pymedphys Documentation

Release 0.6.0

Simon Biggs, Matt Jennings, Paul King, Matthew Sobolewski

Mar 19, 2019

Getting Started

CHAPTER 1

Description

The PyMedPhys Project aims to develop a common core package for Medical Physics in Python and foster an ecosystem of interoperable Medical Physics packages. It is inspired by our physics peers in astronomy and their Astropy Project.

This package is available on PyPi (https://pypi.org/project/pymedphys/), GitHub (https://github.com/pymedphys/pymedphys), and conda-forge (https://anaconda.org/conda-forge/pymedphys). Documentation is at https://pymedphys.com.

CHAPTER 2

Our Team

PyMedPhys is what it is today due to its maintainers and developers. The currently active developers and maintainers of PyMedPhys are given below in alphabetical order along with their affiliation:

- **Matthew Jennings**
  - Royal Adelaide Hospital, Australia
- **Matthew Sobolewski**
  - Riverina Cancer Care Centre, Australia
  - Northern Beaches Cancer Care, Australia
- **Paul King**
  - Anderson Regional Cancer Center, United States
- **Simon Biggs**
  - Riverina Cancer Care Centre, Australia

We want you on this list. We want you, clinical Medical Physicist, to join our team. We want you if you have a desire to create and validate a toolbox we can all use. We want you even if all you feel comfortable contributing to is documentation.

The aim of PyMedPhys is that it will be developed by an open community of contributors. We use a shared copyright model that enables all contributors to maintain the copyright on their contributions. All code is licensed under the AGPLv3+ with additional terms from the Apache-2.0 license.

CHAPTER 3

Beta stage development

These libraries are currently under beta level development. Be prudent with the code in this library.

Throughout the lifetime of this library the following disclaimer will always hold:

> In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

CHAPTER 4

Installation

For instructions on how to install see the documentation at https://pymedphys.com/en/latest/getting-started/installation.html.

CHAPTER 5

Contributing

See the contributor documentation at https://pymedphys.com/en/latest/developer/contributing.html if you wish to create and validate open source Medical Physics tools together.

5.1 Installation

5.1.1 Conda from Conda-Forge

To install use the Anaconda Python distribution with the conda-forge channel

```
conda config --add channels conda-forge
conda install pymedphys
```

5.1.2 Pip from PyPi

You can of course also use pip to install, but you may have trouble with some of the dependencies without conda

```
pip install pymedphys
```

5.1.3 Bleeding edge with GitHub

If you would like to have a bleeding edge installation of pymedphys use the following commands to install the master branch from GitHub.

```
git clone https://github.com/pymedphys/pymedphys.git
cd pymedphys

conda config --add channels conda-forge
conda install pymedphys --only-deps
pip install -e .
```

To use this method you will need to have git on your machine. Instructions for installing git and chocolatey can be found within the relevant section of the contributor docs.

5.2 Licensing Notes

5.2.1 The Licence Agreement

5.2.2 Benefits of using a copyleft license in Medical Physics

For more information on why you as a Medical Physicist might want to use the AGPL-3.0+ license read the the Benefits of AGPL-3.0+ for Medical Physics.

5.2.3 Justification for the inclusion of additional terms

A significant and justifiable fear within the Medical Physics community is that should code be shared the author of the code may be liable for negligence.

Within Australian courts if there is any ambiguity in liability exclusion clauses they will be interpreted narrowly. If liability for negligence is not expressly excluded it may not be read as excluded within an Australian court (https://eprints.qut.edu.au/7404/1/open_source_book.pdf page 80). The same is true for clauses which seek to exclude liability for consequential loss.

The AGPL-3.0+ (nor the MIT license) does not explicitly mention negligence anywhere within its license text. The Apache-2.0 does. The AGPL-3.0+ in Section 7 does define allowable additional terms. The negligence clauses within the Apache-2.0 fall under these allowable additional terms so, as such, they have been included.

There are also other desirable features of the Apache-2.0 license such as contribution, trademark, and warranty requirements. These were also included.

5.2.4 A note about the code sharing license requirement

If you only ever use this code internally within your company to create your own programs the only people who need to have access to the source code are those users whom you distribute the program to. Therefore you do not need to share your code outside of your company if your only users are within your company.

However there are significant benefits from sharing your code with the community. Please read the Benefits of AGPL-3.0+ for Medical Physics.

5.3 MU Density

Determine MU Density given a range of formats.

5.3.1 Available Functions

```
>>> from pymedphys.mudensity import (
...     calc_mu_density)
```

5.3.2 MU Density

Calculate the MU Density given mu, mlc, and jaw control points.

Warning: Although this is a useful tool in the toolbox for patient specific IMRT QA, in and of itself it is not a sufficient stand in replacement. This tool does not verify that the reported dose within the treatment planning system is delivered by the Linac.

Deficiencies or limitations in the agreement between the treatment planning system's beam model and the Linac delivery will not be able to be highlighted by this tool. An example might be an overly modulated beam with many thin sweeping strips, the Linac may deliver those control points with positional accuracy but if the beam model in the TPS cannot sufficiently accurately model the dose effects of those MLC control points the dose delivery will not sufficiently agree with the treatment plan. In this case however, this tool will say everything is in agreement.

It also may be the case that due to a hardware or calibration fault the Linac itself may be incorrectly reporting its MLC and/or Jaw postions. In this case the logfile record can agree exactly with the planned positions while the true real world positions be in significant deviation.

The impact of these issues may be able to be limited by including with this tool an automated independent IMRT 3-D dose calculation tool as well as a daily automated MLC/jaw logfile to EPI to baseline agreement test that moves the EPI so as to measure the full set of leaf pairs and the full range of MLC and Jaw travel.

```
from pymedphys.mudensity import calc_mu_density, get_grid, display_mu_density

leaf_pair_widths = (5, 5, 5)
max_leaf_gap = 10

mu = [0, 2, 5, 10]
mlc = [
    [
        [1, 1],
        [2, 2],
        [3, 3]],
    [
        [2, 2],
        [3, 3],
        [4, 4]],
    [
```

(continues on next page)

(continued from previous page)

```
        [-2, 3],
        [-2, 4],
        [-2, 5]],
    [
        [0, 0],
        [0, 0],
        [0, 0]]]

jaw = [
    [7.5, 7.5],
    [7.5, 7.5],
    [-2, 7.5],
    [0, 0]
]

grid = get_grid(max_leaf_gap=max_leaf_gap, leaf_pair_widths=leaf_pair_widths)

mu_density = calc_mu_density(
    mu, mlc, jaw, max_leaf_gap=max_leaf_gap, leaf_pair_widths=leaf_pair_widths)

display_mu_density(grid, mu_density)
```

```
import matplotlib.pyplot as plt

from pymedphys.mudensity import single_mlc_pair

mlc_left = (-2.3, 3.1)  # (start position, end position)
mlc_right = (0, 7.7)

x, mu_density = single_mlc_pair(mlc_left, mlc_right)
plt.plot(x, mu_density, '-o')
plt.title('A single MLC')
plt.xlabel('MLC direction (mm)')
plt.ylabel('Open fraction')
```

5.3.3 API

pymedphys.mudensity.**calc_mu_density**(*mu, mlc, jaw, grid_resolution=1, max_leaf_gap=400, leaf_pair_widths=(5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5), min_step_per_pixel=10*)

Determine the MU Density.

Both jaw and mlc positions are defined in bipolar format for each control point. A negative value indicates travel over the isocentre. All positional arguments are defined at the isocentre projection with the units of mm.

Parameters

mu [numpy.ndarray] 1-D array containing an MU value for each control point.

mlc [numpy.ndarray] 3-D array containing the MLC positions

axis 0: control point
axis 1: mlc pair

axis 2: leaf bank

jaw [numpy.ndarray] 2-D array containing the jaw positions.

axis 0: control point
axis 1: diaphragm

grid_resolution [float, optional] The calc grid resolution. Defaults to 1 mm.

max_leaf_gap [float, optional] The maximum possible distance between opposing leaves. Defaults to 400 mm.

leaf_pair_widths [tuple, optional] The widths of each leaf pair in the MLC limiting device. The number of entries in the tuples defines the number of leaf pairs. Each entry itself defines that particular leaf pair width. Defaults to 80 leaf pairs each 5 mm wide.

min_step_per_pixel [int, optional] The minimum number of time steps used per pixel for each control point. Defaults to 10.

Returns

mu_density [numpy.ndarray] 2-D array containing the calculated mu density.

axis 0: jaw direction
axis 1: mlc direction

Examples

```
>>> import numpy as np
>>>
>>> from pymedphys.mudensity import (
...     calc_mu_density, get_grid, display_mu_density)
>>>
>>> leaf_pair_widths = (5, 5, 5)
>>> max_leaf_gap = 10
>>> mu = np.array([0, 2, 5, 10])
>>> mlc = np.array([
...     [
...         [1, 1],
...         [2, 2],
...         [3, 3]
...     ],
...     [
...         [2, 2],
...         [3, 3],
...         [4, 4]
...     ],
...     [
...         [-2, 3],
...         [-2, 4],
...         [-2, 5]
...     ],
...     [
...         [0, 0],
...         [0, 0],
...         [0, 0]
...     ]
... ])
>>> jaw = np.array([
```

(continues on next page)

(continued from previous page)

```
...      [7.5, 7.5],
...      [7.5, 7.5],
...      [-2, 7.5],
...      [0, 0]
... ])
>>>
>>> grid = get_grid(
...     max_leaf_gap=max_leaf_gap, leaf_pair_widths=leaf_pair_widths)
>>> grid['mlc']
array([-5., -4., -3., -2., -1.,  0.,  1.,  2.,  3.,  4.,  5.])
>>>
>>> grid['jaw']
array([-8., -7., -6., -5., -4., -3., -2., -1.,  0.,  1.,  2.,  3.,  4.,
        5.,  6.,  7.,  8.])
>>>
>>> mu_density = calc_mu_density(
...     mu, mlc, jaw, max_leaf_gap=max_leaf_gap,
...     leaf_pair_widths=leaf_pair_widths)
>>> display_mu_density(grid, mu_density)
>>>
>>> np.round(mu_density, 1)
array([[0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ],
       [0. , 0. , 0. , 0.3, 1.9, 2.2, 1.9, 0.4, 0. , 0. , 0. ],
       [0. , 0. , 0. , 0.4, 2.2, 2.5, 2.2, 0.6, 0. , 0. , 0. ],
       [0. , 0. , 0. , 0.4, 2.4, 2.8, 2.5, 0.8, 0. , 0. , 0. ],
       [0. , 0. , 0. , 0.4, 2.5, 3.1, 2.8, 1. , 0. , 0. , 0. ],
       [0. , 0. , 0. , 0.4, 2.5, 3.4, 3.1, 1.3, 0. , 0. , 0. ],
       [0. , 0. , 0.4, 2.3, 3.2, 3.7, 3.7, 3.5, 1.6, 0. , 0. ],
       [0. , 0. , 0.4, 2.3, 3.2, 3.8, 4. , 3.8, 1.9, 0.1, 0. ],
       [0. , 0. , 0.4, 2.3, 3.2, 3.8, 4.3, 4.1, 2.3, 0.1, 0. ],
       [0. , 0. , 0.4, 2.3, 3.2, 3.9, 5.2, 4.7, 2.6, 0.2, 0. ],
       [0. , 0. , 0.4, 2.3, 3.2, 3.8, 5.4, 6.6, 3.8, 0.5, 0. ],
       [0. , 0.3, 2.2, 3. , 3.5, 4. , 5.1, 7.5, 6.7, 3.9, 0.5],
       [0. , 0.3, 2.2, 3. , 3.5, 4. , 4.7, 6.9, 6.7, 3.9, 0.5],
       [0. , 0.3, 2.2, 3. , 3.5, 4. , 4.5, 6.3, 6.4, 3.9, 0.5],
       [0. , 0.3, 2.2, 3. , 3.5, 4. , 4.5, 5.6, 5.7, 3.8, 0.5],
       [0. , 0.3, 2.2, 3. , 3.5, 4. , 4.5, 5.1, 5.1, 3.3, 0.5],
       [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ]])
```

MU Density from a Mosaiq record

```
>>> from pymedphys.mudensity import (
...     calc_mu_density, get_grid, display_mu_density)
>>>
>>> from pymedphys.msq import (
...     mosaiq_connect, multi_fetch_and_verify_mosaiq)
>>>
>>> def mu_density_from_mosaiq(msq_server_name, field_id):
...     with mosaiq_connect(msq_server_name) as cursor:
...         delivery_data = multi_fetch_and_verify_mosaiq(
...             cursor, field_id)
...
...
...     mu = delivery_data.monitor_units
...     mlc = delivery_data.mlc
...     jaw = delivery_data.jaw
...
```

(continues on next page)

(continued from previous page)

```
...     grid = get_grid()
...     mu_density = calc_mu_density(mu, mlc, jaw)
...     display_mu_density(grid, mu_density)
>>>
>>> mu_density_from_mosaiq('a_server_name', 11111) # doctest: +SKIP
```

MU Density from a logfile at a given filepath

```
>>> from pymedphys.mudensity import (
...     calc_mu_density, get_grid, display_mu_density)
>>>
>>> from pymedphys.trf import delivery_data_from_logfile
>>>
>>> def mu_density_from_logfile(filepath):
...     delivery_data = delivery_data_from_logfile(filepath)
...
...     mu = delivery_data.monitor_units
...     mlc = delivery_data.mlc
...     jaw = delivery_data.jaw
...
...     grid = get_grid()
...     mu_density = calc_mu_density(mu, mlc, jaw)
...     display_mu_density(grid, mu_density)
>>>
>>> mu_density_from_logfile(r"a/path/goes/here") # doctest: +SKIP
```

pymedphys.mudensity.**calc_single_control_point**(*mlc, jaw, delivered_mu=1, leaf_pair_widths=(5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5), grid_resolution=1, min_step_per_pixel=10*)

Calculate the MU Density for a single control point.

Examples

```
>>> import numpy as np
>>> from pymedphys.mudensity import (
...     calc_single_control_point, display_mu_density)
>>>
>>> leaf_pair_widths = (2, 2)
>>> mlc = np.array([
...     [
...         [1, 1],
...         [2, 2],
...     ],
...     [
...         [2, 2],
...         [3, 3],
...     ]
... ])
>>> jaw = np.array([
...     [1.5, 1.2],
```

(continues on next page)

(continued from previous page)

```
...     [1.5, 1.2]
... ])
>>> grid, mu_density = calc_single_control_point(
...     mlc, jaw, leaf_pair_widths=leaf_pair_widths)
>>> display_mu_density(grid, mu_density)
>>>
>>> grid['mlc']
array([-3., -2., -1.,  0.,  1.,  2.,  3.])
>>>
>>> grid['jaw']
array([-1.5, -0.5,  0.5,  1.5])
>>>
>>> np.round(mu_density, 2)
array([[0.  , 0.07, 0.43, 0.5 , 0.43, 0.07, 0.  ],
       [0.  , 0.14, 0.86, 1.  , 0.86, 0.14, 0.  ],
       [0.14, 0.86, 1.  , 1.  , 1.  , 0.86, 0.14],
       [0.03, 0.17, 0.2 , 0.2 , 0.2 , 0.17, 0.03]])
```

pymedphys.mudensity.**single_mlc_pair**(*left_mlc*, *right_mlc*, *grid_resolution=1*, *min_step_per_pixel=10*)

Calculate the MU Density of a single leaf pair.

Examples

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>>
>>> from pymedphys.mudensity import single_mlc_pair
>>>
>>> mlc_left = (-2.3, 3.1)  # (start position, end position)
>>> mlc_right = (0, 7.7)
>>>
>>> x, mu_density = single_mlc_pair(mlc_left, mlc_right)
>>> fig = plt.plot(x, mu_density, '-o')
>>>
>>> x
array([-2., -1.,  0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.])
>>>
>>> np.round(mu_density, 3)
array([0.064, 0.244, 0.408, 0.475, 0.53 , 0.572, 0.481, 0.352, 0.224,
       0.096, 0.004])
```

pymedphys.mudensity.**get_grid**(*max_leaf_gap=400*, *grid_resolution=1*, *leaf_pair_widths=(5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5)*)

Get the MU Density grid for plotting purposes.

Examples

See *pymedphys.mudensity.calc_mu_density*.

pymedphys.mudensity.**find_relevant_control_points**(*mu*)

Removes control points that will not contribute to the MU Density.

pymedphys.mudensity.**display_mu_density**(*grid*, *mu_density*, *grid_resolution=None*)
Prints a colour plot of the MU Density.

Examples

See *pymedphys.mudensity.calc_mu_density*.

5.4 Gamma Calculation

A range of functions for calculating gamma.

5.4.1 Available Functions

```
>>> from pymedphys.gamma import (
...     gamma_shell, gamma_dicom, gamma_filter_numpy, gamma_filter_brute_force)
```

5.4.2 API

pymedphys.gamma.**gamma_shell**(*coords_reference*, *dose_reference*, *coords_evaluation*, *dose_evaluation*, *dose_percent_threshold*, *distance_mm_threshold*, *lower_percent_dose_cutoff=20*, *interp_fraction=10*, *max_gamma=inf*, *local_gamma=False*, *global_normalisation=None*, *skip_once_passed=False*, *random_subset=None*, *ram_available=None*, *quiet=False*)
Compare two dose grids with the gamma index.

Parameters

coords_reference [tuple] The reference coordinates.

dose_reference [np.array] The reference dose grid. Each point in the reference grid becomes the centre of a Gamma ellipsoid. For each point of the reference, nearby evaluation points are searched at increasing distances.

coords_evaluation [tuple] The evaluation coordinates.

dose_evaluation [np.array] The evaluation dose grid. Evaluation here is defined as the grid which is interpolated and searched over at increasing distances away from each reference point.

dose_percent_threshold [float] The percent dose threshold

distance_mm_threshold [float] The gamma distance threshold. Units must match of the coordinates given.

lower_percent_dose_cutoff [float, optional] The percent lower dose cutoff below which gamma will not be calculated. This is only applied to the reference grid.

interp_fraction [float, optional] The fraction which gamma distance threshold is divided into for interpolation. Defaults to 10 as recommended within <http://dx.doi.org/10.1118/1.2721657>. If a 3 mm distance threshold is chosen this default value would mean that the evaluation grid is interpolated at a step size of 0.3 mm.

max_gamma [float, optional] The maximum gamma searched for. This can be used to speed up calculation, once a search distance is reached that would give gamma values larger than this parameter, the search stops. Defaults to `np.inf`

local_gamma Designates local gamma should be used instead of global. Defaults to False.

global_normalisation [float, optional] The dose normalisation value that the percent inputs calculate from. Defaults to the maximum value of `dose_reference`.

random_subset [int, optional] Used to only calculate a random subset of the reference grid. The number chosen is how many random points to calculate.

ram_available [int, optional] The number of bytes of RAM available for use by this function. Defaults to 0.8 times your total RAM as determined by psutil.

quiet [bool, optional] Used to quiet informational printing during function usage. Defaults to False.

Returns

gamma [np.ndarray] The array of gamma values the same shape as that given by the reference coordinates and dose.

5.5 Electron Factor Modelling

Model insert factors and parameterise inserts as equivalent ellipses.

5.5.1 Available Functions

```
>>> from pymedphys.electronfactors import (
...     parameterise_insert, spline_model, calculate_deformability,
...     spline_model_with_deformability,
...     calculate_percent_prediction_differences,
...     visual_alignment_of_equivalent_ellipse)
```

5.5.2 API

pymedphys.electronfactors.**parameterise_insert**(*x*, *y*, *callback=None*)

Return the parameterisation of an insert given x and y coords.

pymedphys.electronfactors.**spline_model**(*width_test*, *ratio_perim_area_test*, *width_data*, *ratio_perim_area_data*, *factor_data*)

Return the result of the spline model.

The bounding box is chosen so as to allow extrapolation. The spline orders are two in the width direction and one in the perimeter/area direction. For justification on using this method for modelling electron insert factors see the *Methods: Bivariate spline model* section within <http://dx.doi.org/10.1016/j.ejmp.2015.11.002>.

Parameters

width_test [np.ndarray] The width point(s) which are to have the electron insert factor interpolated.

ratio_perim_area_test [np.ndarray] The perimeter/area which are to have the electron insert factor interpolated.

width_data [np.ndarray] The width data points for the relevant applicator, energy and ssd.

ratio_perim_area_data [np.ndarray] The perimeter/area data points for the relevant applicator, energy and ssd.

factor_data [np.ndarray] The insert factor data points for the relevant applicator, energy and ssd.

Returns

result [np.ndarray] The interpolated electron insert factors for width_test and ratio_perim_area_test.

pymedphys.electronfactors.**calculate_deformability**(*x_test*, *y_test*, *x_data*, *y_data*, *z_data*)

Return the result of the deformability test.

This function takes an array of test points and loops over `_single_calculate_deformability`.

The deformability test applies a shift to the spline to determine whether or not sufficient information for modelling is available. For further details on the deformability test see the *Methods: Defining valid prediction regions of the spline* section within <http://dx.doi.org/10.1016/j.ejmp.2015.11.002>.

Parameters

x_test [np.ndarray] The x coordinate of the point(s) to test

y_test [np.ndarray] The y coordinate of the point(s) to test

x_data [np.ndarray] The x coordinate of the model data to test

y_data [np.ndarray] The y coordinate of the model data to test

z_data [np.ndarray] The z coordinate of the model data to test

Returns

deformability [float] The resulting deformability between 0 and 1 representing the ratio of deviation the spline model underwent at the point in question by introducing an outlier at the point in question.

pymedphys.electronfactors.**spline_model_with_deformability**(*width_test*, *ratio_perim_area_test*, *width_data*, *ratio_perim_area_data*, *factor_data*)

Return the spline model for points with sufficient deformability.

Calls both `spline_model` and `calculate_deformabilty` and then adjusts the result so that points with deformability greater than 0.5 return `np.nan`.

Parameters

width_test [np.ndarray] The width point(s) which are to have the electron insert factor interpolated.

ratio_perim_area_test [np.ndarray] The perimeter/area which are to have the electron insert factor interpolated.

width_data [np.ndarray] The width data points for the relevant applicator, energy and ssd.

ratio_perim_area_data [np.ndarray] The perimeter/area data points for the relevant applicator, energy and ssd.

factor_data [np.ndarray] The insert factor data points for the relevant applicator, energy and ssd.

Returns

model_factor [np.ndarray] The interpolated electron insert factors for width_test and ratio_perim_area_test with points outside the valid prediction region set to `np.nan`.

`pymedphys.electronfactors.`**`calculate_percent_prediction_differences`**(*width_data*, *ratio_perim_area_data*, *factor_data*)

Return the percent prediction differences.

Calculates the model factor for each data point with that point removed from the data set. Used to determine an estimated uncertainty for prediction.

Parameters

width_data [np.ndarray] The width data points for a specific applicator, energy and ssd.

ratio_perim_area_data [np.ndarray] The perimeter/area data points for a specific applicator, energy and ssd.

factor_data [np.ndarray] The insert factor data points for a specific applicator, energy and ssd.

Returns

percent_prediction_differences [np.ndarray] The predicted electron insert factors for each data point with that given data point removed.

`pymedphys.electronfactors.`**`visual_alignment_of_equivalent_ellipse`**(*x*, *y*, *width*, *length*, *callback*)

Visually align the equivalent ellipse to the insert.

5.6 DICOM Toolbox

A Dicom toolbox built ontop of the pydicom library.

5.6.1 Available Functions

```
>>> from pymedphys.dicom import (
...     anonymise_dicom_dataset,
...     extract_iec_patient_xyz,
...     extract_iec_fixed_xyz,
...     extract_dicom_patient_xyz,
...     load_dose_from_dicom,
...     load_xyz_from_dicom,
...     find_dose_within_structure,
...     create_dvh,
...     get_structure_aligned_cube)
```

5.6.2 API

`pymedphys.dicom.`**`anonymise_dicom_dataset`**(*ds*, *replace_values=True*, *delete_private_tags=True*, *keywords_to_keep=None*, *ignore_unknown_tags=False*, *copy=True*)

A simple tool to anonymise a DICOM file.

Parameters

ds The DICOM file to be anonymised. *ds* must represent a valid DICOM file in the form of a *pydicom Dataset* - ordinarily returned by *pydicom.dcmread()*.

replace_values [bool, optional] If set to *True*, anonymised tag values will be replaced with dummy "anonymous" values. This is often required for successful reading of anonymised DICOM files in commercial software. If set to False, anonymised tags are simply given empty string values. Defaults to *True*.

delete_private_tags [bool, optional] A boolean to flag whether or not to remove all private (non-standard) DICOM tags from the DICOM file. These may also contain identifying information. Defaults to *True*.

keywords_to_keep [sequence, optional] A sequence of DICOM keywords (corresponding to tags) to exclude from anonymisation. Empty by default.

ignore_unknown_tags [bool, optional] If *pydicom* has updated its DICOM dictionary, this function will raise an error since a new identifying tag may have been introduced. Set to *True* to ignore this error. Defaults to *False*.

Returns

ds_anon An anonymised copy of the input DICOM file as a *pydicom Dataset*

Raises

ValueError Raised if *ignore_unknown_tags* is set to False and unrecognised, non-private DICOM tags are detected in *ds*

pymedphys.dicom.**extract_iec_patient_xyz**(*ds*)

Returns the x, y and z coordinates of a DICOM RT Dose file's dose grid in the IEC patient coordinate system

Parameters

ds A *pydicom Dataset* - ordinarily returned by *pydicom.dcmread()*. Must represent a valid DICOM RT Dose file.

Returns

(x, y, z) A tuple containing three *ndarrays* corresponding to the *x*, *y* and *z* coordinates of the DICOM RT Dose file's dose grid in the IEC patient coordinate system [1]:

x corresponds to the patient's left-right axis and increases toward the left hand side of the patient.

y corresponds to the patient's superoinferior axis and increases toward the head of the patient.

z corresponds to the patient's anteroposterior axis and increases to the anterior side of the patient.

Notes

Supported scan orientations [2]:

Orientation	ImageOrientationPatient
Feet First Decubitus Left	[0, 1, 0, 1, 0, 0]
Feet First Decubitus Right	[0, -1, 0, -1, 0, 0]
Feet First Prone	[1, 0, 0, 0, -1, 0]
Feet First Supine	[-1, 0, 0, 0, 1, 0]
Head First Decubitus Left	[0, -1, 0, 1, 0, 0]
Head First Decubitus Right	[0, 1, 0, -1, 0, 0]
Head First Prone	[-1, 0, 0, 0, -1, 0]
Head First Supine	[1, 0, 0, 0, 1, 0]

References

[1], [2]

pymedphys.dicom.**extract_iec_fixed_xyz**(*ds*)

Returns the x, y and z coordinates of a DICOM RT Dose file's dose grid in the IEC fixed coordinate system.

Parameters

ds A *pydicom Dataset* - ordinarily returned by *pydicom.dcmread()*. Must represent a valid DICOM RT Dose file.

Returns

(x, y, z) A tuple containing three *ndarrays* corresponding to the *x*, *y* and *z* coordinates of the DICOM RT Dose file's dose grid in the IEC fixed coordinate system [1]:

x An *np.ndarray* of coordinates on the horizontal plane that runs orthogonal to the gantry axis of rotation. The x axis increases towards a viewer's right hand side when facing the gantry standing at the isocentre. The isocentre has an x value of 0.

y An *np.ndarray* of coordinates on the horizontal plane that runs parallel to the gantry axis of rotation. The positive direction of the y axis travels from the isocentre to the gantry. The isocentre has a y value of 0.

z An *np.ndarray* of coordinates aligned vertically and passing through the isocentre. The positive direction of the z axis travels upwards and the isocentre has a z value of 0.

Notes

Supported scan orientations [2]:

Orientation	ImageOrientationPatient
Feet First Decubitus Left	[0, 1, 0, 1, 0, 0]
Feet First Decubitus Right	[0, -1, 0, -1, 0, 0]
Feet First Prone	[1, 0, 0, 0, -1, 0]
Feet First Supine	[-1, 0, 0, 0, 1, 0]
Head First Decubitus Left	[0, -1, 0, 1, 0, 0]
Head First Decubitus Right	[0, 1, 0, -1, 0, 0]
Head First Prone	[-1, 0, 0, 0, -1, 0]
Head First Supine	[1, 0, 0, 0, 1, 0]

References

[1], [2]

pymedphys.dicom.**extract_dicom_patient_xyz**(*ds*)

Returns the x, y and z coordinates of a DICOM RT Dose file's dose grid in the DICOM patient coordinate system

Parameters

ds A *pydicom Dataset* - ordinarily returned by *pydicom.dcmread()*. Must represent a valid DICOM RT Dose file.

Returns

(x, y, z) A tuple containing three *ndarrays* corresponding to the *x*, *y* and *z* coordinates of the DICOM RT Dose file's dose grid in the DICOM patient coordinate system [1]:

x corresponds to the patient's left-right axis and increases to the left hand side of the patient.

y corresponds to the patient's anteroposterior axis and increases toward the posterior side of the patient.

z corresponds to the patient's superoinferior axis and increases toward the head of the patient.

Notes

Supported scan orientations [2]:

Orientation	ImageOrientationPatient
Feet First Decubitus Left	[0, 1, 0, 1, 0, 0]
Feet First Decubitus Right	[0, -1, 0, -1, 0, 0]
Feet First Prone	[1, 0, 0, 0, -1, 0]
Feet First Supine	[-1, 0, 0, 0, 1, 0]
Head First Decubitus Left	[0, -1, 0, 1, 0, 0]
Head First Decubitus Right	[0, 1, 0, -1, 0, 0]
Head First Prone	[-1, 0, 0, 0, -1, 0]
Head First Supine	[1, 0, 0, 0, 1, 0]

References

[1], [2]

pymedphys.dicom.**load_dose_from_dicom**(*ds*, *set_transfer_syntax_uid=True*, *reshape=True*)

Extract the dose grid from a DICOM RT Dose file.

Deprecated since version 0.5.0: *load_dose_from_dicom* will be removed in a future version of PyMedPhys. It is replaced by *extract_dose*, which provides additional dose-related information and conforms to a new coordinate system handling convention.

pymedphys.dicom.**load_xyz_from_dicom**(*ds*)

Extract the coordinates of a DICOM RT Dose file's dose grid.

Deprecated since version 0.5.0: *load_xyz_from_dicom* will be removed in a future version of PyMedPhys. It is replaced by *extract_dicom_patient_coords*, *extract_iec_patient_coords* and *extract_iec_fixed_coords*, which explicitly work in their respective coordinate systems.

pymedphys.dicom.**find_dose_within_structure**(*structure*, *dcm_struct*, *dcm_dose*)

pymedphys.dicom.**create_dvh**(*structure*, *dcm_struct*, *dcm_dose*)

pymedphys.dicom.**get_structure_aligned_cube**(*x0: numpy.ndarray*, *structure_name: str*, *dcm_struct: pydicom.dataset.FileDataset*, *quiet=False*, *niter=10*)

Align a cube to a dicom structure set.

Designed to allow arbitrary references frames within a dicom file to be extracted via contouring a cube.

Parameters

x0 A 3x3 array with each row defining a 3-D point in space. These three points are used as initial conditions to search for a cube that fits the contours. Choosing initial values close to the structure set, and in the desired orientation will allow consistent results. See examples within pymedphys.geometry.cubify_cube_definition on what the effects of each of the three points are on the resulting cube.

structure_name The DICOM label of the cube structure

dcm_struct The pydicom reference to the DICOM structure file.

quiet [bool] Tell the function to not print anything. Defaults to False.

Returns

cube_definition_array Four 3-D points the define the vertices of the cube.

vectors The vectors between the points that can be used to traverse the cube.

Examples

```
>>> import numpy as np
>>> import pydicom
>>> from pymedphys.dicom import get_structure_aligned_cube
>>>
>>> struct_path = 'tests/data/dicom_struct/example_structures.dcm'
>>> dcm_struct = pydicom.dcmread(struct_path, force=True)
>>>
>>> x0 = np.array([
...     -270, -35, -20,
...     -270, 30, -20,
...     -260, -35, 40
... ])
>>>
>>> structure_name = 'ANT Box'
>>> cube_definition_array, vectors = get_structure_aligned_cube(
...     x0, structure_name, dcm_struct, quiet=True, niter=1)
>>> np.round(cube_definition_array)
array([[-276.,  -31.,  -16.],
       [-275.,   29.,  -17.],
       [-266.,  -31.,   43.],
       [-217.,  -32.,  -26.]])
>>>
>>> np.round(vectors, 1)
array([[ 0.7, 59.9, -0.5],
       [ 9.7,  0.4, 59.1],
       [59.1, -0.8, -9.7]])
```

5.7 Geometry Toolbox

A geometry toolbox.

5.7.1 Available Functions

```
>>> from pymedphys.geometry import cubify_cube_definition, plot_cube
```

5.7.2 API

pymedphys.geometry.**cubify_cube_definition**(*cube_definition*)

Convertes a set of 3-D points into the vertices that define a cube.

Each point is defined as a length 3 tuple.

Parameters

cube_definition [str] A list containing three 3-D points.

cube_definition[0]: The origin of the cube.
cube_definition[1]: Point that primarily determines the cube edge lengths.
cube_definition[2]: Point that primarily defines the cube rotation.

Returns

final_points A list containing four 3-D points on the vertices of a cube.

Examples

```
>>> import numpy as np
>>> from pymedphys.geometry import cubify_cube_definition
>>>
>>> cube_definition = [(0, 0, 0), (0, 1, 0), (0, 0, 1)]
>>> np.array(cubify_cube_definition(cube_definition))
array([[0., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.]])
```

The second point has primary control over the resulting edge lengths.

```
>>> cube_definition = [(0, 0, 0), (0, 3, 0), (0, 0, 1)]
>>> np.array(cubify_cube_definition(cube_definition))
array([[0., 0., 0.],
       [0., 3., 0.],
       [0., 0., 3.],
       [3., 0., 0.]])
```

The third point has control over the final cube rotation.

```
>>> cube_definition = [(0, 0, 0), (0, 1, 0), (1, 0, 0)]
>>> np.array(cubify_cube_definition(cube_definition))
array([[ 0.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 1.,  0.,  0.],
       [ 0.,  0., -1.]])
```

pymedphys.geometry.**plot_cube**(*cube_definition*)

5.8 xlwings Toolbox

xlwings is a powerful tool for implementing python script within Microsoft Excel.

5.8.1 Installation of PyMedPhys and xlwings

Download and install the latest Anaconda Python 3 version from here. This comes with xlwings included in the installation. Follow xlwings' guide on enabling UDFs within Excel.

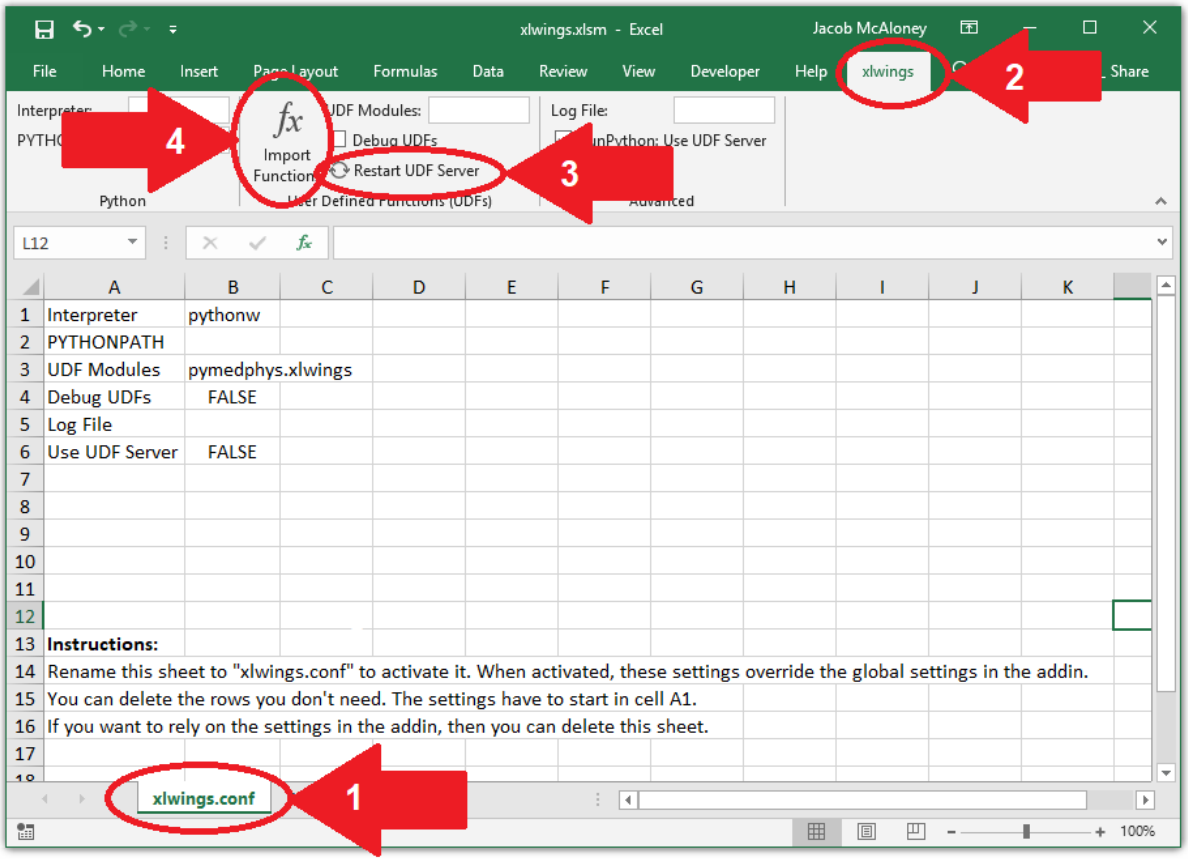
5.8.2 Setting up xlwings

If you wish to create a new xlwings file, use their template creation tool documented at https://docs.xlwings.org/en/stable/udfs.html#workbook-preparation. Otherwise you can use some PyMedPhys example spreadsheets (work in progress).

Within the template file navigate to the "xlwings.conf" tab of the spreadsheet (arrow 1 in below image) and ensure the interpreter and UDF modules are correct (see image below for reference). Rename the sheet to xlwings.conf to activate the file.

Now you can run Python scripts as defined in PyMedPhys in Excel. Whenever changes occur or if things don't appear to be working properly navigate to the xlwings add-on tab in the Excel ribbon (arrow 2 in below image). Click on "restart UDF servers" followed by "import functions", arrows 3 and 4 respectively in the below image. Then try to execute a function within the Excel spreadsheet.

If a file path or reference has changed since the last time the Excel file was saved functions running python within Excel may return false responses. This is usually cleared by re-running a few functions which should trigger the rest within the workbook to update.



5.9 Recommended Setup for Contribution

5.9.1 Assumptions

These instructions assume that you are using a Windows machine and that you have administrator rights on your machine. Although this document is tailored to Windows users, PyMedPhys itself works on Windows, macOS and Linux.

5.9.2 Your mission

Your mission, should you choose to accept it, is to complete all the tasks within this document. While doing so, please take notes of the pain points. Write down what feedback you have. By the end, instead of you emailing that feedback to me, we'd like you to use your new set up to edit this file and submit a merge request!

5.9.3 Python & Anaconda

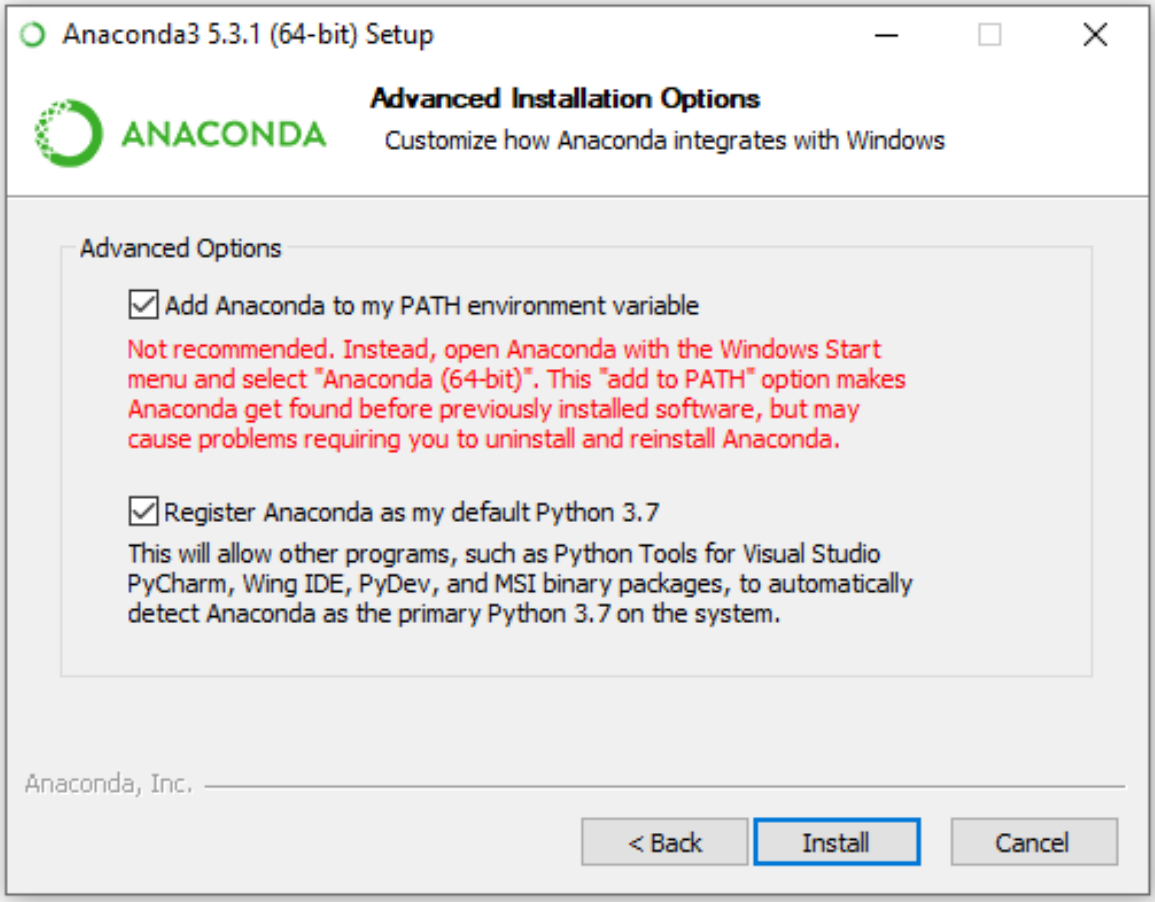
Anaconda is a free, open source, optimized Python (and R) distrubution. It includes:

- conda, a powerful package and environment management system.
- Python

- Over 100 automatically installed scientific packages (*numpy*, *scipy*, etc.) that have been tested to work well together, along with their dependencies.

Download the latest Anaconda Python **3** version from here

When installing Anaconda make sure to install it for your user only, and tick the option "add to path".



You might notice that Microsoft VS Code is available to be installed via the Anaconda installation. However, we recommend installing VS Code from its official install location as outlined below in *Text Editor*. The Anaconda installer does not provide the opportunity to tick the "Open with Code" boxes detailed below.

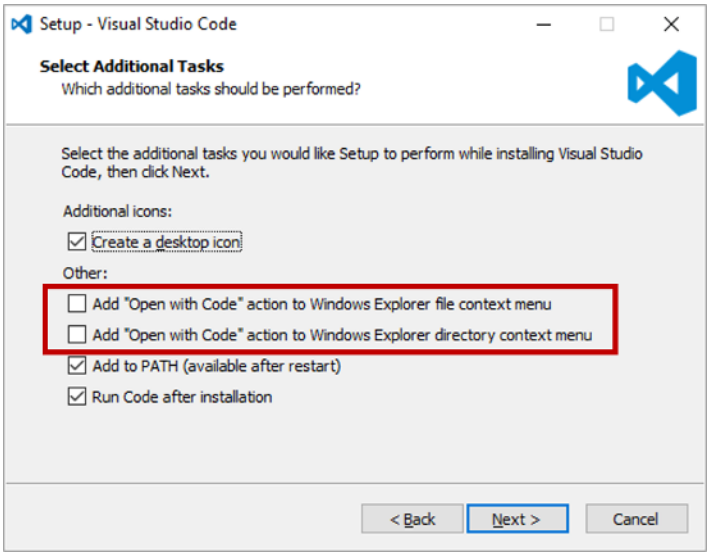
Once you have installed Anaconda, add the conda-forge channel to your machine using the following command in a new command prompt:

```
conda config --add channels conda-forge
```

5.9.4 Text Editor

Visual Studio Code is an excellent, free, open-source code editor. It comes with many great features for both Python and Git. You can download the official release here.

When installing VS Code, make sure to tick the "Open with Code" boxes:



You will need to install a few extensions in VS Code to complete your set up. This is very easy to do via the Extensions Marketplace once VS Code is installed. With VS Code running, access the marketplace by clicking this symbol on the left toolbar:

Search for the "Anaconda Extension Pack" and install it. Reload VS Code when installation has finished and you're ready to go with Python in VS Code!

We also recommend the "GitLens" extension to further enhance your VS Code experience! It comes with a number of useful tools for using Git within VS Code itself.

5.9.5 A Command Prompt

Cmder is a good command prompt that fills the massive gap on Windows machines. One would think that syntax highlighting, copy/paste, and window resizing would be commonplace in terminals everywhere - but apparently not! Nevertheless, cmder to the rescue. Install the mini version of cmder from here.

Once you've downloaded cmder, follow the steps given here to obtain the ability to open a terminal in any directory by right clicking in the file browser.

5.9.6 Chocolatey

Chocolatey is a package manager for Windows. It makes installing software development tools quite a breeze. Follow these instructions to install Chocolatey.

5.9.7 Git, Git LFS, and Yarn

Use Chocolatey within an administrator command prompt to install Git, Git LFS, and yarn like so:

```
choco install git git-lfs yarn
```

5.9.8 GitHub Account

Make a GitHub account here. Once you have an account, you will need commit rights to this repository in order to make contributions. Create an issue on GitHub within the PyMedPhys repository here and include "request for commit rights" or similar in your issue's content, along with "@SimonBiggs" and "@Centrus007" to ensure it is seen!

Whenever you wish to discuss anything about PyMedPhys, please create an issue on GitHub. It can be to ask for help, suggest a change, provide feedback, or anything else regarding PyMedPhys. Write "@" followed by someone's username if you would like to talk to someone specifically.

The real power of GitHub comes from Git itself. A great piece of Git documentation can be found here. Use this documentation to begin to get a feel for what Git is.

5.9.9 Some useful resources

At this point you might find some of the following resources useful:

- Numpy for Matlab users (Scipy.org).
- NumPy for MATLAB users (Mathesaurus).
- Playground and cheatsheet for learning Python.
- Don't be afraid to commit: Git and GitHub.
- Chapter 2 of The Pragmatic Programmer.

The "Don't be afraid to commit" resource will be invaluable for these next few steps.

5.9.10 Authenticate your computer to be able to access your GitHub account

Before setting your SSH keys, I recommend permanently setting your HOME variable. This can clear up some potentially confusing issues. Do this by running the following where *yourusername* is your Windows domain user name.

```
setx HOME "C:\Users\yourusername"
```

Follow these instructions to create and add an SSH key to your GitHub account. Since you already have ssh built into cmder, you can skip the first steps of the tutorial.

If all has gone well you should be able to run the following without being prompted for a password.

```
git clone git@github.com:pymedphys/pymedphys.git
cd pymedphys
```

This will download all of PyMedPhys to your local machine.

5.9.11 Set up nbstripout

nbstripout is used to make it so that you do not post Jupyter Notebook outputs online. Depending on how you use notebooks these outputs may contain private and/or sensitive information.

> **Warning:** In the event that you uninstall Python, it is possible that *nbstripout* ends up disabled. Stay prudent, and be extra cautious when working with sensitive information stored within a notebook in a Git repository.

To install *nbstripout* run the following within the pymedphys directory:

```
conda install nbstripout
nbstripout --install
nbstripout --status
```

Make sure that the output of *nbstipout –status* starts with:

```bash nbstripout is installed in repository `
```

5.9.12 Install the development version of PyMedPhys

Begin by installing the dependencies of the online version of PyMedPhys with conda. With cmder open in the pymedphys directory, run:

```
conda install pymedphys --only-deps
pip install -e .
```

5.9.13 Update this document

Note: If you've made it this far, well done!

Now that you've got this far, you have a copy of the code on your machine.

First thing's first: make a branch. If you don't know what that is, head on back over to Don't be afraid to commit and scrub up on your terminology.

To make a branch, open cmder in the pymedphys directory and run the following:

```
git checkout -b your-name-edit-contributing-document
```

Once you've run that you are now free to make some changes.

Right click on the top level pymedphys directory, and press "Open with Code". This document that you're reading is located at *docs/developer/contributing.rst*. Use VS Code to navigate to that file and begin making your changes.

Once your changes are complete, reopen your cmder and run:

```
git add -A
git commit -m "my first commit"
git push --set-upstream origin your-name-edit-contributing-document
```

Now, you have successfully sent your branch online.

Now you need to open a merge request. Open one here, select the source branch to be *pymedphys/your-name-edit-contributing-document* and set the target branch to be *pymedphys/master*.

At that point, we'll get notified and we can begin discussing the changes you've made.

Thank you! Welcome to the team!

5.10 Editing documentation

This documentation site uses a toolbox called Sphinx. This particular document aims to help contributors to improve the PyMedPhys documentation.

It presumes you have gone through the base contributing documentation.

5.10.1 Prerequisites

To work on the documentation on your machine you will need the following extra libraries installed:

```
pip install sphinx sphinx-autobuild numpydoc sphinx_rtd_theme
```

5.10.2 Starting a live update documentation server

Within the root pymedphys directory run the following command:

```
sphinx-autobuild docs docs/_build/html
```

Then within a web browser go to http://127.0.0.1:8000

You may now edit the documentation within the docs directory and see the changes live update within your browser.

5.10.3 Docstring extraction

Some documentation is best written right within a given function itself. This type of documentation is called a **docstring**. However, this documentation should also be available in the main documentation and it is exceptionally important that this isn't written more than once. Duplicating documentation increases software entropy. As time goes by, documentation updates may result in one or more copies being missed and becoming obsolete or inconsistent with the up-to-date copy. Even if all copies are updated correctly, unnecessary duplication adds to ongoing maintenance requirements. See the DRY programming philosophy for more on this.

To solve this problem, most of PyMedPhys' documentation is written as docstrings, which are then automatically extracted into the main documentation pages. For this to work properly, docstrings need to be formatted according to the numpy style. See the following sites for examples of how to conform to that style:

- Napoleon Docs - Example NumPy Style Python Docstrings
- numpydoc docstring guide

See existing examples within PyMedPhys for how to include new function docstrings into the main PyMedPhys documentation.

5.11 Physical File Structure and Design

5.11.1 John Lakos and Physical Design

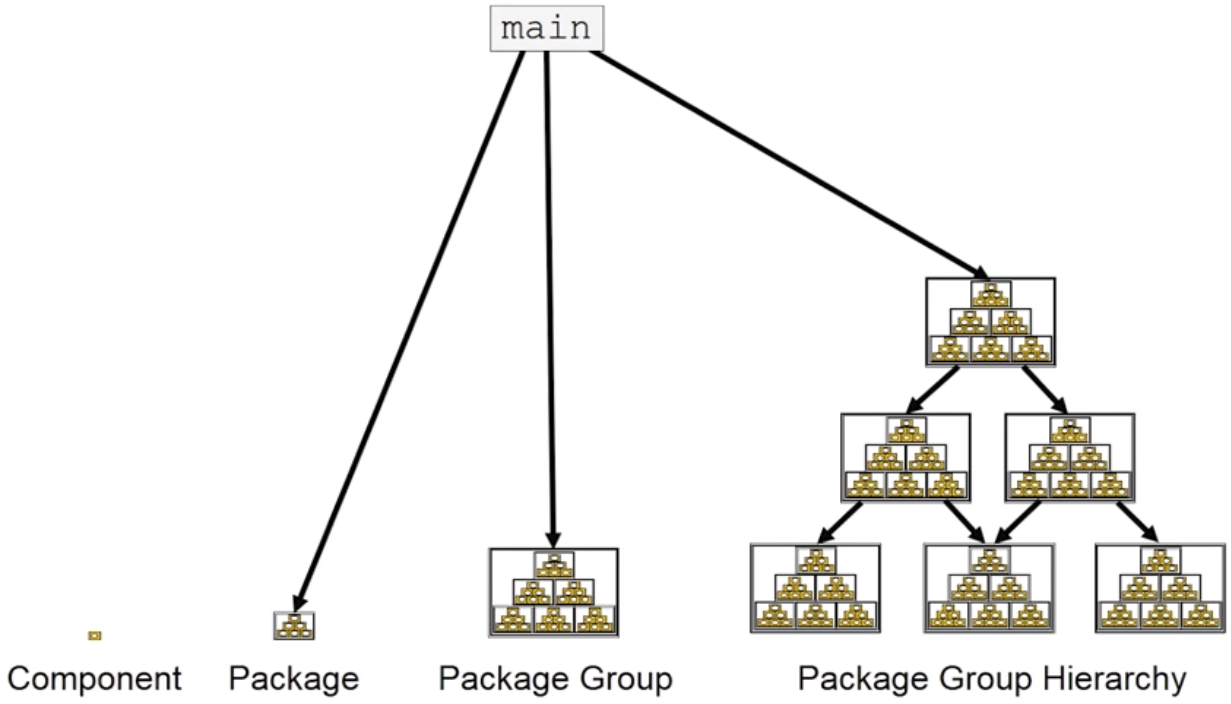
The physical design of PyMedPhys is inspired by John Lakos at Bloomberg, writer of Large-Scale C++ Software Design. He describes this methodology in a talk he gave which is available on YouTube:

The aim is to have an easy to understand hierarchy of component and package dependencies that continues to be easy to hold in ones head even when there are a very large number of these items.

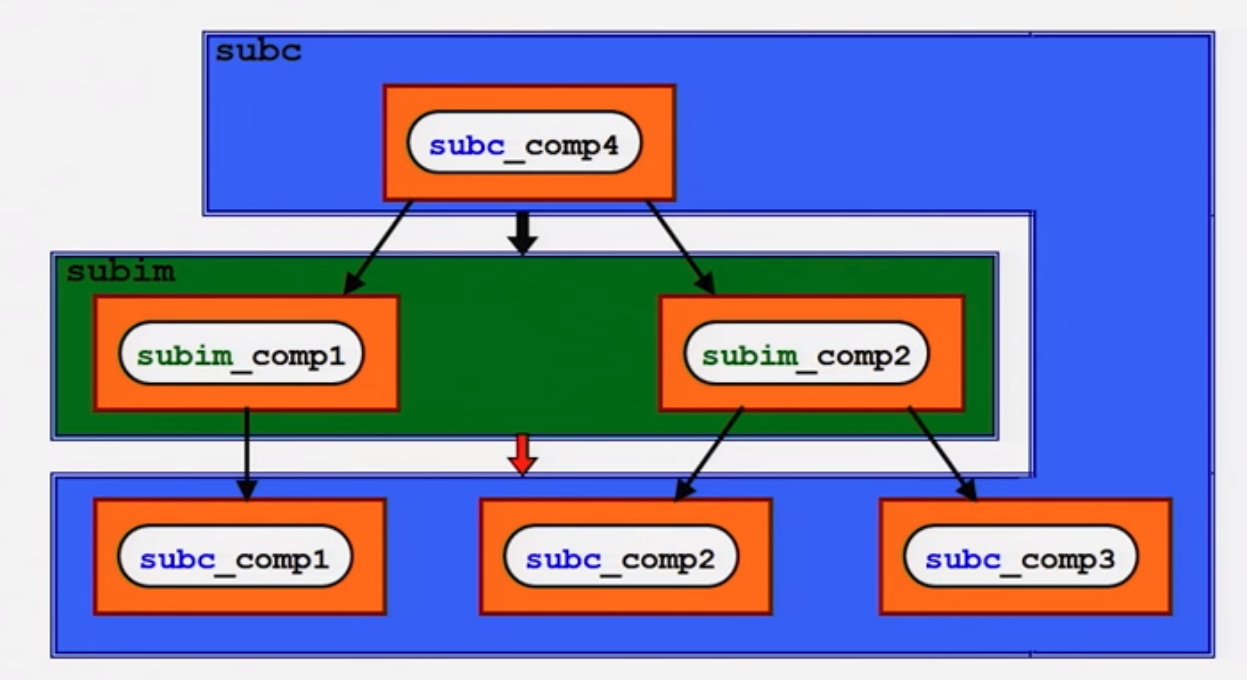
This is achieved by levelling. The idea is that in each type of aggregation there are only three levels, and each level can only depend on the levels lower than it. Never those higher, nor those the same level. So as such, Level 1 components or packages can only depend on external dependencies. Level 2 can depend on Level 1 or external, and then Level3 can depend ong Level 1, Level 2, or external.

John Lakos uses three aggregation terms, component, package, and package group. Primarily PyMedPhys avoids object oriented programming choosing functional methods where appropriate. However within Python, a single python file itself can act as a module object. This module object contains public and private functions (or methods) and largely acts like an object in the object oriented paradime. So the physical and logical component within PyMedPhys is being interpreted as a single *.py* file that contains a range of functions. A set of related components are levelled and grouped together in a package, and then the set of these packages make up the package group of PyMedPhys itself.

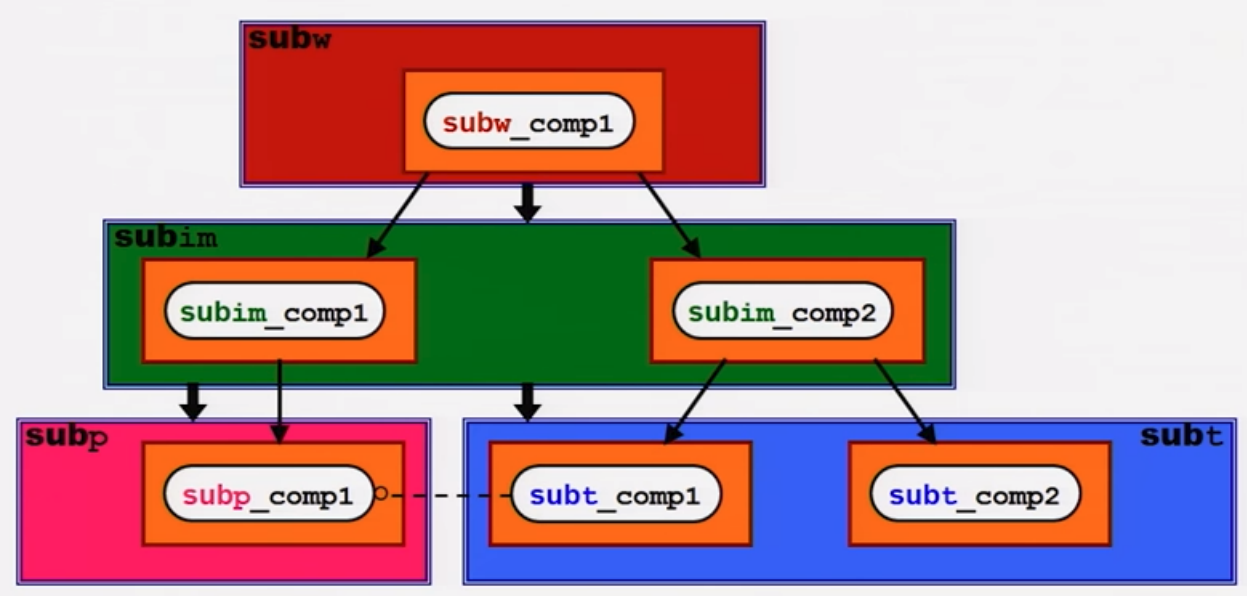
He presents the following diagram:



It is important that the packages themselves are levelled. See in the following image, even though the individual components themselves form a nice dependency tree, the packages to which those components belong end up interdependending on one another:



In this case, it might be able to be solved by appropriately dividing the components up into differently structured packages:



5.11.2 The Layout of PyMedPhys

All source code for PyMedPhys is contained within *src/pymedphys*. Packages are levelled and placed with respect to their dependency level by directory name. The packages within _pack1 only depend on external packages, packages within _pack2 only depend on _pack1 or external, and so on.

Inside each package, the component files are split up into directories by level1, level2, level3. Once again level1 can only depend on either an external dependency or a lower package. Those within level2 can depend on lower pacakges, external dependencies, or level1, and so on.

So that the api to PyMedPhys doesn't vary with the physical structure of the dependencies, each package is imported out to be directly accessable within the top namespace of *src/pymedphys*.

5.12 Benefits of AGPL-3.0+ for Medical Physics

The aim of this document is to outline some of the benefits of Medical Physics code being released under an open source license. In particular the benefits of the AGPL-3.0+ licence. An example of a well known code base that uses this open source licence is EGSnrc (https://www.nrc-cnrc.gc.ca/eng/solutions/advisory/egsnrc_index.html).

A summary of this license and what it entails is available at https://choosealicense.com/licenses/agpl-3.0/.

5.12.1 Software is less dependent on a single software developer

A significant issue with software that is built by a small software development team is the "bus factor". In the event that key team members are lost the software is no longer able to be easily maintained. By opening the code up to the community the number of people who understand the code base is increased. This reduces the dependence on any single employee for the ongoing maintenance of the code packages.

Additionally software development by single developers carry any weaknesses that developer has as a programmer and its only strengths will be limited to those of that programmer. Medical Physicists are generally not trained software engineers, and when it comes to programming we often have many weaknesses that need a light shined upon.

This was the justification provided by the radiation therapy QATrack+ team (http://qatrackplus.com/) for open sourcing their tools (http://randlet.com/static/downloads/papers/QATrack+%20Odette%20Cancer%20Centre.pdf).

5.12.2 Safer and higher quality software

By leveraging the community in software production there will be more minds, more programmers, and more readers of the code. Together there will be more combined time to implement software engineering best practices and there are physicists in the community who are keen to support best practice software engineering. The community will be able to fix bugs, implement programmatic testing, highlight security issues, and generally all round produce safer and higher quality software than a sole physicist turned programmer ever could.

5.12.3 Software that is more compatible with a wide range of systems

Users of open source code who are also programmers who have systems which differ to the author's will be able to improve compatibility issues themselves. This iterative process with the community makes it so that the software has a more seamless interaction with the range of systems in use.

An example of this is the Pylinac quality assurance tool (http://pylinac.readthedocs.io/en/latest/index.html). It was built by a physicist who works at a Varian site. Another physicist submitted code improvements to make the software tool compatible with Elekta (https://github.com/jrkerns/pylinac/pull/67).

5.12.4 Still possible to potentially monetise

The AGPL-3.0+ requires that code include programmatic build and installation instructions. This means that users who are also programmers will likely be able to put in the work to get the software running by themselves without payment to the authors. However in this case there is no one being paid to provide user support and the default is no included warranty.

As a result there still is likely a market for providing the full toolset in a user friendly, warranty included, batteries included, fully supported package. An example of a product and company that uses exactly this business model is the medical imaging DICOM server Orthanc which offers commercial services through the company Osimis (http://www.osimis.io/en/products.html).

There is also the opportunity for advertising revenue.

5.12.5 Protection against another actor creating a closed source competitor

By releasing the code under the AGPL-3.0+ license any future work that makes use of the open source code must also be released under the same license. This means that should another actor create a competing product to the original author's should they ever distribute the code they must also release all changes under the AGPL-3.0+ license along with that distribution. In effect unless that company is willing to release all of their own code that is bundled with the author's code to the community they cannot use the original source code to compete.

5.12.6 Improving our work as Physicists

The software we write is written for the purpose of improving our work as physicists. Having the community improve the software we write directly improves the quality of the work we undergo.

5.12.7 Improving the programming skill of employees through community feedback

Programming skills are significantly benefited by having competent programmers read the code and provide feedback. Community feedback on code itself is invaluable for improving the skill of the programmers writing the original code. The depth and breadth of feedback from the community on programming practices would unable to be matched within a small team.

5.12.8 Software that has more applications and more features

As members of the community have needs those with programming skill may extend the code that is provided to meet those needs. This results in the code having more applications and features than would be possible should the code be kept in house.

5.12.9 Improvements will by default be in a compatible format

Should the community create code built on top of the released software tools, those improvements will built on top of the format that is already implemented. This allows those improvements to be integrated within already implemented systems with minimal friction.

5.13 Release Notes

All notable changes to this project will be documented in this file.

This project adheres to Semantic Versioning.

5.13.1 Unreleased

Breaking Changes

- `anonymise_dicom` has been renamed to `anonymise_dicom_dataset`

New Features

- Implementing a suite of Dicom objects, currently a work in progress:
  - `DicomBase`, a base DICOM class that wraps `pydicom`'s `Dataset` object. This class includes additions such as an anonymisation method.
  - `DicomImage`, designed to hold a single DICOM image slice. Might someday contain methods such as `resample` and the like.
  - `DicomSeries`, a series of `DicomImage` objects creating a CT dataset.
  - `DicomStructure`, designed to house DICOM structure datasets.
  - `DicomPlan`, a class that holds RT plan DICOM datasets.
  - `DicomDose`, a class that to hold RT DICOM dose datasets. It has helper functions and parameters such as coordinate transforms built into it.
  - `DicomStudy`, a class designed to hold an interrelated set of `DicomDose`, `DicomPlan`, `DicomStructure`, and `DicomSeries`. Not every type is required to create a `DicomStudy`. Certain methods will be available on `DicomStudy` depending what is housed within it. For example having both `DicomDose` and `DicomStructure` should enable DVH based methods.
  - `DicomCollection`, a class that can hold multiple studies, interrelated or not. A common use case that will likely be implemented is `DicomCollection.from_directory(directory_path)` which would pull all DICOM files nested within a directory and sort them into `DicomStudy` objects based on their header UIDs.

Bug Fixes

- nil

Code Refactoring

- nil

Performance Improvements

- nil

5.13.2 0.6.0 – 2019/03/15

Breaking Changes

- All uses of "dcm" in directory names, module names, function names, etc. have been converted to "dicom". Anything that makes use of this code will need to be adjusted accordingly. Required changes include:

  - `pymedphys.dcm` -> `pymedphys.dicom`
  - `coords_and_dose_from_dcm()` -> `coords_and_dose_from_dicom()`
  - `dcmfromdict()` -> `dicom_dataset_from_dict()`
  - `gamma_dcm()` -> `gamma_dicom()`
- MU Density related functions are no longer available under the `pymedphys.coll` package, instead they are found within `pymedphys.mudensity` package.
- The DICOM coordinate extraction functions now return simple tuples rather than `Coords` namedtuples:
  - `extract_dicom_patient_xyz()`
  - `extract_iec_patient_xyz()`
  - `extract_iec_fixed_xyz()`

New Features

- DICOM anonymisation now permits replacing deidentified values with suitable "dummy" values. This helps to maintain compatibility with DICOM software that includes checks (beyond those specified in the DICOM Standard) of valid DICOM tag values. Replacing tags with dummy values upon anonymisation is now the default behaviour.
- A set of 3D coordinate transformation functions, including rotations (passive or active) and translations. Transformations may be applied to a single coordinate triplet (an `ndarray`) or a list of arbitrarily many coordinate triplets (a 3 x n `ndarray`). **NB**: Documentation forthcoming.

Code Refactoring

- All uses of `dcm` as a variable name for instances of PyDicom Datasets have been converted to `ds` to match PyDicom convention.

5.13.3 0.5.1 – 2019/01/05

New Features

- Began keeping record of changes in `changelog.md`

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Bibliography

[1] “IEC 61217:2.0 Radiotherapy equipment – Coordinates, movements and scales”

[2] O. McNoleg, “Generalized coordinate transformations for Monte Carlo (DOSXYZnrc and VMC++) verifications of DICOM compatible radiotherapy treatment plans”, arXiv:1406.0014, Table 1, https://arxiv.org/ftp/arxiv/papers/1406/1406.0014.pdf

[1] “IEC 61217:2.0 Radiotherapy equipment – Coordinates, movements and scales”

[2] O. McNoleg, “Generalized coordinate transformations for Monte Carlo (DOSXYZnrc and VMC++) verifications of DICOM compatible radiotherapy treatment plans”, arXiv:1406.0014, Table 1, https://arxiv.org/ftp/arxiv/papers/1406/1406.0014.pdf

[1] “C.7.6.2.1.1 Image Position and Image Orientation”, “DICOM PS3.3 2016a - Information Object Definitions”, http://dicom.nema.org/MEDICAL/dicom/2016a/output/chtml/part03/sect_C.7.6.2.html#sect_C.7.6.2.1.1

[2] O. McNoleg, “Generalized coordinate transformations for Monte Carlo (DOSXYZnrc and VMC++) verifications of DICOM compatible radiotherapy treatment plans”, arXiv:1406.0014, Table 1, https://arxiv.org/ftp/arxiv/papers/1406/1406.0014.pdf

Python Module Index

p

Index