```
# Authors: Liam Martin, Nicholas Pho, Michael Clancy, Steven Abramowitch, Megan Routzong
# Purpose: Calculates and returns significant modes of variation of a set of point clouds using
a principal component
# analysis. Only functions with .vtk file types. Based on Mathematica script (developed by
Megan Routzong and Steven
# Abramowitch) to allow for better data and method sharing between research groups.
# Date of Last Edit: 6/29/22 - Liam Martin

# Future Feature List:
# 1) Full graphing support (Pre-release)
#     i) Normal distribution plots
#    ii) Save plots
#   iii) Color mapping on shapes based on deviation
# 2) Statistical analysis (Pre-release)
#     i) Assigning of Groups
#    ii) ANOVA
#   iii) ttest
#    iv) MANCOVA
#  3) Add table handling for patient data (Pre-release)
#     i) Age
#    ii) Parity
#  4) Other filetype support
#     i) Other filetypes (post-release)
#    ii) New vtk file format (pre-release)
#  5) Table size dynamically change with window (Pre-release)
#  6) Set random variation at constant variation (Pre-release)
#  7) Deformetrica support
#     i) Integrated support (post-release, needs additional permissions)
#    ii) Setup support (pre-release)
#   iii) Assessing deformetrica fit
#  8) 2 dimension support (pre-release)
#  9) Skip procrustes scaling (pre-release)
# 10) Convert filenames to list of numbers using existing code base (pre-release)
# 11) Display capabilities for all of the procrustes shapes
# 12) Display shapes by eigVa and eigVe, code at bottom of Mathematica
# 13) Integrate TICP code

import json  # Long term data storage, all project files are stored in .json file format
import natsort  # Better sorting algorithm, avoids 1, 10, 11, ..., 19, 2, 20 sorting
import tkinter.filedialog  # GUI toolkit
import tkinter.messagebox
from tkinter import *
from tkinter import ttk

from tkintertable import TableCanvas, TableModel  # tkinter table, what shows up in display
import os  # Filepath editing
import numpy as np  # Matrices
import vtk  # vtk file format library. Load, save, display vtk files
from vtk.util import numpy_support
from vtkmodules.vtkRenderingCore import (vtkActor, vtkPolyDataMapper, vtkRenderWindow,
vtkRenderWindowInteractor,
                                         vtkRenderer)
from sklearn.decomposition import PCA  # Principal component analysis
import matplotlib.pyplot as plt  # Plots
import pandas
import scipy.stats
from scipy import stats

# Global variables.
dimensions = 3
version = 1.1


# GUI class, allows for best variable practice.
class ShapeModelGUI(object):
    # Class initialization, called to start GUI.
    def __init__(self, root):
```

```
        # All of the data that is eventually saved for long term storage.
        self.importedData = np.empty(1)
        self.scaledData = np.empty(1)
        self.randomVariation = np.empty(1)
        self.dataVariation = np.empty(1)
        self.PCScores = np.empty(1)
        self.sigModes = 0
        self.polygons = vtk.vtkCellArray()
        self.meanShape = np.empty(1)
        self.namesOfVTKs = []
        self.tableDict = {}

        self.root = root

        # Sets up the frame that is used for the table later on.
        self.tframe = Frame(root, width=450, height=225)
        self.tframe.pack()
        self.table = TableCanvas(self.tframe, rows=0, cols=0)

        # Sets up main window for GUI.
        self.root.geometry('500x500')
        self.root.title('Statistical Shape Model (version 1.1)')

        # Declares the menubar.
        self.mainMenuBar = Menu(root)

        # All of the file menu options and calls.
        self.fileMainMenu = Menu(self.mainMenuBar, tearoff=0)
        self.fileMainMenu.add_command(label="New Dataset", command=self.newDataset)
        self.fileMainMenu.add_command(label="Save Dataset", command=self.saveData)
        self.fileMainMenu.add_command(label="Open Dataset", command=self.loadData)
        self.fileMainMenu.add_separator()
        self.fileMainMenu.add_command(label="Import Data", command=self.dataImporter)
        self.fileMainMenu.add_separator()
        self.fileMainMenu.add_command(label="Exit", command=self.quitter)
        self.fileMainMenu.add_separator()
        self.fileMainMenu.add_command(label="Clear Table", command=self.clearTable)
        self.mainMenuBar.add_cascade(label="File", menu=self.fileMainMenu)

        # All of the procrustes menu options and calls.
        self.procrustesMenuBar = Menu(self.mainMenuBar, tearoff=0)
        self.procrustesMenuBar.add_command(label="Run Procrustes", command=self.procrustes)
        self.procrustesMenuBar.add_separator()
        self.procrustesMenuBar.add_command(label="Export Procrustes Shapes",
command=self.exportProcrustesShape)
        self.procrustesMenuBar.add_command(label="View Average Shape",
command=self.meanDataVisualization)
        self.procrustesMenuBar.add_command(label="Export Average Shape",
                                           command=lambda:
self.exportAverageShape(self.meanShape, True, ''))
        self.mainMenuBar.add_cascade(label="Procrustes", menu=self.procrustesMenuBar)

        # All of the file PCA options and calls.
        self.principalComponentMenuBar = Menu(self.mainMenuBar, tearoff=0)
        self.principalComponentMenuBar.add_command(
            label="Run Principal Component Analysis", command=self.principalComponentAnalysis
        )
        self.principalComponentMenuBar.add_separator()
        self.principalComponentMenuBar.add_command(
            label="Show Scree Plot", command=self.screePlot
        )
        self.mainMenuBar.add_cascade(label="Principal Component Analysis",
menu=self.principalComponentMenuBar)

        self.statisticsMenuBar = Menu(self.mainMenuBar, tearoff=0)
        self.statisticsMenuBar.add_command(label="T-test", command=self.stats_ttest)
        self.statisticsMenuBar.add_command(label="ANOVA", command=None)
```

```
        self.statisticsMenuBar.add_separator()
        self.statisticsMenuBar.add_command(label='Display Distribution of Modes of Variation',
                                           command=None)
        self.mainMenuBar.add_cascade(label="Statistics", menu=self.statisticsMenuBar)

        # Initializes the menubar
        root.config(menu=self.mainMenuBar)

        root.mainloop()

    # Imports and formats data to work on.
    def dataImporter(self):
        # Sets defualt extension as .vtk as they are the supported filetype.
        extension = 'vtk'

        # Select the files to be included in the PCA. Sets names as an array with dimensions
(number of files x 1).
        self.namesOfVTKs = np.asarray(tkinter.filedialog.askopenfilenames(
            title="Select files to include in the shape model"))
        includedFiles = len(self.namesOfVTKs)

        # Confirms that all files are .vtk, removes them from the list otherwise.
        for i in self.namesOfVTKs:
            if not i.endswith(extension):
                includedFiles -= 1
                self.namesOfVTKs = np.delete(self.namesOfVTKs, np.where(self.namesOfVTKs ==
i))

        # Sorts the list into true numerical order. Avoids 1.vtk, 10.vtk, 11.vtk, ..., 19.vtk,
2.vtk.
        self.namesOfVTKs = natsort.natsorted(self.namesOfVTKs)

        # Set up the .vtk reader to read files.
        reader = vtk.vtkPolyDataReader()
        reader.ReadAllScalarsOn()
        reader.ReadAllVectorsOn()
        reader.SetFileName(self.namesOfVTKs[0])
        reader.Update()

        # Reads the output from the first .vtk file to store the connectivity. All shapes need
to have the same
        # connectivity in order to function.
        vtkData = reader.GetOutput()
        self.polygons = vtkData.GetPolys()

        # Pre-initializes the matrix for all of the data to go in
        data = np.empty((len(self.namesOfVTKs), int((vtkData.GetPoints().GetData().GetMaxId()
+ 1) / 3), 3))

        # Main loading loop. All of the point data is imported here.
        for i in range(len(self.namesOfVTKs)):
            reader.SetFileName(self.namesOfVTKs[i])
            reader.Update()
            # Actually Human Readable Data
            numpyPointArray =
vtk.util.numpy_support.vtk_to_numpy(vtkData.GetPoints().GetData())
            data[i] = numpyPointArray

        print("data loaded with", includedFiles, "." + extension, "files")

        self.initTable()
        self.importedData = data

    # Conducts procrustes analysis.
    def procrustes(self):
        # Loads a window for procrustes progress bar.
        procrustes_Win = Toplevel(self.root)
```

```
        procrustes_Win.geometry('200x100')
        procrustes_Win.title('Procrustes Scaling Progress')

        procrustes_Win.procrustesProgressBar = ttk.Progressbar(
            procrustes_Win,
            orient='horizontal',
            mode='determinate',
            length=180
        )

        procrustes_Win.procrustesProgressBar.grid(column=0, row=0, columnspan=2, padx=10,
pady=10)
        procrustes_Win.update()

        # Find out the number of landmarks (number of points*3).
        nLandmarks = self.importedData.shape[1]

        # Number of degrees of freedom.
        identityMatrix = np.identity(nLandmarks)
        populatedMatrix = np.full((nLandmarks, nLandmarks), 1 / nLandmarks)

        # Precalculates the difference between the identity and populated matrix.
        diff = identityMatrix - populatedMatrix

        # Calculates scale for each shape (sum of squares).
        sumSquare = np.empty(len(self.importedData))

        # Calculates the scale for each shape, approximatly the sum of squares.
        for index in range(len(self.importedData)):
            sumSquare[index] = np.sqrt(
                np.trace(
                    np.linalg.multi_dot([diff, self.importedData[index],
self.importedData[index].T, diff])
                )
            )

        # Update progress bar.
        procrustes_Win.procrustesProgressBar['value'] = 20
        procrustes_Win.procrustesProgressBar.update()

        # Normalizes each shape by scale.
        for index in range(len(self.importedData)):
            self.importedData[index] = np.matmul(diff, self.importedData[index]) /
sumSquare[index]

        # Update progress bar.
        procrustes_Win.procrustesProgressBar['value'] = 35
        procrustes_Win.procrustesProgressBar.update()

        # Calculates the mean of all shapes.
        meanShape = np.mean(self.importedData, axis=0)

        # Subtracts the mean for each scaled shape to centralize them.
        residSS = 0
        for index in range(len(self.importedData)):
            residSS += np.sum((self.importedData[index] - meanShape) ** 2)

        procrustes_Win.procrustesProgressBar['value'] = 50
        procrustes_Win.procrustesProgressBar.update()

        # Realigns each shape relative to the mean shape.
        while True:
            for i in range(len(self.importedData)):
                [u, w, v] = np.linalg.svd(np.matmul(np.transpose(self.importedData[i]),
meanShape))
                transMean = np.linalg.multi_dot([u, np.diag(w) @
np.linalg.inv(abs(np.diag(w))), v])
```

```
                self.importedData[i] = np.dot(self.importedData[i], transMean)

            meanShape = np.mean(self.importedData, axis=0)

            newResid = 0.0
            for index in range(len(self.importedData)):
                newResid += np.sum((self.importedData[index] - meanShape) ** 2)

            if abs(newResid - residSS) < 0.0001:
                break
            else:
                residSS = newResid

        # Update progress bar.
        procrustes_Win.procrustesProgressBar['value'] = 80
        procrustes_Win.procrustesProgressBar.update()

        pcaNorm = PCA(n_components=dimensions)
        x = pcaNorm.fit_transform(
            np.reshape(self.importedData.flatten(), (nLandmarks * len(self.importedData),
dimensions)))
        x[:, 1:3] = x[:, 1:3] * -1

        partiMatrix = np.reshape(x, (len(self.importedData), dimensions * nLandmarks))
        meanP = np.mean(partiMatrix, axis=0)
        partiMatrix = partiMatrix @ (np.identity(dimensions * nLandmarks) - meanP @ meanP)
        avgScale = np.mean(sumSquare)
        partiMatrix = partiMatrix * avgScale

        self.meanShape = meanShape
        self.scaledData = partiMatrix

        procrustes_Win.procrustesProgressBar['value'] = 100
        procrustes_Win.procrustesProgressBar.update()

    # Allows for visualization of mean shape after procrustes.
    def meanDataVisualization(self):
        points = vtk.vtkPoints()
        points.SetNumberOfPoints(len(self.meanShape))

        for i in range(len(self.meanShape)):
            points.SetPoint(i, self.meanShape[i][0], self.meanShape[i][1], self.meanShape[i]
[2])

        meanShape = vtk.vtkPolyData()
        meanShape.SetPoints(points)
        meanShape.SetPolys(self.polygons)

        polygonMapper = vtkPolyDataMapper()
        polygonMapper.SetInputData(meanShape)
        polygonMapper.Update()

        polygonActor = vtkActor()
        polygonActor.SetMapper(polygonMapper)
        ren = vtkRenderer()
        ren.AddActor(polygonActor)

        ren.ResetCamera()

        renWin = vtkRenderWindow()
        renWin.SetWindowName('Average Shape')
        renWin.AddRenderer(ren)
        renWin.SetSize(300, 300)

        iren = vtkRenderWindowInteractor()
        iren.SetRenderWindow(renWin)
        iren.Initialize()
```

```
        iren.Start()

    # Export the scaled shapes in procrustes so they are ready for external view.
    def exportProcrustesShape(self):
        partiMatrix = np.reshape(
            self.scaledData, (len(self.scaledData), int(len(self.scaledData[0]) / dimensions),
dimensions))

        for i in range(len(self.namesOfVTKs)):
            filename = os.path.join(
                os.path.dirname(
                    self.namesOfVTKs[i]),
                os.path.basename(
                    self.namesOfVTKs[i])[:-4] + '_procrustes' +
os.path.basename(self.namesOfVTKs[i])[-4:])
            self.exportShape(partiMatrix[i], False, filename)

    # "Heavy lifter" function, does the PCA analysis. PC scores calculated here.
    def principalComponentAnalysis(self):
        PCA_win = Toplevel(self.root)
        PCA_win.geometry('200x100')
        PCA_win.title('Principal Component Analysis Progress')

        PCA_win.principalComponentProgressBar = ttk.Progressbar(
            PCA_win,
            orient='horizontal',
            mode='determinate',
            length=180
        )

        PCA_win.principalComponentProgressBar.grid(column=0, row=0, columnspan=2, padx=10,
pady=10)

        PCA_win.principalComponentProgressBar['value'] = 0.0
        PCA_win.update()

        meanShapes = np.mean(self.scaledData, axis=0)
        residuals = self.scaledData - meanShapes
        (nShapes, nLandmarks) = self.scaledData.shape

        pPCA = PCA()
        pPCA.fit_transform(residuals)
        eVectors = pPCA.components_ * -1
        dV = pPCA.explained_variance_ratio_

        its = 10000
        rV = np.zeros((its, nShapes))

        i = 0
        while i < its:
            moDepts = np.random.multivariate_normal(np.full(nShapes, 0), np.identity(nShapes),
nLandmarks)
            newResiduals = np.transpose(moDepts)
            pPCA.fit_transform(newResiduals)
            newpercentVariance = pPCA.explained_variance_ratio_
            rV[i, :nShapes] = newpercentVariance
            i += 1
            if i % 100 == 0:
                # txt = "Random generation of noise is {percentComplete:.2f}% completed."
                # print(txt.format(percentComplete=i / its * 100))
                PCA_win.principalComponentProgressBar['value'] += 1
                PCA_win.update()

        rV = np.mean(rV, axis=0)

        self.sigModes = len([num for num in dV.round(2) - rV.round(2) if num > 0])
```

```
        def pcaScoreCalc():
            return np.dot(residuals, eVectors[0:self.sigModes, :].T)

        PCS = pcaScoreCalc()

        self.PCScores = PCS
        self.randomVariation = rV
        self.dataVariation = dV
        self.displayModes()
        self.screePlot()

    # Display screePlot. For determination of significant variation modes.
    def screePlot(self):
        plt.cla()
        plt.plot(
            range(len(self.randomVariation)),
            self.randomVariation,
            label="Random Variation",
            color="blue",
            linestyle=":",
            markevery=True
        )

        plt.plot(
            range(len(self.dataVariation)),
            self.dataVariation,
            label="Data Variation",
            color="red",
            markevery=True
        )

        plt.legend()
        plt.show()

    # Save data for all steps. Sets unused variables to empty variables for saving.
    def saveData(self):
        def writeToJSONFile(data, location):
            with open(location, 'w') as f:
                json.dump(data, f, indent=1)

        polygonData = np.delete(vtk.util.numpy_support.vtk_to_numpy(self.polygons.GetData()),
slice(None, None, 4))

        savingData = {
            "version": version,
            "namesOfVTKs": self.namesOfVTKs,
            "rawData": self.importedData.tolist(),
            "scaledData": self.scaledData.tolist(),
            "randomVariation": self.randomVariation.tolist(),
            "dataVariation": self.dataVariation.tolist(),
            "PCScores": self.PCScores.tolist(),
            "meanShape": self.meanShape.tolist(),
            "sigModes": self.sigModes,
            "polygonData": polygonData.tolist(),
            "tableData": self.table.model.data
        }

        files = [('JSON File', '*.json')]
        file = tkinter.filedialog.asksaveasfilename(filetypes=files, defaultextension='json')
        writeToJSONFile(savingData, file)

    # Loads data from properly formatted .json file.
    def loadData(self):
        def loadFromJSONFile(location):
            with open(location, 'r') as f:
                data = json.load(f)
```

```
                if version != data["version"]:
                    print("Loading Data From a previous version. Errors may occur")

                self.namesOfVTKs = np.array(data["namesOfVTKs"])
                self.importedData = np.array(list(data["rawData"]))
                self.scaledData = np.array(list(data["scaledData"]))
                self.randomVariation = np.array(list(data["randomVariation"]))
                self.dataVariation = np.array(list(data["dataVariation"]))
                self.PCScores = np.array(list(data["PCScores"]))
                self.meanShape = np.array(list(data["meanShape"]))
                self.sigModes = data["sigModes"]
                self.polygons.SetData(3,
vtk.util.numpy_support.numpy_to_vtk(np.array(list(data["polygonData"]))))
                self.tableDict = data["tableData"]

        if self.table.model.getColumnCount() > 0:
            self.savePrompt(prompt='Load Data')
            self.clearData()
            self.clearTable()

        files = [('JSON File', '*.json')]
        file = tkinter.filedialog.askopenfilename(filetypes=files, defaultextension='json')
        loadFromJSONFile(file)

        self.initTable()

    def savePrompt(self, prompt):
        shouldSave = tkinter.messagebox.askyesno(prompt, 'Do you want to save the current
dataset?')

        if shouldSave:
            self.saveData()

    # Create new dataset.
    def newDataset(self):
        self.savePrompt(prompt='New Dataset')
        self.table.clearData()
        self.clearData()
        self.clearTable()

    # Sets all data based variables to empty variables. Only call during resets.
    def clearData(self):
        self.importedData = np.empty(1)
        self.scaledData = np.empty(1)
        self.randomVariation = np.empty(1)
        self.dataVariation = np.empty(1)
        self.PCScores = np.empty(1)
        self.meanShape = np.empty(1)

    # Uncalled function. For testing purposes.
    def printer(self):
        print(self.importedData)
        print(self.scaledData)

    # Quits program.
    def quitter(self):
        self.savePrompt('Save Dataset')

        self.root.quit()
        self.root.destroy()

    # Helper function for exporting shapes. Average shape is handled differently than all
others.
    def exportShape(self, pointCloud, isAverageShape, exportedShapeFilename):
        if isAverageShape:
            exportedShapeFilename = tkinter.filedialog.asksaveasfilename(
                filetypes=[('VTK Files', '*.vtk')], defaultextension='vtk')
```

```
        points = vtk.vtkPoints()
        points.SetNumberOfPoints(len(pointCloud))

        for i in range(len(pointCloud)):
            points.SetPoint(i, pointCloud[i][0], pointCloud[i][1], pointCloud[i][2])

        shape = vtk.vtkPolyData()
        shape.SetPoints(points)
        shape.SetPolys(self.polygons)

        writer = vtk.vtkPolyDataWriter()
        writer.SetFileVersion(42)
        writer.SetFileName(exportedShapeFilename)
        writer.SetInputData(shape)
        writer.Write()

    # Initializes the table.
    def initTable(self):
        if self.tableDict != {}:
            model = self.table.model
            model.importDict(self.tableDict)
        else:
            namesOfVTKs_Dict = {}

            for i in range(len(self.namesOfVTKs)):
                namesOfVTKs_Dict['rec' + str(i + 1)] = {"Filenames":
os.path.basename(self.namesOfVTKs[i])}

            model = self.table.model
            model.importDict(namesOfVTKs_Dict)
            self.displayModes()

        self.table.show()

    # Adds modes of variation to the table.
    def displayModes(self):
        for j in range(self.sigModes):
            existingCols = self.table.model.getColumnCount()
            self.table.addColumn("Mode " + str(j + 1))
            for k in range(len(self.namesOfVTKs)):
                self.table.model.setValueAt(str(round(self.PCScores[k][j], 2)), k,
existingCols)

    def clearTable(self):
        self.tframe.destroy()
        self.tframe = Frame(self.root, width=450, height=225)
        self.tframe.pack()
        self.table = TableCanvas(self.tframe, rows=0, cols=0)

    def stats_ttest(self):
        ttestWin = Toplevel(self.root)
        ttestWin.geometry("500x500")
        ttestWin.title('T-Test Menu')
        cols = []

        for i in range(self.table.model.getColumnCount()):
            cols.append(self.table.model.getColumnName(i))

        StatDataFrame = pandas.DataFrame.from_dict(self.table.model.data).transpose()
        StatDataFrame.groupby('Category')

        def valueGetter(temp):
            out = []
            for j in range(len(temp)):
                out.append(float(temp['Mode 1'][j]))
            return out
```

```
        group1 = StatDataFrame[StatDataFrame['Category'] == '1']
        group2 = StatDataFrame[StatDataFrame['Category'] == '0']

        group1 = valueGetter(group1)
        group2 = valueGetter(group2)

    def getDummyVar(self, window):
        dummyCols = []
        cols = self.table.model.getColumnCount()
        for i in range(cols):
            name = self.table.model.getColumnName(i)
            if name != 'Filenames' and not name.startswith('Mode'):
                dummyCols.append(name)

        theDummyCol = StringVar(window)
        dropDown = OptionMenu(window, theDummyCol, *dummyCols)
        dropDown.grid(column=1, row=0)

        def selectedCol():
            self.colName = theDummyCol.get()

        selectButton = Button(window, text="Set Dummy Variable", command=selectedCol)
        selectButton.grid(column=0, row=1)


if __name__ == '__main__':
    Root = Tk()
    Procrustes = ShapeModelGUI(Root)
    Root.mainloop()
```