Candidate Number: 091388 Rename HVG.py to hvg.py 07d10b3 5 hours ag

1 contributor

253 lines (186 sloc) 6.65 KB

```
from math import *
from Coursework_2.logisticmap import logistic_map
from random import uniform


def get_series(n, stype = 0):
    """
    Produces either a monotonic(0), alternating(1) or sinusoid(2) series
        of length 'n'

    :param n: Length of the series
    :param stype: Type of the series
    :return: The series of length n
    """

    series = []

    for i in range(n):

        # CREATE MONOTONIC SERIES
        if stype == 0:
            series.append(i)

        # CREATE ALTERNATING SERIES
        elif stype == 1:
            if i % 2 == 0:
                series.append(1)
            else:
                series.append(0)

        # CREATE SINUSOID SERIES
        elif stype == 2:
            term = sin((2*pi)*500*(i/10000))
            series.append(term)

    return series


def horizontal_visibility_graph(series, weighted = False):
    """
    Take a series and works out the adjacency of the terms, then returning
        an adjacency matrix

    :param series: A series of numbers
    :param weighted: Whether the function should produce a weighted matrix
    :return: An adjacency matrix
    """

    # BUILD MATRIX FULL OF 0'S
    size_of_each_row = len(series)
    amount_of_lists = len(series)
    matrix = [[0 for j in range(size_of_each_row)] for i in range(amount_of_lists)]

    if weighted:

        # POPULATE MATRIX
        for i in range(len(series)):
            for j in range(len(series)):
```

```
                # FIND VALUES FOR COMPARISON
                items_between = find_items_between(i, j, series)
                highest_between = highest_value_in_list(items_between)
                comparator = get_comparator(i, j, series)

                # SET VALUES
                if i == j:
                    matrix[i][j] = 0
                elif i == (j + 1):
                    matrix[i][j] = round(1/(sqrt(((i-j)**2)+(series[i]-series[j])**2)), 3)
                elif i == (j - 1):
                    matrix[i][j] = round(1/(sqrt(((i-j)**2)+(series[i]-series[j])**2)), 3)
                elif highest_between < comparator:
                    matrix[i][j] = round(1/(sqrt(((i-j)**2)+(series[i]-series[j])**2)), 3)
                else:
                    matrix[i][j] = 0

    else:

        # POPULATE MATRIX
        for i in range(len(series)):
            for j in range(len(series)):

                # FIND VALUES FOR COMPARISON
                items_between = find_items_between(i, j, series)
                highest_between = highest_value_in_list(items_between)
                comparator = get_comparator(i, j, series)

                # SET VALUES
                if i == j:
                    matrix[i][j] = 0
                elif i == (j + 1):
                    matrix[i][j] = 1
                elif i == (j - 1):
                    matrix[i][j] = 1
                elif highest_between < comparator:
                    matrix[i][j] = 1
                else:
                    matrix[i][j] = 0

    return matrix


def find_items_between(start_index, end_index, series):
    """
    Returns a list of everything between a specified
        start and end point in a list

    :param start_index: The starting index that specifies where in the list
        to take the data from
    :param end_index: The ending index that specifies where in the list
        to take the data from
    :param series: The list of terms
    :return: A list of all items between the two indexes specified
    """

    # SWAP INDEXES IF STARTING > ENDING
    temp = end_index
    if start_index > end_index:
        end_index = start_index
        start_index = temp

    # FIND VALUES BETWEEN INDEXES
    items_between = []
    for i in range(len(series)):

        if i == start_index:
            continue
        elif i == end_index:
            continue
```

```
        elif start_index <= i <= end_index:
            items_between.append(series[i])

    return items_between


def highest_value_in_list(series):
    """
    Finds the highest numerical value in a list

    :param series: A series of numerical values
    :return: The highest numerical value
    """

    greatest = 0

    if len(series) > 0:
        greatest = series[0]

    for n in series:
        if n >= greatest:
            greatest = n

    return greatest


def get_comparator(index_1, index_2, series):
    """
    Finds the lowest item between two values at given indexes in a
        list and returns lowest

    :param index_1: First index point to compare the value of
    :param index_2: Second index point to compare the value of
    :param series: The list of values
    :return: The lowest of the two values at the specified indexes
    """

    # FIND VALUES AT INDEXES TO BE COMPARED
    value_1 = series[index_1]
    value_2 = series[index_2]
    comparator = 0

    # TAKE LOWEST VALUE OF THE TWO FOR COMPARISON
    if value_1 >= value_2:
        comparator = value_2
    elif value_1 <= value_2:
        comparator = value_1

    return comparator


def print_hvg(hvg):
    """
    Prints a hvg, replacing the 0's with a space, in a table format

    :param hvg: A hvg in the form of an adjacency matrix
    """
    # REPLACE 0'S WITH SPACES
    for i in range(len(hvg)):
        for j in range(len(hvg[i])):
            if hvg[i][j] == 0:
                hvg[i][j] = " "

    # PRINT EVERY ITEM IN HVG
    for i in range(len(hvg)):
        for j in range(len(hvg)):
            print(hvg[i][j], end="")
        print("")

    print("")

```

```
201
202 def process_logisticmap(params, steps = 100):
203     """
204     Creates a dictionary, from an adjacency matrix, produced by a
205         logistic map of the passed parameters
206
207     :param params: The parameters to be passed to logistic map
208     :param steps: Number of y coordinates to be produced when
209         logistic map is called
210     :return: A dictionary of key value pairs of parameters: matrices
211     """
212
213     dictionary = {}
214
215     for i in params:
216         hvg = horizontal_visibility_graph(logistic_map(uniform(0, 1), steps, i))
217         dictionary[i] = hvg
218
219     return dictionary
220
221
222 # MAIN PROGRAM
223 if __name__ == "__main__":
224
225     # CALCULATE SERIES
226     monotonic = get_series(30, 0)
227     alternating = get_series(30, 1)
228     sinusoid = get_series(30, 2)
229
230     # PRINT ADJACENCY MATRICES
231     print("MONOTONIC SERIES")
232     print_hvg(horizontal_visibility_graph(monotonic))
233
234     print("ALTERNATING SERIES")
235     print_hvg(horizontal_visibility_graph(alternating, True))
236
237     print("SINUSOID SERIES")
238     print_hvg(horizontal_visibility_graph(sinusoid))
239
240     # LOGISTIC MAPPINGS
241     logistic_parameters = [3.0, 3.4, 3.6785, 3.84, 4]
242     logisticmap_dictionary = process_logisticmap(logistic_parameters)
243
244     # PRINT KEY VALUE PAIRS
245     for key, value in logisticmap_dictionary.items():
246         print("KEY: ", key)
247         print_hvg(value)
248
249
250
251
252
```