label (*label*)
Get indices of voxels labeled `label` in this *DoseFrame*.

voxel_weights
Vector of weights assigned to each (meta-)voxel.

Setter will also use dimension of input vector to set voxel dimensions (*DoseFrame.voxels*) if not already assigned at call time.

Raises `ValueError` – If provided vector dimensions inconsistent with known frame dimensions.

voxels
Number of voxels in dose frame.

If *DoseFrame.voxel_weights* has not been assigned at call time, the setter will initialize it to the 1 vector.

Raises `ValueError` – If *DoseFrame.voxels* already determined. Voxel dimension is a write-once property.

class `physics.`**Physics** (*voxels=None*, *beams=None*, *dose_matrix=None*, *dose_grid=None*, *voxel_labels=None*, ***options*)
Class managing all dose-related information for treatment planning.

A *Physics* instance includes one or more DoseFrames, each with attached data including voxel dimensions, beam dimensions, a voxel-to-structure mapping, and a dose influence matrix. The class also provides an interface for adding and switching between frames, and extracting data from the active frame.

A *Physics* instance optionally has an associated VoxelGrid that represents the dose grid used for dose matrix calculation, and that provides the necessary geometric information for reconstructing and rendering the 3-D dose distribution (or 2-D slices thereof).

add_dose_frame (*key*, ***frame_args*)
Add new *DoseFrame* representation of a dosing configuration.

Parameters

- **key** – A new *DoseFrame* will be added to the *Physics* object's dictionary with the key key.
- ****frame_args** – Keyword arguments passed to *DoseFrame* initializer.

Returns None

Raises ValueError – If key corresponds to an existing key in the *Physics* object's dictionary of dose frames.

available_frames
List of keys to dose frames already attached to *Physics*.

beam_weights_by_label (*label*)
Subvector of beam weights, filtered by label.

beams
Number of beams in current *Physics.frame*.

change_dose_frame (*key*)
Switch between dose frames already attached to *Physics*.

data_loaded
True if a client has seen data from the current dose frame.

dose_grid
Three-dimensional grid.

dose_matrix
Dose influence matrix for current *Physics.frame*.

dose_matrix_by_label (*voxel_label=None*, *beam_label=None*)
Submatrix of dose matrix, filtered by voxel and beam labels.

Parameters

- **voxel_label** (*optional*) – Label for which to build/retrieve submatrix of current *Physics.dose_matrix* based on row indices for which voxel_label matches the entries of *Physics.voxel_labels*. All rows returned if no label provided.
- **beam_label** (*optional*) – Label for which to build/retrieve submatrix of current *Physics.dose_matrix* based on column indices for which beam_label matches the entries of Physics.frame.beam_labels. All columns returned if no label provided.

Returns Submatrix of dose matrix attached to current *Physics.frame*, based on row indices for which Physics.frame.voxel_labels matches the queried voxel_label, and column indices for which Physics.frame.beam_labels matches the queried beam_label.

frame
Handle to *DoseFrame* representing current dosing configuration.

mark_data_as_loaded()
Allow clients to mark dose frame data as seen.

plannable
True if current frame has both dose matrix and voxel label data

unique_frames
List of unique dose frames attached to *Physics*.

voxel_labels
Vector mapping voxels to structures in current *Physics.frame*.

voxel_weights_by_label(*label*)
Subvector of voxel weights, filtered by `label`.

voxels
Number of voxels in current *Physics.frame*.

Optimization

Treatment Planning as a Convex Problem

Define *PlanningProblem*, interface between `Case` and solvers.

class problem.**PlanningProblem**
Interface between `Case` and convex solvers.

Builds and solves specified treatment planning problem using fastest available solver, then extracts solution data and solver metadata (e.g., timing results) for use by clients of the *PlanningProblem* object (e.g., a `Case`).

solver_cvxpy
`SolverCVXPY` or `NoneType` – `cvxpy`-baed solver, if available.

solver_pogs
`SolverOptkit` or `NoneType` – POGS solver, if available.

solve(*structures*, *run_output*, *slack=True*, *exact_constraints=False*, ***options*)
Run treatment plan optimization.

Parameters

- **structures** – Iterable collection of `Structure` objects with attached objective, constraint, and dose matrix information. Build convex model of treatment planning problem using these data.
- **run_output** (`RunOutput`) – Container for saving solver results.
- **slack** (`bool`, optional) – If `True`, build dose constraints with slack.
- **exact_constraints** (`bool`, optional) – If `True` *and* at least one structure has a percentile-type dose constraint, execute the two-pass planning algorithm, using convex restrictions of the percentile constraints on the firstpass, and exact versions of the constraints on the second pass.
- ****options** – Abitrary keyword arguments, passed through to `PlanningProblem.solver.init_problem()` and `PlanningProblem.solver.build()`.

Returns Number of feasible solver runs performed: `0` if first pass infeasible, `1` if first pass feasible, `2` if two-pass method requested and both passes feasible.

Return type int

Raises ValueError – If no solvers avaialable.

solver
Get active solver (CVXPY or OPTKIT/POGS).

Convex Solvers

CVXPY solver interface

POGS solver interface

Define *Case*, the top level interface for treatment planning.

class case.**Case**(*anatomy=None*, *physics=None*, *prescription=None*, *suppress_rx_constraints=False*)
Top level interface for treatment planning.

A *Case* has four major components.

Case.physics is of type Physics, and contains physical information for the case, including the number of voxels, beams, beam layout, voxel labels and dose influence matrix.

Case.anatomy is of type Antomy, and manages the structures in the patient anatomy, including optimization objectives and dose constraints applied to each structure.

Case.prescription is of type Prescription, and specifies a clinical prescription for the case, including prescribed doses for target structures and prescribed dose constraints (e.g., RTOG recommendations).

Case.problem is of type PlanningProblem, and is a tool that forms and manages the mathematical representation of treatment planning problem specified by case anatomy, physics and prescription; it serves as the interface to convex solvers that run the treatment plan optimization.

A
Dose matrix from current planning frame of *Case.physics*.

add_constraint(*structure_label*, *constraint*)
Add constraint to structure specified by structure_label.

Parameters

- **structure_label** – Must correspond to label or name of a Structure in *Case.anatomy*.
- **constraint** (conrad.medicine.Constraint) – Dose constraint to add to constraint list of specified structure.

Returns None

anatomy
Container for all planning structures.

anatomy_for_frame(*frame_name*)

calculate_doses(*x*)
Calculate voxel doses for each structure in *Case.anatomy*.

Parameters **x** – Vector-like np.array of beam intensities.

Returns None

change_constraint(*constr_id*, *threshold=None*, *direction=None*, *dose=None*)
Modify constraint in *Case*.

If `constr_id` is a valid key to a constraint in the `ConstraintList` attached to one of the structures in *Case.anatomy*, that constraint will be modified according to the remaining arguments. Call is no-op if key does not exist.

Parameters

- **constr_id** – Key to a constraint on one of the structures in *Case.anatomy*.
- **threshold** (*optional*) – If constraint in question is a `PercentileConstraint`, percentile threshold set to this value. No effect otherwise.
- **direction** (`str`, optional) – Constraint direction set to this value. Should be one of: '<' or '>'.
- **dose** (`DeliveredDose`, optional) – Constraint dose level set to this value.

Returns None

change_objective(*label*, ***objective_parameters*)
Modify objective for structure in *Case*.

Parameters

- **label** – Label or name of a `Structure` in *Case.anatomy*.
- ****options** –

Returns None

clear_constraints()
Remove all constraints from all structures in *Case*.

Parameters **None** –

Returns None

drop_constraint(*constr_id*)
Remove constraint from case.

If `constr_id` is a valid key to a constraint in the `ConstraintList` attached to one of the structures in *Case.anatomy*, that constraint will be removed from the structure's constraint list. Call is no-op if key does not exist.

Parameters **constr_id** – Key to a constraint on one of the structures in *Case.anatomy*.

Returns None

gather_physics_from_anatomy()
Gather dose matrices from structures.

Parameters **None** –

Returns None

Raises `AttributeError` – If `case.physics.dose_matrix` is already set.

load_physics_to_anatomy(*overwrite=False*, *frame='active'*)
Transfer data from physics to each structure.

The label associated with each structure in *Case.anatomy* is used to retrieve the dose matrix data and voxel weights from *Case.physics* for the voxels bearing that label.

The method marks the `Case.physics.dose_matrix` as seen, in order to prevent redundant data transfers.

Parameters **overwrite** (bool, optional) – If True, dose matrix data from *Case.physics* will overwrite dose matrices assigned to each structure in *Case.anatomy*.

Returns None

Raises ValueError – If *Case.anatomy* has assigned dose matrices, *Case.physics* not marked as having updated dose matrix data, and flag overwrite set to False.

n_beams
Number of beams in current planning frame of *Case.physics*.

n_structures
Number of structures in *Case.anatomy*.

n_voxels
Number of voxels in current planning frame of *Case.physics*.

physics
Patient anatomy, contains all dose physics information.

plan(*use_slack=True*, *use_2pass=False*, ***options*)
Invoke numerical solver to optimize plan, given state of *Case*.

At call time, the objectives, dose constraints, dose matrix, and other relevant data associated with each structure in *Case.anatomy* is passed to *Case.problem* to build and solve a convex optimization problem.

Parameters

- **use_slack** (bool, optional) – Allow slacks on each dose constraint.
- **use_2pass** (bool, optional) – Execute two-pass planing method to enforce exact versions, rather than convex restrictions of any percentile-type dose constraints included in the plan.
- ****options** – Arbitrary keyword arguments. Passed through to Case.problem.solve().

Returns Tuple with bool indicator of planning problem feasibility and a RunRecord with data from the setup, execution and output of the planning run.

Return type tuple

Raises ValueError – If case not plannable due to missing information.

plannable
True if case meets minimum requirements for *Case.plan()* call.

Parameters **None** –

Returns True if anatomy has one or more target structures and dose matrices from the case physics.

Return type bool

plotting_data(*x=None*, *constraints_only=False*, *maxlength=None*)
Dictionary of matplotlib-compatible plotting data.

Includes data for dose volume histograms, prescribed doses, and dose volume (percentile) constraints for each structure in *Case.anatomy*.

Parameters

- **x** (*optional*) – Vector of beam intensities from which to calculate structure doses prior to emitting plotting data.

- **constraints_only** (bool, optional) – If True, only include each structure's constraint data in returned dictionary.
- **maxlength** (int, optional) – If specified, re-sample each structure's DVH plotting data to have a maximum series length of maxlength.

Returns Plotting data for each structure, keyed by structure label.

Return type dict

prescription
Container for clinical goals and limits.

Structure list from prescription used to populate *Case.anatomy* if anatomy is empty when *Case.prescription* setter is invoked.

problem
Object managing numerical optimization setup and results.

propagate_doses(*y*)
Split voxel dose vector y into doses for each structure in *Case.anatomy*.

Parameters **y** – Vector-like np.array of voxel doses, or dictionary mapping structure labels to voxel dose subvectors,

structures
Dictionary of structures contained in *Case.anatomy*.

transfer_rx_constraints_to_anatomy()
Push constraints in prescription onto structures in anatomy.

Assume each structure label represented in *Case.prescription* is represented in *Case.anatomy*. Any existing constraints on structures in *Case.anatomy* are preserved.

Parameters **None** –

Returns None

Treatment Planning Workflow

Treatment Planning Workflow

Planning History

Define classes used to record solver inputs/outputs and maintain a treatment planning history.

class history.**PlanningHistory**
Class for tracking treatment plans generated by a Case.

runs
list of *RunRecord* – List of treatment plans in history, in chronological order.

run_tags
dict – Dictionary mapping tags of named plans to their respective indices in *PlanningHistory.runs*

last_feasible
Solver feasibility flag from most recent treatment plan.

last_info
Solver info from most recent treatment plan.

last_solvetime
Solver runtime from most recent treatment plan.

last_solvetime_exact
Second-pass solver runtime from most recent treatment plan.

last_x
Vector of beam intensities from most recent treatment plan.

last_x_exact
Second-pass beam intensities from most recent treatment plan.

no_run_check (*property_name*)
Test whether history includes any treatment plans.

Helper method for property getter methods.

Parameters **property_name** (str) – Name to use in error message if exception raised.

Returns None

Raises ValueError – If no treatment plans exist in history, i.e., *PlanningHistory.runs* has length zero.

tag_last (*tag*)
Tag most recent treatment plan in history.

Parameters **tag** – Name to apply to most recently added treatment plan. Plan can then be retrieved with slicing syntax:

```
# (history is a :class:`PlanningHistory` instance)
history[tag]
```

Returns None

Raises ValueError – If no treatment plans exist in history.

class history.**RunOutput**
Record of solver outputs associated with a treatment planning run.

optimal_variables
dict – Dictionary of optimal variables returned by solver. At a minimum, has entries for the beam intensity vectors for the first-pass and second-pass solver runs. May include entries for:

- x (beam intensities),
- y (voxel doses),
- mu (dual variable for constraint x>= 0), and
- nu (dual variable for constraint Ax == y).

optimal_dvh_slopes
dict – Dictionary of optimal slopes associated with the convex restriction of each percentile-type dose constraint. Keyed by constraint ID.

solver_info
dict – Dictionary of solver information. At a minimum, has entries solver run time (first pass/restricted constraints, and second pass/exact constraints).

solvetime
Run time for first-pass solve (restricted dose constraints).

solvetime_exact
Run time for second-pass solve (exact dose constraints).

x
Optimal beam intensities from first-pass solve.

x_exact
Optimal beam intensities from second-pass solve.

class history.**RunProfile**(*structures=None*, *use_slack=True*, *use_2pass=False*, *gamma='default'*)
Record of solver input associated with a treatment planning run.

use_slack
bool – True if solver allowed to construct convex problem with slack variables for each dose constraint.

use_2pass
bool – True if solver requested to construct and solve two problems, one incorporating convex restrictions of all percentile-type dose constraints, and a second problem formulating exact constraints based on the feasible output of the first solver run.

objectives
dict – Dictionary of objective data associated with each structure in plan, keyed by structure labels.

constraints
dict – Dictionary of constraint data for each dose constraint on each structure in plan, keyed by constraint ID.

gamma
Master scaling applied to slack penalty term in objective when dose constraint slacks allowed.

pull_constraints(*structures*)
Extract and store dictionaries of constraint data from structures.

Parameters **structures** – Iterable collection of Structure objects.

Returns None

pull_objectives(*structures*)
Extract and store dictionaries of objective data from structures.

Parameters **structures** – Iterable collection of Structure objects.

Returns None

class history.**RunRecord**(*structures=None*, *use_slack=True*, *use_2pass=False*, *gamma='default'*)

profile
RunProfile – Record of the objective weights, dose constraints, and relevant solver options passed to the convex solver prior to planning.

output
RunOutput – Output from the solver, including optimal beam intensities, i.e., the treatment plan.

plotting_data
dict – Dictionary of plotting data from case, with entries corresponding to the first (and potentially only) plan formed by the solver, as well as the exact-constraint version of the same plan, if the two-pass planning method was invoked.

feasible
Solver feasibility flag from solver output.

info
Solver information from solver output.

nonzero_beam_count
Number of active beams in first-pass solution.

nonzero_beam_count_exact
Number of active beams in second-pass solution.

solvetime
Run time for first-pass solve (restricted dose constraints).

solvetime_exact
Run time for second-pass solve (exact dose constraints).

x
Optimal beam intensitites from first-pass solution.

x_exact
Optimal beam intensitites from second-pass solution.

x_pass1
Alias for *RunRecord.x*.

x_pass2
Alias for *RunRecord.x_exact*.

Visualization

Dose volume histogram plotting utilities.

Provides `CasePlotter` for conveniently plotting DVH curve data generated by calling `Case.plan()`.

If `matplotlib` *is* available, plotting types such as `CasePlotter` types are defined normally.

This switch allows `conrad` to install, load and operate without Python plotting capabilities, and exempts `matplotlib` from being a load-time requirement.

Saving and Loading Cases

Python Module Index

Index

T

U

V

W

X

Y