Riemannian_metrics

July 27, 2024

This notebook attemps to recreate some experiments from [1], where Riemannian metrics for ODFs are proposed. Unfortunately, the paper does not provide any reference implementation AND does not seem to present some "baseline" results, ie interpolating fODFs SH coefficients as-is.

References: [1]: Cheng, J., Ghosh, A., Jiang, T., & Deriche, R. (2009). A Riemannian framework for orientation distribution function computing. In International Conference on Medical Image Computing and Computer-Assisted Intervention (pp. 911-918). Berlin, Heidelberg: Springer Berlin Heidelberg.

First, some function definitions:

[1]:
```
import nibabel as nib
import numpy as np

from IPython.display import Image
from matplotlib import pyplot as plt

from dipy.core.interpolation import trilinear_interpolate4d
from dipy.data import get_sphere
from dipy.reconst.shm import sf_to_sh, sh_to_sf
from dipy.sims.voxel import single_tensor_odf
from dipy.viz import window, actor

from scilpy.reconst.sh import peaks_from_sh, maps_from_sh
from scilpy.reconst.fodf import log_u, exp_u


def single_fiber_frf():
    return [0.00139919, 0.0003007,  0.0003007]


def isotropic_frf():
    return [sum(single_fiber_frf()) / 3] * 3


def get_single_fiber_projection_odf():
    evals = single_fiber_frf()
    evecs = np.array([[0, 0, 1], [0, 1, 0], [1, 0, 0]]).T
```

```
    response_odf = single_tensor_odf(get_sphere().vertices, evals, evecs)
    sh = sf_to_sh(response_odf, get_sphere(), 6)

    return sh


def get_single_fiber_association_odf():
    evals = single_fiber_frf()
    evecs = np.array([[0, 1, 0], [1, 0, 0], [0, 0, 1]]).T

    response_odf = single_tensor_odf(get_sphere().vertices, evals, evecs)
    sh = sf_to_sh(response_odf, get_sphere(), 6)

    return sh


def get_single_fiber_commissural_odf():
    evals = single_fiber_frf()
    evecs = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]).T

    response_odf = single_tensor_odf(get_sphere().vertices, evals, evecs)
    sh = sf_to_sh(response_odf, get_sphere(), 6)

    return sh


def get_crossing_commisural_association_odf():
    evals = single_fiber_frf()
    evecs1 = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]).T
    evecs2 = np.array([[0, 1, 0], [1, 0, 0], [0, 0, 1]]).T

    response_odf1 = single_tensor_odf(get_sphere().vertices, evals, evecs1)
    response_odf2 = single_tensor_odf(get_sphere().vertices, evals, evecs2)
    response_odf = (response_odf1 + response_odf2) / 2
    sh = sf_to_sh(response_odf, get_sphere(), 6)

    return sh


def get_crossing_association_projection_odf():
    evals = single_fiber_frf()
    evecs1 = np.array([[0, 1, 0], [1, 0, 0], [0, 0, 1]]).T
    evecs2 = np.array([[0, 0, 1], [0, 1, 0], [1, 0, 0]]).T

    response_odf1 = single_tensor_odf(get_sphere().vertices, evals, evecs1)
    response_odf2 = single_tensor_odf(get_sphere().vertices, evals, evecs2)
    response_odf = (response_odf1 + response_odf2) / 2
```

```
    sh = sf_to_sh(response_odf, get_sphere(), 6)

    return sh

def get_isotropic_odf():
    evals = isotropic_frf()
    evecs = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]).T

    response_odf = single_tensor_odf(get_sphere().vertices, evals, evecs)
    sh = sf_to_sh(response_odf, get_sphere(), 6)

    return sh


def _odf_img(sh):
    sh_img = nib.Nifti1Image(sh, np.eye(4))
    return sh_img

def interpolate_map(map1, n_steps):

    resampled_map1 = np.asarray([
        trilinear_interpolate4d(map1, np.asarray([0, i/n_steps, 0]))
        for i in range(n_steps+1)])[None, ..., None]

    return resampled_map1


def interpolate_between_odfs_euclidean(odf1, odf2, n_steps):
    stack = np.stack([odf1, odf2], axis=0)[None, :, None, :]
    resampled_odf1 = np.asarray([trilinear_interpolate4d(
        stack, np.asarray([0, i/n_steps, 0]))
        for i in range(n_steps+1)])[None, :, None, :]
    return resampled_odf1


def interpolate_between_odfs_riemannian(odf1, odf2, n_steps):
    """ Interpolate between two ODFs using the Riemannian framework.

    Parameters
    ----------
    odf1 : ndarray
    First ODF.
    odf2 : ndarray
    Second ODF.
    n_steps : int
    Number of steps for the interpolation.
```

```
    Returns
    -------
    ndarray
    Interpolated ODF.
    """

    stack = np.stack([odf1, odf2], axis=0)[None, :, None, :]
    log_stack, norm = log_u(stack)

    resampled_odf1 = np.asarray([trilinear_interpolate4d(
        log_stack, np.asarray([0, i/n_steps, 0]))
        for i in range(n_steps+1)])[None, :, None, :]
    resampled_norm = interpolate_map(norm, n_steps)
    return exp_u(resampled_odf1, resampled_norm)


def assemble_odf(odf1, odf2):
    """ Assemble two ODFs into a single ODF volume.

    Parameters
    ----------
    odf1 : ndarray
    First ODF.
    odf2 : ndarray
    Second ODF.

    Returns
    -------
    ndarray
    Assembled ODF.
    """

    return np.concatenate([odf1, odf2], axis=0)


def compute_ga(odf):
    """ Compute the Geometric Anisotropy (GA) of an ODF.

    Parameters
    ----------
    odf : ndarray
    ODF.

    Returns
    -------
    ndarray
    GA.
```

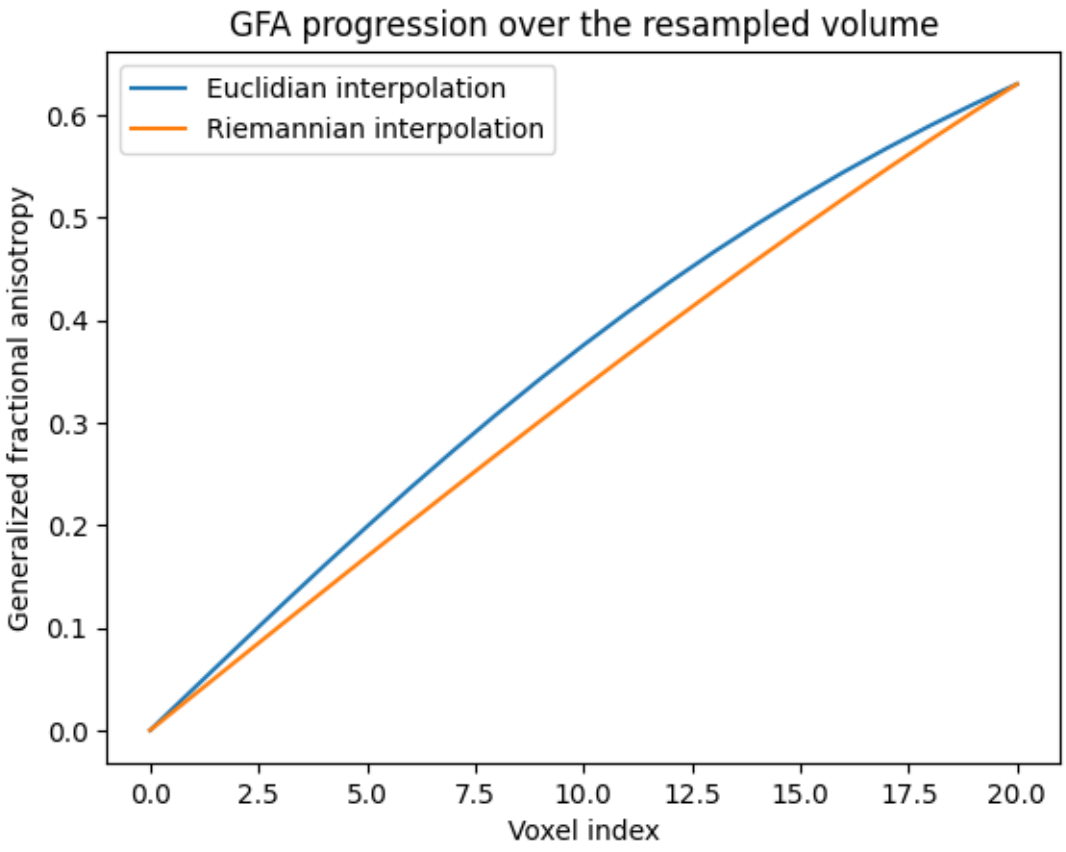
```
    """
    #
    # Computing peaks
    peak_dirs, peak_values, peak_indices = peaks_from_sh(odf,
                                                         get_sphere())
    # Computing maps
    nufo_map, afd_max, afd_sum, rgb_map, ga_map, gfa_map, _ = \
        maps_from_sh(odf, peak_dirs, peak_values, peak_indices,
                     get_sphere())
    return ga_map.squeeze()


def compute_gfa(odf):
    """ Compute the Generalized Fractional Anisotropy (GFA) of an ODF.

    Parameters
    ----------
    odf : ndarray
    ODF.

    Returns
    -------
    ndarray
    GFA.
    """
    #
    # Computing peaks
    peak_dirs, peak_values, peak_indices = peaks_from_sh(odf,
                                                         get_sphere())
    # Computing maps
    nufo_map, afd_max, afd_sum, rgb_map, ga_map, gfa_map, _ = \
        maps_from_sh(odf, peak_dirs, peak_values, peak_indices,
                     get_sphere())
    return gfa_map.squeeze()


def plot_ga(ga1, label1, ga2, label2):

    # Displaying GA progression over the resampled volume as a line
    # Title: GA progression over the resampled volume
    plt.title('GA progression over the resampled volume')
    # X-axis label: Voxel index
    plt.xlabel('Voxel index')
    # Y-axis label: Generalized anisotropy
    plt.ylabel('Geometric anisotropy')
    # Plotting GA progression over the resampled volume
    # First plot: without projection
```

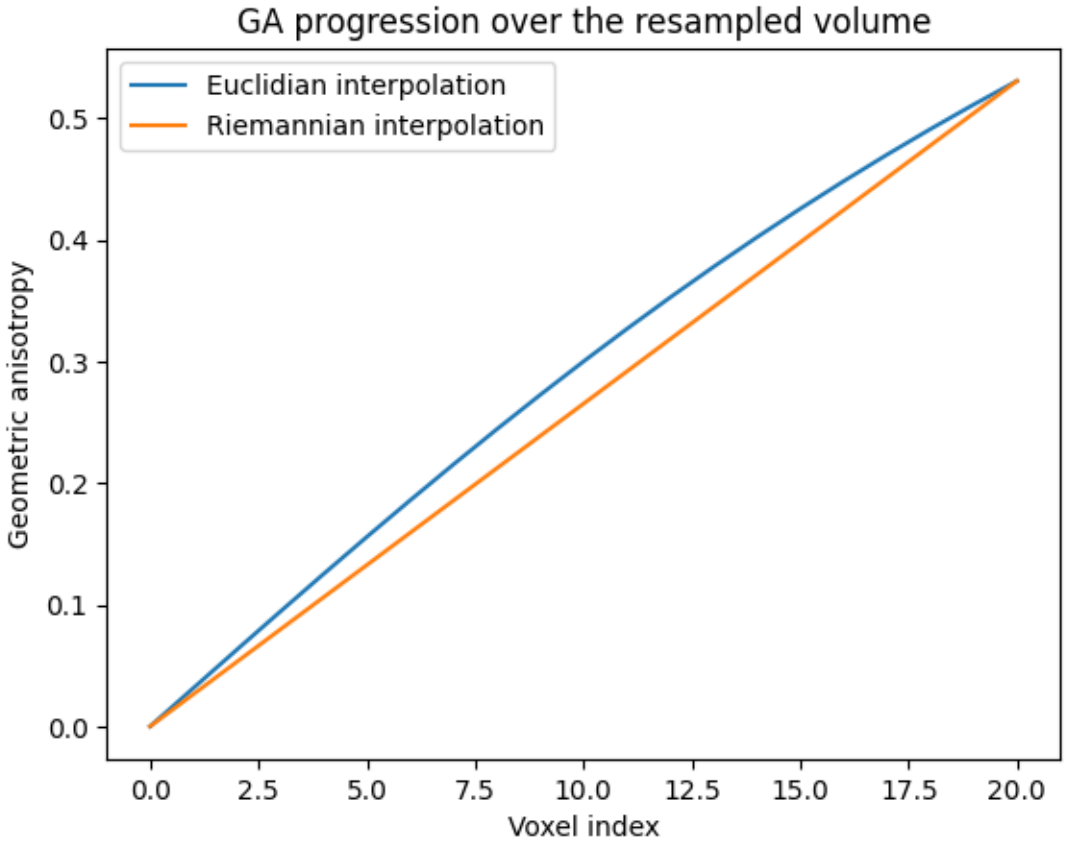
```
    # Second plot: with projection
    plt.plot(ga1 / (np.arccos(1/np.sqrt(4*np.pi))))
    plt.plot(ga2 / (np.arccos(1/np.sqrt(4*np.pi))))
    # Adding a legend
    plt.legend([label1, label2])
    plt.show()


def plot_gfa(gfa1, label1, gfa2, label2):

    # Displaying GA progression over the resampled volume as a line
    # Title: GA progression over the resampled volume
    plt.title('GFA progression over the resampled volume')
    # X-axis label: Voxel index
    plt.xlabel('Voxel index')
    # Y-axis label: Generalized anisotropy
    plt.ylabel('Generalized fractional anisotropy')
    # Plotting GA progression over the resampled volume
    # First plot: without projection
    # Second plot: with projection
    plt.plot(gfa1)
    plt.plot(gfa2)
    # Adding a legend
    plt.legend([label1, label2])
    plt.show()



def viz_odf_interp_1d(sh, metric=None):
    """ Visualize an ODF. If metric, use it to colorize the ODF.
    """
    scene = window.Scene()
    sf = sh_to_sf(sh, get_sphere(), 6)
    if metric is not None:
        slicer = actor.slicer(metric[..., None], opacity=0.5)
        scene.add(slicer)
    response_actor = actor.odf_slicer(
        sf, sphere=get_sphere(), affine=np.eye(4), norm=False, scale=1.5)
    scene.add(response_actor)
    # Rotate the camera 90 degrees to the left
    scene.set_camera(position=[-0.125, 0, 1], focal_point=[0, 0, 0],
                     view_up=[0, 0, -1])
    # Set the background color to white
    scene.SetBackground((1, 1, 1))
    window.record(scene, out_path='interp.png', size=(2048, 1024))
```

Let's start with something similar to "Experiment 1" of [1], where we interpolate from an isotropic

ODF to a very anisotropic “single-fiber” ODF:

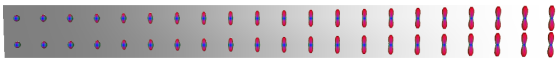
[2]:
```
# Generate the ODFs
sh1 = get_isotropic_odf()
sh2 = get_single_fiber_commissural_odf()
# Interpolate between the two ODFs using different metrics
interp_euc = interpolate_between_odfs_euclidean(sh1, sh2, 20)
interp_rie = interpolate_between_odfs_riemannian(sh1, sh2, 20)
# Assemble the two interpolations into a single "image"
interp = assemble_odf(interp_euc, interp_rie)
# Compute the GA and GFA of both interpolations
euc_ga = compute_ga(interp_euc)
rie_ga = compute_ga(interp_rie)

euc_gfa = compute_gfa(interp_euc)
rie_gfa = compute_gfa(interp_rie)
# Plot the progression of both metrics "over time"

plot_gfa(euc_gfa, 'Euclidian interpolation',
         rie_gfa, 'Riemannian interpolation')

plot_ga(euc_ga, 'Euclidian interpolation',
        rie_ga, 'Riemannian interpolation')
```





[3]:
```
# Display the interpolation "over time" for both
metrics = np.stack((euc_ga, rie_ga), axis=0)
viz_odf_interp_1d(interp, metric=metrics)
Image(filename='interp.png')
```

[3]:

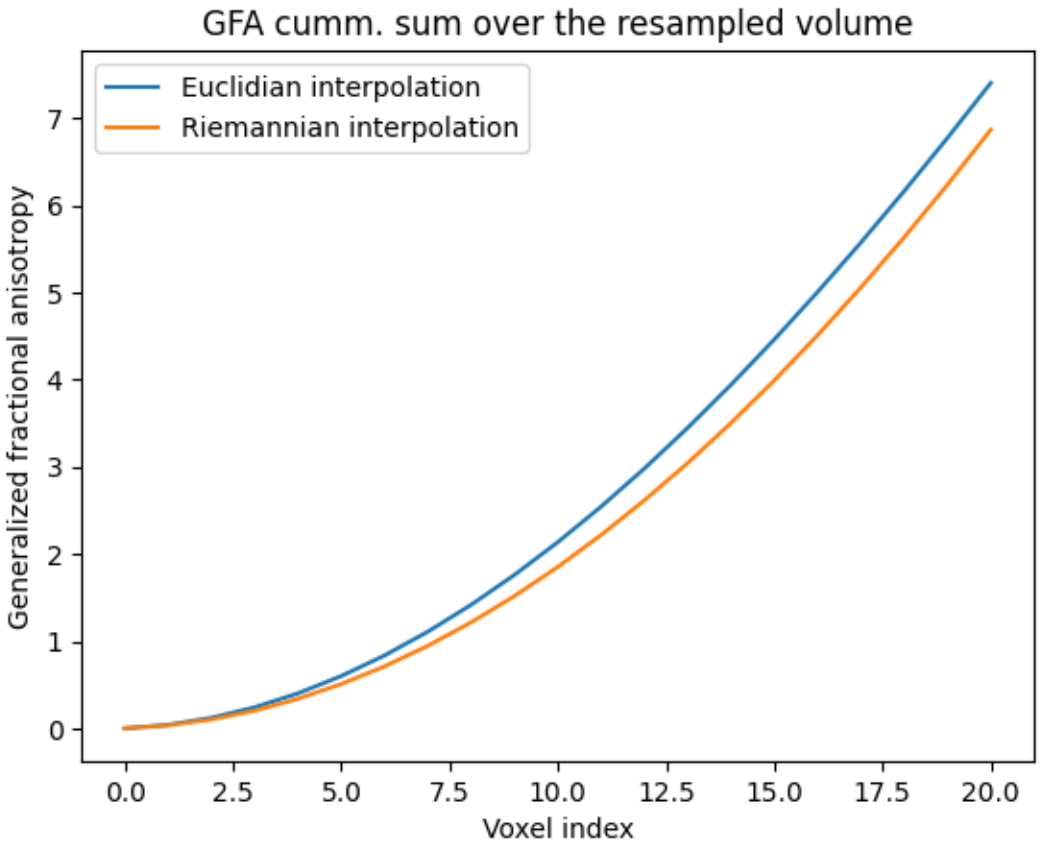


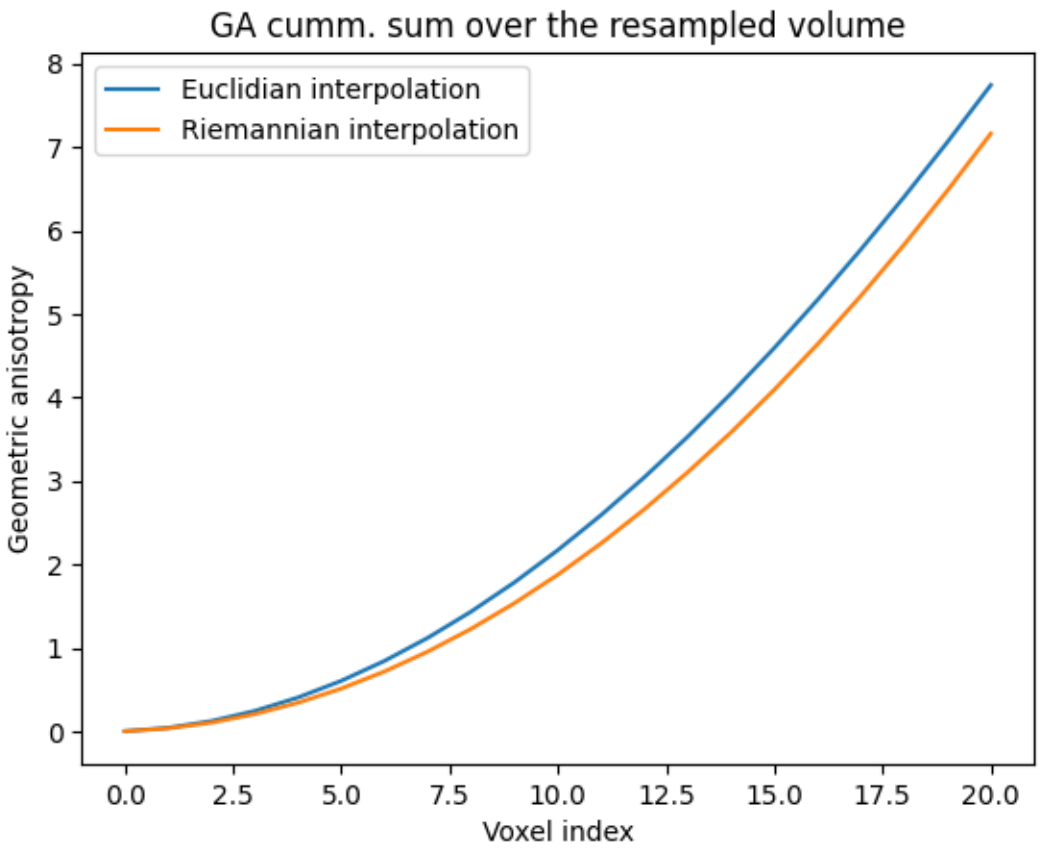
Top row is euclidean interpolation, bottom is "log-eucidean" interpolation. Doesn't seem to change much visually, but the GA and GFA plots atleast show that both are increasing linearly with proper metrics, whereas some "swelling" can be encountered from euclidean interpolation. This may be more evident by plotting the "cummulative sum" of GA and GFA as interpolation goes on:

[4]:
```
def plot_ga_cumsum(ga1, label1, ga2, label2):

    # Displaying GA progression over the resampled volume as a line
    # Title: GA progression over the resampled volume
    plt.title('GA cumm. sum over the resampled volume')
    # X-axis label: Voxel index
    plt.xlabel('Voxel index')
    # Y-axis label: Generalized anisotropy
    plt.ylabel('Geometric anisotropy')
    # Plotting GA progression over the resampled volume
    # First plot: without projection
    # Second plot: with projection
    plt.plot(np.cumsum(ga1))
    plt.plot(np.cumsum(ga2))
    # Adding a legend
    plt.legend([label1, label2])
    plt.show()


def plot_gfa_cumsum(gfa1, label1, gfa2, label2):

    # Displaying GA progression over the resampled volume as a line
    # Title: GA progression over the resampled volume
    plt.title('GFA cumm. sum over the resampled volume')
```

```
    # X-axis label: Voxel index
    plt.xlabel('Voxel index')
    # Y-axis label: Generalized anisotropy
    plt.ylabel('Generalized fractional anisotropy')
    # Plotting GA progression over the resampled volume
    # First plot: without projection
    # Second plot: with projection
    plt.plot(np.cumsum(gfa1))
    plt.plot(np.cumsum(gfa2))
    # Adding a legend
    plt.legend([label1, label2])
    plt.show()
```

[5]:
```
plot_gfa_cumsum(euc_gfa, 'Euclidian interpolation',
         rie_gfa, 'Riemannian interpolation')

plot_ga_cumsum(euc_ga, 'Euclidian interpolation',
        rie_ga, 'Riemannian interpolation')
```

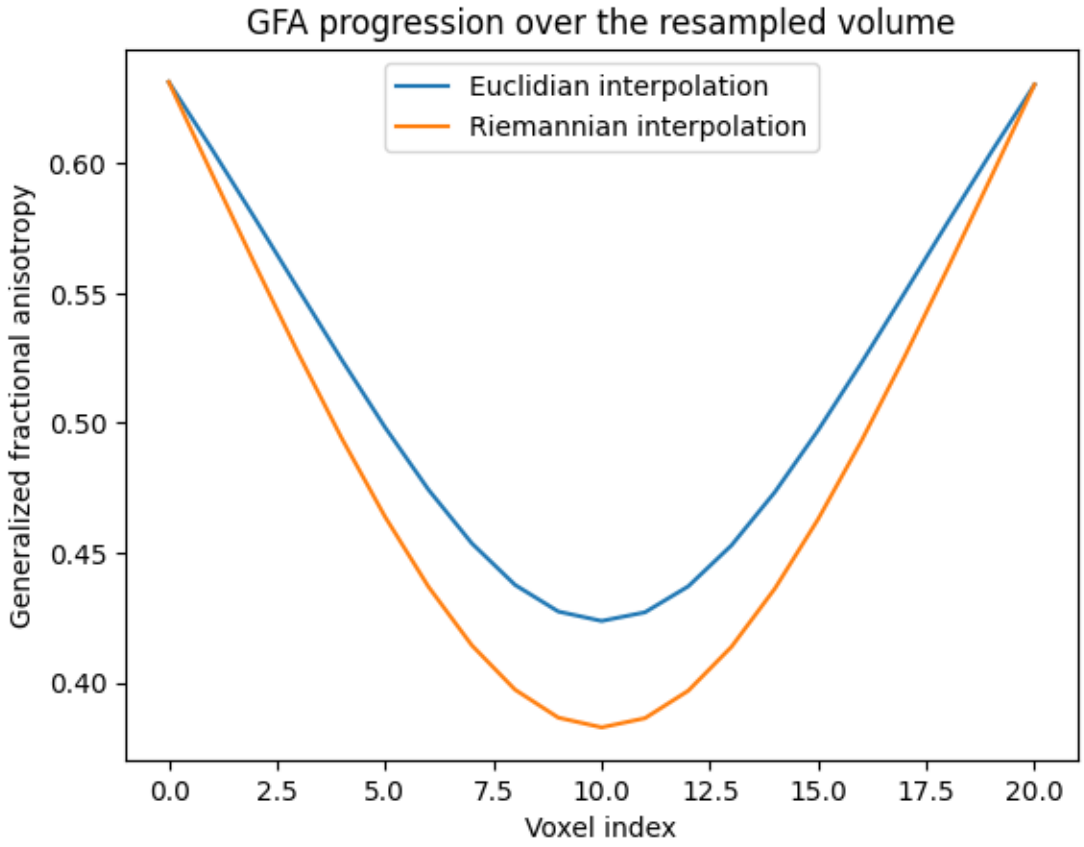


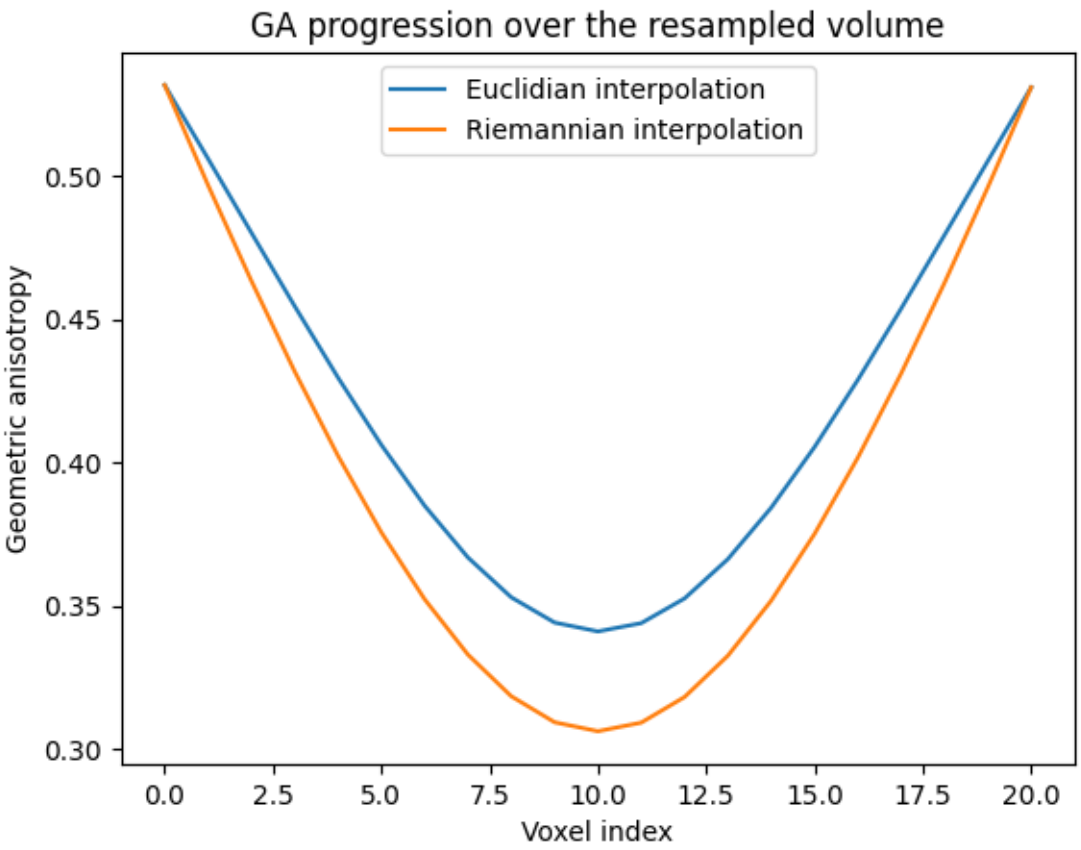


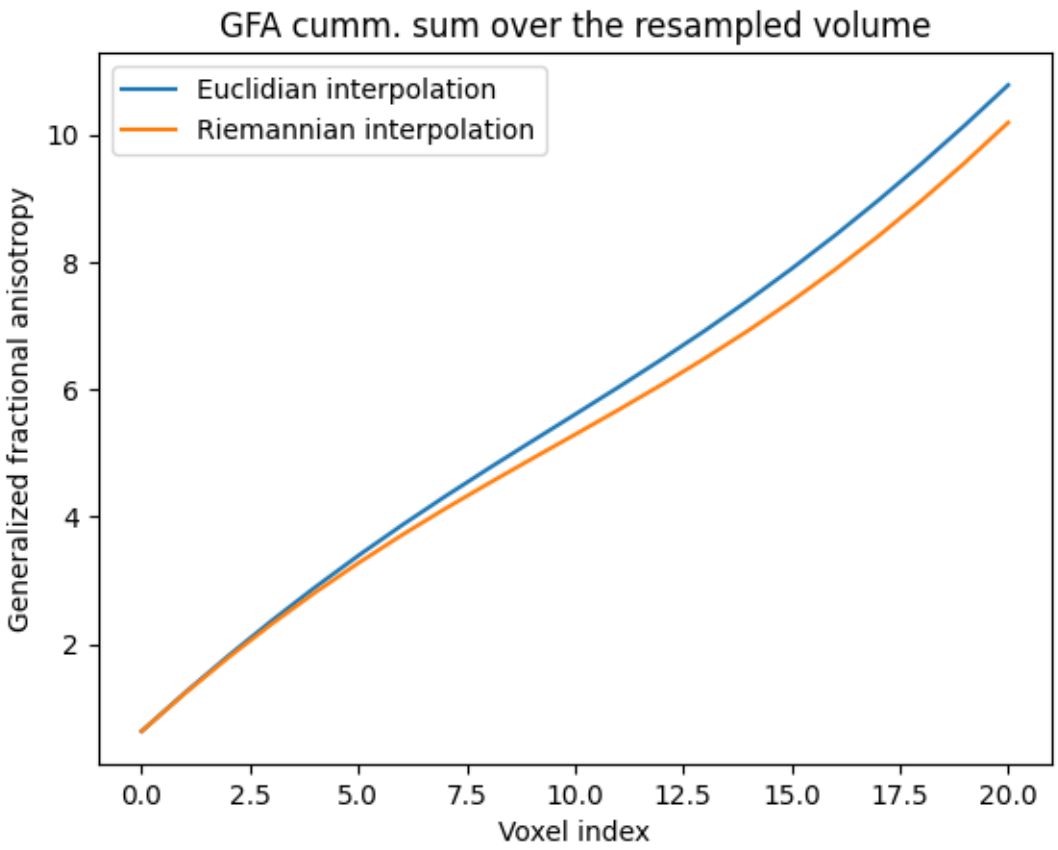
Next, let's try to do something similar to Experiment B, going from single-fiber to single-fiber, but different orientations.

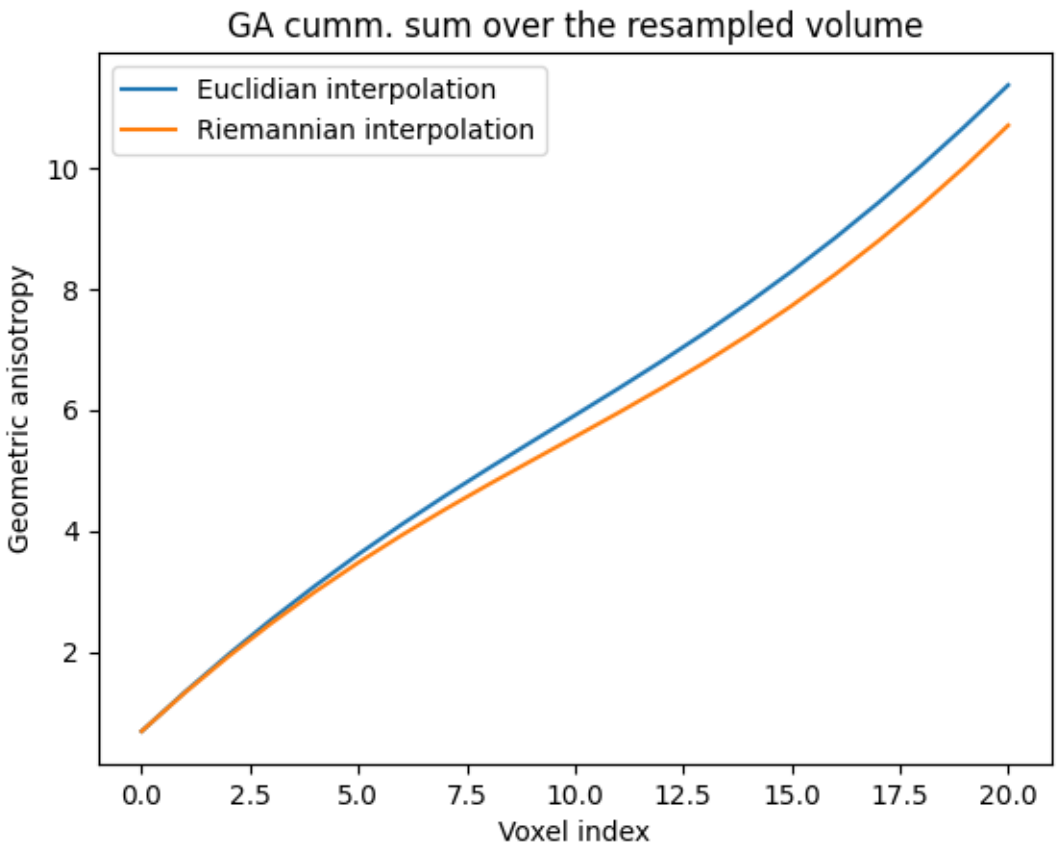
[6]:
```
# Generate the ODFs
sh1 = get_single_fiber_association_odf()
sh2 = get_single_fiber_commissural_odf()
# Interpolate between the two ODFs using different metrics
interp_euc = interpolate_between_odfs_euclidean(sh1, sh2, 20)
interp_rie = interpolate_between_odfs_riemannian(sh1, sh2, 20)
# Assemble the two interpolations into a single "image"
interp = assemble_odf(interp_euc, interp_rie)
# Compute the GA and GFA of both interpolations
euc_ga = compute_ga(interp_euc)
rie_ga = compute_ga(interp_rie)

euc_gfa = compute_gfa(interp_euc)
rie_gfa = compute_gfa(interp_rie)
# Plot the progression of both metrics "over time"

plot_gfa(euc_gfa, 'Euclidian interpolation',
```

```
         rie_gfa, 'Riemannian interpolation')

plot_ga(euc_ga, 'Euclidian interpolation',
        rie_ga, 'Riemannian interpolation')

plot_gfa_cumsum(euc_gfa, 'Euclidian interpolation',
         rie_gfa, 'Riemannian interpolation')

plot_ga_cumsum(euc_ga, 'Euclidian interpolation',
        rie_ga, 'Riemannian interpolation')
```



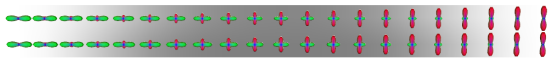

GA progression over the resampled volume
Euclidian interpolation
Riemannian interpolation
Geometric anisotropy
0.50
0.45
0.40
0.35
0.30
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Voxel index


GFA cumm. sum over the resampled volume
Euclidian interpolation
Riemannian interpolation
Generalized fractional anisotropy
10
8
6
4
2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Voxel index



[7]:
```
# Display the interpolation "over time" for both
metrics = np.stack((euc_ga, rie_ga), axis=0)
viz_odf_interp_1d(interp, metric=metrics)
Image(filename='interp.png')
```

[7]:

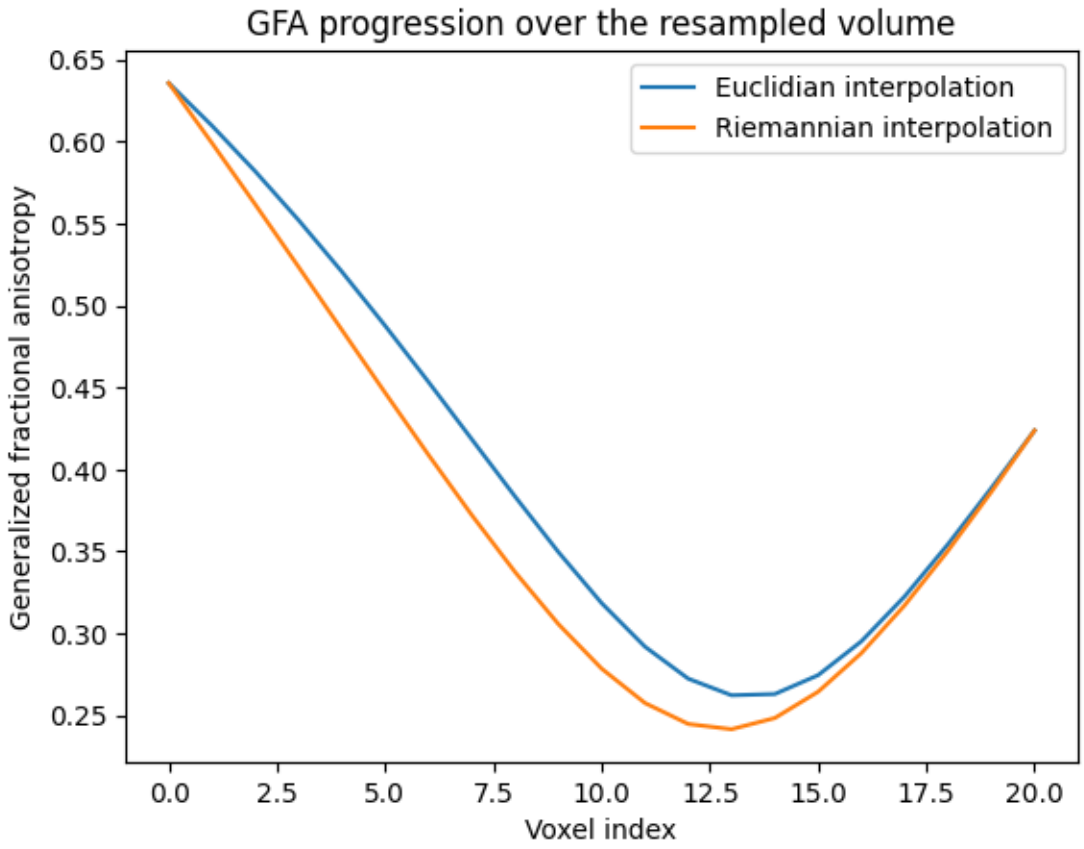


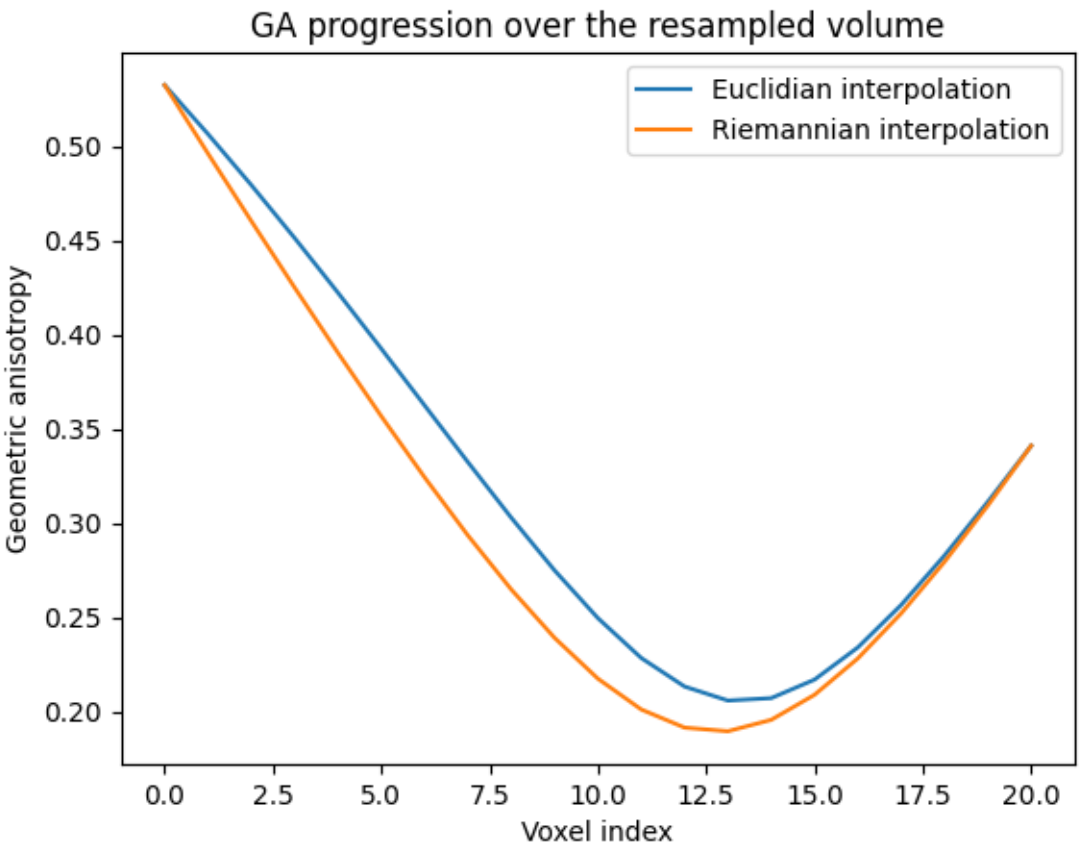
Again, top row is using Euclidean metrics and bottom row is Riemannian metrics. This is a bit harder to interpret, but atleast the results are consistent with the glyphs and plots of [1] in Experiment B. One thing we can appreciate is that the overall "amplitude" of the ODF seems to stay more consistent with riemannian metrics, whereas the top row gets "smaller" in the middle.

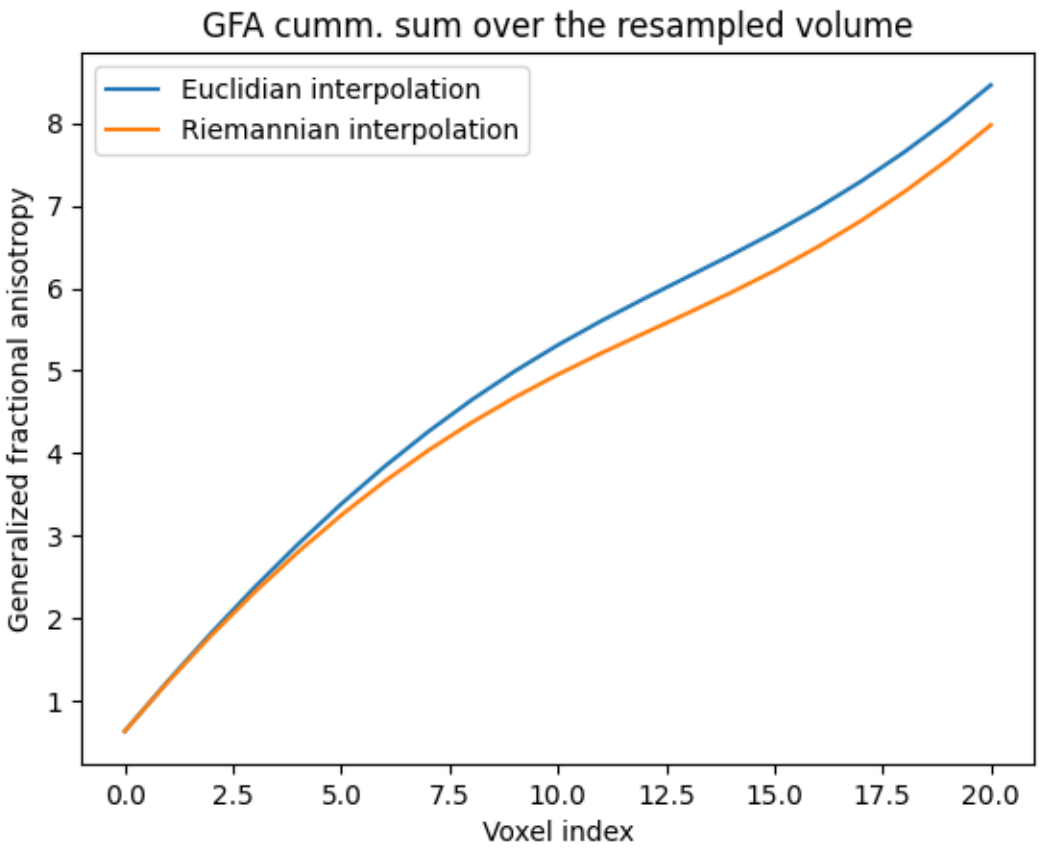
Still on experiment B, let's go from single fiber to two-fiber populations

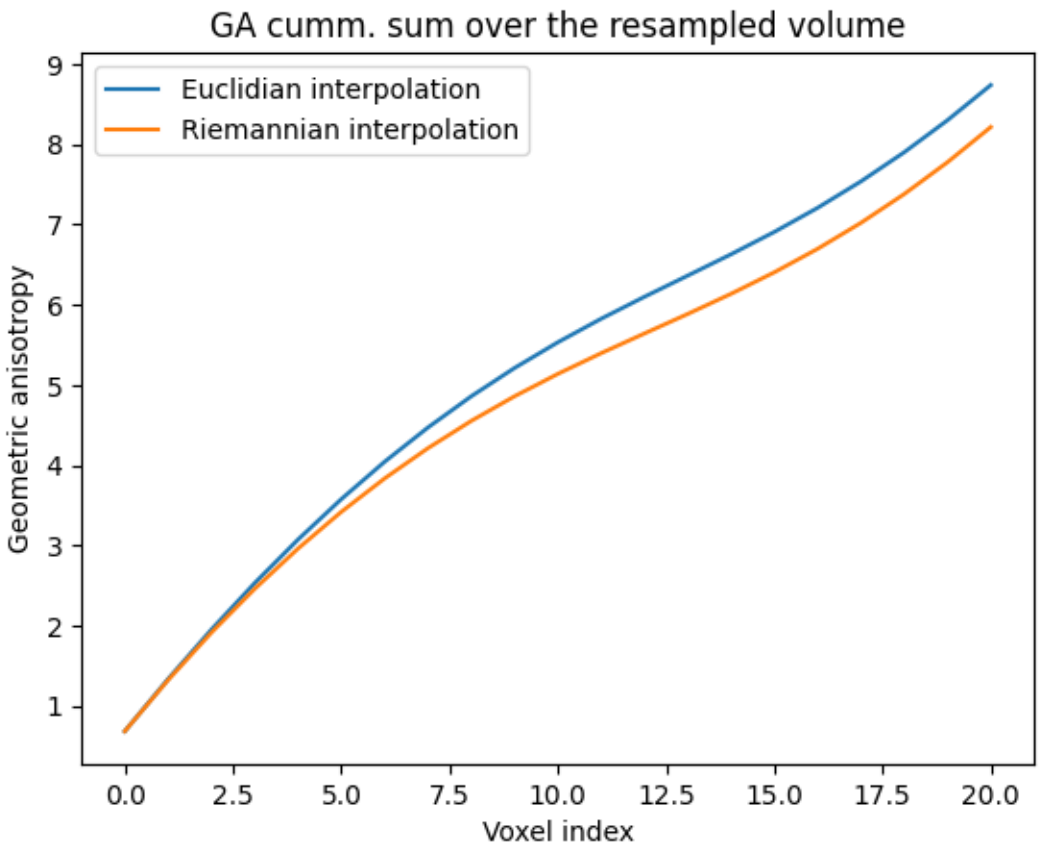
[8]:
```
# Generate the ODFs
sh1 = get_single_fiber_projection_odf()
sh2 = get_crossing_commisural_association_odf()
# Interpolate between the two ODFs using different metrics
interp_euc = interpolate_between_odfs_euclidean(sh1, sh2, 20)
interp_rie = interpolate_between_odfs_riemannian(sh1, sh2, 20)
# Assemble the two interpolations into a single "image"
interp = assemble_odf(interp_euc, interp_rie)
# Compute the GA and GFA of both interpolations
euc_ga = compute_ga(interp_euc)
rie_ga = compute_ga(interp_rie)

euc_gfa = compute_gfa(interp_euc)
rie_gfa = compute_gfa(interp_rie)
# Plot the progression of both metrics "over time"

plot_gfa(euc_gfa, 'Euclidian interpolation',
         rie_gfa, 'Riemannian interpolation')

plot_ga(euc_ga, 'Euclidian interpolation',
        rie_ga, 'Riemannian interpolation')
```

```
plot_gfa_cumsum(euc_gfa, 'Euclidian interpolation',
         rie_gfa, 'Riemannian interpolation')

plot_ga_cumsum(euc_ga, 'Euclidian interpolation',
        rie_ga, 'Riemannian interpolation')
```



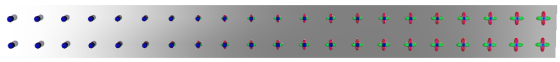

GA progression over the resampled volume
Euclidian interpolation
Riemannian interpolation
Geometric anisotropy
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Voxel index


GFA cumm. sum over the resampled volume
Euclidian interpolation
Riemannian interpolation
Generalized fractional anisotropy
Voxel index
1
2
3
4
5
6
7
8
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



[9]:
```
# Display the interpolation "over time" for both
metrics = np.stack((euc_ga, rie_ga), axis=0)
viz_odf_interp_1d(interp, metric=metrics)
Image(filename='interp.png')
```

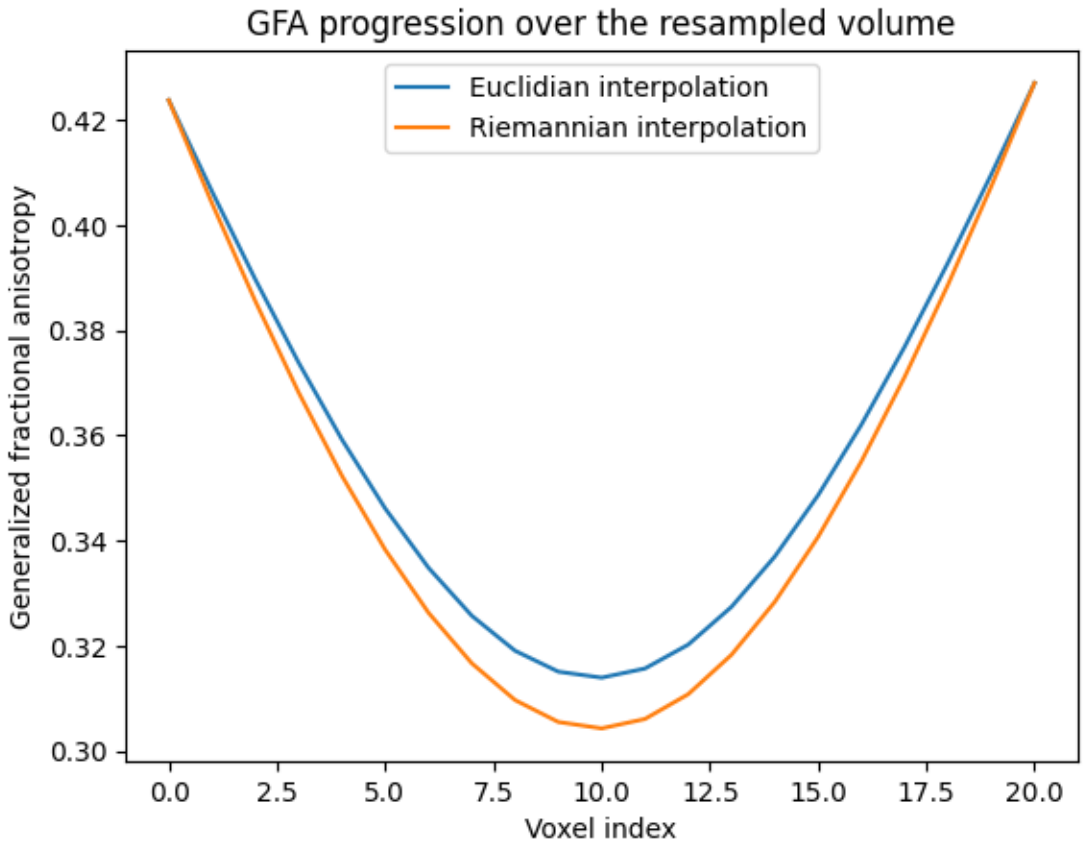
[9]:

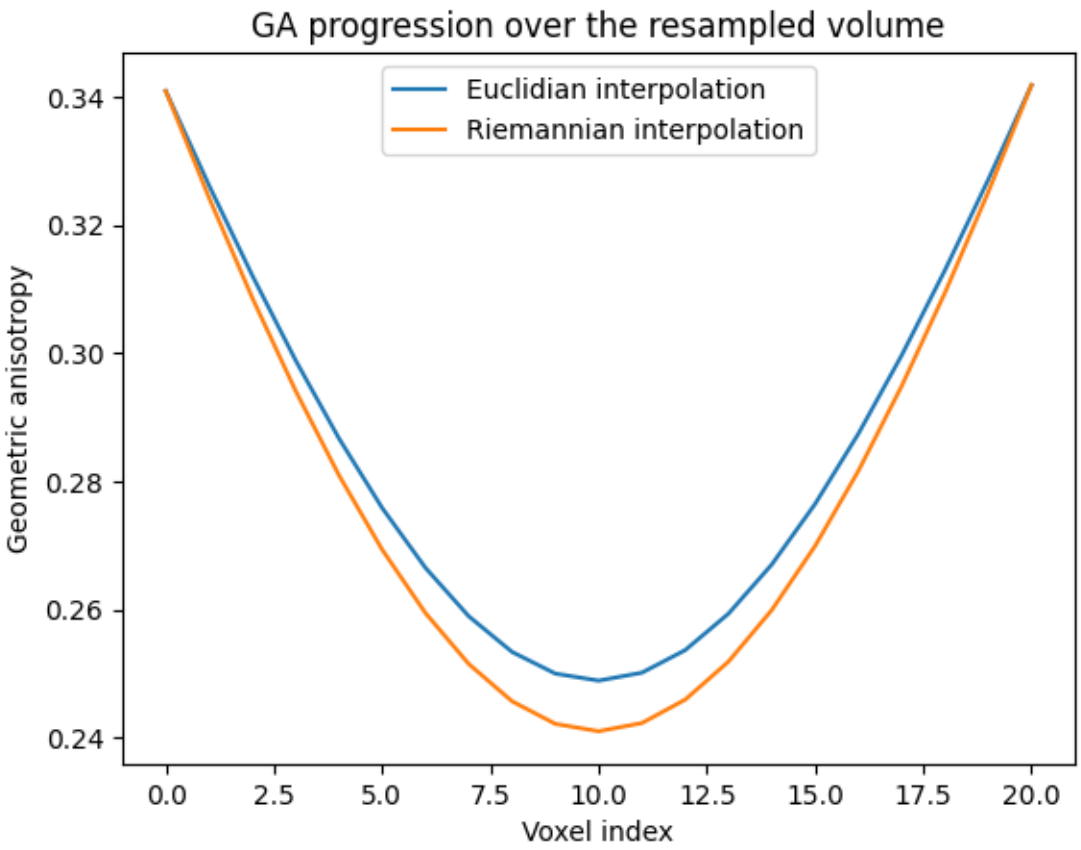


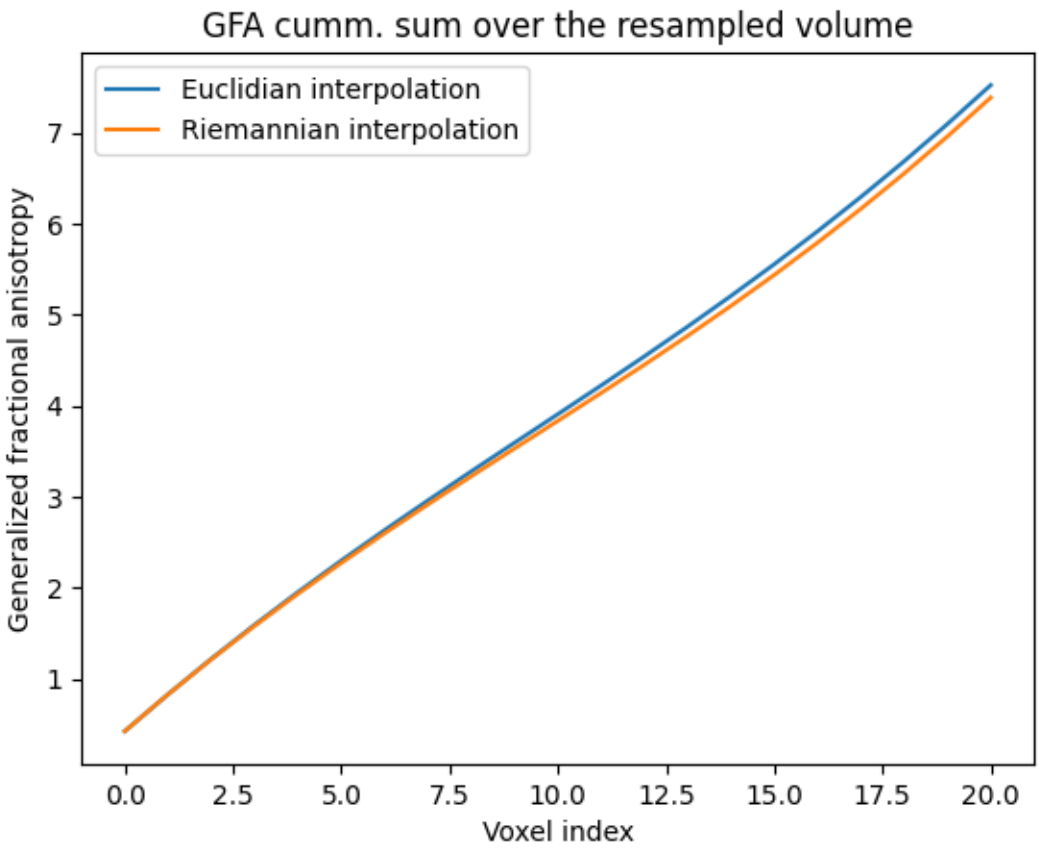
Again, results are consistent with the paper and euclidean interpolations leads to more GA and GFA over time. Last plots for experiment B, two-fiber to two-fiber interpolation !

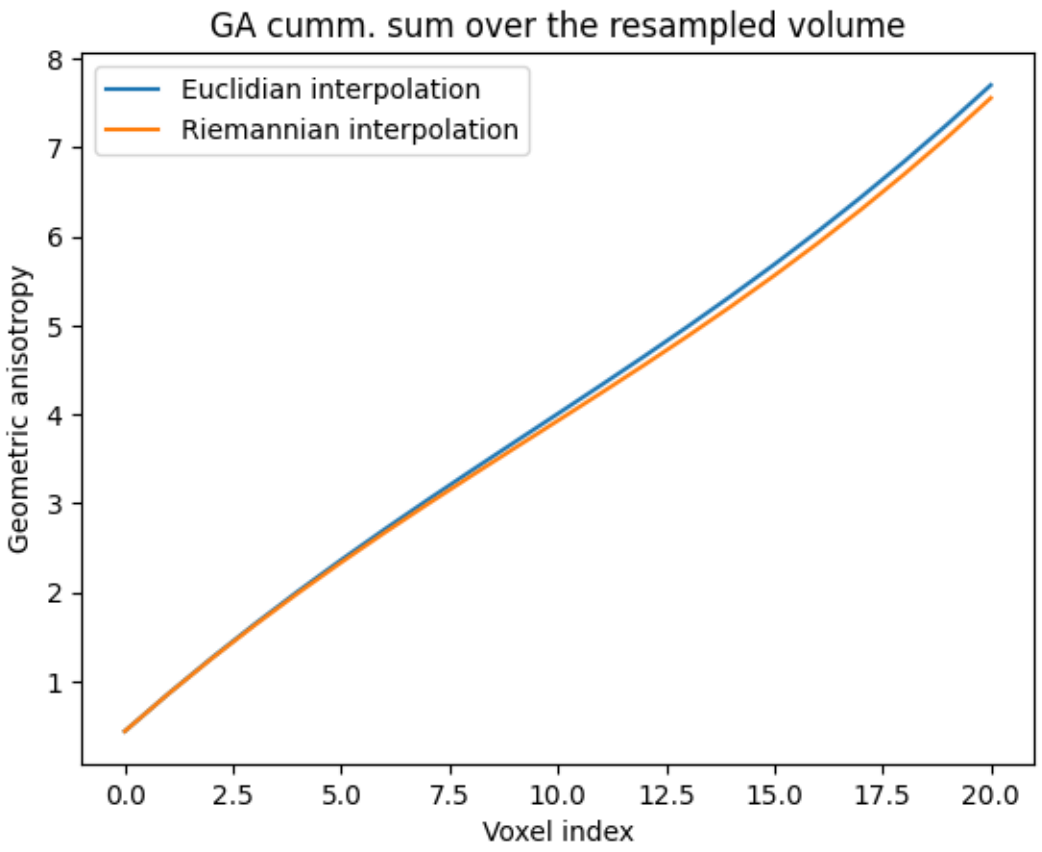
[10]:
```
# Generate the ODFs
sh1 = get_crossing_commisural_association_odf()
sh2 = get_crossing_association_projection_odf()
# Interpolate between the two ODFs using different metrics
interp_euc = interpolate_between_odfs_euclidean(sh1, sh2, 20)
interp_rie = interpolate_between_odfs_riemannian(sh1, sh2, 20)
# Assemble the two interpolations into a single "image"
interp = assemble_odf(interp_euc, interp_rie)
# Compute the GA and GFA of both interpolations
euc_ga = compute_ga(interp_euc)
rie_ga = compute_ga(interp_rie)

euc_gfa = compute_gfa(interp_euc)
rie_gfa = compute_gfa(interp_rie)
# Plot the progression of both metrics "over time"

plot_gfa(euc_gfa, 'Euclidian interpolation',
         rie_gfa, 'Riemannian interpolation')

plot_ga(euc_ga, 'Euclidian interpolation',
        rie_ga, 'Riemannian interpolation')

plot_gfa_cumsum(euc_gfa, 'Euclidian interpolation',
         rie_gfa, 'Riemannian interpolation')

plot_ga_cumsum(euc_ga, 'Euclidian interpolation',
```

```
        rie_ga, 'Riemannian interpolation')
```



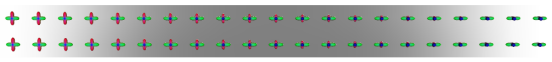

GA progression over the resampled volume
Euclidian interpolation
Riemannian interpolation
Geometric anisotropy
0.34
0.32
0.30
0.28
0.26
0.24
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Voxel index


GFA cumm. sum over the resampled volume
Euclidian interpolation
Riemannian interpolation
Generalized fractional anisotropy
1
2
3
4
5
6
7
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Voxel index



[11]:
```
# Display the interpolation "over time" for both
metrics = np.stack((euc_ga, rie_ga), axis=0)
viz_odf_interp_1d(interp, metric=metrics)
Image(filename='interp.png')
```

[11]:



Doesn't seem to do much in this case.

[ ]: