Why we should use Jupyter notebook in medical image analysis

Serena Bonaretti, PhD
Transparent Musculoskeletal Research
https://tmskr.github.io/

serena.bonaretti.research@gmail.com
@serenabonaretti

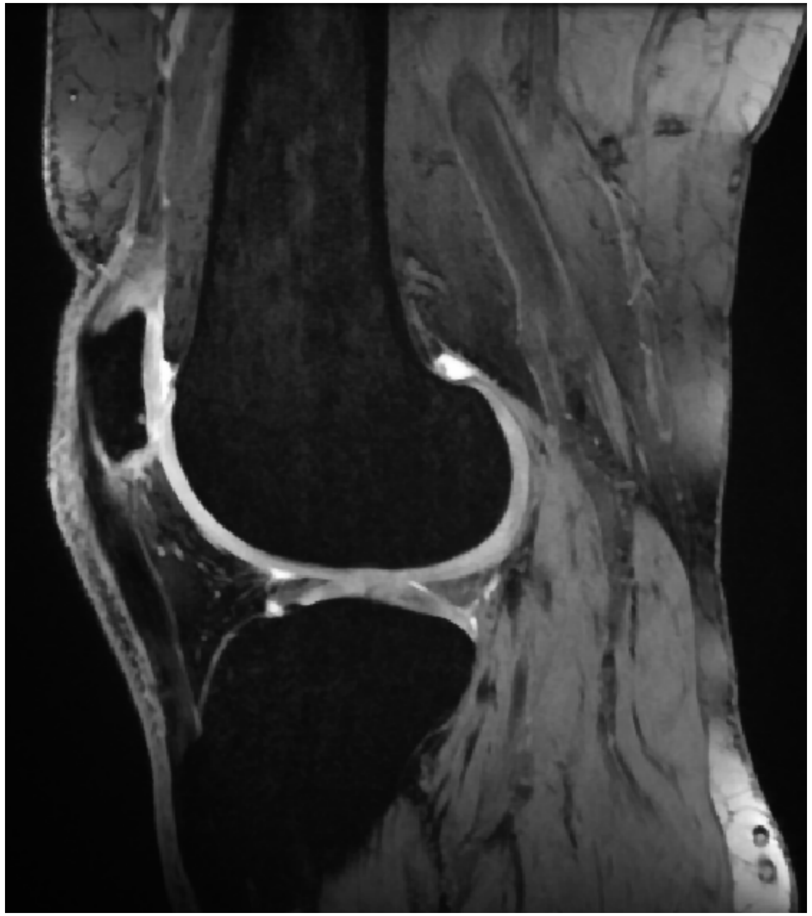
Slides at: tinyurl.com/TOR2020jupyter

tMSKr

Sharing my experience with Jupyter notebook in musculoskeletal imaging

I had to segment and analyze some knee images…

- Collaboration with scientists with limited experience in medical imaging

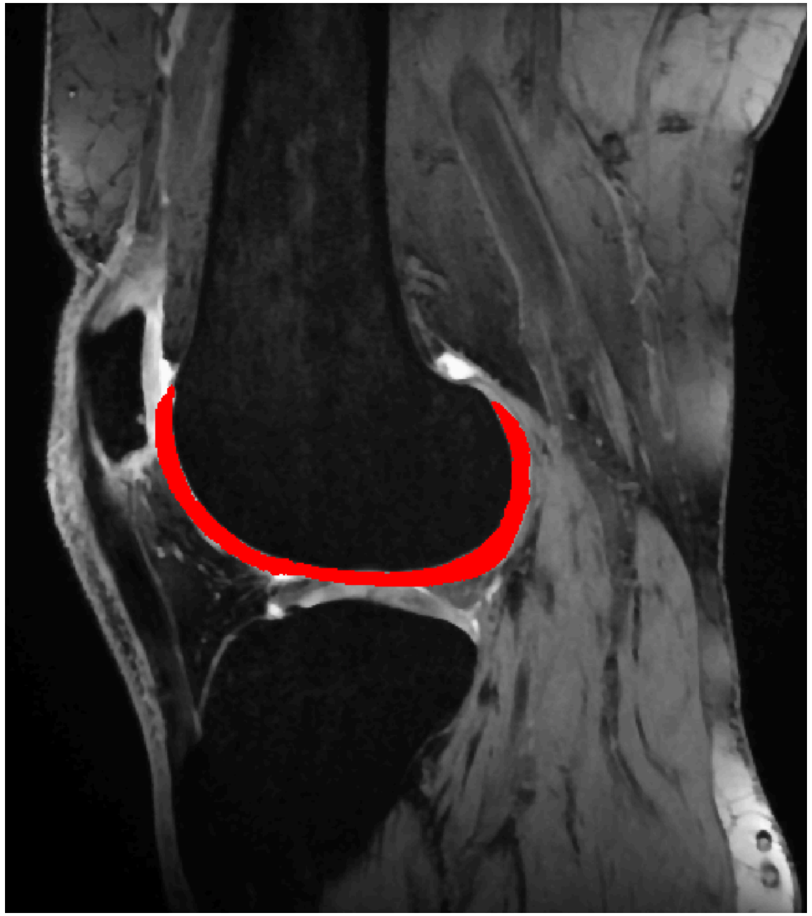


I had to segment and analyze some knee images…

- Collaboration with scientists with limited experience in medical imaging

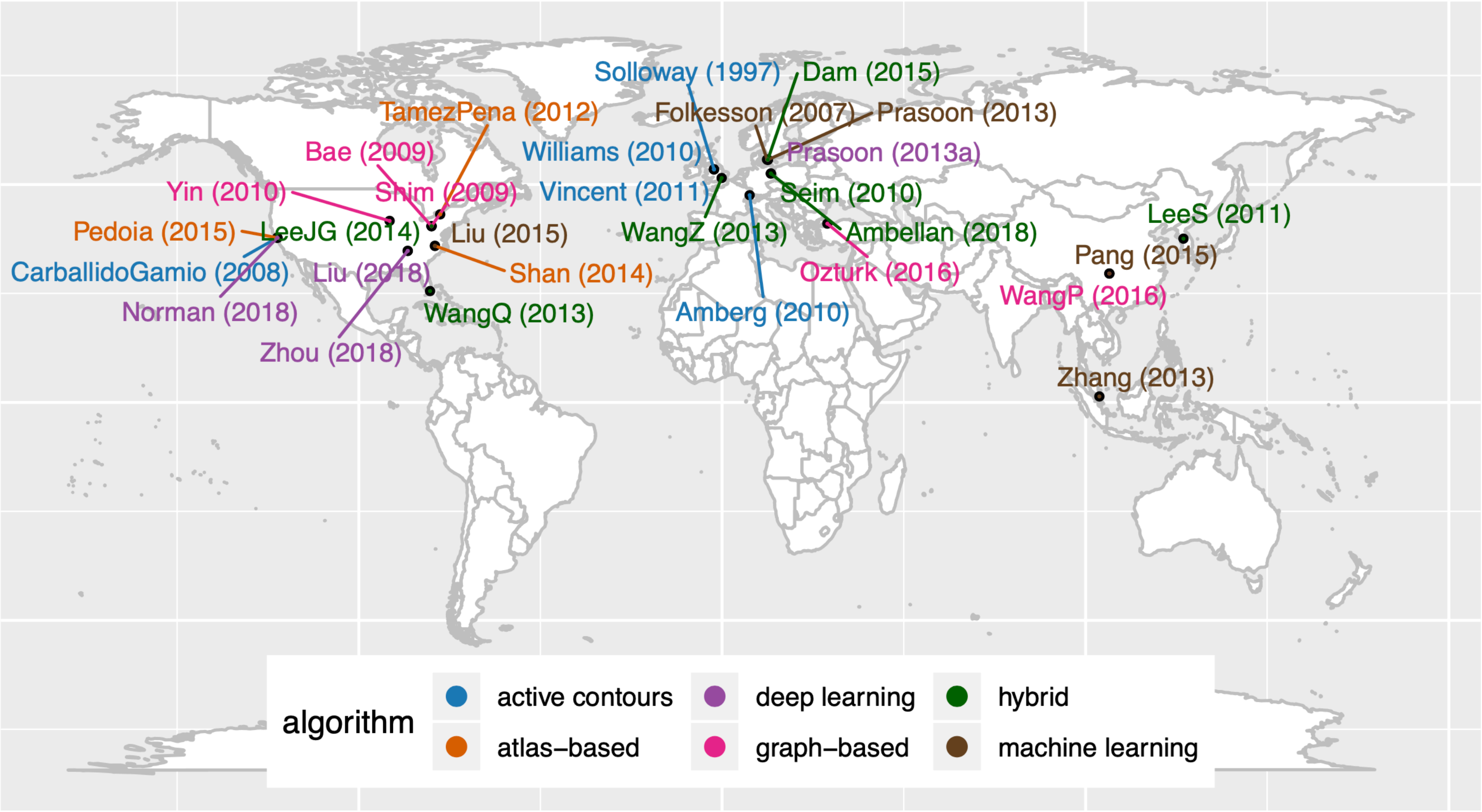
- They ***needed code*** to extract measures of OA progression
  - Femoral cartilage thickness and relaxation times

➡ First thing: Segmentation!



I looked for existing algorithms around…

Literature map of femoral knee cartilage segmentation

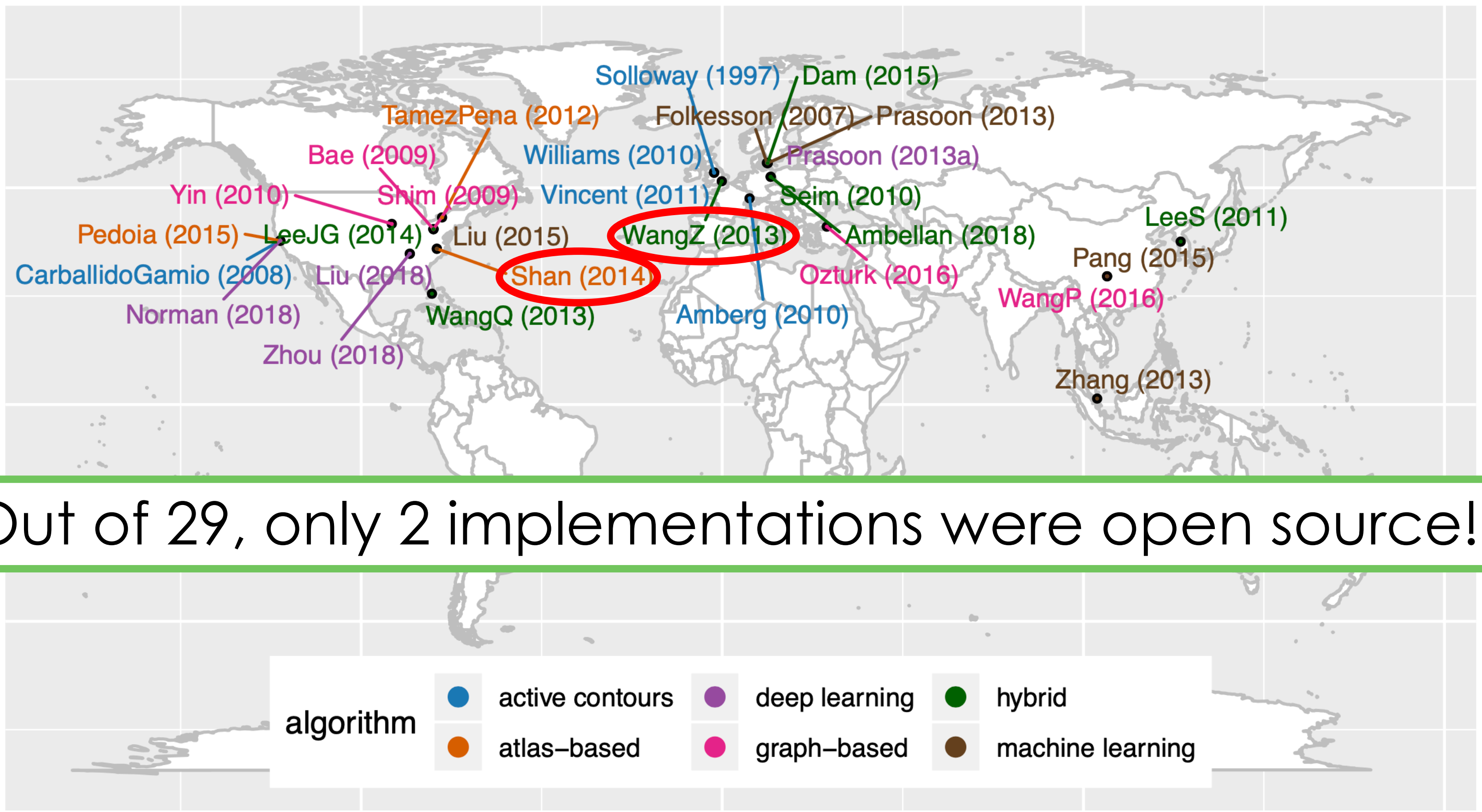


tMSKr

Bonaretti S, Gold GE, Beaupre GS (2020) pyKNEEr: An image analysis workflow for open and reproducible research on femoral knee cartilage. PLoS ONE 15(1): e0226501

I looked for existing algorithms around…

Literature map of femoral knee cartilage segmentation



Out of 29, only 2 implementations were open source!!!

tMSKr

Bonaretti S, Gold GE, Beaupre GS (2020) pyKNEEr: An image analysis workflow for open and reproducible research on femoral knee cartilage. PLoS ONE 15(1): e0226501

What to do?

- I was not interested to create another algorithm
  - There were already 29 around!

- I needed to create a ***pipeline*** to preprocess, segment, and analyze knee
  - Use of already existing algorithms
  - Focus on “putting the pieces together” and ***ease of use***

Image from: https://dribbble.com/shots/964040-Blog-Illo-gif

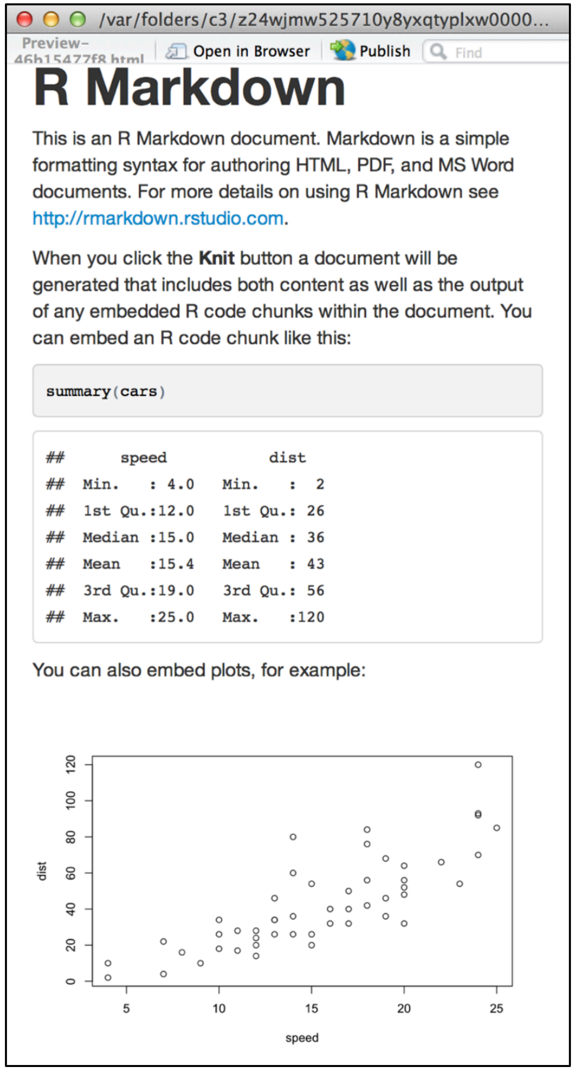
Somehow I started…

- Initially, I wanted to use Matlab
  - It was what I knew, and I already had some code
  - *But* it is not open source! Not everybody has a Matlab license
  - I did not want to write new closed source code (and be the 28th!)

- So I started in C++
  - I could use open libraries: ITK and elastix
  - *But* I had to create executables in Windows - I work in MacOS!
  - Command lines are not ideal for people with limited coding experience and coding in C++ is hard for me
  - Pipeline still fragmented (e.g. code vs. visualization)

But I was still looking for a better solution…

- A statistician showed me reproducible workflow using R markdown



R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2  
##  1st Qu.:12.0   1st Qu.: 26  
##  Median :15.0   Median : 36  
##  Mean   :15.4   Mean   : 43  
##  3rd Qu.:19.0   3rd Qu.: 56  
##  Max.   :25.0   Max.   :120  
```

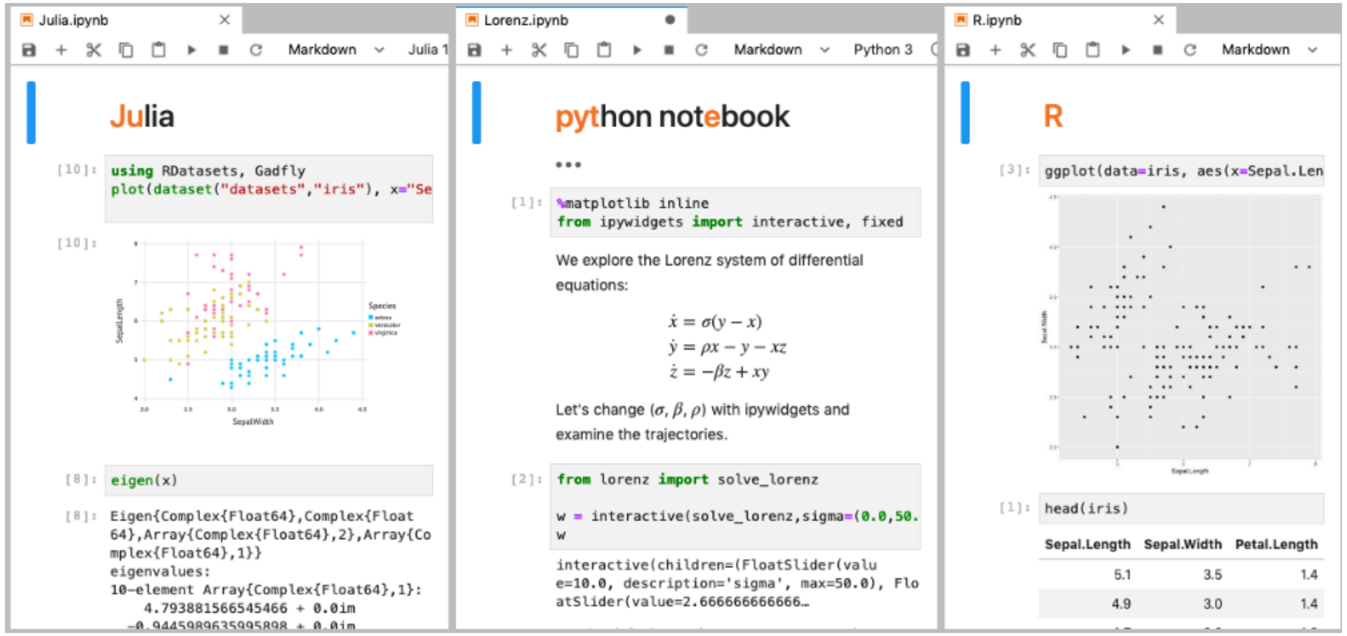
You can also embed plots, for example:

But I was still looking for a better solution…

- A statistician showed me reproducible workflow using R markdown

- There was something similar for python, something new which was getting more and more popular…

Jupyter notebook!!!



Julia

```
[10]: using RDatasets, Gadfly
      plot(dataset("datasets","iris"), x="Se
```

```
[8]: eigen(x)
```

```
[8]: Eigen{Complex{Float64},Complex{Float
     64},Array{Complex{Float64},2},Array{Co
     mplex{Float64},1}}
     eigenvalues:
     10-element Array{Complex{Float64},1}:
         4.793881566545466 + 0.0im
       -0.9445989635995898 + 0.0im
```

python notebook

•••

```
[1]: %matplotlib inline
     from ipywidgets import interactive, fixed
```

We explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

Let's change (σ, β, ρ) with ipywidgets and examine the trajectories.

```
[2]: from lorenz import solve_lorenz

     w = interactive(solve_lorenz,sigma=(0.0,50.
     w
```

```
interactive(children=(FloatSlider(valu
e=10.0, description='sigma', max=50.0), Flo
atSlider(value=2.666666666666…
```

R

```
[3]: ggplot(data=iris, aes(x=Sepal.Len
```

```
[1]: head(iris)
```

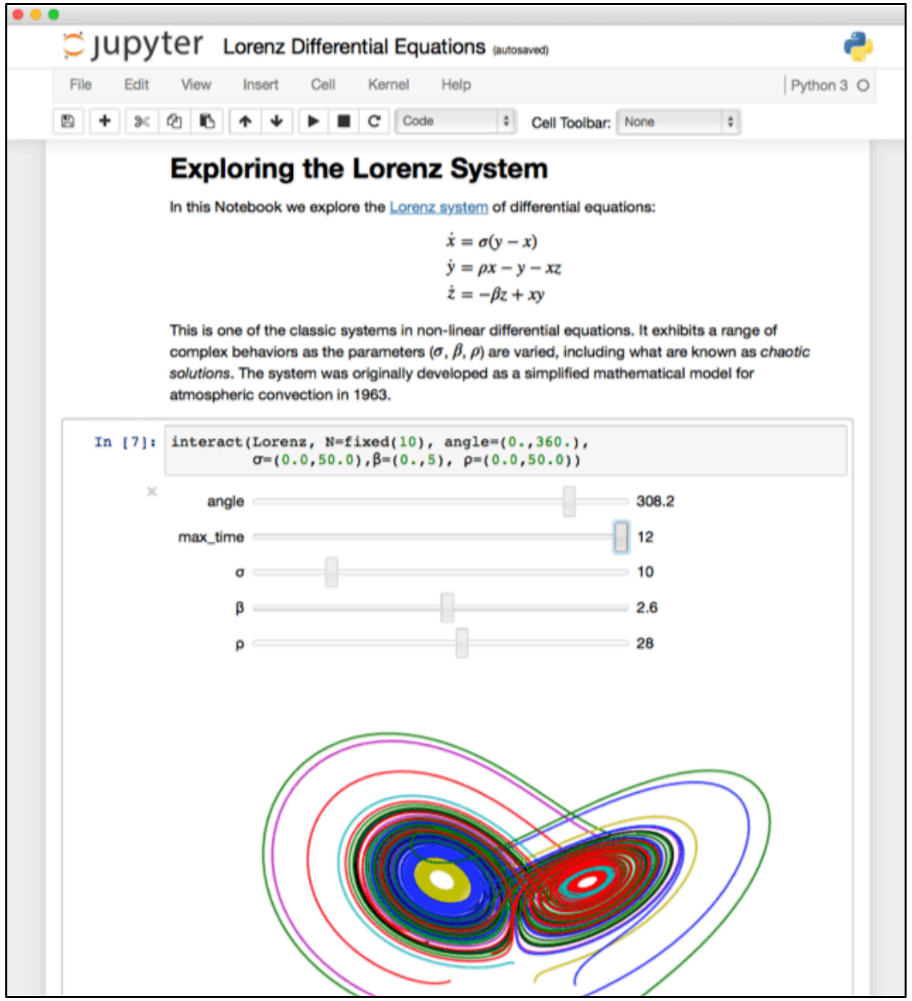
Sepal.Length	Sepal.Width	Petal.Length
5.1	3.5	1.4
4.9	3.0	1.4

What is Jupyter notebook?

- Web-based application integrating:
  - Narrative: text, equations, figures
  - Live code
    - Several kernels: python, C++, R, …
  - Visualizations (graphs, 3D renderings, …)

- Favors:
  - Reproducibility of computations
  - Sharing among researchers
  - Integration in publication

- Advantages:
  - Works for all OS
  - Runs on laptops and clusters



Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                 σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0))
```

angle 308.2
max_time 12
σ 10
β 2.6
ρ 28

So where did I start?

- I had to learn:
  - Jupyter and its environment
  - python
  - python packages:
    - numpy, matplotlib, SimpleITK

➡

- I started learning from:
  - Online tutorials - YouTube
  - Blogs
  - Live coding, e.g.
    - https://www.learnpython.org/
    - https://www.w3schools.com/python/

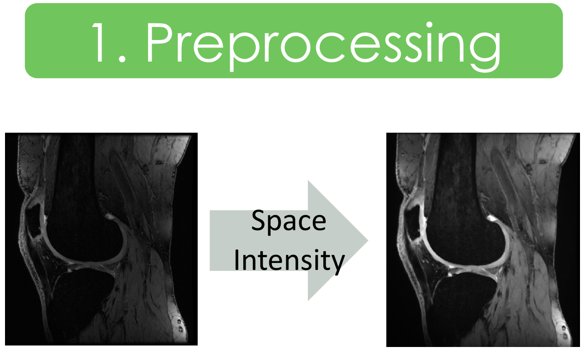
- I had to implement a workflow:
  - Image preprocessing
  - Cartilage segmentation
  - Analysis

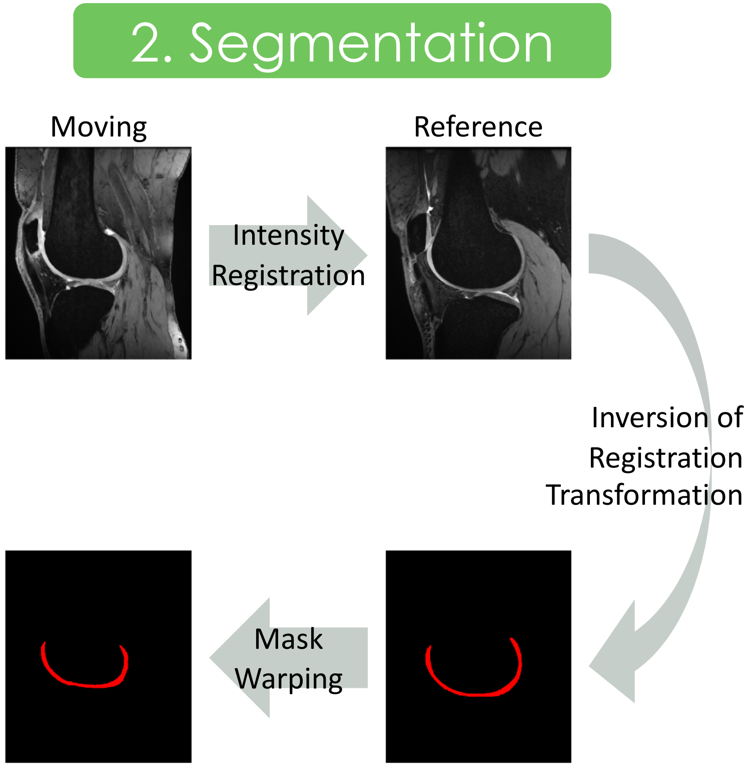
➡

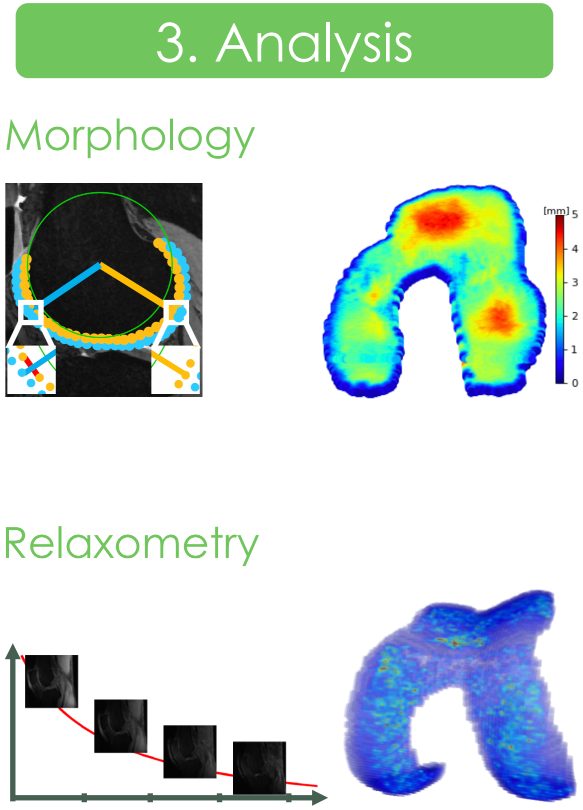
- I started by:
  - Translating the code I already had
    - Focus on language, not algorithm
  - Looking for solutions online
    - Blogs, stack overflow, etc.

And this is how created pyKNEEr!

- An image analysis workflow for ***open*** and ***reproducible*** research on femoral knee cartilage





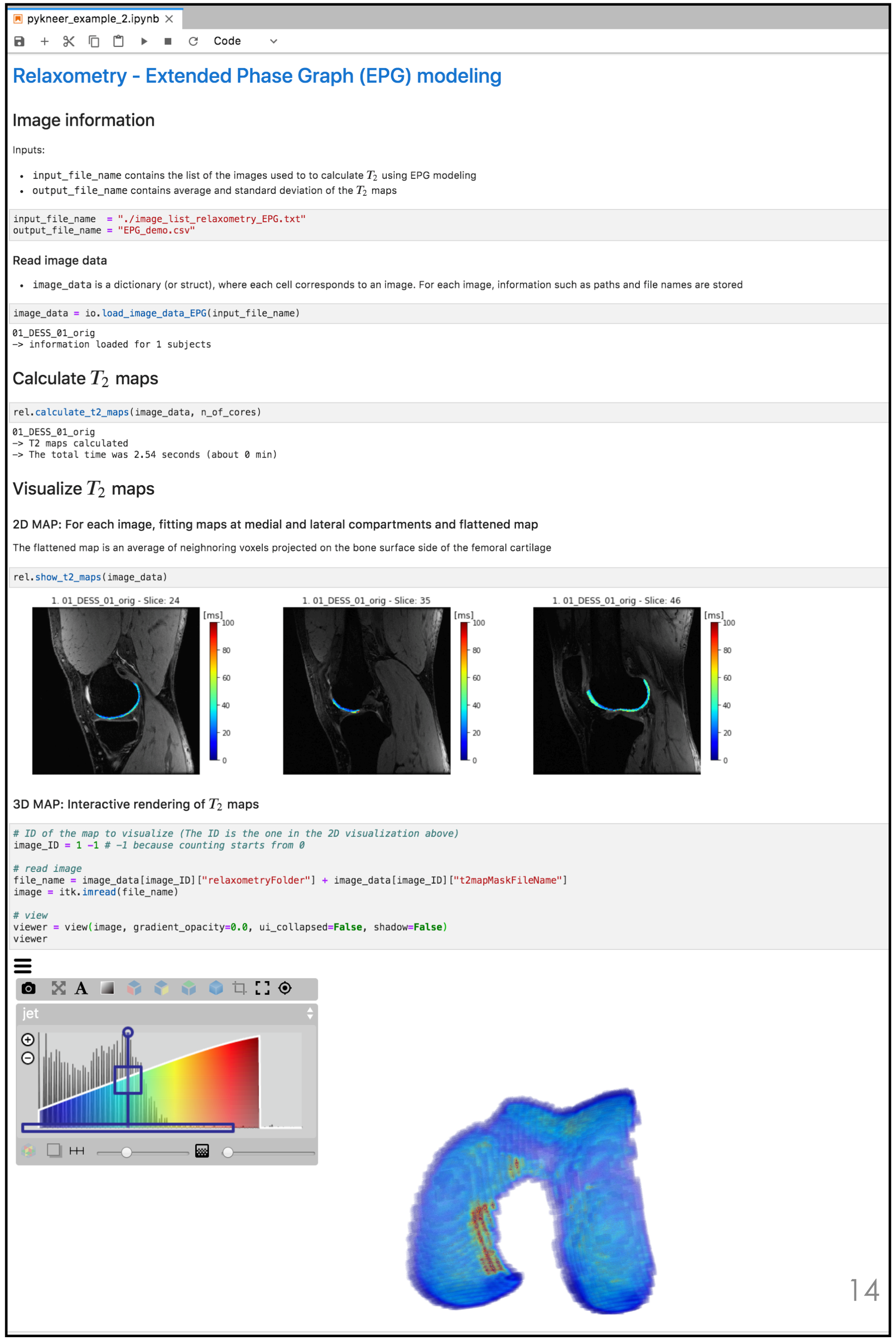


Structure of pyKNEEr

- Each part has one (or two) Jupyter notebooks as a user-interface
  - From data upload to result visualization in one file

- “Behind” the notebooks there is the pyKNEEr python package
  - Divided in modules
  - Contains core functions

- User has just to load her/his own images and run the notebook



pykneer_example_2.ipynb

Relaxometry - Extended Phase Graph (EPG) modeling

Image information

Inputs:

- `input_file_name` contains the list of the images used to to calculate T_2 using EPG modeling
- `output_file_name` contains average and standard deviation of the T_2 maps

```
input_file_name  = "./image_list_relaxometry_EPG.txt"
output_file_name = "EPG_demo.csv"
```

Read image data

- `image_data` is a dictionary (or struct), where each cell corresponds to an image. For each image, information such as paths and file names are stored

```
image_data = io.load_image_data_EPG(input_file_name)
```

```
01_DESS_01_orig
-> information loaded for 1 subjects
```

Calculate T_2 maps

```
rel.calculate_t2_maps(image_data, n_of_cores)
```

```
01_DESS_01_orig
-> T2 maps calculated
-> The total time was 2.54 seconds (about 0 min)
```

Visualize T_2 maps

2D MAP: For each image, fitting maps at medial and lateral compartments and flattened map

The flattened map is an average of neighnoring voxels projected on the bone surface side of the femoral cartilage

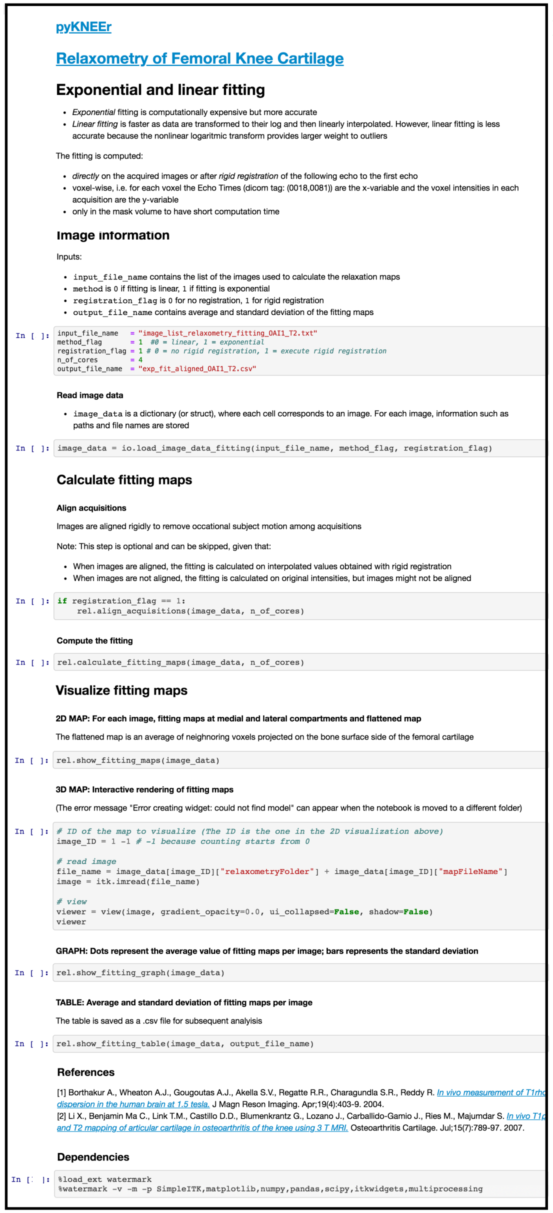
```
rel.show_t2_maps(image_data)
```

Code
1. 01_DESS_01_orig - Slice: 24
1. 01_DESS_01_orig - Slice: 35
1. 01_DESS_01_orig - Slice: 46
[ms]
100
80
60
40
20
0
jet

3D MAP: Interactive rendering of T_2 maps

```
# ID of the map to visualize (The ID is the one in the 2D visualization above)
image_ID = 1 -1 # -1 because counting starts from 0

# read image
file_name = image_data[image_ID]["relaxometryFolder"] + image_data[image_ID]["t2mapMaskFileName"]
image = itk.imread(file_name)

# view
viewer = view(image, gradient_opacity=0.0, ui_collapsed=False, shadow=False)
viewer
```

Structure of Jupyter notebooks in pyKNEEr



pyKNEEr

Relaxometry of Femoral Knee Cartilage

Exponential and linear fitting

- *Exponential* fitting is computationally expensive but more accurate
- *Linear fitting* is faster as data are transformed to their log and then linearly interpolated. However, linear fitting is less accurate because the nonlinear logaritmic transform provides larger weight to outliers

The fitting is computed:

- *directly* on the acquired images or after *rigid registration* of the following echo to the first echo
- voxel-wise, i.e. for each voxel the Echo Times (dicom tag: (0018,0081)) are the x-variable and the voxel intensities in each acquisition are the y-variable
- only in the mask volume to have short computation time

Image information

Inputs:

- `input_file_name` contains the list of the images used to calculate the relaxation maps
- `method` is `0` if fitting is linear, `1` if fitting is exponential
- `registration_flag` is `0` for no registration, `1` for rigid registration
- `output_file_name` contains average and standard deviation of the fitting maps

```
In [ ]: input_file_name   = "image_list_relaxometry_fitting_OAI1_T2.txt"
        method_flag       = 1  #0 = linear, 1 = exponential
        registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
        n_of_cores        = 4
        output_file_name  = "exp_fit_aligned_OAI1_T2.csv"
```

Read image data

- `image_data` is a dictionary (or struct), where each cell corresponds to an image. For each image, information such as paths and file names are stored

```
In [ ]: image_data = io.load_image_data_fitting(input_file_name, method_flag, registration_flag)
```

Calculate fitting maps

Align acquisitions

Images are aligned rigidly to remove occational subject motion among acquisitions

Note: This step is optional and can be skipped, given that:

- When images are aligned, the fitting is calculated on interpolated values obtained with rigid registration
- When images are not aligned, the fitting is calculated on original intensities, but images might not be aligned

```
In [ ]: if registration_flag == 1:
            rel.align_acquisitions(image_data, n_of_cores)
```

Compute the fitting

```
In [ ]: rel.calculate_fitting_maps(image_data, n_of_cores)
```

Visualize fitting maps

2D MAP: For each image, fitting maps at medial and lateral compartments and flattened map

The flattened map is an average of neighnoring voxels projected on the bone surface side of the femoral cartilage

```
In [ ]: rel.show_fitting_maps(image_data)
```

3D MAP: Interactive rendering of fitting maps

(The error message "Error creating widget: could not find model" can appear when the notebook is moved to a different folder)

```
In [ ]: # ID of the map to visualize (The ID is the one in the 2D visualization above)
        image_ID = 1 -1 # -1 because counting starts from 0

        # read image
        file_name = image_data[image_ID]["relaxometryFolder"] + image_data[image_ID]["mapFileName"]
        image = itk.imread(file_name)

        # view
        viewer = view(image, gradient_opacity=0.0, ui_collapsed=False, shadow=False)
        viewer
```

GRAPH: Dots represent the average value of fitting maps per image; bars represents the standard deviation

```
In [ ]: rel.show_fitting_graph(image_data)
```

TABLE: Average and standard deviation of fitting maps per image

The table is saved as a .csv file for subsequent analyisis

```
In [ ]: rel.show_fitting_table(image_data, output_file_name)
```

References

[1] Borthakur A., Wheaton A.J., Gougoutas A.J., Akella S.V., Regatte R.R., Charagundla S.R., Reddy R. *In vivo measurement of T1rh dispersion in the human brain at 1.5 tesla.* J Magn Reson Imaging. Apr;19(4):403-9. 2004.

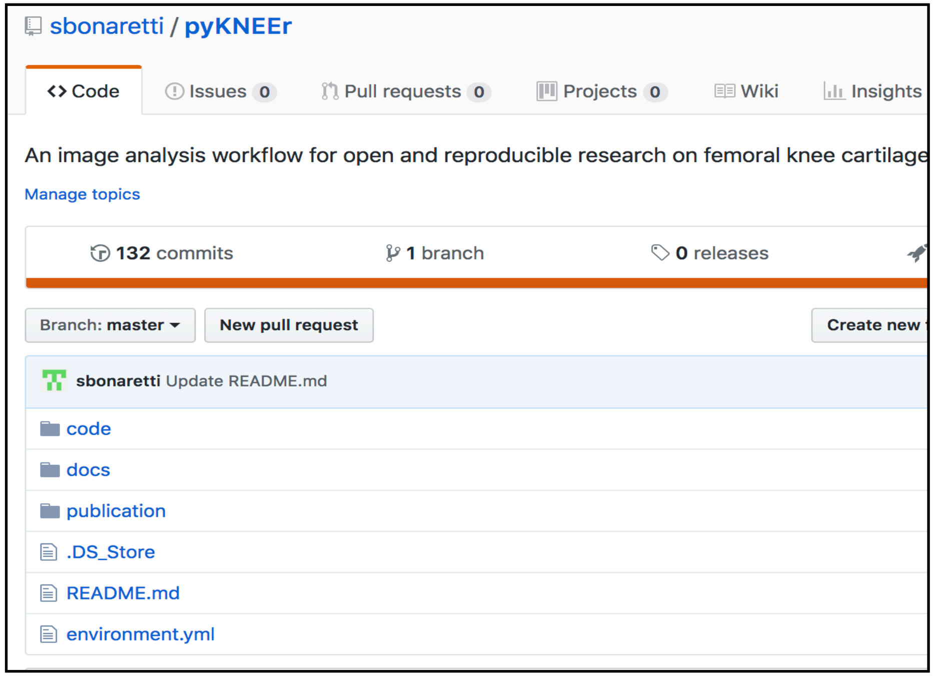
[2] Li X., Benjamin Ma C., Link T.M., Castillo D.D., Blumenkrantz G., Lozano J., Carballido-Gamio J., Ries M., Majumdar S. *In vivo T1ρ and T2 mapping of articular cartilage in osteoarthritis of the knee using 3 T MRI.* Osteoarthritis Cartilage. Jul;15(7):789-97. 2007.

Dependencies

```
In [ ]: %load_ext watermark
        %watermark -v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

1. Link to GitHub repository

https://github.com/sbonaretti/pyKNEEr



sbonaretti / **pyKNEEr**

An image analysis workflow for open and reproducible research on femoral knee cartilage

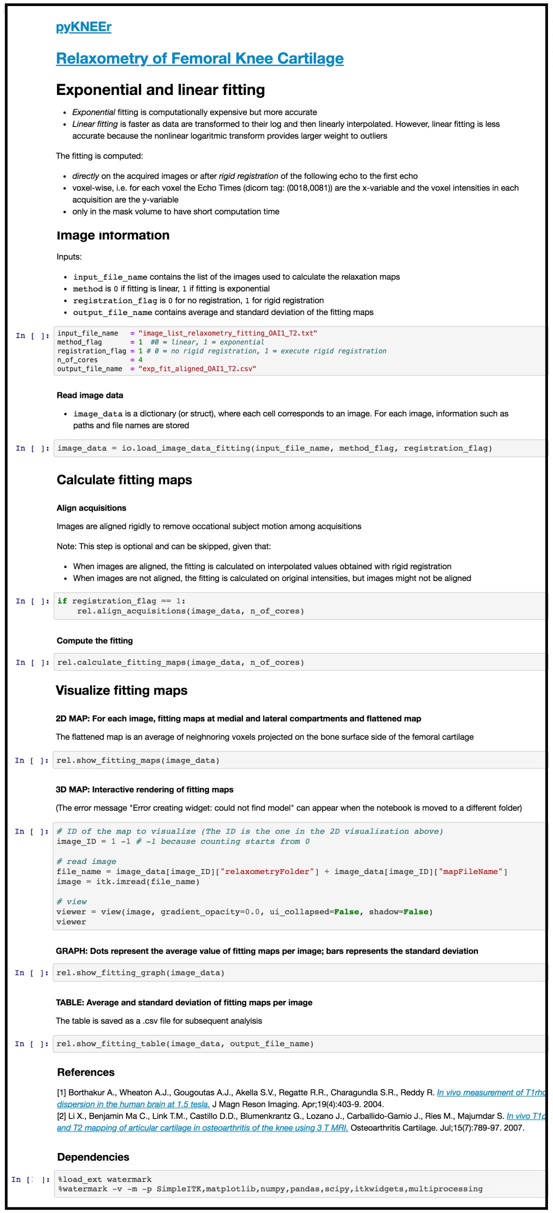
Manage topics

132 commits · **1** branch · **0** releases

sbonaretti Update README.md

- code
- docs
- publication
- .DS_Store
- README.md
- environment.yml

Structure of Jupyter notebooks in pyKNEEr



pyKNEEr

Relaxometry of Femoral Knee Cartilage

Exponential and linear fitting

- *Exponential* fitting is computationally expensive but more accurate
- *Linear fitting* is faster as data are transformed to their log and then linearly interpolated. However, linear fitting is less accurate because the nonlinear logaritmic transform provides larger weight to outliers

The fitting is computed:

- *directly* on the acquired images or after *rigid registration* of the following echo to the first echo
- voxel-wise, i.e. for each voxel the Echo Times (dicom tag: (0018,0081)) are the x-variable and the voxel intensities in each acquisition are the y-variable
- only in the mask volume to have short computation time

Image information

Inputs:

- `input_file_name` contains the list of the images used to calculate the relaxation maps
- `method` is `0` if fitting is linear, `1` if fitting is exponential
- `registration_flag` is `0` for no registration, `1` for rigid registration
- `output_file_name` contains average and standard deviation of the fitting maps

```
In [ ]: input_file_name   = "image_list_relaxometry_fitting_OAI1_T2.txt"
        method_flag       = 1  #0 = linear, 1 = exponential
        registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
        n_of_cores        = 4
        output_file_name  = "exp_fit_aligned_OAI1_T2.csv"
```

Read image data

- `image_data` is a dictionary (or struct), where each cell corresponds to an image. For each image, information such as paths and file names are stored

```
In [ ]: image_data = io.load_image_data_fitting(input_file_name, method_flag, registration_flag)
```

Calculate fitting maps

Align acquisitions

Images are aligned rigidly to remove occational subject motion among acquisitions

Note: This step is optional and can be skipped, given that:

- When images are aligned, the fitting is calculated on interpolated values obtained with rigid registration
- When images are not aligned, the fitting is calculated on original intensities, but images might not be aligned

```
In [ ]: if registration_flag == 1:
            rel.align_acquisitions(image_data, n_of_cores)
```

Compute the fitting

```
In [ ]: rel.calculate_fitting_maps(image_data, n_of_cores)
```

Visualize fitting maps

2D MAP: For each image, fitting maps at medial and lateral compartments and flattened map

The flattened map is an average of neighnoring voxels projected on the bone surface side of the femoral cartilage

```
In [ ]: rel.show_fitting_maps(image_data)
```

3D MAP: Interactive rendering of fitting maps

(The error message "Error creating widget: could not find model" can appear when the notebook is moved to a different folder)

```
In [ ]: # ID of the map to visualize (The ID is the one in the 2D visualization above)
        image_ID = 1 -1 # -1 because counting starts from 0

        # read image
        file_name = image_data[image_ID]["relaxometryFolder"] + image_data[image_ID]["mapFileName"]
        image = itk.imread(file_name)

        # view
        viewer = view(image, gradient_opacity=0.0, ui_collapsed=False, shadow=False)
        viewer
```

GRAPH: Dots represent the average value of fitting maps per image; bars represents the standard deviation

```
In [ ]: rel.show_fitting_graph(image_data)
```

TABLE: Average and standard deviation of fitting maps per image

The table is saved as a .csv file for subsequent analyisis

```
In [ ]: rel.show_fitting_table(image_data, output_file_name)
```

References

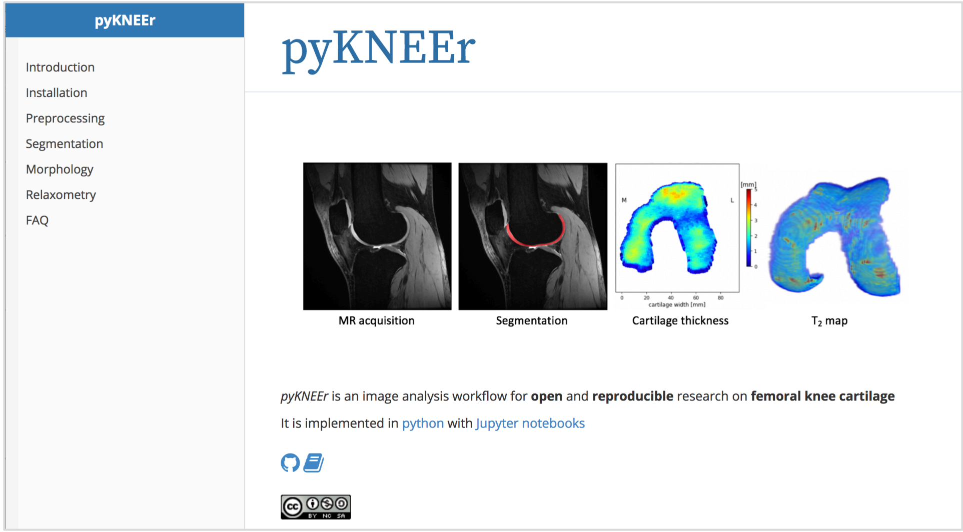
[1] Borthakur A., Wheaton A.J., Gougoutas A.J., Akella S.V., Regatte R.R., Charagundla S.R., Reddy R. *In vivo measurement of T1rho dispersion in the human brain at 1.5 tesla.* J Magn Reson Imaging. Apr;19(4):403-9. 2004.
[2] Li X., Benjamin Ma C., Link T.M., Castillo D.D., Blumenkrantz G., Lozano J., Carballido-Gamio J., Ries M., Majumdar S. *In vivo T1ρ and T2 mapping of articular cartilage in osteoarthritis of the knee using 3 T MRI.* Osteoarthritis Cartilage. Jul;15(7):789-97. 2007.

Dependencies

```
In [ ]: %load_ext watermark
        %watermark -v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

1. Link to GitHub repository
2. Link to documentation

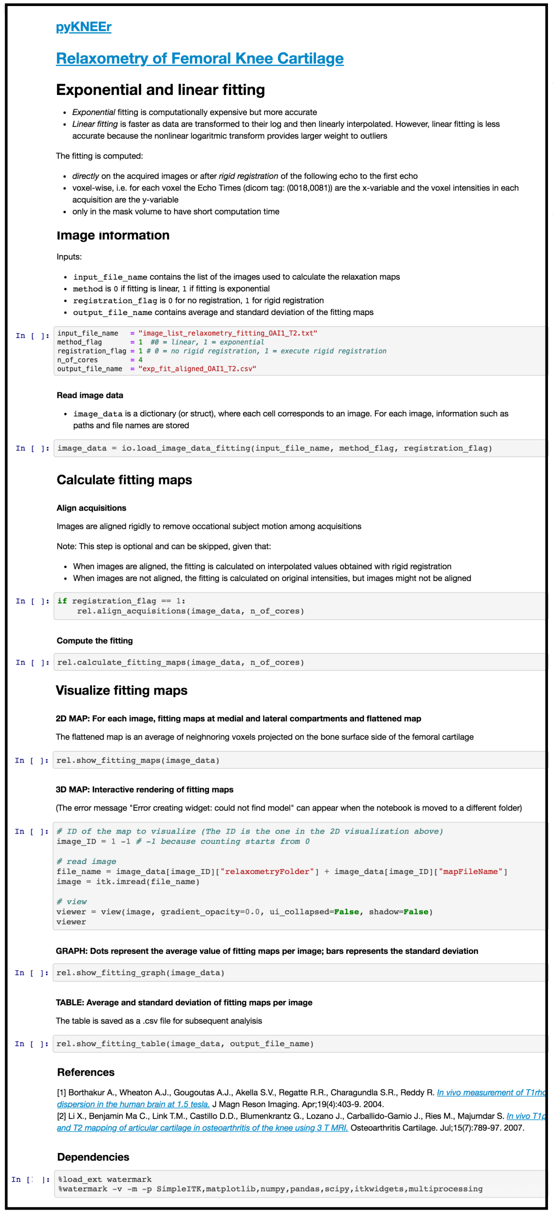
https://sbonaretti.github.io/pyKNEEr/



pyKNEEr is an image analysis workflow for **open** and **reproducible** research on **femoral knee cartilage**

It is implemented in python with Jupyter notebooks

Structure of Jupyter notebooks in pyKNEEr



pyKNEEr

Relaxometry of Femoral Knee Cartilage

Exponential and linear fitting

- *Exponential* fitting is computationally expensive but more accurate
- *Linear fitting* is faster as data are transformed to their log and then linearly interpolated. However, linear fitting is less accurate because the nonlinear logaritmic transform provides larger weight to outliers

The fitting is computed:

- *directly* on the acquired images or after *rigid registration* of the following echo to the first echo
- voxel-wise, i.e. for each voxel the Echo Times (dicom tag: (0018,0081)) are the x-variable and the voxel intensities in each acquisition are the y-variable
- only in the mask volume to have short computation time

Image information

Inputs:

- `input_file_name` contains the list of the images used to calculate the relaxation maps
- `method` is `0` if fitting is linear, `1` if fitting is exponential
- `registration_flag` is `0` for no registration, `1` for rigid registration
- `output_file_name` contains average and standard deviation of the fitting maps

In [ ]:
```
input_file_name   = "image_list_relaxometry_fitting_OAI1_T2.txt"
method_flag       = 1  #0 = linear, 1 = exponential
registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
n_of_cores        = 4
output_file_name  = "exp_fit_aligned_OAI1_T2.csv"
```

Read image data

- `image_data` is a dictionary (or struct), where each cell corresponds to an image. For each image, information such as paths and file names are stored

In [ ]:
```
image_data = io.load_image_data_fitting(input_file_name, method_flag, registration_flag)
```

Calculate fitting maps

Align acquisitions

Images are aligned rigidly to remove occational subject motion among acquisitions

Note: This step is optional and can be skipped, given that:

- When images are aligned, the fitting is calculated on interpolated values obtained with rigid registration
- When images are not aligned, the fitting is calculated on original intensities, but images might not be aligned

In [ ]:
```
if registration_flag == 1:
    rel.align_acquisitions(image_data, n_of_cores)
```

Compute the fitting

In [ ]:
```
rel.calculate_fitting_maps(image_data, n_of_cores)
```

Visualize fitting maps

2D MAP: For each image, fitting maps at medial and lateral compartments and flattened map

The flattened map is an average of neighnoring voxels projected on the bone surface side of the femoral cartilage

In [ ]:
```
rel.show_fitting_maps(image_data)
```

3D MAP: Interactive rendering of fitting maps

(The error message "Error creating widget: could not find model" can appear when the notebook is moved to a different folder)

In [ ]:
```
# ID of the map to visualize (The ID is the one in the 2D visualization above)
image_ID = 1 -1 # -1 because counting starts from 0

# read image
file_name = image_data[image_ID]["relaxometryFolder"] + image_data[image_ID]["mapFileName"]
image = itk.imread(file_name)

# view
viewer = view(image, gradient_opacity=0.0, ui_collapsed=False, shadow=False)
viewer
```

GRAPH: Dots represent the average value of fitting maps per image; bars represents the standard deviation

In [ ]:
```
rel.show_fitting_graph(image_data)
```

TABLE: Average and standard deviation of fitting maps per image

The table is saved as a .csv file for subsequent analyisis

In [ ]:
```
rel.show_fitting_table(image_data, output_file_name)
```

References

[1] Borthakur A., Wheaton A.J., Gougoutas A.J., Akella S.V., Regatte R.R., Charagundla S.R., Reddy R. *In vivo measurement of T1rh dispersion in the human brain at 1.5 tesla.* J Magn Reson Imaging. Apr;19(4):403-9. 2004.
[2] Li X., Benjamin Ma C., Link T.M., Castillo D.D., Blumenkrantz G., Lozano J., Carballido-Gamio J., Ries M., Majumdar S. *In vivo T1 and T2 mapping of articular cartilage in osteoarthritis of the knee using 3 T MRI.* Osteoarthritis Cartilage. Jul;15(7):789-97. 2007.

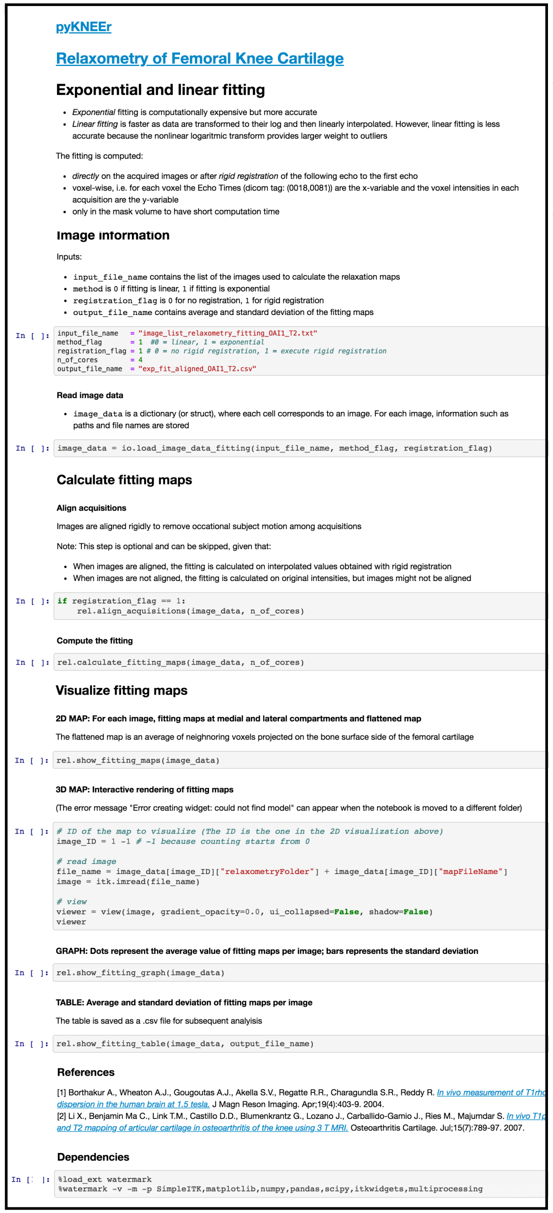
Dependencies

In [ ]:
```
%load_ext watermark
%watermark -v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

1. Link to GitHub repository
2. Link to documentation

3. Introduction

Structure of Jupyter notebooks in pyKNEEr



[pyKNEEr]

[Relaxometry of Femoral Knee Cartilage]

Exponential and linear fitting

- *Exponential* fitting is computationally expensive but more accurate
- *Linear fitting* is faster as data are transformed to their log and then linearly interpolated. However, linear fitting is less accurate because the nonlinear logaritmic transform provides larger weight to outliers

The fitting is computed:

- *directly* on the acquired images or after *rigid registration* of the following echo to the first echo
- voxel-wise, i.e. for each voxel the Echo Times (dicom tag: (0018,0081)) are the x-variable and the voxel intensities in each acquisition are the y-variable
- only in the mask volume to have short computation time

Image information

Inputs:

- `input_file_name` contains the list of the images used to calculate the relaxation maps
- `method` is `0` if fitting is linear, `1` if fitting is exponential
- `registration_flag` is `0` for no registration, `1` for rigid registration
- `output_file_name` contains average and standard deviation of the fitting maps

```
In [ ]: input_file_name   = "image_list_relaxometry_fitting_OAI1_T2.txt"
        method_flag       = 1  #0 = linear, 1 = exponential
        registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
        n_of_cores        = 4
        output_file_name  = "exp_fit_aligned_OAI1_T2.csv"
```

Read image data

- `image_data` is a dictionary (or struct), where each cell corresponds to an image. For each image, information such as paths and file names are stored

```
In [ ]: image_data = io.load_image_data_fitting(input_file_name, method_flag, registration_flag)
```

Calculate fitting maps

Align acquisitions

Images are aligned rigidly to remove occational subject motion among acquisitions

Note: This step is optional and can be skipped, given that:

- When images are aligned, the fitting is calculated on interpolated values obtained with rigid registration
- When images are not aligned, the fitting is calculated on original intensities, but images might not be aligned

```
In [ ]: if registration_flag == 1:
            rel.align_acquisitions(image_data, n_of_cores)
```

Compute the fitting

```
In [ ]: rel.calculate_fitting_maps(image_data, n_of_cores)
```

Visualize fitting maps

2D MAP: For each image, fitting maps at medial and lateral compartments and flattened map

The flattened map is an average of neighnoring voxels projected on the bone surface side of the femoral cartilage

```
In [ ]: rel.show_fitting_maps(image_data)
```

3D MAP: Interactive rendering of fitting maps

(The error message "Error creating widget: could not find model" can appear when the notebook is moved to a different folder)

```
In [ ]: # ID of the map to visualize (The ID is the one in the 2D visualization above)
        image_ID = 1 -1 # -1 because counting starts from 0

        # read image
        file_name = image_data[image_ID]["relaxometryFolder"] + image_data[image_ID]["mapFileName"]
        image = itk.imread(file_name)

        # view
        viewer = view(image, gradient_opacity=0.0, ui_collapsed=False, shadow=False)
        viewer
```

GRAPH: Dots represent the average value of fitting maps per image; bars represents the standard deviation

```
In [ ]: rel.show_fitting_graph(image_data)
```

TABLE: Average and standard deviation of fitting maps per image

The table is saved as a .csv file for subsequent analyisis

```
In [ ]: rel.show_fitting_table(image_data, output_file_name)
```

References

[1] Borthakur A., Wheaton A.J., Gougoutas A.J., Akella S.V., Regatte R.R., Charagundla S.R., Reddy R. *In vivo measurement of T1rh* *dispersion in the human brain at 1.5 tesla.* J Magn Reson Imaging. Apr;19(4):403-9. 2004.
[2] Li X., Benjamin Ma C., Link T.M., Castillo D.D., Blumenkrantz G., Lozano J., Carballido-Gamio J., Ries M., Majumdar S. *In vivo T1ρ* *and T2 mapping of articular cartilage in osteoarthritis of the knee using 3 T MRI.* Osteoarthritis Cartilage. Jul;15(7):789-97. 2007.

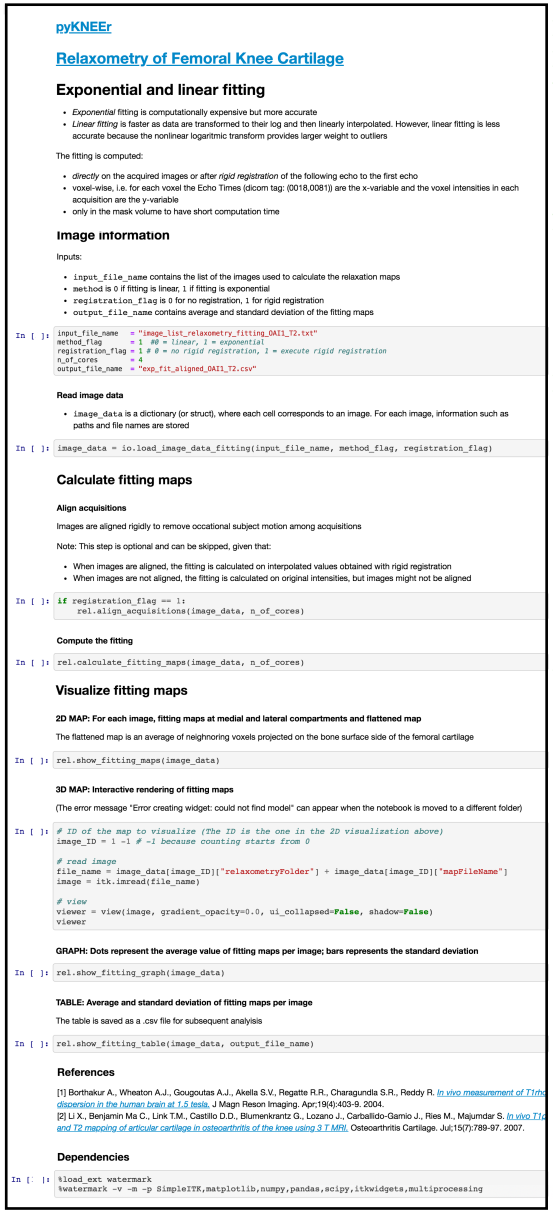
Dependencies

```
In [ ]: %load_ext watermark
        %watermark -v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

1. Link to GitHub repository
2. Link to documentation
3. Introduction
4. User inputs

```
input_file_name

./original/
001/BL
left
002/BL
left
003/BL
right
004/BL
left
005/BL
right
006/BL
left
007/BL
left
008/BL
left
009/BL
right
010/BL
left
011/BL
left
012/BL
right


n_of_cores

output_file_name

...
```

Structure of Jupyter notebooks in pyKNEEr



[pyKNEEr](#)

[Relaxometry of Femoral Knee Cartilage](#)

Exponential and linear fitting

- *Exponential* fitting is computationally expensive but more accurate
- *Linear fitting* is faster as data are transformed to their log and then linearly interpolated. However, linear fitting is less accurate because the nonlinear logaritmic transform provides larger weight to outliers

The fitting is computed:

- *directly* on the acquired images or after *rigid registration* of the following echo to the first echo
- voxel-wise, i.e. for each voxel the Echo Times (dicom tag: (0018,0081)) are the x-variable and the voxel intensities in each acquisition are the y-variable
- only in the mask volume to have short computation time

Image information

Inputs:

- `input_file_name` contains the list of the images used to calculate the relaxation maps
- `method` is `0` if fitting is linear, `1` if fitting is exponential
- `registration_flag` is `0` for no registration, `1` for rigid registration
- `output_file_name` contains average and standard deviation of the fitting maps

```
In [ ]: input_file_name   = "image_list_relaxometry_fitting_OAI1_T2.txt"
        method_flag       = 1  #0 = linear, 1 = exponential
        registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
        n_of_cores        = 4
        output_file_name  = "exp_fit_aligned_OAI1_T2.csv"
```

Read image data

- `image_data` is a dictionary (or struct), where each cell corresponds to an image. For each image, information such as paths and file names are stored

```
In [ ]: image_data = io.load_image_data_fitting(input_file_name, method_flag, registration_flag)
```

Calculate fitting maps

Align acquisitions

Images are aligned rigidly to remove occational subject motion among acquisitions

Note: This step is optional and can be skipped, given that:

- When images are aligned, the fitting is calculated on interpolated values obtained with rigid registration
- When images are not aligned, the fitting is calculated on original intensities, but images might not be aligned

```
In [ ]: if registration_flag == 1:
            rel.align_acquisitions(image_data, n_of_cores)
```

Compute the fitting

```
In [ ]: rel.calculate_fitting_maps(image_data, n_of_cores)
```

Visualize fitting maps

2D MAP: For each image, fitting maps at medial and lateral compartments and flattened map

The flattened map is an average of neighnoring voxels projected on the bone surface side of the femoral cartilage

```
In [ ]: rel.show_fitting_maps(image_data)
```

3D MAP: Interactive rendering of fitting maps

(The error message "Error creating widget: could not find model" can appear when the notebook is moved to a different folder)

```
In [ ]: # ID of the map to visualize (The ID is the one in the 2D visualization above)
        image_ID = 1 -1 # -1 because counting starts from 0

        # read image
        file_name = image_data[image_ID]["relaxometryFolder"] + image_data[image_ID]["mapFileName"]
        image = itk.imread(file_name)

        # view
        viewer = view(image, gradient_opacity=0.0, ui_collapsed=False, shadow=False)
        viewer
```

GRAPH: Dots represent the average value of fitting maps per image; bars represents the standard deviation

```
In [ ]: rel.show_fitting_graph(image_data)
```

TABLE: Average and standard deviation of fitting maps per image

The table is saved as a .csv file for subsequent analyisis

```
In [ ]: rel.show_fitting_table(image_data, output_file_name)
```

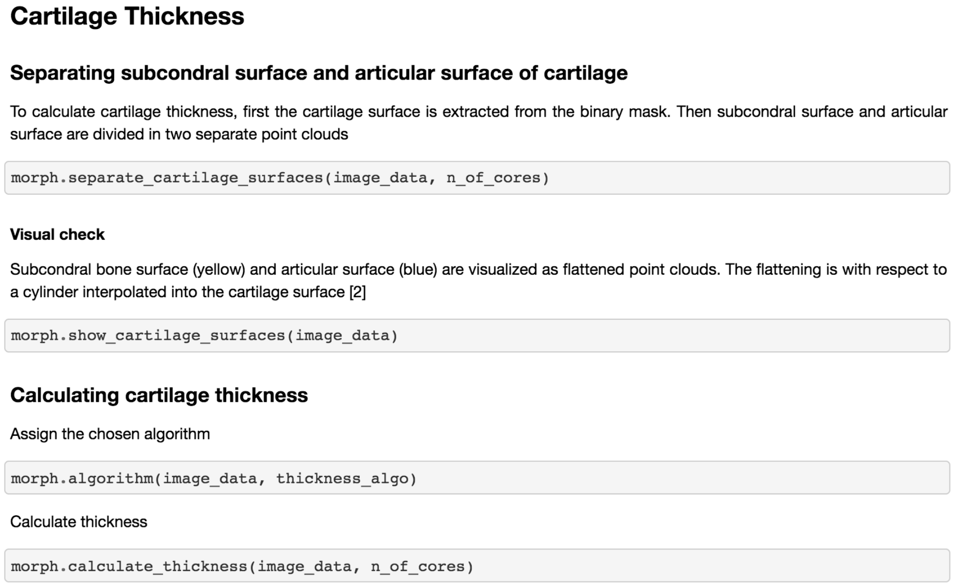
References

[1] Borthakur A., Wheaton A.J., Gougoutas A.J., Akella S.V., Regatte R.R., Charagundla S.R., Reddy R. *In vivo measurement of T1rh dispersion in the human brain at 1.5 tesla.* J Magn Reson Imaging. Apr;19(4):403-9. 2004.
[2] Li X., Benjamin Ma C., Link T.M., Castillo D.D., Blumenkrantz G., Lozano J., Carballido-Gamio J., Ries M., Majumdar S. *In vivo T1ρ and T2 mapping of articular cartilage in osteoarthritis of the knee using 3 T MRI.* Osteoarthritis Cartilage. Jul;15(7):789-97. 2007.

Dependencies

```
In [ ]: %load_ext watermark
        %watermark -v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

1. Link to GitHub repository
2. Link to documentation
3. Introduction
4. User inputs
5. Commands with narrative



Cartilage Thickness

Separating subcondral surface and articular surface of cartilage

To calculate cartilage thickness, first the cartilage surface is extracted from the binary mask. Then subcondral surface and articular surface are divided in two separate point clouds

```
morph.separate_cartilage_surfaces(image_data, n_of_cores)
```

Visual check

Subcondral bone surface (yellow) and articular surface (blue) are visualized as flattened point clouds. The flattening is with respect to a cylinder interpolated into the cartilage surface [2]

```
morph.show_cartilage_surfaces(image_data)
```

Calculating cartilage thickness

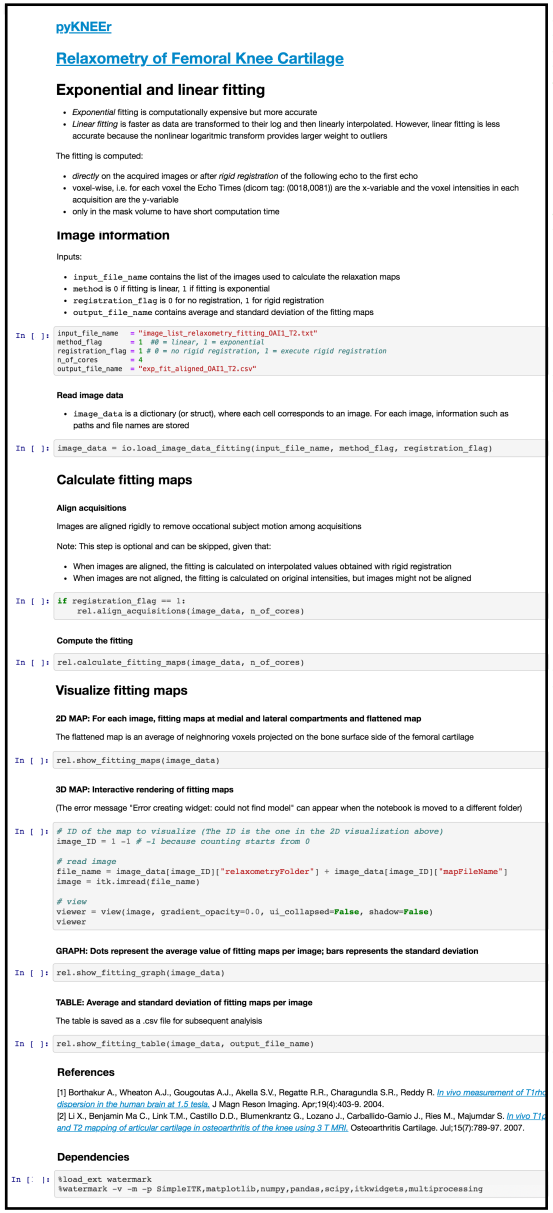
Assign the chosen algorithm

```
morph.algorithm(image_data, thickness_algo)
```

Calculate thickness

```
morph.calculate_thickness(image_data, n_of_cores)
```

Structure of Jupyter notebooks in pyKNEEr



pyKNEEr

Relaxometry of Femoral Knee Cartilage

Exponential and linear fitting

- *Exponential* fitting is computationally expensive but more accurate
- *Linear fitting* is faster as data are transformed to their log and then linearly interpolated. However, linear fitting is less accurate because the nonlinear logaritmic transform provides larger weight to outliers

The fitting is computed:

- *directly* on the acquired images or after *rigid registration* of the following echo to the first echo
- voxel-wise, i.e. for each voxel the Echo Times (dicom tag: (0018,0081)) are the x-variable and the voxel intensities in each acquisition are the y-variable
- only in the mask volume to have short computation time

Image information

Inputs:

- `input_file_name` contains the list of the images used to calculate the relaxation maps
- `method` is 0 if fitting is linear, 1 if fitting is exponential
- `registration_flag` is 0 for no registration, 1 for rigid registration
- `output_file_name` contains average and standard deviation of the fitting maps

```
In [ ]: input_file_name   = "image_list_relaxometry_fitting_OAI1_T2.txt"
        method_flag       = 1  #0 = linear, 1 = exponential
        registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
        n_of_cores        = 4
        output_file_name  = "exp_fit_aligned_OAI1_T2.csv"
```

Read image data

- `image_data` is a dictionary (or struct), where each cell corresponds to an image. For each image, information such as paths and file names are stored

```
In [ ]: image_data = io.load_image_data_fitting(input_file_name, method_flag, registration_flag)
```

Calculate fitting maps

Align acquisitions

Images are aligned rigidly to remove occational subject motion among acquisitions

Note: This step is optional and can be skipped, given that:

- When images are aligned, the fitting is calculated on interpolated values obtained with rigid registration
- When images are not aligned, the fitting is calculated on original intensities, but images might not be aligned

```
In [ ]: if registration_flag == 1:
            rel.align_acquisitions(image_data, n_of_cores)
```

Compute the fitting

```
In [ ]: rel.calculate_fitting_maps(image_data, n_of_cores)
```

Visualize fitting maps

2D MAP: For each image, fitting maps at medial and lateral compartments and flattened map

The flattened map is an average of neighnoring voxels projected on the bone surface side of the femoral cartilage

```
In [ ]: rel.show_fitting_maps(image_data)
```

3D MAP: Interactive rendering of fitting maps

(The error message "Error creating widget: could not find model" can appear when the notebook is moved to a different folder)

```
In [ ]: # ID of the map to visualize (The ID is the one in the 2D visualization above)
        image_ID = 1 -1 # -1 because counting starts from 0

        # read image
        file_name = image_data[image_ID]["relaxometryFolder"] + image_data[image_ID]["mapFileName"]
        image = itk.imread(file_name)

        # view
        viewer = view(image, gradient_opacity=0.0, ui_collapsed=False, shadow=False)
        viewer
```

GRAPH: Dots represent the average value of fitting maps per image; bars represents the standard deviation

```
In [ ]: rel.show_fitting_graph(image_data)
```

TABLE: Average and standard deviation of fitting maps per image

The table is saved as a .csv file for subsequent analyisis

```
In [ ]: rel.show_fitting_table(image_data, output_file_name)
```

References

[1] Borthakur A., Wheaton A.J., Gougoutas A.J., Akella S.V., Regatte R.R., Charagundla S.R., Reddy R. *In vivo measurement of T1rho dispersion in the human brain at 1.5 tesla.* J Magn Reson Imaging. Apr;19(4):403-9. 2004.

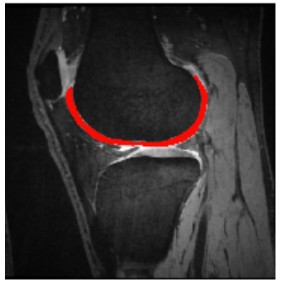
[2] Li X., Benjamin Ma C., Link T.M., Castillo D.D., Blumenkrantz G., Lozano J., Carballido-Gamio J., Ries M., Majumdar S. *In vivo T1ρ and T2 mapping of articular cartilage in osteoarthritis of the knee using 3 T MRI.* Osteoarthritis Cartilage. Jul;15(7):789-97. 2007.

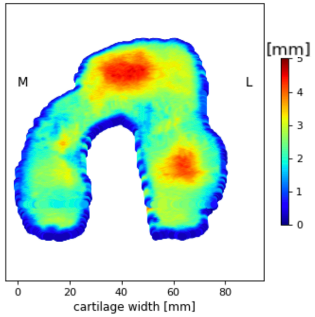
Dependencies

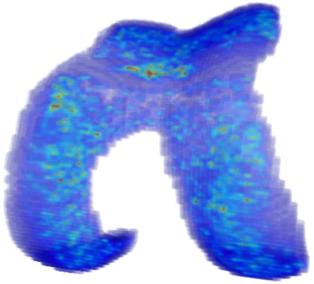
```
In [ ]: %load_ext watermark
        %watermark -v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

1. Link to GitHub repository
2. Link to documentation
3. Introduction
4. User inputs
5. Commands with narrative
6. Visualization of outputs

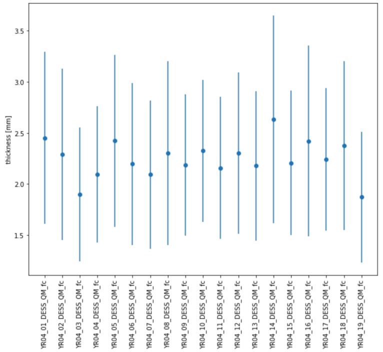
Qualitative visualizations

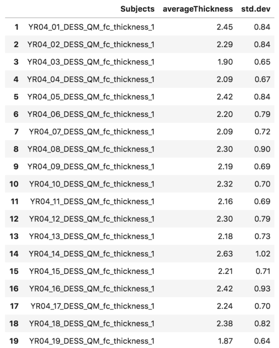






Quantitative visualizations

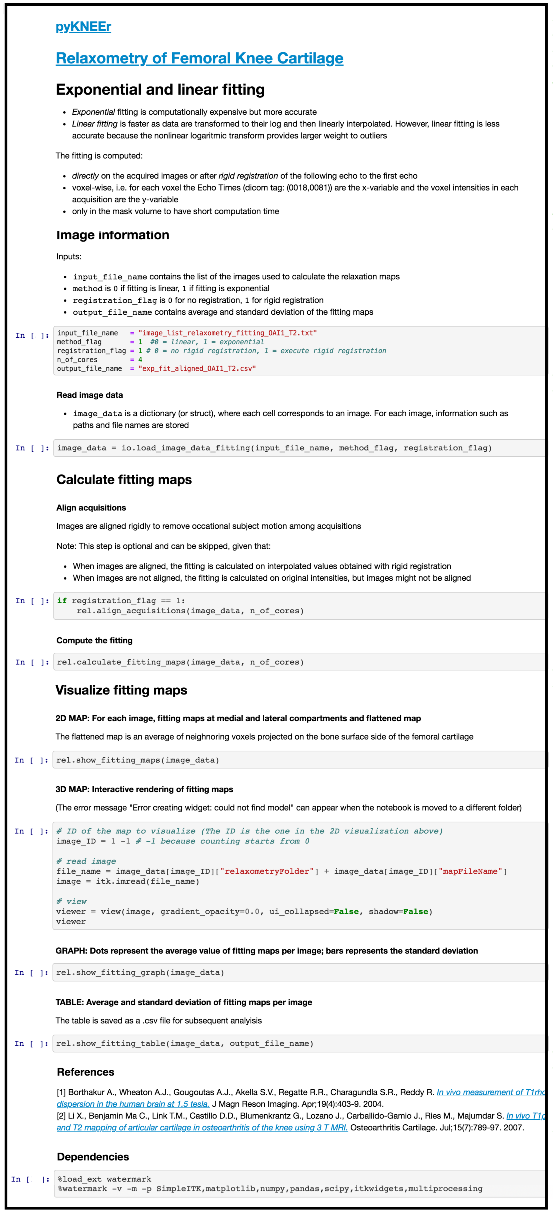




	Subjects	averageThickness	std.dev
1	YR04_01_DESS_QM_fc_thickness_1	2.45	0.84
2	YR04_02_DESS_QM_fc_thickness_1	2.29	0.84
3	YR04_03_DESS_QM_fc_thickness_1	1.90	0.65
4	YR04_04_DESS_QM_fc_thickness_1	2.09	0.67
5	YR04_05_DESS_QM_fc_thickness_1	2.42	0.84
6	YR04_06_DESS_QM_fc_thickness_1	2.20	0.79
7	YR04_07_DESS_QM_fc_thickness_1	2.09	0.72
8	YR04_08_DESS_QM_fc_thickness_1	2.30	0.90
9	YR04_09_DESS_QM_fc_thickness_1	2.19	0.69
10	YR04_10_DESS_QM_fc_thickness_1	2.32	0.70
11	YR04_11_DESS_QM_fc_thickness_1	2.16	0.69
12	YR04_12_DESS_QM_fc_thickness_1	2.30	0.79
13	YR04_13_DESS_QM_fc_thickness_1	2.18	0.73
14	YR04_14_DESS_QM_fc_thickness_1	2.63	1.02
15	YR04_15_DESS_QM_fc_thickness_1	2.21	0.71
16	YR04_16_DESS_QM_fc_thickness_1	2.42	0.93
17	YR04_17_DESS_QM_fc_thickness_1	2.24	0.70
18	YR04_18_DESS_QM_fc_thickness_1	2.38	0.82
19	YR04_19_DESS_QM_fc_thickness_1	1.87	0.64

tMSKr

Structure of Jupyter notebooks in pyKNEEr



pyKNEEr

Relaxometry of Femoral Knee Cartilage

Exponential and linear fitting

- *Exponential* fitting is computationally expensive but more accurate
- *Linear fitting* is faster as data are transformed to their log and then linearly interpolated. However, linear fitting is less accurate because the nonlinear logaritmic transform provides larger weight to outliers

The fitting is computed:

- *directly* on the acquired images or after *rigid registration* of the following echo to the first echo
- voxel-wise, i.e. for each voxel the Echo Times (dicom tag: (0018,0081)) are the x-variable and the voxel intensities in each acquisition are the y-variable
- only in the mask volume to have short computation time

Image information

Inputs:

- `input_file_name` contains the list of the images used to calculate the relaxation maps
- `method` is `0` if fitting is linear, `1` if fitting is exponential
- `registration_flag` is `0` for no registration, `1` for rigid registration
- `output_file_name` contains average and standard deviation of the fitting maps

In [ ]:

```
input_file_name   = "image_list_relaxometry_fitting_OAI1_T2.txt"
method_flag       = 1  #0 = linear, 1 = exponential
registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
n_of_cores        = 4
output_file_name  = "exp_fit_aligned_OAI1_T2.csv"
```

Read image data

- `image_data` is a dictionary (or struct), where each cell corresponds to an image. For each image, information such as paths and file names are stored

In [ ]:

```
image_data = io.load_image_data_fitting(input_file_name, method_flag, registration_flag)
```

Calculate fitting maps

Align acquisitions

Images are aligned rigidly to remove occational subject motion among acquisitions

Note: This step is optional and can be skipped, given that:

- When images are aligned, the fitting is calculated on interpolated values obtained with rigid registration
- When images are not aligned, the fitting is calculated on original intensities, but images might not be aligned

In [ ]:

```
if registration_flag == 1:
    rel.align_acquisitions(image_data, n_of_cores)
```

Compute the fitting

In [ ]:

```
rel.calculate_fitting_maps(image_data, n_of_cores)
```

Visualize fitting maps

2D MAP: For each image, fitting maps at medial and lateral compartments and flattened map

The flattened map is an average of neighnoring voxels projected on the bone surface side of the femoral cartilage

In [ ]:

```
rel.show_fitting_maps(image_data)
```

3D MAP: Interactive rendering of fitting maps

(The error message "Error creating widget: could not find model" can appear when the notebook is moved to a different folder)

In [ ]:

```
# ID of the map to visualize (The ID is the one in the 2D visualization above)
image_ID = 1 -1 # -1 because counting starts from 0

# read image
file_name = image_data[image_ID]["relaxometryFolder"] + image_data[image_ID]["mapFileName"]
image = itk.imread(file_name)

# view
viewer = view(image, gradient_opacity=0.0, ui_collapsed=False, shadow=False)
viewer
```

GRAPH: Dots represent the average value of fitting maps per image; bars represents the standard deviation

In [ ]:

```
rel.show_fitting_graph(image_data)
```

TABLE: Average and standard deviation of fitting maps per image

The table is saved as a .csv file for subsequent analyisis

In [ ]:

```
rel.show_fitting_table(image_data, output_file_name)
```

References

[1] Borthakur A., Wheaton A.J., Gougoutas A.J., Akella S.V., Regatte R.R., Charagundla S.R., Reddy R. *In vivo measurement of T1rho dispersion in the human brain at 1.5 tesla.* J Magn Reson Imaging. Apr;19(4):403-9. 2004.

[2] Li X., Benjamin Ma C., Link T.M., Castillo D.D., Blumenkrantz G., Lozano J., Carballido-Gamio J., Ries M., Majumdar S. *In vivo T1ρ and T2 mapping of articular cartilage in osteoarthritis of the knee using 3 T MRI.* Osteoarthritis Cartilage. Jul;15(7):789-97. 2007.

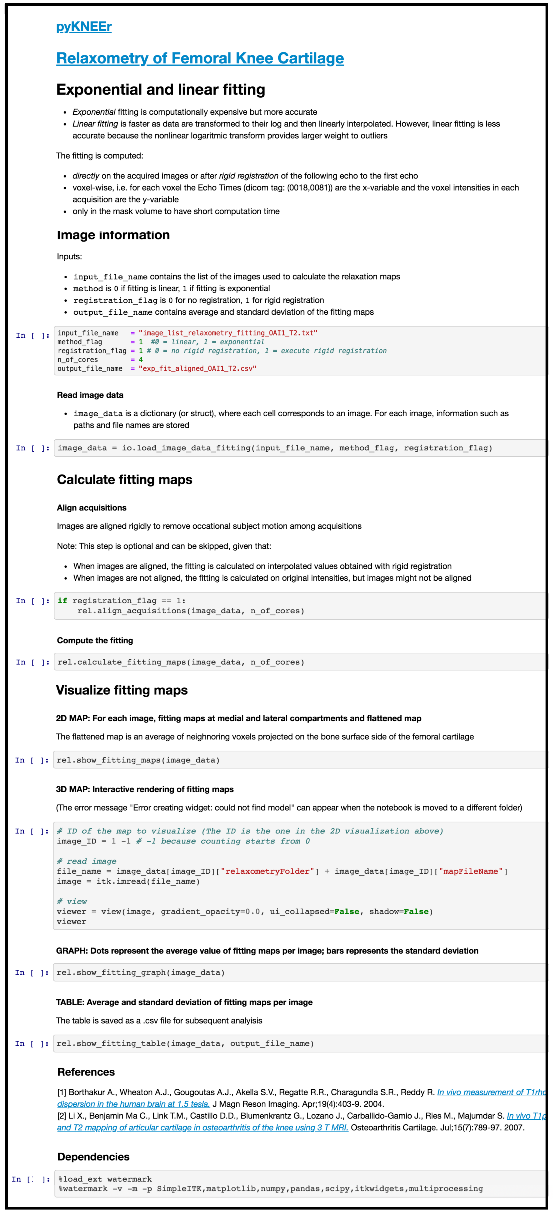
Dependencies

In [ ]:

```
%load_ext watermark
%watermark -v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

1. Link to GitHub repository
2. Link to documentation
3. Introduction
4. User inputs
5. Commands with narrative
6. Visualization of outputs
7. References

Structure of Jupyter notebooks in pyKNEEr



pyKNEEr

Relaxometry of Femoral Knee Cartilage

Exponential and linear fitting

- *Exponential* fitting is computationally expensive but more accurate
- *Linear fitting* is faster as data are transformed to their log and then linearly interpolated. However, linear fitting is less accurate because the nonlinear logaritmic transform provides larger weight to outliers

The fitting is computed:

- *directly* on the acquired images or after *rigid registration* of the following echo to the first echo
- voxel-wise, i.e. for each voxel the Echo Times (dicom tag: (0018,0081)) are the x-variable and the voxel intensities in each acquisition are the y-variable
- only in the mask volume to have short computation time

Image information

Inputs:

- `input_file_name` contains the list of the images used to calculate the relaxation maps
- `method` is `0` if fitting is linear, `1` if fitting is exponential
- `registration_flag` is `0` for no registration, `1` for rigid registration
- `output_file_name` contains average and standard deviation of the fitting maps

```
In [ ]: input_file_name   = "image_list_relaxometry_fitting_OAI1_T2.txt"
        method_flag       = 1  #0 = linear, 1 = exponential
        registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
        n_of_cores        = 4
        output_file_name  = "exp_fit_aligned_OAI1_T2.csv"
```

Read image data

- `image_data` is a dictionary (or struct), where each cell corresponds to an image. For each image, information such as paths and file names are stored

```
In [ ]: image_data = io.load_image_data_fitting(input_file_name, method_flag, registration_flag)
```

Calculate fitting maps

Align acquisitions

Images are aligned rigidly to remove occational subject motion among acquisitions

Note: This step is optional and can be skipped, given that:

- When images are aligned, the fitting is calculated on interpolated values obtained with rigid registration
- When images are not aligned, the fitting is calculated on original intensities, but images might not be aligned

```
In [ ]: if registration_flag == 1:
            rel.align_acquisitions(image_data, n_of_cores)
```

Compute the fitting

```
In [ ]: rel.calculate_fitting_maps(image_data, n_of_cores)
```

Visualize fitting maps

2D MAP: For each image, fitting maps at medial and lateral compartments and flattened map

The flattened map is an average of neighnoring voxels projected on the bone surface side of the femoral cartilage

```
In [ ]: rel.show_fitting_maps(image_data)
```

3D MAP: Interactive rendering of fitting maps

(The error message "Error creating widget: could not find model" can appear when the notebook is moved to a different folder)

```
In [ ]: # ID of the map to visualize (The ID is the one in the 2D visualization above)
        image_ID = 1 -1 # -1 because counting starts from 0

        # read image
        file_name = image_data[image_ID]["relaxometryFolder"] + image_data[image_ID]["mapFileName"]
        image = itk.imread(file_name)

        # view
        viewer = view(image, gradient_opacity=0.0, ui_collapsed=False, shadow=False)
        viewer
```

GRAPH: Dots represent the average value of fitting maps per image; bars represents the standard deviation

```
In [ ]: rel.show_fitting_graph(image_data)
```

TABLE: Average and standard deviation of fitting maps per image

The table is saved as a .csv file for subsequent analyisis

```
In [ ]: rel.show_fitting_table(image_data, output_file_name)
```

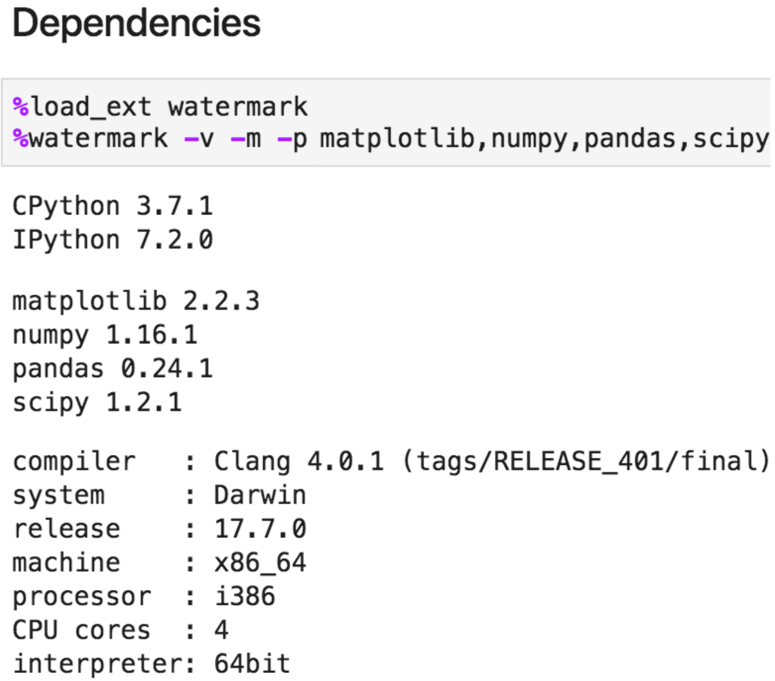
References

[1] Borthakur A., Wheaton A.J., Gougoutas A.J., Akella S.V., Regatte R.R., Charagundla S.R., Reddy R. *In vivo measurement of T1rho dispersion in the human brain at 1.5 tesla.* J Magn Reson Imaging. Apr;19(4):403-9. 2004.
[2] Li X., Benjamin Ma C., Link T.M., Castillo D.D., Blumenkrantz G., Lozano J., Carballido-Gamio J., Ries M., Majumdar S. *In vivo T1ρ and T2 mapping of articular cartilage in osteoarthritis of the knee using 3 T MRI.* Osteoarthritis Cartilage. Jul;15(7):789-97. 2007.

Dependencies

```
In [ ]: %load_ext watermark
        %watermark -v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

1. Link to GitHub repository
2. Link to documentation
3. Introduction
4. User inputs
5. Commands with narrative
6. Visualization of outputs
7. References
8. Dependencies



Dependencies

```
%load_ext watermark
%watermark -v -m -p matplotlib,numpy,pandas,scipy
```

```
CPython 3.7.1
IPython 7.2.0

matplotlib 2.2.3
numpy 1.16.1
pandas 0.24.1
scipy 1.2.1

compiler   : Clang 4.0.1 (tags/RELEASE_401/final)
system     : Darwin
release    : 17.7.0
machine    : x86_64
processor  : i386
CPU cores  : 4
interpreter: 64bit
```

→ Reproducibility of computational environment

tMSKr

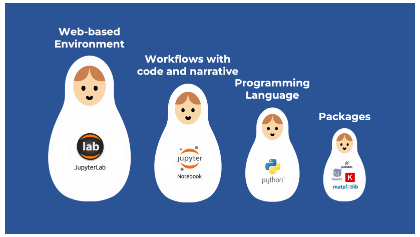
Jupyter Community | MSK

- We got a Jupyter Community Workshop grant!
  - To start building the Jupyter community in MSK imaging
  - Workshop, June 7-9, 2020 → Online meeting June 7, 2020
- We span across the globe
  - UCSF, CU Denver, U. Calgary, UNC, ITK, U. Leuven, U. Lund, I.O. Rizzoli, U. Melbourne
- We aim to create open and reproducible workflows by
  - Combining existing code to overcome fragmentation
  - Creating new code with structured guidelines

Practical tips to create workflows with Jupyter notebook

Learn from free online material

Video tutorials



Jupyter notebook and python for scientists

SciPy

PyData

Scipy, pyData

Hands-on tutorials

SimpleITK notebooks

SPIE 2019 workshop

OpenMR Benelux 2020

Imperial College Course

Notebook Examples

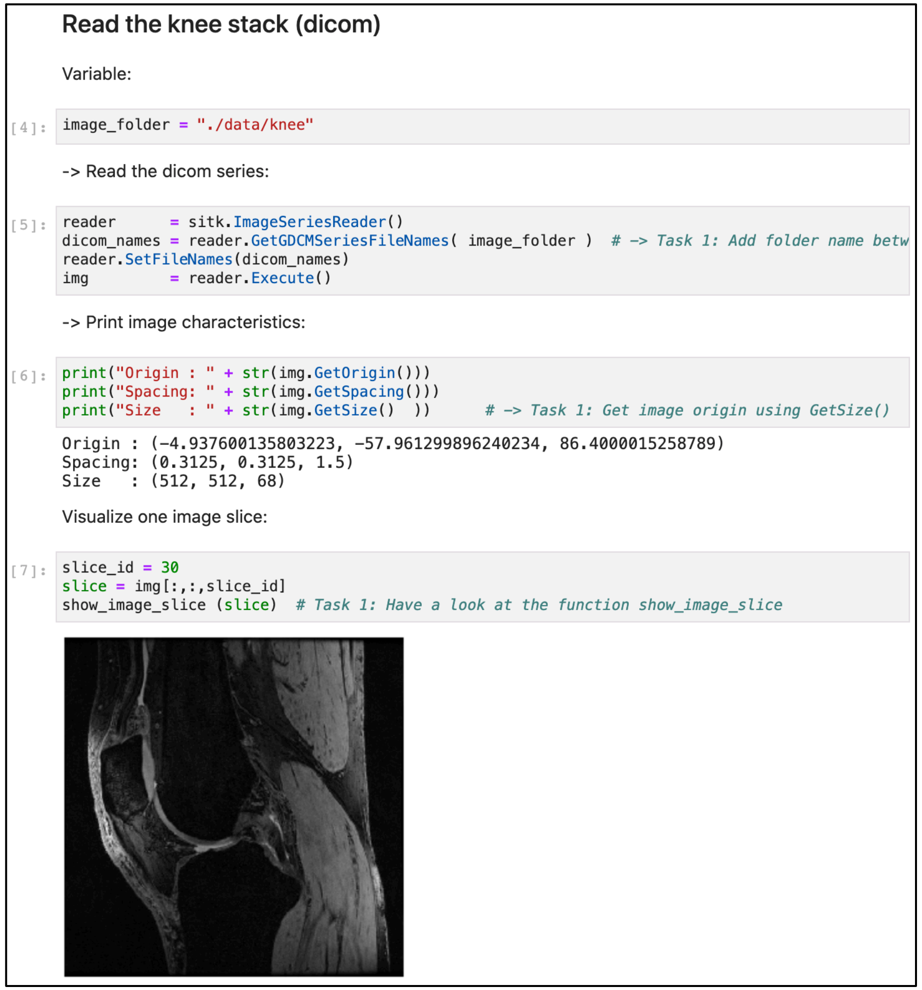
Nipype

Deep Learning Toolkit

pyKNEEr

Open a new notebook…

- Write the narrative
- Write code
- Run cells in sequence



Read the knee stack (dicom)

Variable:

```
[4]: image_folder = "./data/knee"
```

-> Read the dicom series:

```
[5]: reader      = sitk.ImageSeriesReader()
     dicom_names = reader.GetGDCMSeriesFileNames( image_folder )  # -> Task 1: Add folder name betw
     reader.SetFileNames(dicom_names)
     img         = reader.Execute()
```

-> Print image characteristics:

```
[6]: print("Origin : " + str(img.GetOrigin()))
     print("Spacing: " + str(img.GetSpacing()))
     print("Size   : " + str(img.GetSize()  ))      # -> Task 1: Get image origin using GetSize()

     Origin : (-4.937600135803223, -57.961299896240234, 86.4000015258789)
     Spacing: (0.3125, 0.3125, 1.5)
     Size   : (512, 512, 68)
```

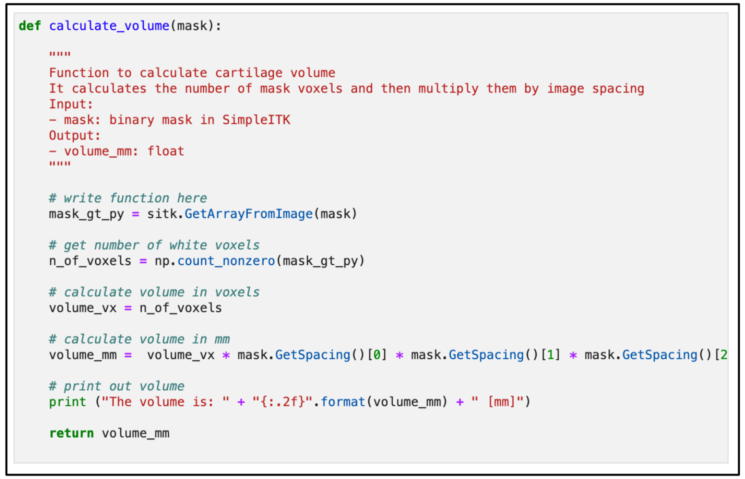
Visualize one image slice:

```
[7]: slice_id = 30
     slice = img[:,:,slice_id]
     show_image_slice (slice)  # Task 1: Have a look at the function show_image_slice
```

Start optimizing the code...

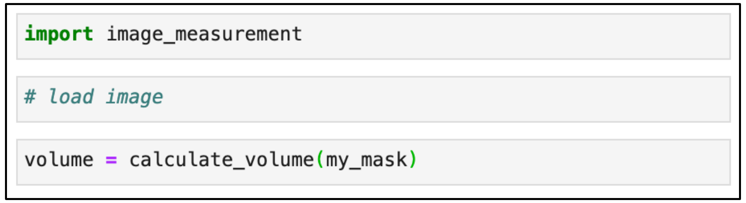
- Move reusable functions to a python module

- In the notebook:
  - Import the module
  - Call the function



```
def calculate_volume(mask):

    """
    Function to calculate cartilage volume
    It calculates the number of mask voxels and then multiply them by image spacing
    Input:
    - mask: binary mask in SimpleITK
    Output:
    - volume_mm: float
    """

    # write function here
    mask_gt_py = sitk.GetArrayFromImage(mask)

    # get number of white voxels
    n_of_voxels = np.count_nonzero(mask_gt_py)

    # calculate volume in voxels
    volume_vx = n_of_voxels

    # calculate volume in mm
    volume_mm =  volume_vx * mask.GetSpacing()[0] * mask.GetSpacing()[1] * mask.GetSpacing()[2

    # print out volume
    print ("The volume is: " + "{:.2f}".format(volume_mm) + " [mm]")

    return volume_mm
```

`image_measurements.py`



```
import image_measurement
```

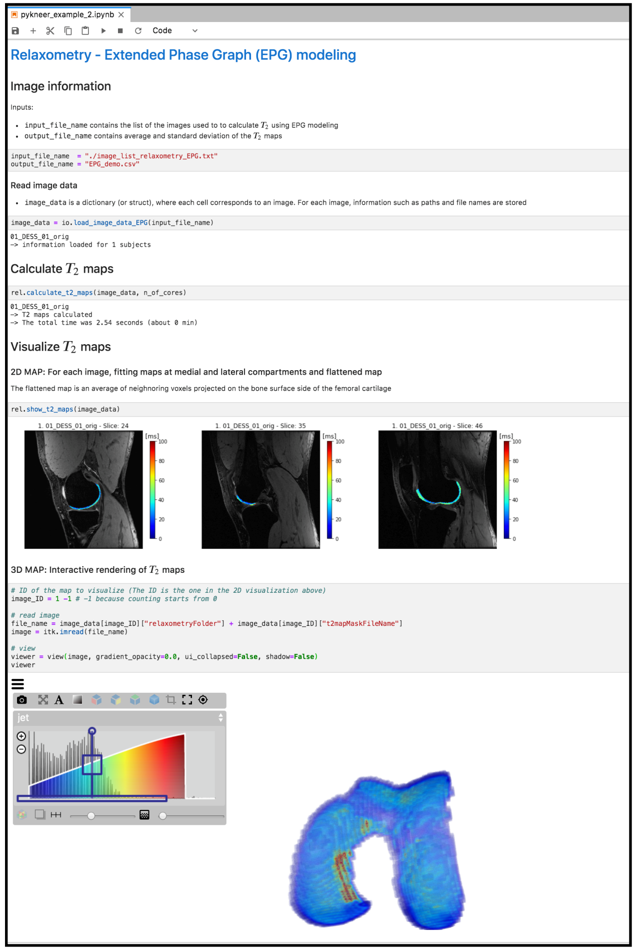
```
# load image
```

```
volume = calculate_volume(my_mask)
```

tMSKr

Keep the notebook human readable

- Organize the narrative:
  - Title of the notebook
  - Divide in paragraphs with subtitles
  - Introduce code
  - Comment the results you obtain
- Organize the code:
  - Package imports
  - Functions
  - Variables
  - Workflow body
  - Dependencies



pykneer_example_2.ipynb
Code
1. 01_DESS_01_orig - Slice: 24
1. 01_DESS_01_orig - Slice: 35
1. 01_DESS_01_orig - Slice: 46
[ms]
100
80
60
40
20
0
jet

Relaxometry - Extended Phase Graph (EPG) modeling

Image information

Inputs:

- input_file_name contains the list of the images used to to calculate T_2 using EPG modeling
- output_file_name contains average and standard deviation of the T_2 maps

```
input_file_name  = "./image_list_relaxometry_EPG.txt"
output_file_name = "EPG_demo.csv"
```

Read image data

- image_data is a dictionary (or struct), where each cell corresponds to an image. For each image, information such as paths and file names are stored

```
image_data = io.load_image_data_EPG(input_file_name)
```

```
01_DESS_01_orig
-> information loaded for 1 subjects
```

Calculate T_2 maps

```
rel.calculate_t2_maps(image_data, n_of_cores)
```

```
01_DESS_01_orig
-> T2 maps calculated
-> The total time was 2.54 seconds (about 0 min)
```

Visualize T_2 maps

2D MAP: For each image, fitting maps at medial and lateral compartments and flattened map

The flattened map is an average of neighnoring voxels projected on the bone surface side of the femoral cartilage

```
rel.show_t2_maps(image_data)
```

3D MAP: Interactive rendering of T_2 maps

```
# ID of the map to visualize (The ID is the one in the 2D visualization above)
image_ID = 1 -1 # -1 because counting starts from 0

# read image
file_name = image_data[image_ID]["relaxometryFolder"] + image_data[image_ID]["t2mapMaskFileName"]
image = itk.imread(file_name)

# view
viewer = view(image, gradient_opacity=0.0, ui_collapsed=False, shadow=False)
viewer
```

For practical example, see video tutorial: Jupyter notebook and python for scientists

Make the notebook reproducible

- Automatically download data from a repository

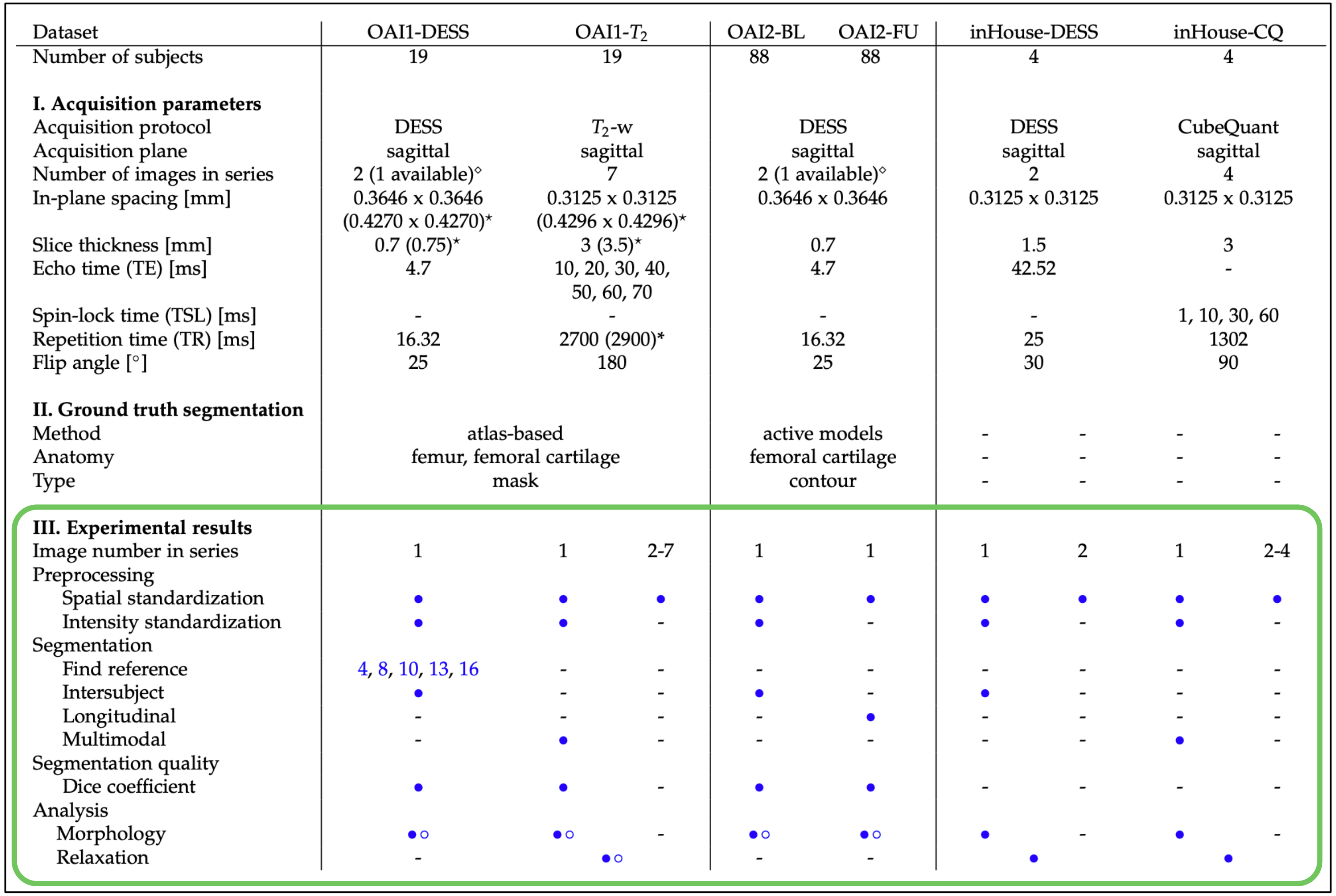
- Automate data manipulation

- Define seeds to generate random numbers

- Print dependences

Sandve et al., (2013) Ten Simple Rules for Reproducible Computational Research, PLoS Comput Biol. 9(10): e1003285.
Rule et al. (2019) Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks. Comput Biol. 15(7): e1007007.
See practical tutorial: How to create a reproducible Jupyter notebook?

Attach the notebook to the paper



Dataset	OAI1-DESS	OAI1-T_2		OAI2-BL	OAI2-FU	inHouse-DESS		inHouse-CQ	
Number of subjects	19	19		88	88	4		4	
I. Acquisition parameters									
Acquisition protocol	DESS	T_2-w		DESS		DESS		CubeQuant	
Acquisition plane	sagittal	sagittal		sagittal		sagittal		sagittal	
Number of images in series	2 (1 available)$^\diamond$	7		2 (1 available)$^\diamond$		2		4	
In-plane spacing [mm]	0.3646 x 0.3646 (0.4270 x 0.4270)*	0.3125 x 0.3125 (0.4296 x 0.4296)*		0.3646 x 0.3646		0.3125 x 0.3125		0.3125 x 0.3125	
Slice thickness [mm]	0.7 (0.75)*	3 (3.5)*		0.7		1.5		3	
Echo time (TE) [ms]	4.7	10, 20, 30, 40, 50, 60, 70		4.7		42.52		-	
Spin-lock time (TSL) [ms]	-	-		-		-		1, 10, 30, 60	
Repetition time (TR) [ms]	16.32	2700 (2900)*		16.32		25		1302	
Flip angle [°]	25	180		25		30		90	
II. Ground truth segmentation									
Method	atlas-based			active models		-	-	-	-
Anatomy	femur, femoral cartilage			femoral cartilage		-	-	-	-
Type	mask			contour		-	-	-	-
III. Experimental results									
Image number in series	1	1	2-7	1	1	1	2	1	2-4
Preprocessing									
Spatial standardization	●	●	●	●	●	●	●	●	●
Intensity standardization	●	●	-	●	-	●	-	●	-
Segmentation									
Find reference	4, 8, 10, 13, 16	-	-	-	-	-	-	-	-
Intersubject	●	-	-	●	-	●	-	-	-
Longitudinal	-	-	-	-	●	-	-	-	-
Multimodal	-	●	-	-	-	-	-	●	-
Segmentation quality									
Dice coefficient	●	●	-	●	●	-	-	-	-
Analysis									
Morphology	●○	●○	-	●○	●○	●	-	●	-
Relaxation	-	●○		-	-	●		●	

Bonaretti S, Gold GE, Beaupre GS (2020) pyKNEEr: An image analysis workflow for open and reproducible research on femoral knee cartilage. PLoS ONE 15(1): e0226501

Attach the notebook to the paper

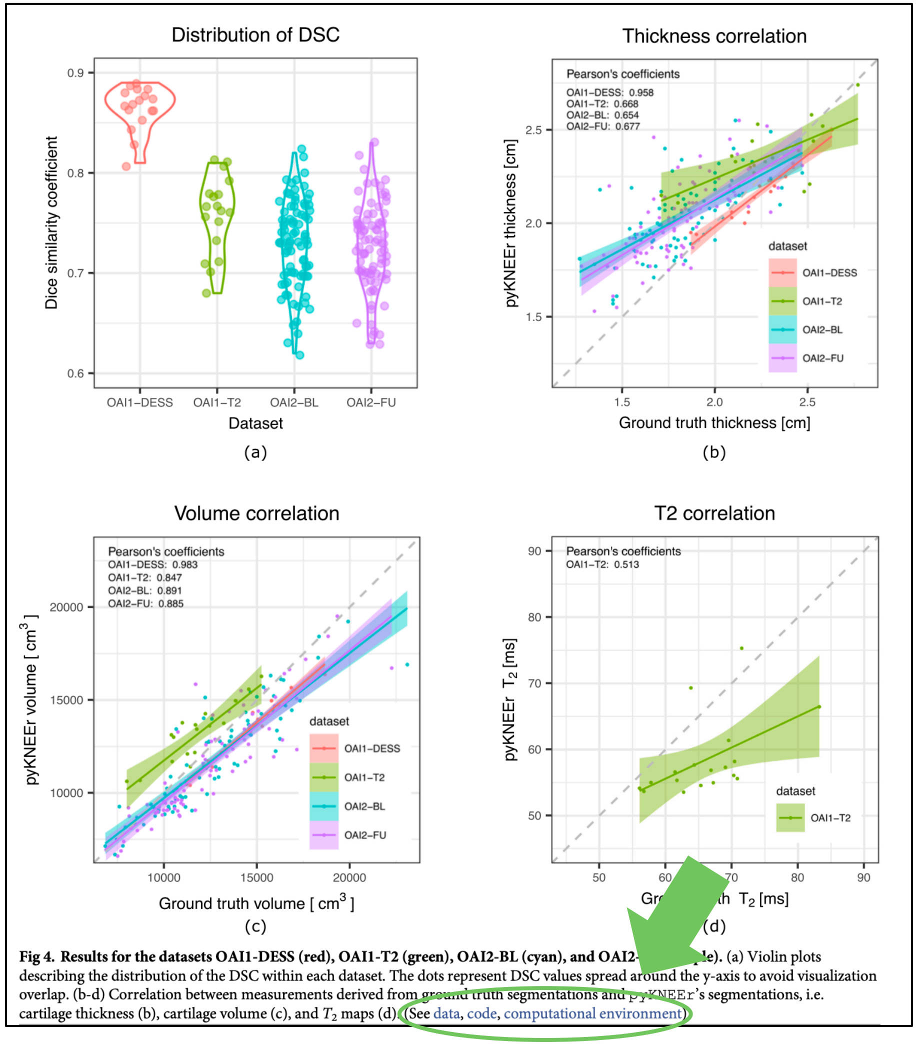
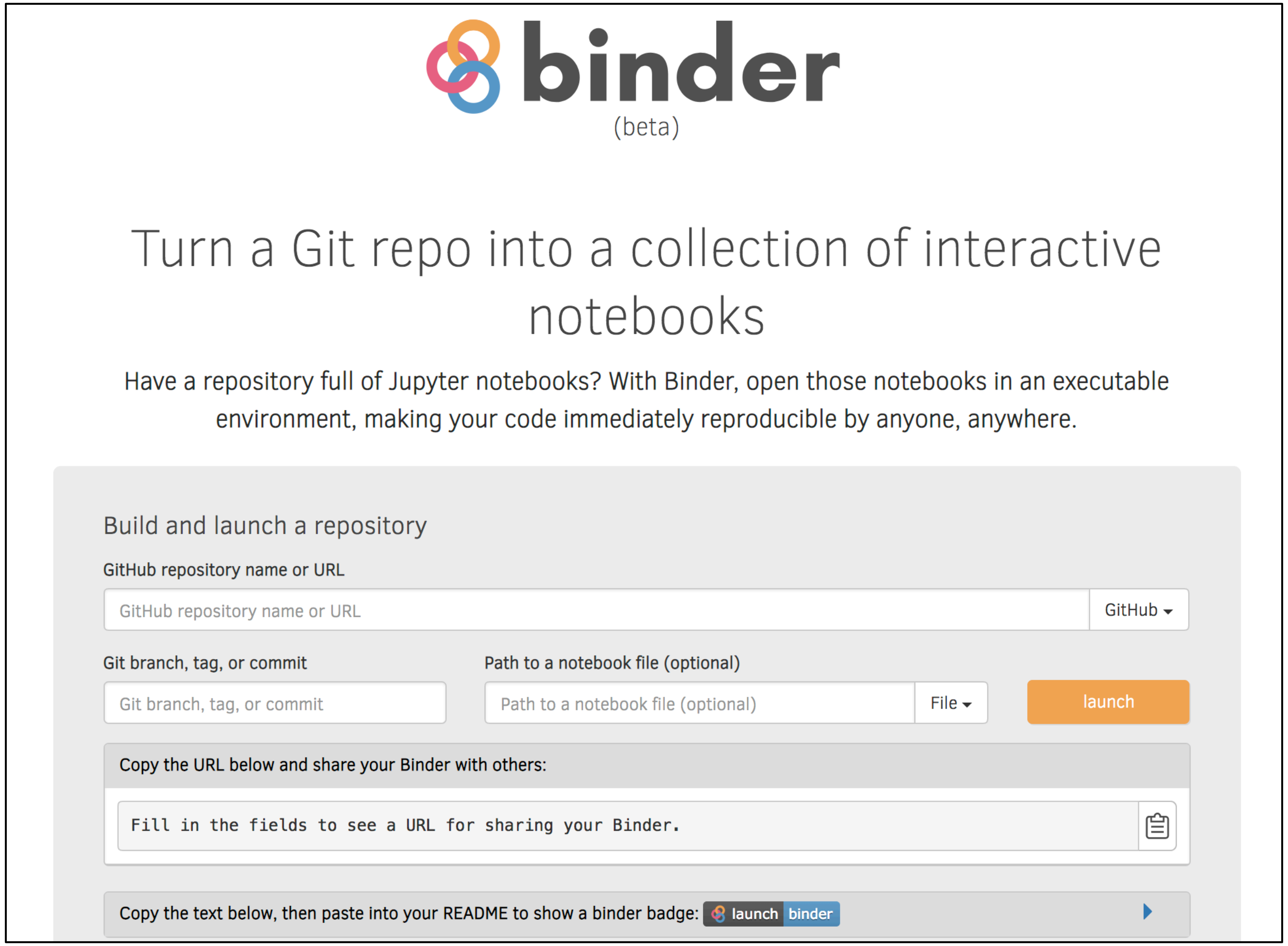


Fig 4. Results for the datasets OAI1-DESS (red), OAI1-T2 (green), OAI2-BL (cyan), and OAI2- le). (a) Violin plots describing the distribution of the DSC within each dataset. The dots represent DSC values spread around the y-axis to avoid visualization overlap. (b-d) Correlation between measurements derived from ground truth segmentations and pyKNEEr's segmentations, i.e. cartilage thickness (b), cartilage volume (c), and T_2 maps (d). (See data, code, computational environment)

tMSKr

Bonaretti S, Gold GE, Beaupre GS (2020) pyKNEEr: An image analysis workflow for open and reproducible research on femoral knee cartilage. PLoS ONE 15(1): e0226501

Share in an executable environment



binder (beta)

Turn a Git repo into a collection of interactive notebooks

Have a repository full of Jupyter notebooks? With Binder, open those notebooks in an executable environment, making your code immediately reproducible by anyone, anywhere.

Build and launch a repository

GitHub repository name or URL: GitHub repository name or URL [GitHub ▾]

Git branch, tag, or commit: Git branch, tag, or commit

Path to a notebook file (optional): Path to a notebook file (optional) [File ▾]

launch

Copy the URL below and share your Binder with others:

```
Fill in the fields to see a URL for sharing your Binder.
```

Copy the text below, then paste into your README to show a binder badge: launch binder

See practical tutorial: [How to share a Jupyter notebook with Binder?]()

So, why should we use Jupyter notebook in medical image analysis?

Because Jupyter notebooks allow us to:

- Do open and reproducible medical image analysis
- Create image analysis workflows that are complete
- Easily integrate our workflows into our papers