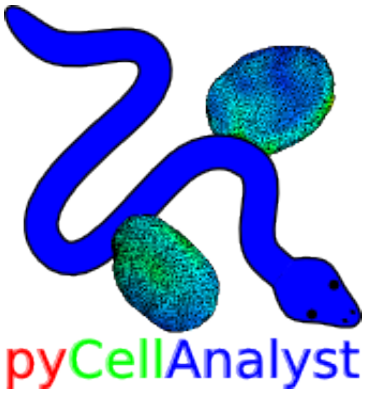
pyCellAnalyst Documentation

Release 1.1

Scott Sibole

Dec 20, 2017

Contents


pyCellAnalyst

CHAPTER 1

Installation

1.1 Anaconda

pyCellAnalyst now requires Anaconda Python from Continuum Analytics. You can install either the full Anaconda package or Miniconda.

Note: pyCellAnalyst is now built for Python 3. If you wish to run it in your root environment, you must install Anaconda3 or Miniconda3. Alternatively, you can create a conda environment using Python 3.

1.2 Installing from Anaconda Cloud

1. Add additional channels to fetch packages from.

```
conda config --add channels conda-forge
conda config --add channels SimpleITK
conda config --add channels siboles
```

2. Install

   To install to your root conda environment:

```
conda install -c siboles pycellanalyst
```

   To install to a conda virtual environment for just pyCellAnalyst:

```
conda create --name pyCell python=3.6 pycellanalyst
```

1.3 Optional

It is still possible to install pyCellAnalyst to a standard Python build. Although the dependencies must be resolved by the user. pyCellAnalyst depends on the following:

Available in PyPi

- numpy
- scipy
- scikit-learn
- matplotlib
- wquantiles
- xlrd
- xlwt
- trimesh
- febio

Others:

- VTK
- SimpleITK
- tetmesh

1.4 Troubleshooting

- If the Anaconda version of Python does not start when you type python in a command terminal you likely need to unset your PYTHONPATH and PYTHONHOME variables.
- Other problems with Anaconda can be researched here
- If the pycellanalyst does not work with conda install on Linux, the user can build the package themself. - Download the source code - Unzip and open a command terminal in the *src* directory containing *build.sh*. - Type the following:

```
conda build .
conda install --use-local pycellanalyst
```

CHAPTER 2

Introduction

2.1 Object Segmentation

An example usage is as follows:

```
from pyCellAnalyst import Volume

# Instantiate a Volume object with default parameters
# This will execute the segmentation and populate all class attributes
# Note: PATH_TO_IMAGE_DIRECTORY must be changed to the location of the either
# a directory containing a single sequence of 2-D TIFF or a single NiFTi format ".nii"
# 3-D image. Likewise, each [int, int, int, int, int, int] must be replaced by the
# position and size of regions of interest (as indices)
# Please consult the pyCellAnalyst.Voluume class reference below to explore parameter␣
↪options.
vol = Volume.Volume("PATH_TO_IMAGE_DIRECTORY", regions=[,[int, int, int, int, int,␣
↪int],...])
# print the volumes of each segmented object
print vol.volumes
```

This code block will create a new directory "PATH_TO_IMAGE_DIRECTORY"+"_results" where the segmented object surfaces are saved in stereolithographic format (.stl). Also, a 3-D label image of the reconstructed objects "labels.nii" is also saved here.

2.2 Deformation Analysis

1. The moment of inertia tensor, $\mathbf{I}$ is calculated discretely for each object in the reference and deformed states. An ellipsoid with the same principal moments of inertia, the eigenvalues of $\mathbf{I}$, is then determined for each reference and deformed object. Deformation can then be characterized as the stretch along the ellipsoid axes between the reference and deformed states. This is analagous to the principal stretches (which of course can also be converted to strains); however, it is impossible to separate rigid body rotation from shear with this approach.

2. The optimal affine transformation mapping the each reference surface to its deformed pair can be calculated using an interative closest point minimization. The objective function to minimize is the sum of the Euclidean distances from the vertices of the reference surface after affine transformation to their nearest neighbor vertices on the deformed surface. By default a rigid body transformation will first be optimized by the same method, before attempting to dfind the best affine transformation. Assuming uniform deformation on the object, the linear transformation matrix from this optimal affine transformation is the deformation gradient, $\mathbf{F}$. The Green-Lagrange strain tensor, $\mathbf{E} = \frac{1}{2}(\mathbf{F}^T.\mathbf{F} - \mathbf{I})$ is then calculated for each object pair.
3. Deformable image registration can be performed to determine the optimal diffeomorphism between the reference and deformed objects. This will yield a displacement vector field in the reference state that is then interpolated to the object vertices. Images are reconstructed from the object surfaces at a user-specified precision expressed as a ratio of the bounding box edge lengths e.g. 0.01 will result in 100 voxels in the i, j, and k directions with length of $\frac{L_x}{100}$, $\frac{L_y}{100}$, and $\frac{L_z}{100}$.

To perform deformation analysis on these segmented objects:

```
from pyCellAnalyst import CellMech

# Instantiate a CellMech object with default parameters
# This will calculate object volumes, ellipsoids of equivalent principal moments of␣
↪inertia,
# and the optimal affine transformation between reference and deformed object pairs.
# Please consult the pyCellAnalyst.CellMech class reference below to explore␣
↪parameter options.
mech = CellMech.CellMech("PATH_TO_REFERENCE_DIRECTORY", "PATH_TO_DEFORMED_DIRECTORY")
```

CHAPTER 3

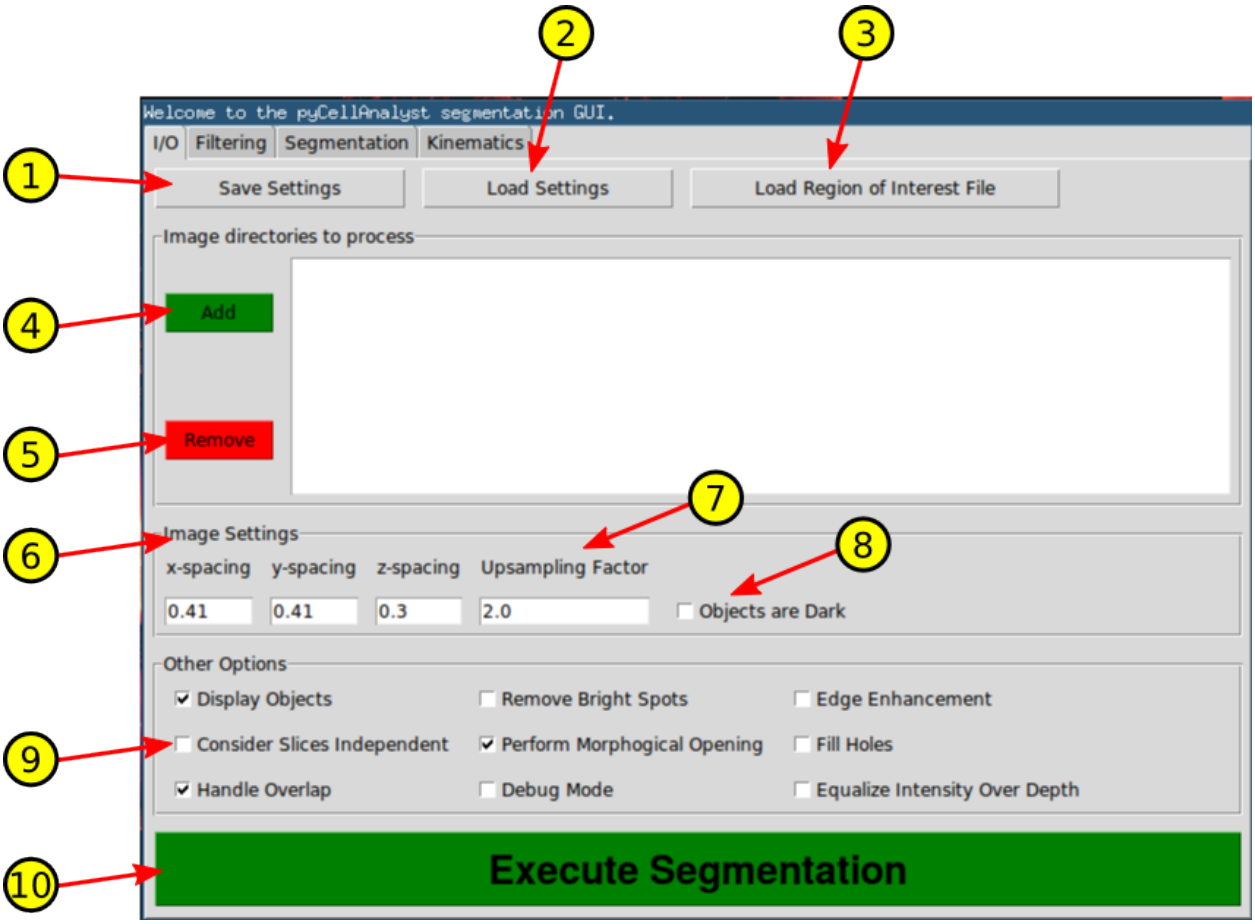
Segmentation and Analysis Graphical User Interface

A graphical user interface (GUI) is packaged within the pyCellAnalyst module to allow for access to all features without a need for a knowledge of the Python language. To start the GUI, open a command terminal anywhere and type:

```
python -m pyCellAnalyst.GUI
```

An example analysis video using ImageJ and this GUI can be found here.

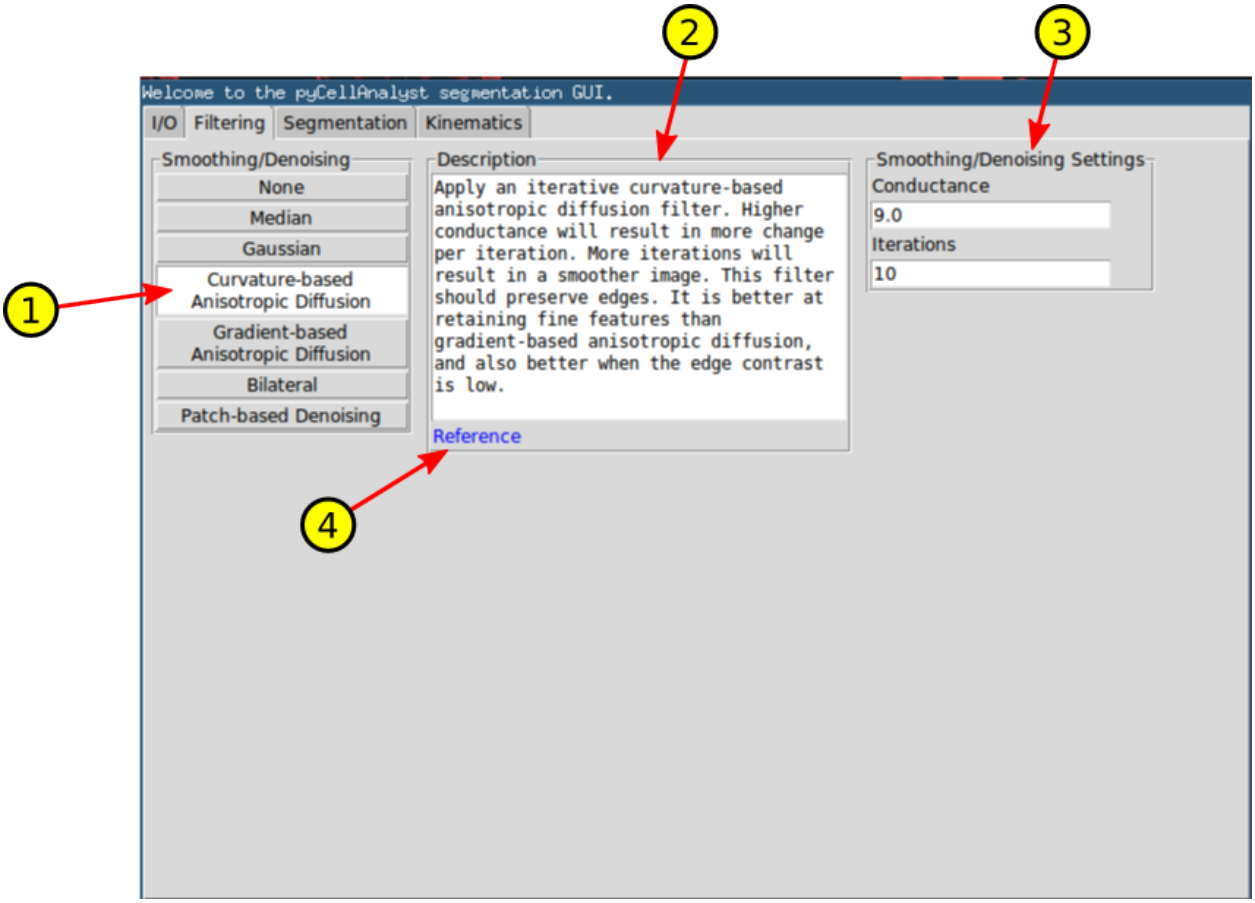
3.1 I/O Tab - Input and Output, Segmentation Execution, Extra Options



1. Save settings used in current analysis to a Python pickle file.
2. Load settings from a previously saved Python pickle file.
3. Load an Excel (.xls) file containing region of interest definitions for each image directory.
   - All regions of interest applicable to a specific image directory must be on a individual sheets.
   - The sheets must be in the order that the image directories are added.
4. Add and the path to a directory containing the images.
   - Directories must be added corresponding to the order of the sheets in the region of interest file.
   - Directory paths will appear in the white box to the right after addtion.
5. Remove selected directory from the list.
6. Physical dimensions of image voxels.
   - If there is a necessary correction physical dimensions of the voxels, e.g., for laser scanning microscopy the z-spacing should be adjusted to account for depth distortion, please account for it here.
7. Upsample the image by this factor. This can help when an active contour model is employed.
8. Check *Objects are Dark* if the objects of interest appear darkest in the image.

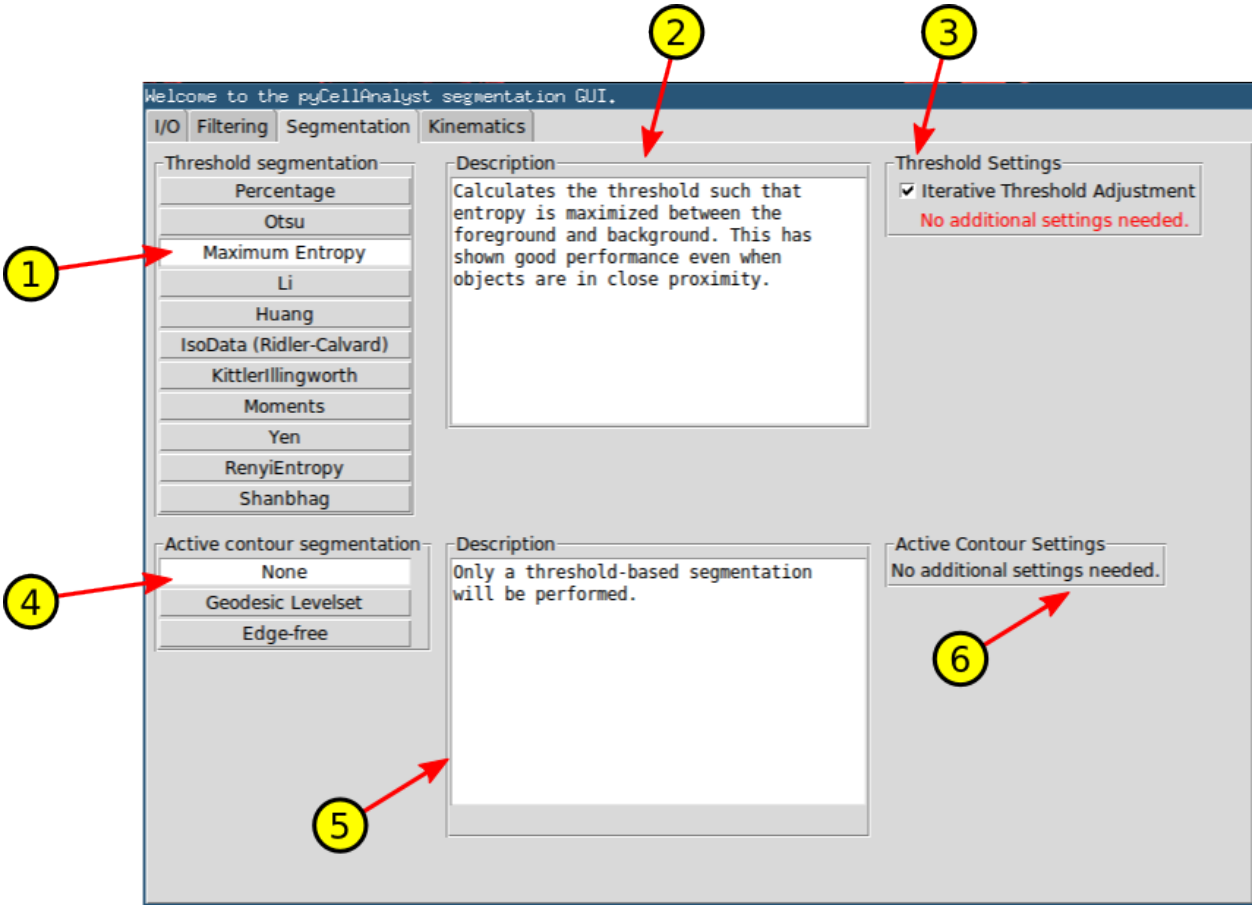
9. Other options for the image processing and segmentation.
    - *Display Objects* - If checked will spawn a 3-D interactive rendering of the reconstructed objects with label indicated by color.
    - *Remove Bright Spots* - Will replace bright spots greater than or equal to the 98:sup:*th* percentile intensity.
    - *Edge Enhancement* - Enhance the edges in each region of interest using Laplacian sharpening.
    - *Consider Slices Independent* - Perform processing and segmentation on each slice in stack independently in 2-D and reassemble results. This is not recommended except for specialized cases.
    - *Perform Morphological Opening* - If checked will perform a single voxel erosion followed by a single voxel dilation. This will remove islands and spurious connections with typically negligible effects on the segmentation. If the object of interest has exceptionally thin features it may be necessary to disable this.
    - *Fill Holes* - If checked all holes completely internal to the segmented objects will be filled.
    - *Handle Overlap* - If checked a support vector machine classification will be performed if any objects overlap.
    - *Debug Mode* - Outputs addtional images to disk for debugging purposes (smoothed image, seed images for active contour models).
    - *Equalize Intensity Over Depth* - Performs a linear correction of intensity changes with depth, so all slices are of similar intensity.
10. Perform the segmentation.

3.2 Filtering Tab - Image Smoothing and Denoising Methods



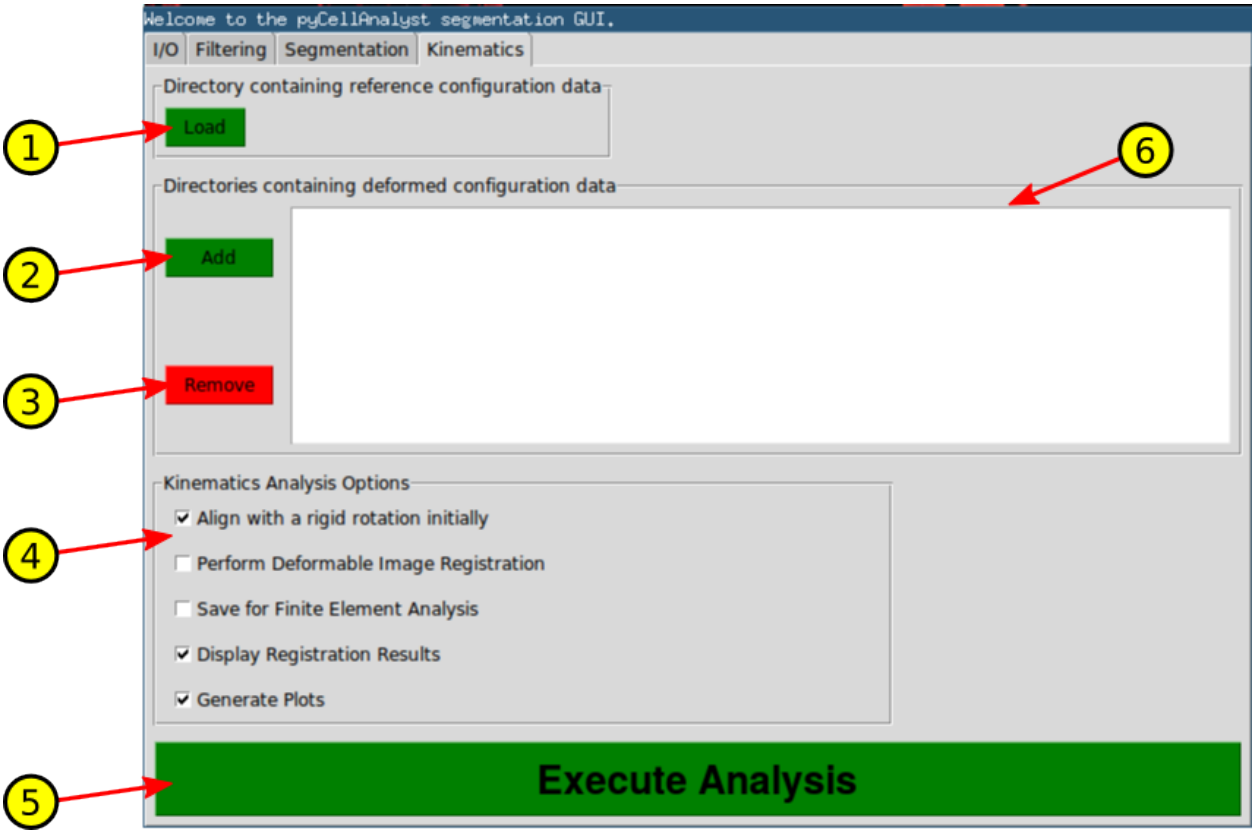
1. Image smoothing and denoising methods.
2. A brief description of the currently selected method, and general advice on its use.
3. The adjustable parameters for the currently selected method.
4. Hyperlinks to the appropriate theoretical reference if available. This requires and internet connection and will open in the OS default web browser.

3.3 Segmentation Tab - Object Segmentation and Display



1. Thresholding method that can either serve as the complete segmentation method, or provide the initial seed for an active contour model.
2. Brief description of the currently selected thresholding method.
3. Threshold settings for currently selected method. All by the *Percentage* method require no additional settings.
    - If *Iterative Threshold Adjustment* is checked, the threshold will be adjusted until the thresholded object no longer touches the edge of the region of interest definition.
4. Selection of active contour model to use.
5. Brief description of the currently selected active contour model with general advice on its usage.
6. Adjustable parameters for the currently selected active contour model.

3.4 Kinematics Tab - Deformation Analysis Methods



1. Load the directory containing the polygonal surfaces resulting from object segmentation of the reference state.
2. Load each directory containing segmented surfaces for each deformed state.
3. Remove the selected deformed directory(ies) from the list.
4. Options for the deformation analysis.
    - Before performing the optimization to determine the best affine transformation between objects, will first find the optimal rigid body translation and rotation to align objects if checked.
    - If checked, perform deformable image registration to find the best non-uniform deformation that maps the reference to the deformed object(s).
    - If checked, save 3-D mesh information and displacements calculated by deformable image registration interpolated to the mesh boundary to a Python pickle file that can later be used to fully generate and solve finite element analyses of the the deformation.
    - If checked, spawns a 3-D interactive rendering of the non-uniform displacements resulting from deformable image registration. The deformation can be animated by pressing the RIGHT-ARROW. Also, the frames of the animation can be saved to disk by pressing the UP-ARROW.
    - If checked, bar plots of the results from the ellipsoidal and affine transformation methods reporting cell strains.

CHAPTER 4

Finite Element Analysis Graphical User Interface

4.1 Pre-requisites

This graphical user intereface (GUI) uses the FEBio finite element solver, which can be downloaded here. FEBio is aimed at solving problems in biomechanics, which often have both geometric and material non-linearity as well as anisotropy and multiple physical phases. This and the fact that FEBio is open-source make it an excellent solver for finite element analysis of the cells.

After running the install wizard, it may be advantage to add the FEBio binary (.exe in Windows, .lnx64 (arbitrary) in Linux) to your system path. This is not necessary for this utility though.

4.2 Peforming an Analysis

Start the GUI, by opening a command terminal and typing:

```
python -m pyCellAnalyst.FEA_GUI
```

4.2.1 Running for the First Time

The GUI looks for a file in your HOME directory called .pyCellAnalystFEA.pth containing the path to your FEBio binary. If this is not found or the path contained within is incorrect, a file browser will spawn instructing you to navigate to the FEBio executable and select it. Do this and the file will be generated or modified.

4.2.2 Importing Model Definition Pickles

When the option to save for FEA is active and a deformable image registration analysis is performed, files with the naming convention **cellFEA{:02d}.pkl** will be written to the corresponding results directory. These files contain all the necessary information to generate a finite element model of the cell and its deformation except for material properties.

Simply click the **Add** button on the Analysis tab and select pickle files for the cell(s) you wish to model. One or more files can be selected at a time. To remove files that were added simply select them from the listbox and click **Remove**

4.2.3 Setting Outputs

Checkboxes are provided to indicate what variables to output as results. Select or deselect these as you wish.

4.2.4 Setting Analysis Options

The GUI will automatically generate plots of the results for each analysis and write them to a directory named FEA_analysis_{TIME_STAMP} one directory level above the pickle folders. The options are as follows:

- Generate Histograms - For each cell, a histogram of each selected output variable will be generated from the values calculated for each element. The histograms are volumetrically weighted, such that the integral area is always 1. This will be done for all mechanical treatments of the cell included in the analysis.
- Tukey Boxplots - For each cell, a volumetrically weighted Tukey boxplot will be generated for each selected output variable. This will be done for all mechanical treatments of the cell included in the analysis.
- Calculate Differences - Since the finite element mesh used for each cell in the analysis is the same for all mechanical treatments (the reference state mesh), paired differences for each output variable can be calculated for all elements. The root-mean-square differences for all combinations of treatments within a cell are written to disk as heatmaps. If *Convert to VTK* is also selected the differences are also saved on each tetrahedron.
- Convert to VTK - An unstructured grid representation of each cell will be written in VTK format (.vtu). All selected output variables will be saved on the tetrahedrons. These can be further visualized and analyzed in software such as Paraview.

4.2.5 Assigning Material Properties

The second tab in the GUI, *Material Model*, provides options to define the material properties of the cells. The simplest model would be an isotropic ground substance such as a neo-Hookean or Mooney-Rivlin material with no tensile network.

To attempt to model the cytoskeleton, a transversely-isotropic ground substance can be selected to represent the microtubules. The symmetry axes for this material are oriented perpendicular to the local iso-distance contours measured from the cell surface.

To model the actin filaments, *Tensile Fibres* can be added. The contribution of these is modelled as a probability density function in spherical space. **ksi1** is the axis of an ellipsoid oriented tangent to the local iso-distance contour, and can be thought of as the fibre stiffness in that direction. **ksi2** and **ksi3** are the other ellipsoidal axes and are forced to be equal. Likewise, the parameters **beta1**, **beta2**, and **beta3**, also represent the ellipsoidal axes of a probability density function, but these govern the non-linearity of the fibre stiffnesses. Since a derivative is taken, values of 2 for **beta** represent linear stiffness behaviour, and also the lower bound allowed for the value.

CHAPTER 5

Module Classes

The autodocumentation for the pyCellAnalyst module is provided below.

class pyCellAnalyst.**Volume**(*vol_dir, output_dir=None, regions=None, pixel_dim=[0.411, 0.411, 0.6835], stain='Foreground', segmentation='Geodesic', smoothing_method='Curvature Diffusion', smoothing_parameters={}, two_dim=False, bright=False, enhance_edge=False, depth_adjust=False, display=True, handle_overlap=True, debug=False, opening=True, fillholes=True*)

This class will segment objects from 3-D images using user-specified routines. The intended purpose is for laser scanning fluorescence microscopy of chondrocytes and/or their surrounding matrices. Nevertheless, this can be generalized to any 3-D object using any imaging modality; however, it is likely the segmentation parameters will need to be adjusted. Therefore, in this case, the user should set segmentation='User' during Class instantiation, and call the segmentaion method with appropriate parameters.

Parameters

- **vol_dir** (*str*) – This is required. Currently it is the path to a directory containing a stack of TIFF images or a single NifTi (.nii) file. Other formats may be supported in the future.
- **output_dir** (*str, optional*) – The directory in which to save results. If not specified, a directory **vol_dir** + '_results' will be created and used.
- **regions** (*[,[int, int, int, int, int, int], ..], optional*) – Cropped regions bounding a single object to segment. In terms of voxel indices the order for each region is: [top left corner x, y, z, box edge length Lx, Ly, Lz]. If not specified, the entire image is considered as the region.
- **pixel_dim** (*[float=0.411, float=0.411, float=0.6835], optional*) – The physical dimensions of the voxel ordered x, y, and z. If there is a need to correct a dimesion such as for the depth distortion in laser scanning microscopy, it should be incorporated here. Defaults to [0.411, 0.411, 0.6835]
- **stain** (*str='Foreground', optional*) –
  - 'Foreground' indicates the objects of interest appear bright in the image.
  - 'Background' indicates the objects of interest appear dark.

- **segmentation** (*str='Geodesic', optional*) –
  - 'Threshold' – indicates to threshold the image at $0.4 \times intensity_{max}$.
  - 'Geodesic' – (default) peform a geodesic active contour segmentation with default settings.
  - 'Edge-Free' – perform an edge-free active contour segmentation with default dettins.
  - 'User' – The user will invoke calls to segmentation function with custom settings.
- **smoothing_method** (*str='Curvature Diffusion', optional*) – Smoothing method to use on regions of interest.
  - 'None' – No smoothing will be performed.
  - 'Gaussian' – Perform Gaussian smoothing.
  - 'Median' – Apply a median filter.
  - 'Curvature Diffusion' – Perform curvature-based anisotropic diffusion smoothing.
  - 'Gradient Diffusion' – Peform classical anisotropic diffusion smoothing.
  - 'Bilateral' – Apply a bilateral filter.
  - 'Patch-based' – Perform patch-based denoising.
- **smoothing_parameters** (*dict, optional*) – Depends on **smoothing_method**. Field keys are documented in methods **smoothRegion()**.
  - **'Gaussian' – fields are:**
    - 'sigma': float=0.5
  - **'Median' – fields are:**
    - 'radius': (int=1, int=1, int=1)
  - **'Curvature Diffusion' – fields are:**
    - 'iterations': int=10
    - 'conductance': float=9.0
  - **'Gradient Diffusion' – fields are:**
    - 'iterations': int=10
    - 'conductance': float=9.0
    - 'time step': float=0.01
  - **'Bilateral' – fields are:**
    - 'domainSigma': float=1.5
    - 'rangeSigma': float=40.0
    - 'samples': int=100
  - **'Patch-based' – fields are:**
    - 'radius': int=3
    - 'iterations': int=10
    - 'patches': int=20
    - **'noise model: str='poisson'**

· options: ('none', 'gaussian', 'poisson', 'rician')

- **two_dim** (*bool=False, optional*) – If *True*, will consider each 2-D slice in stack independently in smoothing and segmentation. This is not recommended except in special cases.
- **bright** (*bool=False, optional*) – If *True*, will perform bright spot removal replacing voxels with intensities $\geq 98^{th}$ percentile with median filtered (radius=6) value.
- **enhance_edge** (*bool=False, optional*) – If *True*, will enhance edges after smoothing using Laplacian sharpening.
- **depth_adjust** (*bool=False, optional*) – If *True*, will perform a linear correction for intensity degradation with depth.
- **opening** (*bool=True, optional*) – If *True*, will perform a morphological binary opening following thresholding to remove spurious connections and islands. If object of interest is thin, this may cause problems in which case, setting this to *False* may help.
- **fillholes** (*bool=False, optional*) – If *True*, all holes completely internal to the segmented object will be considered as part of the object.
- **display** (*bool=True, optional*) – If *True*, will spawn a 3-D interactive window rendering of segmented object surfaces.
- **handle_overlap** (*bool=True, optional*) – If *True*, overlapping segmented objects will be reclassified using a Support Vector Machine.
- **debug** (*bool=False, optional*) – If *True*, will write additional images to disk in NifTi format for debugging purposes.

cells
SimpleITK Image – An image containing the segmented objects as integer labels. Has the same properties as the input image stack.

thresholds
[, int, . . .] – The threshold level for each cell

volumes
[, float, . . .] – List of the physical volumes of the segmented objects.

centroids
[,[float, float, float], . . .]1 – List of centroids of segmented objects in physical space.

surfaces
[,vtkPolyData, . . .] – List containing VTK STL objects.

dimensions
[,[float, float, float],. . .] – List containing the ellipsoid axis lengths of segmented objects. These are determined from the segmented binary images. It is recommended to use the values calculated from a 3-D mesh in the **CellMech** class.

_classifyShared(*i*, *cells*, *previous*)
If segmented objects overlap and **handle_overlap** is *True*, this will attempt to reclassify the shared voxels using the thresholded seed to train a support vector machine. Of course, this relies on the seed to not overlap. The user strategy to get good results from this would be to use an active contour method, with an aggressive thresholding method to produce the seed.

Returns

- *A modified version of cells attribute with the overlapping objects*
- *reclassified.*

_flattenBorder(*img*)
To help ensure the segmentation does not touch the cropped region border, the voxel intensities of the 6 border slices are replaced by the intensity of their 1st percentile.

_getLabelShape(*img*)

_getMinMax(*img*)

_parseStack()

_replaceSeed(*seed*)

adjustForDepth()
Iterates over slices in 3-D image stack and appends each slice's maximum pixel intensity to a list. Then performs a weighted linear curve fit with slices below the 2^{nd} percentile and above the 98^{th} percentile assigned zero weights with all others equally weighted. Ratios are then calculated from this fit for all z-depths as:

$$\frac{a_0}{a_1 z + a_0}$$

and each slice of the image is multiplied by its corresponding weight and reassembled into a 3-D image.

Returns

- *Replaces image read from disk with and image that is corrected for*
- *intensity change with depth.*

edgeFreeSegmentation(*upsampling=2*, *seed_method='Percentage'*, *adaptive=True*, *ratio=0.4*, *lambda1=1.0*, *lambda2=1.1*, *curvature=0.0*, *iterations=20*)
Performs a segmentation using the SimpleITK implementation of the Active Contours Without Edges method described in (Chan and Vese. 2001.) Please also consult SimpleITK's documentation of ScalarChanAndVeseDenseLevelSetImageFilter.

Parameters

- **upsampling** (*int=2, optional*) – Resample image splitting original voxels this many times. Resampling will always be performed to make voxels isotropic, because anisotropic voxels can degrade the performance of this algorithm.
- **seed_method** (*str='Percentage'*) – Thresholding method used to determine seed image; same as **thresholdSegmentation()** method parameter. Please consult its documentation.
- **adaptive** (*bool=True*) – If true will adaptively adjust threshold the threshold value until resulting segmentation no longer touches the region of interest bounds.
- **ratio** (*float=0.7*) – The ratio to use with 'Percentage' threshold method. This plays no role with other seed methods.
- **lambda1** (*float=1.0*) – Weight for internal levelset term contribution to the total energy.
- **lambda2** (*float=1.1*) – Weight for external levelset term contribution to the total energy.
- **curvature** (*float=0.0*) – Weight for curvature. Higher results in smoother levelsets, but less ability to capture fine features.
- **iterations** (*int=20*) – The number of iterations the active contour method will conduct.

geodesic2D(*seed*, *simg*, *cannyLower*, *cannyUpper*, *canny_variance*, *upsampling*, *active_iterations*, *rms*, *propagation*, *curvature*, *advection*)

A 2-D implementation of **geodesicSegmentation()** that operates on each slice in the 3-D stack independently.

geodesicSegmentation(*upsampling=2*, *seed_method='Percentage'*, *adaptive=True*, *ratio=0.7*, *canny_variance=(0.05*, *0.05*, *0.05)*, *cannyUpper=0.0*, *cannyLower=0.0*, *propagation=0.15*, *curvature=0.2*, *advection=1.0*, *rms=0.01*, *active_iterations=200*)

Performs a segmentation using the SimpleITK implementation of the Geodesic Active Contour Levelset Segmentation method described in (Caselles et al. 1997.) Please also consult SimpleITK's documentation of GeodesicActiveContourLevelSetImageFilter. This method will establish the initial levelset function by calling the **thresholdSegmentation()** method, and calculating a distance map from the resulting binary image.

Parameters

- **propagation** (*float=0.15*) – Weight for propagation term in active contour functional. Higher values result in faster expansion.
- **curvature** (*float=0.2*) – Weight for curvature term in active contour functional. Higher values result in smoother segmentation.
- **advection** (*float=1.0*) – Weight for advective term in active contour functional. Higher values causes the levelset evolution to be drawn and stick to edges.
- **rms** (*float=0.01*) – The change in root-mean-square difference at which iterations will terminate. This value is divided by the **upsampling** value to account the effect of voxel size.
- **active_iterations** (*int=200*) – The maximum number of iterations the active contour will conduct.
- **upsampling** (*int=2, optional*) – Resample image splitting original voxels this many times. Resampling will always be performed to make voxels isotropic, because anisotropic voxels can degrade the performance of this algorithm.
- **seed_method** (*str='Percentage'*) – Thresholding method used to determine seed image; same as **thresholdSegmentation() method** parameter. Please consult its documentation.
- **adaptive** (*bool=True*) – If true will adaptively adjust threshold the threshold value until resulting segmentation no longer touches the region of interest bounds.
- **ratio** (*float=0.7*) – The ratio to use with 'Percentage' threshold method. This plays no role with other seed methods.
- **canny_variance** (*[float=0.05, float=0.05, float=0.05]*) – The Gaussian variance for canny edge detection used to generate the edge map for this method. Gaussian smoothing is performed during edge detection, but if another smoothing method was already performed this can be set low. High values results in smoother edges, but risk losing edges when other objects are close.
- **cannyUpper** (*float=0.0*) – Ensures voxels in the image gradient with a value higher than this will always be considered edges, and never discarded.
- **cannyLower** (*float=0.0*) – Ensures voxels in the image gradient with a value lower than this will be discarded.

getDimensions()

scale2D(*img*, *thresh*)

smooth2D(*img*)

smoothRegion(*img*)

threshold2D(*img*, *thres*, *ratio*)

thresholdSegmentation(*method='Percentage'*, *adaptive=True*, *ratio=0.4*)
Segments object of interest from image using user-specified method.

Parameters

- **method** (*str='Percentage'*) – The thresholding method to use. Options are:
  - 'Percentage' – Threshold at percentage of the maximum voxel intensity.
  - 'Otsu' – Threshold using Otsu's method
  - 'Huang' –
  - 'IsoData'
  - 'Li'
  - 'MaxEntropy' – Sets the threshold value such that the sum of information entropy (Shannon) in the foreground and background is maximized.
  - 'KittlerIllingworth'
  - 'Moments'
  - 'Yen'
  - 'RenyiEntropy' – The same as 'MaxEntropy', but uses the Renyi entropy function.
  - 'Shanbhag' – Extends upon the entropy methods with fuzzy set theory.
- **ratio** (*float=0.4*) – Ratio of maximum voxel intensity in the region of interest to threshold at. Only used if 'Percentage' method is given.
- **adaptive** (*bool=True*) – If *True* will adaptively adjust the determined threshold value until the segmented object does not the region boundaries.

writeLabels()

writeSurfaces()

class pyCellAnalyst.**CellMech**(*ref_dir=None*, *def_dir=None*, *rigidInitial=True*, *deformable=False*, *saveFEA=False*, *deformableSettings={'Maximum RMS': 0.01*, *'Displacement Smoothing': 3.0*, *'Iterations': 200*, *'Precision': 0.01}*, *display=False*)
Quantifies deformation between objects in reference and deformed states.

Object geometries are obtained by importing polygonal sufaces saved in the STL format. The user indicates a single reference directory containing the files and all corresponding deformed directories. The STL files must be named the same in each directory, so they are matched appropriately. In most cases, these will have been generated by pyCellAnalyst.Volume(), and by default will be named [image_dirname]_results.

Parameters

- **ref_dir** (*str*) – The directory containing the STL files corresponding to the reference (undeformed) state.
- **def_dir** (*str*) – The directory containing the STL files corresponding to the deformed state.
- **rigidInitial** (*bool, optional*) – If *True* do an initial rigid body transformation to align objects.

- **deformable** (*bool, optional*) – If *True* deformable image registration will be performed. This will call deformableRegistration(), which will calculate a displacement map between images reconstructed from reference and deformed surfaces.
- **saveFEA** (*bool, optional*) – If *True* will save nodes, elements, surface nodes, and displacement boundary conditions in a dictionary to *cell{:02d}.pkl*. This information can then be used to run finite element analysis in whatever software the user desires.
- **deformableSettings** (*dict, optional*) –

  Settings for deformable image registration with fields:

  - **Iterations:** (int, 200) The maximum number of iterations for algorithm.
  - **Maximum RMS:** (float, 0.01) Will terminate iterations if root-mean-square error is less than.
  - **Displacement Smoothing:** (float, 3.0) Variance for Gaussian smoothing of displacement field result.
  - **Precision:** (float, 0.01) The fraction of the object bounding box in each dimension spanned by 1 voxel.
- **display** (*bool, optional*) – If *True* will display 3-D interactive rendering of displacement fields.

rsurfs
[,vtkPolyData,...] – Polygonal surfaces of objects in reference state.

dsurfs
[,vtkPolyData,...] – Polygonal surfaces of objects in deformed state.

rmeshes
[,vtkUnstructuredGrid,...] – Tetrahedral meshes of undeformed geometries.

rcentroids
[,ndarray(3, float),...] – Volumetric centroids of objects in reference state.

dcentroids
[,ndarray(3, float),...] – Volumetric centroids of objects in deformed state.

cell_strains
[,ndarray((3,3), float) – Green-Lagrange strain tensors for object assuming uniform deformation.

vstrains
[, float, ...] – Volumetric strains of the analyzed objects.

ecm_strain
ndarray((3,3), float) – Green-Lagrange strain tensor for extracellular (extra-object) matrix assuming uniform deformation.

rvols
[,float,...] – Volumes of objects in reference state.

dvols
[,float,...] – Volumes of objects in deformed state.

raxes
[,ndarray(3, float),...] – Lengths of axes for ellipsoid with equivalent principal moments of inertia to object in reference state.

daxes
[,ndarray(3, float),...] – Lengths of axes for ellisoid with equivalent principal moments of inertia to object in deformed state.

cell_fields
[,vtkUnstructuredGrid,...] – Displacement vectors determined by deformable image registration interpolated to the vertices of object mesh in reference state.

rigid_transforms
[,vtkTransform,...] – Initial rigid body transforms applied to object in reference state to align with deformed state. This is only populated if **rigidInitial** is *True*.

_deform()
Calculates the affine transform that best maps the reference polygonal surface to its corresponding deformed surface. This transform is calculated through an interactive closest point optimization, that seeks to minimize the sum of distances between the reference surface vertices and the current affine transformed surface.

Assuming a uniform deformation, the non-translational elements of this affine transform compose the deformation gradient $\mathbf{F}$. The Green-Lagrange strain tensor is then defined as

$\mathbf{E} = \frac{1}{2}(\mathbf{F}^T.\mathbf{F} - \mathbf{1})$,

where $\mathbf{1}$ is the identity.

Returns

Return type *cell_strains*

_getECMstrain()
Generates tetrahedrons from object centroids in the reference and deformed states. The highest quality tetrahedron (edge ratio closest to 1) is used to construct a linear system of equations,

$\|\mathbf{w}\|^2 - \|\mathbf{W}\|^2 = \mathbf{W}.\mathbf{E}.\mathbf{W}$,

where, $\mathbf{W}$ are the reference tetrahedron edges (as vectors) and $\mathbf{w}$ are the deformed tetrahedron edges, to solve for Green-Lagrange strain, $\mathbf{E}$.

Returns

Return type *ecm_strain*

_make3Dmesh(*filename*, *frame*, *vConst*)
Generates a 3-D tetrahedral mesh using tetmesh module build on CGAL. These meshes are then used to determine the object's volume, centroid, and the axes of the ellipsoid that has equivalent principal moments of inertia.

Parameters

- **filename** (*str*) – The path and filename of the STL surface currently being analyzed. This is necessary since MeshPy has to read the STL from disk in its native format.
- **frame** (*str*) – Indicates the curent state, 'MATERIAL' or 'SPATIAL'.

Returns

- ***if frame=='MATERIAL'** –
  - rmeshes
  - rcentroids
  - rvols
  - raxes
- ***else** –
  - dcentroids

- dvols
- daxes

_poly2img(*ind*)

Helper function called by **deformableRegistration()** that generates images from polygonal surfaces in reference/deformed pairs. The voxel dimension of these images is determined by the value for **Precision** in **deformableSettings**.

Parameters **ind** (*int*) – The list index for the current object pair being analyzed.

Returns

Return type (Reference Image, Deformed Image, Tranformed Reference Surface)

_readstls()

Reads in all STL files contained in directories indicated by **ref_dir** and **def_dir**. Also calls **_make3Dmesh()** to create 3-D tetrahedral meshes.

Returns

Return type *rsurfs*, *dsurfs*

animate(*pd*, *ind*)

Helper function called by **deformableRegistration** if **animate** is *True*. Spawns a window with an interactive 3-D rendering of the current analyzed object in its reference state. The displacements calculated from the deformable image registration can be applied to this object to animate the deformation by pressing the RIGHT-ARROW. Pressing the UP-ARROW will animate and also save the frames to disk.

Parameters

- **pd** (*vtkPolyData*) – The current analyzed object's reference geometry.
- **ind** (*int*) – The index of the current polydata in **rsurfs**. Necessary for naming directory created if animation frames are saved.

deformableRegistration()

Performs deformable image registration on images reconstructed from polygonal surfaces at a user-specified precision. If **animate** parameter is *True*, this also spawns a VTK interative rendering that can animate the deformation.

Returns

Return type *cell_fields*

CHAPTER 6

License

The MIT License (MIT)

Copyright (c) 2014 Scott Sibole

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

pyCellAnalyst is a Python module aimed at the segmentation of cells imaged with 3-D microscopy under different mechanical conditions, and then quantifying the deformations resulting from those conditions. Many image processing, segmentation, and registration methods from the extensive and powerful *C++* Visualization and Simple Insight Toolkits are provided. With these tools, objects in challenging configurations can be segmented in 3-D.

Following object segmentation, the resulting polygonal surfaces in reference and deformed states can be used to calculate cellular deformations. Three methods are provided for this: equivalent principal moments of inertia ellipsoid, optimal affine transformation, and deformable image registration (more details in Introduction). The results of deformable image registration applied to the example object segmentation above is visualized below.

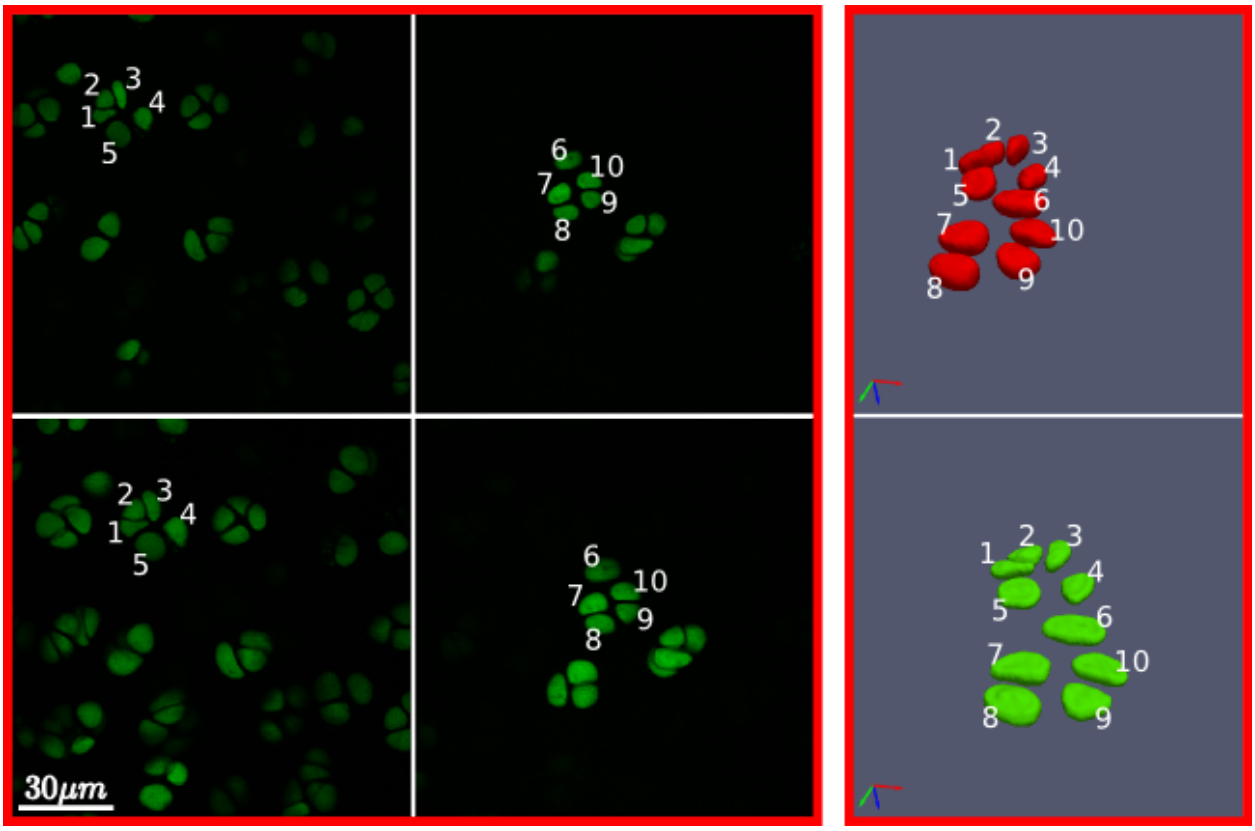


Fig. 6.1: Cells reconstructed from dual photon laser scanning microscopy data in an unloaded (top) and loaded state (bottom). The challenge of separating highly clustered objects is handled robustly by the tools provided.

Fig. 6.2: A visualization of the optimal diffeomorphism mapping the reconstructed referece (red) and deformed (green) cells in the previous figure is shown here. This is one of three methods to characterize deformation provided within pyCellAnalyst.

Python Module Index

p

Index

Symbols

A

C

D

E

G

P

R

S

T

V

W