

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

Serena Bonaretti^{1,2}, Garry E. Gold¹, Gary S. Beaupre^{2,3}

¹Department of Radiology, Stanford University, Stanford, CA, USA

²Musculoskeletal Research Laboratory, VA Palo Alto Health Care System, CA, USA

³Department of Bioengineering, Stanford University, Stanford, CA, USA

<https://sbonaretti.github.io/>

Open and reproducible research

- Openness and computational reproducibility are recent movements in research ([Woelfle 2011](#), [Bollen 2015](#))
- They are essential to researchers to assess value of scientific claims and confidently build on previous work ([Sandve 2013](#))
- Transparent *image-based* research is crucial to provide reliable answers to biological questions ([Poldrack 2017](#))
- Other communities, e.g. neuroimaging, largely benefit from open-access tools ([Gorgolewski 2011](#))

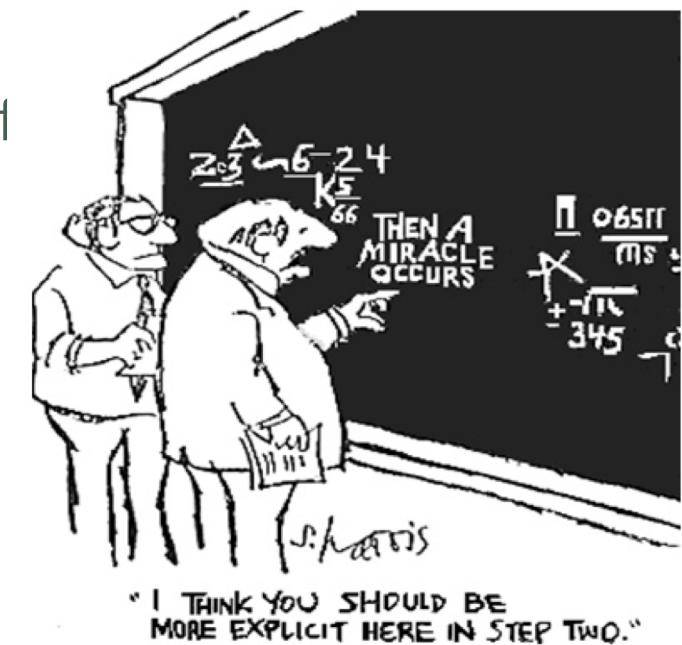
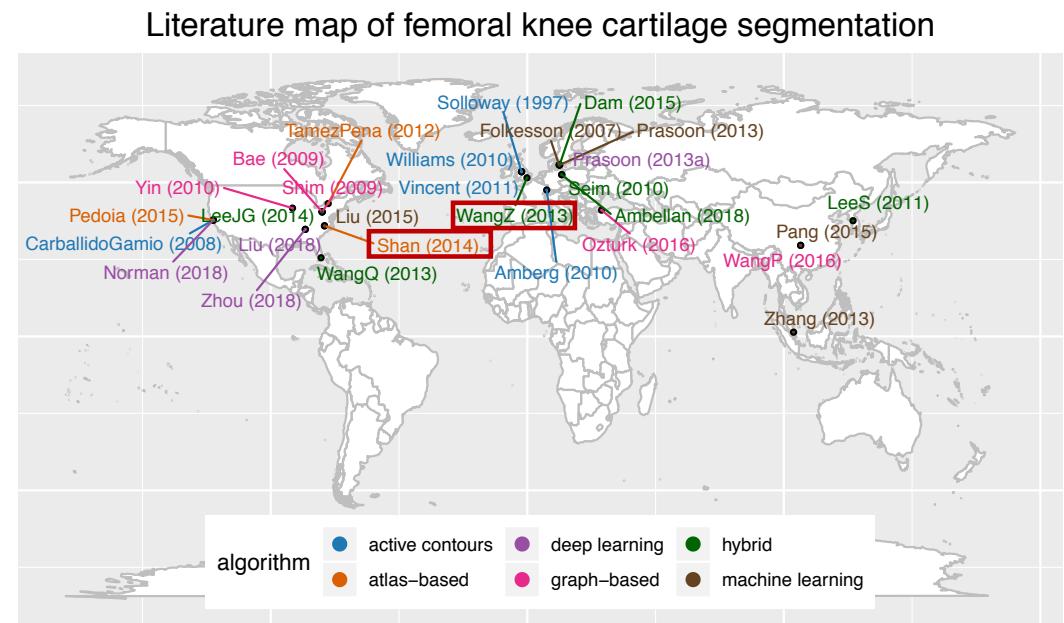


Image-based analysis of femoral knee cartilage

- In knee OA, analysis of femoral cartilage from MR images has assumed an important role ([Hafezi-Nejad 2018](#))
- Analysis workflows include:
 - Preprocessing
 - Segmentation
 - Morphology and Relaxometry
- Segmentation is a major challenge
 - Of 29 segmentation algorithms, only 2 have open source code ([Wang 2013](#), [Shan 2014](#))

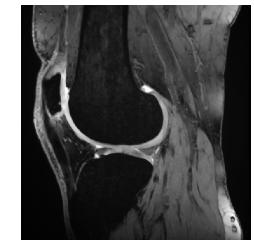


pyKNEEr: An image analysis workflow for open and reproducible research on femoral knee cartilage

Openness of pyKNEEr

- Code written in *python*
- Use of open file formats
 - *.mha, .txt, .csv*
- Modular structure to favor code expansion
- Reuse of developed code
 - Intensity preprocessing from Shan et al., *elastix*
- Available on GitHub
 - GNU GPLv3 license and DOI

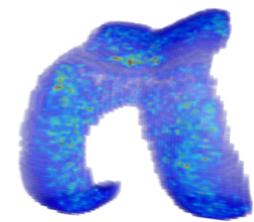
I. Preprocessing



II. Segmentation



III. Analysis



Reproducibility: Jupyter notebook

```
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 logarithmic 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 = "image_list_relaxometry_fitting_0411_72.txt"
• registration_flag = 0 # no registration, 1 = rigid registration
• method = 0 # exponential, 1 = linear
• output_file_name contains average and standard deviation of the fitting maps
In [1]: input_file_name = "image_list_relaxometry_fitting_0411_72.txt"
registration_flag = 0 # no registration, 1 = execute rigid registration
n_of_cores = 4
output_file_name = "exp_fit_aligned_0411_72.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 [1]: 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 occasional 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 [1]: if registration_flag == 1:
    rel.align_acquisitions(image_data, n_of_cores)

Compute the fitting
In [1]: 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 neighboring voxels projected on the bone surface side of the femoral cartilage
In [1]: rel.show_fitting_map(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 [1]: # ID of the map to visualize (This ID is the one in the 2D visualization above)
image_ID = -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 [1]: 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 analysis
In [1]: rel.show_fitting_table(image_data, output_file_name)

References
[1] Borhani A, Weston AJ, Doganay A, Arnett SV, Rajani R, Charniga S, Reddy R. In vivo measurement of T1ho relaxation in the human knee at 3.3 T. Magn Reson Imaging. Apr;19(4):623-4. 2001
[2] LUX, Beumer, Van C, Lin TM, Castillo D, O, Blumenkrantz G, Lazaro J, Comendador-Gamio J, Ries M, Majumdar S. In vivo T1o and T2 mapping of articular cartilage in osteoarthritis of the knee using 3.7 MHz. Osteoarthritis Cartilage. Jul;15(7):789-97. 2007

Dependencies
In [1]: !load_ext watermark
watermark --v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

- Open-source web application integrating
 - Live code
 - Narrative text with equations
 - Visualizations
- Versatile
- Easy to share among researchers

Reproducibility of pyKNEEr: Jupyter notebook

```
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 logarithmic 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 = "image_list_relaxometry_fitting_0411_72.txt"
• registration_flag = 0 # 0 = no registration, 1 = rigid registration
• n_of_cores = 4
• output_file_name = "exp_fit_aligned_0411_72.csv"

In [1]: input_file_name = "image_list_relaxometry_fitting_0411_72.txt"
registration_flag = 1 # 0 = no registration, 1 = execute rigid registration
n_of_cores = 4
output_file_name = "exp_fit_aligned_0411_72.csv"

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

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

Compute the fitting
In [1]: 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 neighboring voxels projected on the bone surface side of the femoral cartilage

In [1]: rel.show_fitting_map(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 [1]: # ID of the map to visualize (This ID is the one in the 2D visualization above)
image_ID = -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 = vtkImageViewer()
viewer.SetInputData(image)
viewer.Update()

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

In [1]: 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 analysis

In [1]: rel.show_fitting_table(image_data, output_file_name)

Dependencies
In [1]: %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

Code Issues Pull requests Projects Wiki Insights

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

Manage topics

132 commits 1 branch 0 releases

Branch: master New pull request Create new I

sbonaretti Update README.md

code docs publication .DS_Store README.md environment.yml

6

Reproducibility of pyKNEEr: Jupyter notebook

```
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 logarithmic 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 = "image_list_relaxometry_fitting_0411_72.txt"
• method_flag = "exp" # execute exponential fitting
• registration_flag = 1 # # = no rigid registration, 1 = execute rigid registration
• n_of_cores = 4
• output_file_name = "exp_fit_aligned_0411_72.csv"

In [1]: input_file_name = "image_list_relaxometry_fitting_0411_72.txt"
registration_flag = 1 # # = no rigid registration, 1 = execute rigid registration
n_of_cores = 4
output_file_name = "exp_fit_aligned_0411_72.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 [1]: 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 occasional 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 [1]: if registration_flag == 1:
    rel.align_acquisitions(image_data, n_of_cores)

Compute the fitting
In [1]: 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 neighboring voxels projected on the bone surface side of the femoral cartilage

In [1]: rel.show_fitting_map(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 [1]: # ID of the map to visualize (The ID is the one in the 2D visualization above)
image_ID = -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 [1]: 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 analysis
In [1]: rel.show_fitting_table(image_data, output_file_name)

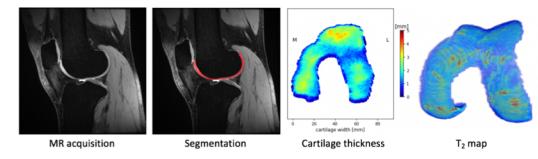
Dependencies
In [1]: !load_ext watermark
!watermark -v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

- 1. Link to GitHub repository
- 2. Link to documentation

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

pyKNEEr

Introduction
Installation
Preprocessing
Segmentation
Morphology
Relaxometry
FAQ



pyKNEEr is an image analysis workflow for **open** and **reproducible** research on **femoral knee cartilage**. It is implemented in **python** with **Jupyter notebooks**.



Reproducibility of pyKNEEr: Jupyter notebook

```
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 logarithmic 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 = "image_list_relaxometry_fitting_0411_72.txt"
• registration_flag = 0 # 0 = no registration, 1 = rigid registration
• n_of_cores = 4
• output_file_name = "exp_fit_aligned_0411_72.csv"

In [1]: input_file_name = "image_list_relaxometry_fitting_0411_72.txt"
registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
n_of_cores = 4
output_file_name = "exp_fit_aligned_0411_72.csv"

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

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

Compute the fitting
In [1]: 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 neighboring voxels projected on the bone surface side of the femoral cartilage

In [1]: rel.show_fitting_map(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 [1]: # ID of the map to visualize (The ID is the one in the 2D visualization above)
image_ID = -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 [1]: 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 analysis

In [1]: rel.show_fitting_table(image_data, output_file_name)

Dependencies
In [1]: %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

Reproducibility of pyKNEEr: Jupyter notebook

```
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 logarithmic 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 [1]: input_file_name = "image_list_relaxometry_fitting_0411_72.txt"
registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
n_of_cores = 4
output_file_name = "exp_fit_aligned_0411_72.csv"

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

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

Compute the fitting
In [1]: 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 neighboring voxels projected on the bone surface side of the femoral cartilage

In [1]: 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 [1]: # ID of the map to visualize (The ID is the one in the 2D visualization above)
image_ID = -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 [1]: 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 analysis
In [1]: rel.show_fitting_table(image_data, output_file_name)

Dependencies
In [1]: %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

...

Reproducibility of pyKNEEr: Jupyter notebook

```
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 logarithmic 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 [1]: input_file_name = "image_list_relaxometry_fitting_0411_72.txt"
registration_flag = 1 # # = no rigid registration, 1 = execute rigid registration
n_of_cores = 4
output_file_name = "exp_fit_aligned_0411_72.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 [1]: 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 occipital 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 [1]: if registration_flag == 1:
    rel.align_acquisitions(image_data, n_of_cores)

Compute the fitting
In [1]: 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 neighboring voxels projected on the bone surface side of the femoral cartilage

In [1]: 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 [1]: # ID of the map to visualize (The ID is the one in the 2D visualization above)
image_ID = -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 [1]: 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 analysis

In [1]: rel.show_fitting_table(image_data, output_file_name)

References
[1] Borhani A, Heenan AJ, Gokcecelci A, Arnett SV, Rajani R, Charniga S, Reddy R. In vivo measurement of T1ho
decays in the human knee at 3.0 Tesla. J Magn Reson Imaging. Apr;19(4):620-5. 2004.
[2] U.K. Bernmark, M.C. Lin, T.M. Castillo D.O. Blumensztajn G, Lázaro J, Comendador-García J, Ries M, Majumdar S. In vivo T1o
and T2 mapping of articular cartilage in osteoarthritis of the knee using 3.7 MHz. Osteoarthritis Cartilage. Jul;15(7):789-97. 2007.

Dependencies
In [1]: !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)
```

Reproducibility of pyKNEEr: Jupyter notebook

```
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 logarithmic 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
• registration_flag is 0 for no registration, 1 for rigid registration
• output_file_name contains average and standard deviation of the fitting maps
registration_flag = 1 # # = no rigid registration, 1 = execute rigid registration
n_of_cores = 4
output_file_name = "exp_fit_aligned_0411_72.csv"

In [1]: input_file_name = "image_list_relaxometry_fitting_0411_72.txt"
registration_flag = 1 # # = no rigid registration, 1 = execute rigid registration
n_of_cores = 4
output_file_name = "exp_fit_aligned_0411_72.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 [1]: 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 occasional 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 [1]: if registration_flag == 1:
    rel.align_acquisitions(image_data, n_of_cores)

Compute the fitting
In [1]: 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 neighboring voxels projected on the bone surface side of the femoral cartilage

In [1]: rel.show_fitting_map(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 [1]: # ID of the map to visualize (This ID is the one in the 2D visualization above)
image_ID = -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 [1]: 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 analysis

In [1]: rel.show_fitting_table(image_data, output_file_name)

References
[1] Borhani A., Heinen A.J., Deograde A.J., Arnett S.V., Ringer R., Charniga S.R., Reddy R. In vivo measurement of T1ho decay in the human knee joint. J. Magn. Reson. Imaging. Apr;19(4):620-5. 2004.
[2] U.X., Bernmarksson C., Lin T.M., Castillo D.O., Blumenkrantz G., Lazzar J., Comendado-Gamio J., Ries M., Majumdar S. In vivo T1 and T2 mapping of articular cartilage in osteoarthritis of the knee using 3.7 MHz. Osteoarthritis Cartilage. Jul;15(7):789-97. 2007.

Dependencies
In [1]: !load_ext watermark
watermark --v -m -p SimpleITK,matplotlib,numpy,pandas,scipy,itkwidgets,multiprocessing
```

- 1. Link to GitHub repository
- 2. Link to documentation

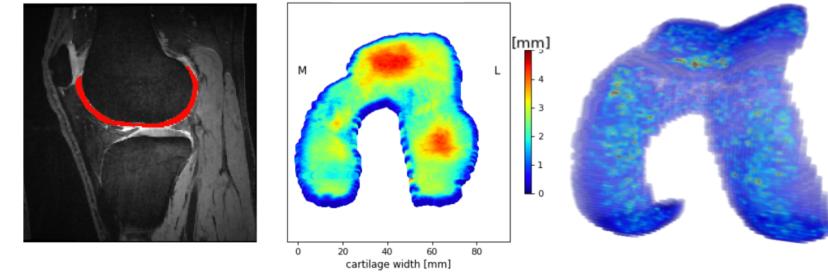
3. Introduction

4. User inputs

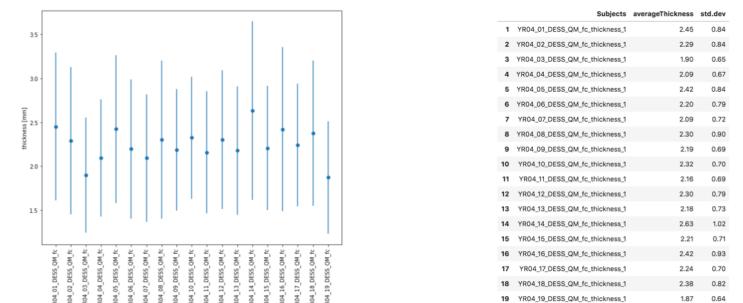
5. Commands with narrative

6. Visualization of outputs

Qualitative visualizations



Quantitative visualizations



Reproducibility of pyKNEEr: Jupyter notebook

```
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 logarithmic 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 = "image_list_relaxometry_fitting_0411_72.txt"
• registration_flag = 1 if # = no rigid registration, 0 = execute rigid registration
• n_of_cores = 4
• output_file_name = "exp_fit_aligned_0411_72.csv"

In [1]: input_file_name = "image_list_relaxometry_fitting_0411_72.txt"
registration_flag = 1 # # = no rigid registration, 0 = execute rigid registration
n_of_cores = 4
output_file_name = "exp_fit_aligned_0411_72.csv"

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

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

Compute the fitting
In [1]: 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 neighboring voxels projected on the bone surface side of the femoral cartilage

In [1]: rel.show_fitting_map(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 [1]: # ID of the map to visualize (This ID is the one in the 2D visualization above)
image_ID = -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 [1]: 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 analysis

In [1]: rel.show_fitting_table(image_data, output_file_name)

Dependencies
In [1]: %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

- [1] Borhani A., Weston A.J., Gopinath A., Anelli S.V., Rajani R., Chirayath S.R., Reddy R. *In vivo measurement of T1 mapping in the human knee at 3.3 T*. *Int J Magn Reson Imaging*. Apr;19(6):63-6. 2001.
- [2] U.X., Bernardo-Ma C., Lin T.M., Castillo D.O., Blumenkrantz G., Lazoza J., Contreras-Garcia J., Ries M., Majumdar S. *In vivo T1 and T2 mapping of articular cartilage in osteoarthritis of the knee using 3.7 MHz*. *Osteoarthritis Cartilage*. Jul;15(7):789-97. 2007.

Reproducibility of pyKNEEr: Jupyter notebook

```
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 logarithmic 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
• registration_flag is 0 for no registration, 1 for rigid registration
• output_file_name contains average and standard deviation of the fitting maps

In [1]: input_file_name = "image_list_relaxometry_fitting_0411_72.txt"
registration_flag = 1 # 0 = no rigid registration, 1 = execute rigid registration
n_of_cores = 4
output_file_name = "exp_fit_aligned_0411_72.csv"

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

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

Compute the fitting
In [1]: 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 neighboring voxels projected on the bone surface side of the femoral cartilage

In [1]: rel.show_fitting_map(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 [1]: # ID of the map to visualize (This ID is the one in the 2D visualization above)
image_ID = -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 = vtkImageViewer()
viewer.SetInputData(image)
viewer.SetGradientOpacity(0.0, ui.collapsed=False, shadow=False)
viewer.Update()

GRAPH: Dots represent the average value of fitting maps per image; bars represents the standard deviation
In [1]: 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 analysis

In [1]: rel.show_fitting_table(image_data, output_file_name)

References
[1] Borhani A, Heinen AJ, Gopinath A, Arnett SV, Rajani R, Charniga S, Reddy R. In vivo measurement of T1 mapping in the human knee at 3.0 Tesla. J Magn Reson Imaging. Apr;19(4):620-5, 2004.
[2] U.X. Behera, M.C. Lin,T.M., Castillo D.O., Blumenkrantz G., Lazaro J., Contreras-Garcia J., Ries M., Majumdar S. In vivo T1 and T2 mapping of articular cartilage in osteoarthritis of the knee using 3.7 MHz. Osteoarthritis Cartilage. Jul;15(7):789-97, 2007.

Dependencies
In [1]: %load_ext watermark
%watermark -v -m -p 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

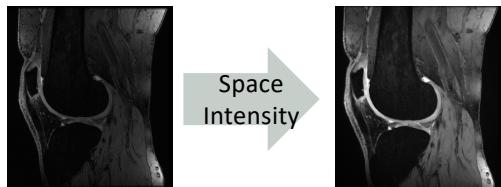
CPU cores : 4

interpreter: 64bit

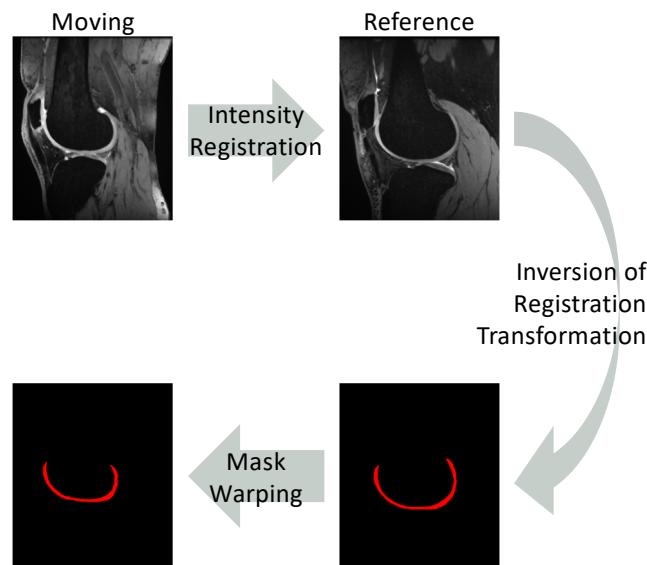
→ Reproducibility of computational environment

Algorithms in pyKNEEr

I. Preprocessing

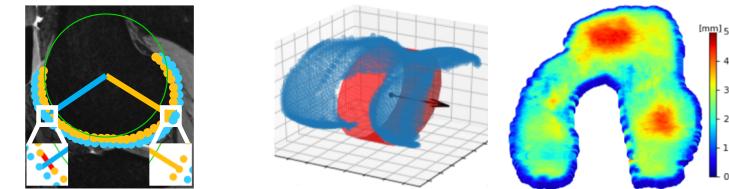


II. Segmentation

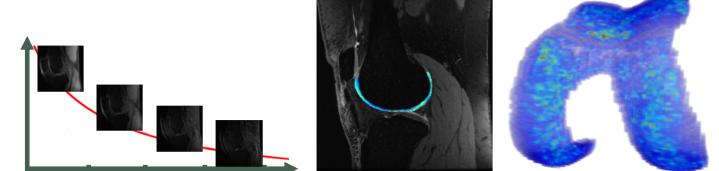


III. Analysis

Morphology



Relaxometry

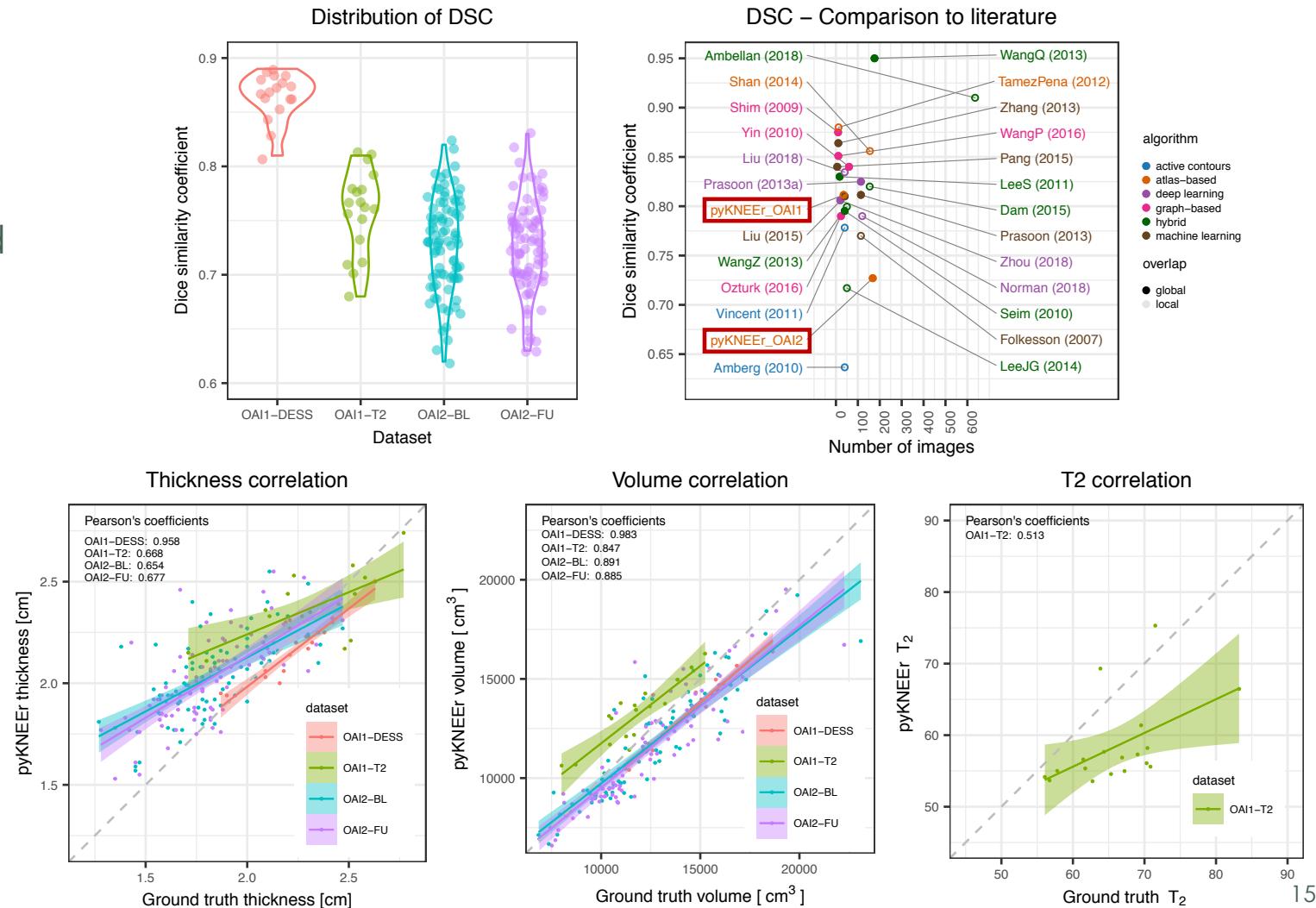


Validation experiments

- OAI1: 19 subjects
 - OAI1-DESS
 - OAI1-T2
 - G. Truth: atlas-based
([TamezPena 2012](#))

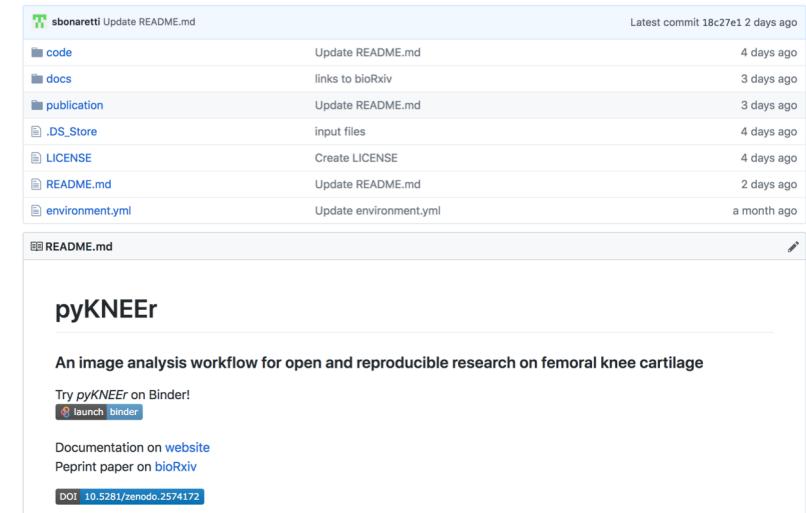
- OAI2: 88 subjects
 - OAI2-DESS BL
 - OAI2-DESS FU
 - G. Truth: AAM
([Vincent 2011](#))

Data on [Zenodo](#)
 Code on [GitHub](#)
 Analysis on [binder](#)
 Paper on [bioRxiv](#)



Discussion

- pyKNEEr can facilitate transparent research
 - Open source
 - Notebooks can be shared among researchers
 - Notebooks can be linked to publications
- Do you want to use pyKNEEr?
 - Go to terminal and type: `pip install pyKNEEr`
 - Download Jupyter notebooks from [GitHub](#)
 - Follow the instruction on the [documentation](#) (demo)
- Do you want to contribute to pyKNEEr?
 - Go to [GitHub](#) and create a branch
 - Implement your contribution
 - Segmentation: Machine and deep learning, hybrid, ...
 - Thickness algorithms: surface normal vectors, potential field lines, ...
 - Other knee tissues: tibial cartilage, patellar cartilage, menisci, ...
 - Send a pull request and your code will be merged to the main project



The screenshot shows the GitHub repository page for 'pyKNEEr' by 'sbonaretti'. The repository has 1 commit, 12 files, and 1 branch. The README.md file was updated 2 days ago. Other files like code, docs, and publication were updated recently. The repository description is 'An image analysis workflow for open and reproducible research on femoral knee cartilage'. It includes a 'Try pyKNEEr on Binder!' button, documentation links, and a DOI link.

<https://github.com/sbonaretti/pyKNEEr>

Acknowledgments

- Uche Monu for her insights on cartilage segmentation
- Bragi Sveinsson for sharing his code on EPG modeling
- Valentina Mazzoli for her insight on relaxometry
- Christof Seiler for software engineering support
- Amy Silder, Julie Kolesar, and Scott Uhlrich for beta testing
- All the researchers that make their code open-source, on whose effort we could create pyKNEEr
- Merit Review Award Number I01 RX001811 from the United States Department of Veterans Affairs Rehabilitation R&D Service
- NIH EB002524 and NIH AR062068

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

Serena Bonaretti^{1,2}, Garry E. Gold¹, Gary S. Beaupre^{2,3}

¹Department of Radiology, Stanford University, Stanford, CA, USA

²Musculoskeletal Research Laboratory, VA Palo Alto Health Care System, CA, USA

³Department of Bioengineering, Stanford University, Stanford, CA, USA

<https://sbonaretti.github.io/>