Script Documentation (Dan Berrebbi)

findRadius (name, index) : This function gives the radius of the contour **name** at the location given by **index**.

[**name** is a contour name in your SV current project, **index** is an int in your contour index list]

- **Return type** : a float (the radius).

canGraft (vessel, graft, maxShrink) : This function determines if a graft is possible at a certain point considering geometric parameters : can the graft vessel enter the main vessel at this location ? (if not we can shrink the end of the graft vessel but respecting the constraint of **maxShrink**).

[**vessel** is the diameter of the main vessel at this location , **graft** is the diameter of the end (one end) of the graft vessel, **maxShrink** is a float between 0 and 1 which specify how much the graft vessel can be shrinked at its end, eg : maxShrink=0.3 means we can Shrink the graftend at 70%]

- **Return type :** this function returns a double (Boolean, float). The Boolean answers the question « Is the graft possible ? », and if True the float is the shrink we have to do at the end of the graft vessel. Example : (True, 0.7) means we have to shrink the end by 30% but the graft is then possible.

shrinkGraft(graft_radii, shrink): This function takes a list of radii of the graft vessel and « **shrink** » , wich comes from the previous function (canGraft) and shrinks the radii of the graft vessel in a smooth way (linear until a certain point) in order to enable the garft.

[**graft_radii** is a list of the radii of the graft vessel, **shrink** is a float between 0 and 1]

- **Return type :** no return, the function just modify the list graft_radii.

coarctPipeline(pathList, radiusList, pathName, contourName, modelName, save) : This function is Tobias' one to build model. I just added one parameter (**save**) to save the model in the current location in order to perform union after.

coarctPipeline2(pathList, radiusList, pathName, contourName, modelName, save) : This function is the same as the previous one but some intern names are changed to avoid « `error registering obj in repository` » in SV. Indeed the function coarctPipeline can be used only once per project, that's why I did this trick.

- **read_centerline(path_name) :** This function returns the path points of the path named « **path_name** ».
    - **Return type :** a list of path points : [[x,y,z] , [u,v,w] , …]

I also used the SPLIPY librairy to design 3D curves with specific constraints : interpolation with some given points and given tangents at some points.

This librairy is not used a lot on Python and have to be installed (as SCIPY) on SV. However it is really conveniant to manipulate 3D curves, to interpolate with a lot of constraints, and to get data afterwork on your curve.

I found most of the useful documentation here :

https://pythonhosted.org/Splipy/factories.html#splipy.curve_factory.Boundary

https://pythonhosted.org/Splipy/basic_classes.html#splipy.Curve.curvature

I chose to use the « tangent » boundary (number 5) and the function cubic_curve in order to build the graft vessel by an interpolation of all the points we want to be on the path (including of course the junction points on the main vessel) and the tangent at the endpoints are specified.

cubic_curve(x=coordinates, boundary=5, t=None, tangents=tgt) : This function does an interpolation.

[**Coordinates** is a numpy array of the points that we want on the curve, **boundary** specify the type of interpolation, I didn't get the role of the 3rd parameter, and **tangents** is a numpy array of the tangents at the endpoints (tangents represented like points, 3 coordinates)]

- **Return type :** a curve (*class* splipy.**Curve**). I then work on the Curve to get the path points of the graft vessel with other Splipy functions that I describe bellow.

SplineObject.evaluate (cb, tab) : This function evaluates the coordinates of the curve **cb** at the points specified by the list tab. One problem I had is to know what mean the index in **tab**. **Tab** is a list of float that should begin at 0 and end at the float given by SplineObject.end(cb)[**0**].

[**cb** is the curve (*class* splipy.**Curve**) returned by the function cubic_curve described above, and tab is a list of floats]

- **Return type :** a numpy array of 3D points.

Before transforming the curve into an array I used function to get the max and min values in each direction, the max curvature and max torsion of the curve. There is a bench of other functions available

to do geometric analysis of the curve, I don't know which ones can be useful for a study of constraints on this graft vessel.

As I said Splipy is very conveniant (contrairy to most of the Python librairies for 3D work) and we can also specify the derivatives at some points… .

MAIN :

- ## Graft_on_coarct :

  Goal : given a main vessel and 2 points input points, we want to do a graft with some geometrics constraints.

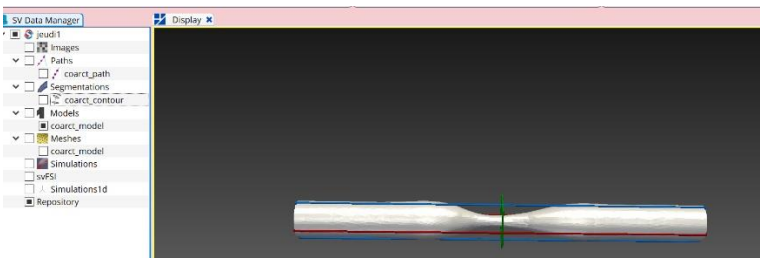
  Inputs :

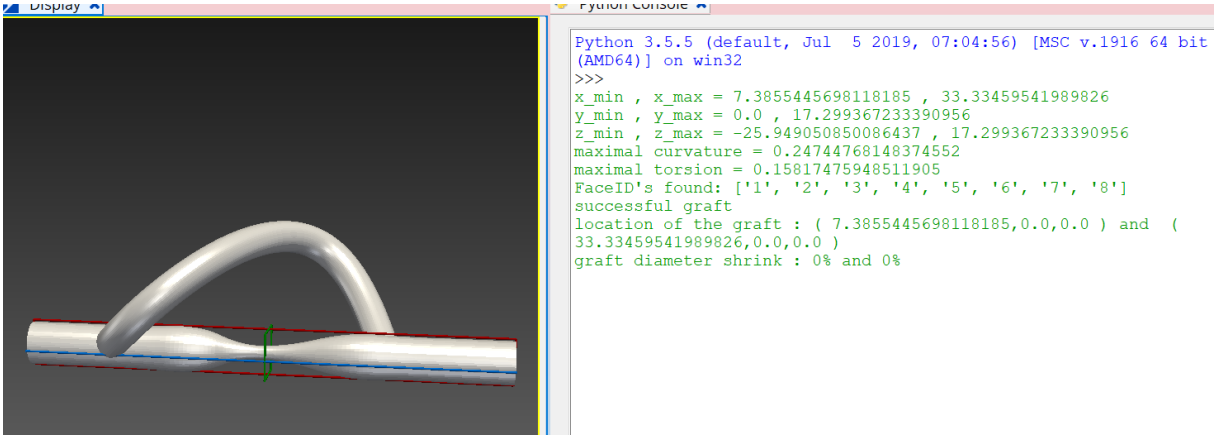
  First the user have to open the sv project with his main vessel model and to give the name of the contour, the adress of the path (.pth) and the index of the path points where he wants the graft to enter.
  Then the user have to enter the number of points he wants on the graft vessel and the list of radii of the graft vessel. The next input is the maximal Shrink the user allows (described in canGraft function), and the last input is an array of tangents the user wants at the endpoints.

  Work :

  Then we find the radius on the location wanted for the graft. If the graft is possible on the first input point, we shrink the first end of the vessel to enable the graft if necessary. Then we look at the other end (tha's why I use the reverse function for the list of radii) and perform the same operations. We can then generate the graft using Splipy functions and then print some parameters such as the shrink at each end, the curvature, the position of the endpoints… .

  Starting point :





This is the result of running this python file and then do a boolean union by hand.
(The automatic boolean union still doesn't work).