pytrip98 Documentation

Release 3.8.0

Author

Nov 14, 2022

Contents

Contents:

CHAPTER 1

Getting Started with PyTRiP

Brief overview of PyTRiP98 and how to install it.

1.1 Introduction

PyTRiP98 is a python package for working with files generated by the treatment planning systems TRiP98 and VIRTUOS/VOXELPLAN. Dicom files are also to some extent supported.

PyTRiP will simplify importing and exporting files, processing the data, and also execute TRiP98 locally or remotely. Thereby it is possible to work with TRiP98 in e.g. a Windows environment, while accessing TRiP98 on a remote server. PyTRiP enables scripting for large parameters studies of treatment plans, and also more advanced and automized manipulation than what commercial treatment planning systems might allow.

Let us for instance assume, that one wants (for whatever reason) to reduce all Hounsfield units in a CT cube with a factor of two and write the result into a new file, this can be realized with a few lines of code.

```
>>> import pytrip as pt
>>> c = pt.CtxCube()
>>> c.read("tst000001.ctx")   # read a .ctx file
```

Where the first line imports the pytrip modules, the second line initialized the CtxCube object. The new object holds (among others) the read() method, which is then being used to load the CT data. Now let's work with the CT data:

```
>>> c *= 0.5   # reduce all HUs inside c with a factor of two
```

and write it to disk.

```
>>> c.write("out0000001.ctx") # write the new file.
```

And that all.

We may want to inspect the data, what is the largest and the smalles HU value found in the cube?

```
>>> print(c.cube.min())
>>> print(c.cube.max())
```

To see all available methods and attributes, one can run the

```
>>> dir(c)
```

command, or read the detailed documentation.

1.2 Quick Installation Guide

PyTRiP is available for python 2.7, 3.5 or later, and can be installed via pip. If you intend to use **pytripgui** you need the python 2.7 version.

We recommend that you run a modern Linux distribution, like: **Ubuntu 16.04** or newer, **Debian 9 Stretch** (currently known as testing) or any updated rolling release (archLinux, openSUSE tumbleweed). In that case, be sure you have **python** and **python-pip** installed. To get them on Debian or Ubuntu, type being logged in as normal user:

```
$ sudo apt-get install python-pip
```

To automatically download and install the pytrip library, type:

```
$ sudo pip install pytrip98
```

NOTE: the package is named **pytrip98**, while the name of library is **pytrip**.

This command will automatically download and install **pytrip** for all users in your system.

For more detailed instruction, see the *Detailed Installation Guide*

To learn how to install **pytripgui** graphical user interface, proceed to following document page: https://github.com/pytrip/pytripgui

1.3 Using PyTRiP

Once installed, the package can be imported at a python command line or used in your own python program with `import pytrip as pt`. See the examples directory for both kinds of uses. Also see the *User's Guide* for more details of how to use the package.

1.4 Support

Bugs can be submitted through the issue tracker. Besides the example directory, cookbook recipes are encouraged to be posted on the wiki page

1.5 Next Steps

To start learning how to use PyTRiP, see the *PyTRiP User's Guide*.

1.6 License

PyTRiP98 is licensed under GPLv3.

CHAPTER 2

Detailed Installation Guide

Installation guide is divided in two phases: checking the prerequisites and main package installation.

2.1 Prerequisites

PyTRiP works under Linux and Mac OSX operating systems.

First we need to check if Python interpreter is installed. Try if one of following commands (printing Python version) works:

```
$ python --version
$ python3 --version
```

At the time of writing Python language interpreter has two popular versions: 2.x (Python 2) and 3.x (Python 3) families. Command `python` invokes either Python 2 or 3, while `python3` can invoke only Python 3.

pytrip supports most of the modern Python versions, mainly: 2.7, 3.5 - 3.10. Check if your interpreter version is supported.

If none of `python` and `python3` commands are present, then Python interpreter has to be installed.

We suggest to use the newest version available (from 3.x family).

Python installers can be found at the python web site (http://python.org/download/).

PyTRiP also relies on these packages:

- NumPy – Better arrays and data processing.
- matplotlib – Needed for plotting.
- paramiko – Needed for remote execution of TRiP98 via SSH.

and if they are not installed beforehand, these will automatically be fetched by pip.

2.2 Installing using pip (all platforms)

The easiest way to install PyTRiP98 is using pip:

```
.. note::
```

Pip comes pre-installed with Python newer than 3.4 and 2.7 (for 2.x family)

2.2.1 Administrator installation (root access)

Administrator installation is very simple, but requires to save some files in system-wide directories (i.e. */usr*):

```
$ sudo pip install pytrip98
```

To upgrade the **pytrip** to newer version, simply type:

```
$ sudo pip install --upgrade pytrip98
```

To completely remove **pytrip** from your system, use following command:

```
$ sudo pip uninstall pytrip98
```

Now all **pytrip** commands should be installed for all users:

```
$ cubeslice --help
```

2.2.2 User installation (non-root access)

User installation will put the **pytrip** under hidden directory *$HOME/.local*.

To install the package, type in the terminal:

```
$ pip install pytrip98 --user
```

If *pip* command is missing on your system, replace *pip* with *pip3* in abovementioned instruction.

To upgrade the **pytrip** to newer version, simply type:

```
$ pip install --upgrade pytrip98 --user
```

To completely remove **pytrip** from your system, use following command:

```
$ pip uninstall pytrip98
```

In most of modern systems all executables found in *$HOME/.local/bin* directory can be called like normal commands (i.e. *ls*, *cd*). It means that after installation you should be able to simply type in terminal:

```
$ cubeslice --help
```

If this is not the case, please prefix the command with *$HOME/.local/bin* and call it in the following way:

```
$ $HOME/.local/bin/cubeslice --help
```

CHAPTER 3

PyTRiP User's Guide

pytrip object model, description of classes, examples

3.1 Using PyTRiP as a library

The full potential of PyTRiP is exposed when using it as a library.

Using the *dir()* and *help()* methods, you may explore what functions are available, check also the index and module tables found in this documentation.

CT and Dose data are handled by the "CtxCube" and "DosCube" classes, respectively. Structures (volume of interests) are handled by the VdxCube class. For instance, when a treatment plan was made the resulting 3D dose distribution (and referred to as a "DosCube").

```
>>> import pytrip as pt
>>> dc = pt.DosCube()
>>> dc.read("foobar.dos")
```

You can display the entire doscube by simply printing its string (str or repr) value:

```
>>> dc
....
```

We recommend you to take a look at the *Examples* and browse the modindex page.

3.2 Converters

A few converters based on PyTRiP are supplied as well. These converters are:

trip2dicom.py converts a Voxelplan formatted file to a Dicom file.

dicom2trip.py converts a Dicom file to a Voxelplan formatted file.

cubeslice.py Generates .png files for each slice found in the given cube.

gd2dat.py Converts a GD formatted plot into a stripped ASCII-file

gd2agr.py Converts a GD formatted plot into a a xmgrace formatted plot.

rst2sobp.py Converts a raster scan file to a file which can be read by FLUKA or SHIELD-HIT12A.

CHAPTER 4

Examples

Code snippets demonstrating PyTRiP capabilities.

4.1 Example 00 - Cube arithmetic

This example demonstrates simple arithmetic on dose- and LET-cubes. Two dose cubes from two fields are summed to generate a new total dose cube.

The two LET-cubes from the two fields are combined to calculate the total dose-averaged LET in the resulting treatment plan. All data are saved to disk.

```python
"""
Simple example of how to do arithmetic on Cube objects in PyTRiP.
"""
import pytrip as pt

# sum two dose cubes, write result:
print("Two half boxes: out.dos")
d1 = pt.DosCube()
d2 = pt.DosCube()
d1.read("box052000.dos")
d2.read("box053000.dos")
d = (d1 + d2)
d.write("out.dos")

# print minium and maximum value found in cubes
print(d1.cube.min(), d1.cube.max())
print(d2.cube.min(), d2.cube.max())

# calculate new dose average LET cube
l1 = pt.LETCube()
l2 = pt.LETCube()
l1.read("box052000.dosemlet.dos")
l2.read("box053000.dosemlet.dos")
```

(continues on next page)

(continued from previous page)

```python

let = ((d1 * l1) + (d2 * l2)) / (d1 + d2)
let.write("out.dosemlet.dos")
```

4.2 Example 01 - Handling structures

This example shows how one can select a region inside a CTX data cube using a VDX file, and perform some manipulation of it.

```python
"""
This example shows how to use contours to select volume of interests inside a CTX␣
↪cube. The VOI is then manipulated.
"""

import logging
import pytrip as pt

# by default logging in PyTRiP98 is disabled, here we enable it to see it with basic␣
↪INFO level
logging.basicConfig(level=logging.DEBUG)

# first define some paths and other important parameters
ctx_path = "/home/bassler/Projects/CTdata/TST000/TST000000.ctx"
vdx_path = "/home/bassler/Projects/CTdata/TST000/TST000000.vdx"
my_target_voi = "GTV"

# load CT cube
my_ctx = pt.CtxCube()
my_ctx.read(ctx_path)

# load VOIs
my_vdx = pt.VdxCube(my_ctx)  # my_vdx is the object which will hold all volumes of␣
↪interest and the meta information
my_vdx.read(vdx_path)  # load the .vdx file
print(my_vdx.voi_names())  # show us all VOIs found in the .vdx file

# Select the requested VOI from the VdxCube object
target_voi = my_vdx.get_voi_by_name(my_target_voi)

# get_voi_cube() returns a DosCube() object, where all voxels inside the VOI holds␣
↪the value 1000, and 0 elsewhere.
voi_cube = target_voi.get_voi_cube()

# Based on the retrieved DosCube() we calculate a three dimensional mask object,
# which assigns True to all voxels inside the Voi, and False elsewhere.
mask = (voi_cube.cube == 1000)

# "The mask object and the CTX cube have same dimensions (they are infact inherited␣
↪from the same top level class).
# Therefore we can apply the mask cube to the ctx cube and work with the values.
# For instance we can set all HUs to zero within the Voi:
my_ctx.cube[mask] = 0
# or add 100 to all HUs of the voxels inside the mask:
# my_ctx.cube[mask] += 100
```

(continues on next page)

(continued from previous page)

```python

# save masked CT to the file in current directory
masked_ctx = "masked.ctx"
my_ctx.write(masked_ctx)
```

Working with dose cubes is fully analogous to the CTX cubes.

4.3 Example 02 - TRiP execution

In this example, we demonstrate how to actually perform a treatment plan using TRiP98. Most of the lines concern with the setup of TRiP.

```python
"""
This example demonstrates how to load a CT cube in Voxelplan format, and the associated contours.
Then a plan is prepared and optimized using TRiP98.
"""
import os
import logging

import pytrip as pt
import pytrip.tripexecuter as pte

logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)  # give some output on what is going on.

# Please adjust these paths according to location of the patient data (CT and contouring) and TRiP98 installation.
# Fist we specify the directory where all our files are:
wdir = "/home/user/workspace"  # working dir must exist.
patient_dir = "/home/user/data/yoda"
trip_path = "/home/user/usr/trip98"

# In TRiP, the patient "TST000" would typically carry the filename "TST000000"
patient_name = "TST000000"

# so we can construc the paths to the CTX and VDX files like this:
ctx_path = os.path.join(patient_dir, patient_name + ".ctx")
vdx_path = os.path.join(patient_dir, patient_name + ".vdx")

# Next we load the CT cube:
c = pt.CtxCube()
c.read(ctx_path)

# And load the contours
v = pt.VdxCube(c)
v.read(vdx_path)

# we may print all contours found in the Vdx file, if we want to
print(v.voi_names())

# We need to specify where the kernel files can be found. The location may depend on the ion we
# want to treat with. This example sets up a kernel model for C-12 ions with a 3 mm Ripple Filter.
```

(continues on next page)

(continued from previous page)

```python
mykernel = pte.KernelModel()
mykernel.projectile = pte.Projectile("C", a=12)
mykernel.ddd_path = trip_path + "/DATA/DDD/12C/RF3MM/*"
mykernel.spc_path = trip_path + "/DATA/SPC/12C/RF3MM/*"
mykernel.sis_path = trip_path + "/DATA/SIS/19981218.sis"
mykernel.rifi_thickness = 3.0  # 3 mm ripple filter. (Only for documentaiton, will not affect dose optimization.)
mykernel.rifi_name = "GSI_1D_3mm"  # Additional free text for documentation.
mykernel.comment = "Carbon-12 ions with 3 mm 1D Ripple Filter"

# Ok, we have the Contours, the CT cube and dose kernels ready. Next we must prepare a plan.
# We may choose any basename for the patient. All output files will be named using
# this basename.
plan = pte.Plan(basename=patient_name, default_kernel=mykernel)

# Plan specific data:
plan.hlut_path = trip_path + "/DATA/HLUT/19990218.hlut"  # Hounsfield lookup table location
plan.dedx_path = trip_path + "/DATA/DEDX/20000830.dedx"  # Stopping power tables
plan.working_dir = wdir

# Set the plan target to the voi called "CTV"
plan.voi_target = v.get_voi_by_name('CTV')

# some optional plan specific parameters (if not set, they will all be zero by default)
plan.bolus = 0.0  # No bolus is applied here. Set this to some value, if you are to optimize very shallow tumours.
plan.offh2o = 1.873  # Some offset mimicing the monitoring ionization chambers and exit window of the beam nozzle.

# Next we need to specify at least one field, and add that field to the plan.
field = pte.Field(kernel=mykernel)  # The ion speicies is selected by passing the corresponding kernel to the field.
field.basename = patient_name  # This name will be used for output filenames, if any field specific output is saved.
field.gantry = 10.0  # degrees
field.couch = 90.0  # degrees
field.fwhm = 4.0  # spot size in [mm]

print(field)  # We can print all parameters of this field, for checking.
plan.fields.append(field)  # attach field to plan. You may attach multiple fields.

# Next, set the flags for what output should be generated, when the plan has completed.
plan.want_phys_dose = True  # We want a physical dose cube, "TST000000.dos"
plan.want_bio_dose = False  # No biological cube (Dose * RBE)
plan.want_dlet = True  # We want to have the dose-averaged LET cube
plan.want_rst = False  # Print the raster scan files (.rst) for all fields.

# print(plan)  # this will print all plan parameters

te = pte.Execute(c, v)  # get the executer object, based on the given Ctx and Vdx cube.

# in the case that TRiP98 is not installed locally, you may have to enable remote execution:
```

(continues on next page)

(continued from previous page)

```
# te.remote = True
# te.servername = "titan.phys.au.dk"
# te.username = "bassler"
# te.password = "xxxxxxxx"  # you can set a password, but this is strongly␣
↪discouraged. Better to exchange SSH keys!
# te.remote_base_dir = "/home/bassler/test"
#
# Depending on the remote .bashrc_profile setup, it may be needed to specify the full␣
↪path
# for the remote TRiP installation. On some systems the $PATH is set, so this line␣
↪can be omitted,
# or shortened to just "TRiP98" :
# te.trip_bin_path = trip_path + "/bin/TRiP98"

te.execute(plan)  # this will run TRiP
# te.execute(plan, False)  # set to False, if TRiP98 should not be executed. Good for␣
↪testing.

# requested results can be found in
# plan.dosecubes[]
# and
# plan.letcubes[]
# and they are also saved to working_dir
```

CHAPTER 5

Credits

5.1 Development

- Niels Bassler - Stockholm University, Sweden
- Leszek Grzanka - IFJ-PAN, Poland <leszek.grzanka@gmail.com>
- Jakob Toftegaard - Aarhus University Hospital, Denmark

5.2 Contributors

None yet. Why not be the first?

5.3 How to cite PyTRiP98

If you use PyTRiP for your research, please cite:

5.4 Others

PyTRiP is using cntr.c code from matplotlib v2.1.2 (code is copied into PyTRiP sources).

CHAPTER 6

pytrip package

6.1 Subpackages

6.1.1 pytrip.res package

Submodules

pytrip.res.contour module

pytrip.res.interpolate module

pytrip.res.point module

pytrip.res.utils module

6.1.2 pytrip.tripexecuter package

Submodules

pytrip.tripexecuter.execparser module

pytrip.tripexecuter.execute module

pytrip.tripexecuter.executor_logger module

pytrip.tripexecuter.field module

pytrip.tripexecuter.kernel module

pytrip.tripexecuter.plan module

pytrip.tripexecuter.projectile module

6.1.3 pytrip.utils package

Submodules

pytrip.utils.bevlet2oer module

pytrip.utils.cubeslice module

pytrip.utils.dicom2trip module

pytrip.utils.dvhplot module

pytrip.utils.gd2agr module

pytrip.utils.gd2dat module

pytrip.utils.rst2sobp module

pytrip.utils.rst_plot module

pytrip.utils.spc2pdf module

pytrip.utils.trip2dicom module

6.2 Submodules

6.2.1 pytrip.ctx module

6.2.2 pytrip.cube module

6.2.3 pytrip.ddd module

6.2.4 pytrip.dicomhelper module

6.2.5 pytrip.dos module

6.2.6 pytrip.error module

6.2.7 pytrip.field module

6.2.8 pytrip.file_parser module

6.2.9 pytrip.let module

6.2.10 pytrip.paths module

6.2.11 pytrip.raster module

6.2.12 pytrip.util module

6.2.13 pytrip.vdx module

CHAPTER 7

Indices and tables

- genindex
- modindex
- search