Documentation for the CrimeStat Libraries 1.1

Ned Levine & Associates
Houston, TX
&
National Institute of Justice
Washington, DC

December 2012

This page is intentionally left blank

Table of Contents

This page is intentionally left blank

CrimeStat® Libraries 1.1

The following provides documentation on the library functions for version 1.1 of the CrimeStat® Libraries. The background to the statistics is presented only in an elementary way. The user should consult the regular CrimeStat manual for more complete information (http://www.icpsr.umich.edu/CrimeStat).

The CrimeStat Libraries 1.1 was developed by ***Ned Levine & Associates***, Houston, TX, under a cooperative development agreement with Grant 2005-IJ-CX-K037 from the Office of Science and Technology***, National Institute of Justice*** (NIJ), Washington, DC. The libraries include routines that were available in CrimeStat 2.0 as well as several additional routines. The programming of the libraries was produced by researchers in the Department of Computer Science and Engineering at the University of Minnesota, Minneapolis, MN, under the direction of Professor Shashi Shekhar, and then refined by us. The developer would like to thank the following for their contribution to the effort:

1. Professor Shashi Shekhar of the University of Minnesota who organized the library development and supervising of the programming. The effort would never have been completed without his expertise and continual oversight over the programming effort.

2. The programmers at the University of Minnesota who worked on the effort over a two and a half year period – Mr. Vijay Gandhi; Mr. Chetan Shivarudrappa; Mr. Pradeep Mohan; and Ms. S.Preethalakshmi.

3. Ms. Haiyan Teng of Houston, TX who evaluated the libraries for consistency with the existing CrimeStat program, conducted quality control tests, added graphical output capability, and ensured that the routines were consistent with the existing program for the comparable routines.

4. Mr. Ronald Wilson, formerly the program manager at the Mapping and Analysis for Public Safety Program (MAPS) at NIJ who supported the project through this development and provided valuable feedback on the libraries. The idea for the development of CrimeStat libraries was proposed by him.

5. The beta testers for the libraries including Mr. Robert Cheetham of Avencia, Inc; Mr. Gaston Pezzuchi of the Buenos Aires Police Department; Mr. Phil Mielke of the City of Redlands, CA; Dr. Issac Van Patten of Radford University; Mr. Jonathon Mayer of Map Analysis Ltd.; and Mr. Rolando Paredes of Spatial Transactions, LLC.

6. Support for the quality control evaluation of the libraries was provided by the South Carolina Research Authority.

7. Mr. Joel Hunt of NIJ who has taken on the oversight role for the distribution of the libraries.

8. The National Archive for Criminal Justice Data (NACJD) at ICPSR in the University of Michigan for distributing the libraries.

License Agreement

CrimeStat® is a registered trademark of Ned Levine & Associates. The libraries were developed under a joint development agreement between Ned Levine and Associates and the National Institute of Justice and are intended for the use of law enforcement agencies, criminal justice, researchers in other fields, and educators. They can be distributed freely for educational or research purposes, but cannot be re-sold. They must be cited correctly in the documentation for a software package that utilizes them as well as in any publication or report that uses their results. The correct citation is:

> Ned Levine, *CrimeStat Libraries (1.1)*. Ned Levine & Associates, Houston, TX, and the National Institute of Justice, Washington, DC, December 2012.

The National Institute of Justice, Office of Justice Programs, United States Department of Justice reserves a royalty-free, non-exclusive, and irrevocable license to reproduce, publish, or otherwise use, and authorize others to use this program for Federal government purposes. This program cannot be distributed without the permission of both Ned Levine and Associates and the National Institute of Justice, except as noted above.

Neither Ned Levine and Associates, the United States Government nor any of their respective employees make any warranty, express or implied, including but not limited to the warranties of merchantability and fitness for a particular purpose. In no event will Ned Levine and Associates, the United States Government or any of their respective employees be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the software or documentation. Neither Ned Levine and Associates, the United States Government, nor their respective employees are responsible for any costs including, but not limited to, those incurred as a result of lost profits or revenue, loss of time or use of software, loss of data, the costs of recovering such software or data, the cost of substitute software, or other similar costs. Any actions taken or documents printed as a result of using this software and its accompanying documentation remain the responsibility of the user.

Any questions about the use of this program should be directed to either:

Dr. Ned Levine
Ned Levine & Associates
Houston, TX
CrimeStat@nedlevine.com

Mr. Joel Hunt
Mapping and Analysis for
Public Safety Program
National Institute of Justice
U. S. Department of Justice
810 7th St, NW
Washington, DC 20531
Joel.Hunt@usdoj.gov

This page is intentionally left blank

User Notes

1. Some users have reported receiving an error message when they try to load one or more of the libraries (e.g., error message HRESULT 0x800736b1. The message appears to be related to not having installed the latest service pack from Microsoft for the MFC and C++ runtime modules. If you get this message, please download and install the latest service pack at:

http://www.microsoft.com/downloads/en/details.aspx?FamilyID=766A6AF7-EC73-40FF-B072-9112BAB119C2&displaylang=en

This page is intentionally left blank

CrimeStat Core Components Library

Data Setup

The CrimeStat Core Components library provides routines for defining the data set and variables for a primary file (required) and a secondary file (optional), identifying a reference grid (required for several routines), and defining measurement parameters (required for several routines). See chapter 3 of the CrimeStat manual for more detail.

Primary File

A primary file is required for CrimeStat. It is a point file with X and Y coordinates. For example, the primary file could be the location of street robberies, each of which have an associated X and Y coordinate. There can be associated weights or intensities, though these are optional. Also, there can be time references, though these are optional. For example, if the points are the locations of police stations, then the intensity variable could be the number of calls for service at each police station while the weighting variable could be service zones. Currently, only a single file can be read. However, we are planning on adding support for multiple files. The time references are used in the space-time analysis routines are defined in terms of hours, days, weeks, months, or years.

Select Files

Select the primary file. The CrimeStat library can currently read dBase '.dbf' files, text files '.txt', ESRI shape '.shp'.. Select the type of file to be selected. Use the browse button to search for a particular file name.

Variables

Define the file that contains the X and Y coordinates. If there are weights or intensities being used, define the file that contains these variables. Certain statistics (e.g., spatial autocorrelation, local Moran) require intensity values and most other statistics can use intensity values. Most other statistics can also use weights. It is possible to have both an intensity variable and a weighting variable, though the user should be cautious in doing this to avoid 'double weighting'. If a time variable is used, it must be an integer or real number (e.g., 1, 36892). Do not use formatted dates (e.g., 01/01/2001, October 1, 2001). Convert these to real numbers before using the space-time analysis routines.

Columns

Select the variables for the X and Y coordinates respectively (e.g., Lon, Lat, Xcoord, Ycoord.) If weights or intensities are being used, select the appropriate variable names. If a time variable is used, select the appropriate variable name.

Missing Values

Identify whether there are any missing values. By default, CrimeStat will ignore records with blank values in any of the eligible fields or records with non-numeric values (e.g., alphanumeric characters, #, *). Blanks will always be excluded unless the user selects **<*none*>**. There are 8 possible options:

1. Blank fields are automatically excluded. This is the default.
2. Indicates that no records will be excluded. If there is a blank field, CrimeStat will treat it as a 0
3. 0 is excluded
4. -1 is excluded
5. 0 and -1 indicates that both 0 and -1 will be excluded
6. 0, -1 and 9999 indicates that all three values (0, -1, 9999) will be excluded
7. Any other numerical value can be treated as a missing value by typing it (e.g., 99)
8. Multiple numerical values can be treated as missing values by typing them, separating each by commas (e.g., 0, -1, 99, 9999, -99)

Directional Coordinates

If the file contains directional coordinates (angles), define the file name and variable name (column) that contains the directional measurements. If directional coordinates are being used, there can be an optional distance variable for the measurement. Define the file name and variable name (column) that contains the distance variable.

Time units

Define the units for the time variable and are defined in terms of hours, days, weeks, months, or years. Time is only used for the primary file. The default value is days. Note, only integer or real numbers can be used (e.g., 1, 36892). Do not use formatted dates (e.g., 01/01/2001, October 1, 2001). Convert these to real or integer numbers before using the space-time analysis routines.

Type of Coordinate System and Data Units

Select the type of coordinate system. If the coordinates are longitudes and latitudes, then a spherical system is being used and data units will automatically be decimal degrees. If the coordinate system is projected (e.g., State Plane, UTM), then data units could be feet (e.g., State Plane), meters (e.g., UTM.), miles, kilometers, or nautical miles. If the coordinate system is directional, then the coordinates are angles and the data unit box will be blanked out. For directions, an optional distance variable can be used which specifies the distance of the incident from an origin location; the units are undefined. Note: if a projected coordinate system is used, but the coordinate system is defined as longitude/latitude (spherical), an error message will appear that says "Found invalid data at row 1 of the primary data set!". Change the coordinate system to Projected (Euclidean)".

Secondary File

A secondary data file is optional. It is also a point file with X and Y coordinates. It is usually used in comparison with the primary file. There can be weights or intensities variables associated, though these are optional. For example, if the primary file is the location of motor vehicle thefts, the secondary file could be the centroid of census block groups that have the population of the block group as the intensity (or weight) variable. In this case, one could compare the distribution of motor vehicle thefts with the distribution of population in, for example, the Ripley's "K" routine or the dual kernel density estimation routine. Currently, only a single file can be read. Time units are not used in the secondary file.

Select files

Select the primary file. The CrimeStat library can currently read dBase '.dbf' files, text files '.txt', ESRI shape '.shp'. Select the type of file to be selected. Use the browse button to search for a particular file name.

Variables

Define the file that contains the X and Y coordinates. If weights or intensities are being used, define the file that contains these variables. Certain statistics (e.g., spatial autocorrelation, local Moran) require intensity values and most other statistics can use intensity values. Most other statistics can use weights. It is possible to have both an intensity variable and a weighting variable, though the user should be cautious in doing this to avoid 'double weighting'. Time units are not used in the secondary file.

Columns

Select the variables for the X and Y coordinates respectively (e.g., Lon, Lat, Xcoord, Ycoord.) If there are weights or intensities being used, select the appropriate variable names. Time units are not used in the secondary file.

Missing values

Identify whether there are any missing values. By default, *CrimeStat* will ignore records with blank values in any of the eligible fields or records with non-numeric values (e.g., alphanumeric characters, #, *). Blanks will always be excluded unless the user selects ***<none>***. There are 8 possible options:

1. ***<blank>*** fields are automatically excluded. This is the default
2. ***<none>*** indicates that no records will be excluded. If there is a blank field, *CrimeStat* will treat it as a 0
3. ***0*** is excluded
4. ***–1*** is excluded
5. ***0 and –1*** indicates that both 0 and -1 will be excluded
6. ***0, -1 and 9999*** indicates that all three values (0, -1, 9999) will be excluded
7. ***Any*** other numerical value can be treated as a missing value by typing it (e.g., 99)
8. ***Multiple*** numerical values can be treated as missing values by typing them, separating each by commas (e.g., 0, -1, 99, 9999, -99)

Type of coordinate system and data units

The secondary file must have the same coordinate system and data units as the primary file. Directional coordinates (angles) are not allowed for the secondary file nor are time variables.

Reference File

For referencing the study area, there is a reference grid, a reference origin, and two X/Y coordinates that define the lower-left and upper-right corners of the grid. The reference file is used in the risk-adjusted nearest neighbor hierarchical clustering routine, journey-to-crime estimation and in the single and dual variable kernel density estimation routines. The file can be an external file that is input or can be generated by *CrimeStat*. It is usually, though not always, a grid which is overlaid on the study area. The reference

origin is used in the directional mean routine. The file can be an external file that is input or can be generated by *CrimeStat*. The coverage is the area of the study region and the length of the street network.

Create reference grid

If allowing *CrimeStat* to generate a true grid, click on 'generated' and then input the lower left and upper right X and Y coordinates of a rectangle placed over the study area. Cells can be defined either by cell size, in the same coordinates and data units as the primary file, or by the number of columns in the grid (the default). In addition, a reference origin can be defined for the directional mean routine. The reference grid can be saved and re-used. Click on 'Save' and enter a file name. To use an already saved file, click on 'Load' and the file name. The coordinates are saved in the registry, but can be re-saved in any directory. With the Load screen open, click on 'Save to file' and then enter a directory and a file name. The default file extension is 'ref.

External reference file

If an external file that stores the coordinates of each grid cell is used, select the name of the reference file. CrimeStat can currently read dbase '.dbf' files though development on reading ESRI '.shp', Microsoft Access 'mdb' files and files formats that correspond to the ODBC standard is envisioned. Use the browse button to search for the file.

Reference origin

A reference origin can be defined for the directional mean routine. The reference origin can be assigned to:

1. Use the lower-left corner defined by the minimumX and Y values. This is the default
2. Use the upper-right corner defined by the maximum X and Y values
3. Use a different origin point. With the later, the user must define the origin

Measurement Parameters

The measurement parameters page defines the measurement units of the coverage and the type of distance measurement to be used. It is currently a part of CInputParam. There are three components that are defined:

Area

First, define the geographical area of the study area in area units (square miles, square nautical miles, square feet, square kilometers, square meters.) Irrespective of the data units that are defined for the primary file, *CrimeStat* can convert to various area measurement units. These units are used in the nearest neighbor, Ripley's "K", nearest neighbor hierarchical clustering, risk-adjusted nearest neighbor hierarchical clustering, Stac, and K-means clustering routines. If no area units are defined, then *CrimeStat* will define a rectangle by the minimum and maximum X and Y coordinates.

Length of street network

Second, define the total length of the street network within the study area or an appropriate network (e.g., freeway system) in distance units (miles, nautical miles, feet, kilometers, meters.) The length of the street network is used in the linear nearest neighbor routine. Irrespective of the data units that are defined for the primary file, *CrimeStat* can convert to distance measurement units. The distance units should be in the same metric as the area units (e.g., miles and square miles/meters and square meters.)

Type of distance measurement

Third, define how distances are to be calculated. There are three choices:

1. **Direct** distance
2. **Indirect** (Manhattan) distance
3. **Network** distance

Direct

If direct distances are used, each distance is calculated as the shortest distance between two points. If the coordinates are spherical (i.e., latitude, longitude), then the shortest direct distance is a 'Great Circle' arc on a sphere. If the coordinates are projected, then the shortest direct distance is a straight line on a Euclidean plane.

Indirect

If indirect distances are used, each distance is calculated as the shortest distance between two points on a grid, that is with distance being constrained to the horizontal or vertical directions (i.e., not diagonal.) This is sometimes called 'Manhattan' metric. If the coordinates are spherical (i.e., latitude, longitude), then the shortest indirect distance is a

modified right angle on a spherical right triangle; see the documentation for more details. If the coordinates are projected, then the shortest indirect distance is the right angle of a right triangle on a two-dimensional plane

Network distance

If network distances are used, each distance is calculated as the shortest path between two points using the network. Alternatives to distance can be used including speed, travel time, or travel cost. Click on 'Network parameters' and identify a network file.

Type of network

Network files can *bi-directiona*l (e.g., a TIGER file) or *single directional* (e.g., a transportation modeling file). In a bi-directional file, travel can be in either direction. In a single directional file, travel is only in one direction. Specify the type of network used.

Network input file

The network file can either be a shape file (line, polyline, or polylineZ file) or another file, either dBase IV 'dbf', Microsoft Access 'mdb', Ascii 'dat', or an ODBC-compliant file. The default is a shape file. If the file is a shape file, the routine will know the locations of the nodes. For a dBase IV or other file, the X and Y coordinate variables of the end nodes must be defined. These are called the "From" node and the "End" node. An optional weight variable is allowed for all file types. The routine identifies nodes and segments and finds the shortest path. If there are one-way streets in a bi-directional file, the flag fields for the "From" and "To" nodes should be defined.

Network weight field

Normally, each segment in the network is not weighted. In this case, the routine calculates the shortest distance between two points using the distance of each segment. However, each segment can be weighted by travel time, speed or travel costs. If travel time is used for weighting the segment, the routine calculates the shortest time for any route between two points. If speed is used for weighting the segment, the routine converts this into travel time by dividing the distance by the speed. Finally, if travel cost is used for weighting the segment, the routine calculates the route with the smallest total travel cost. Specify the weighting field to be used and be sure to indicate the measurement units (distance, speed, travel time, or travel cost) at the bottom of the page. If there is no weighting field assigned, then the routine will calculate using distance.

From one-way flag and To one-way flag

One-way segments can be identified in a bi-directional file by a 'flag' field (it is not necessary in a single directional file). The 'flag' is a field for the end nodes of the segment with values of '0' and '1'. A '0' indicates that travel can pass through that node in either direction whereas a '1' indicates that travel can only pass from the other node of the same segment (i.e., travel cannot occur from another segment that is connected to the node). The default assumption is for travel to be allowed through each node (i.e., there is a '0' assumed for each node). For each one-way street, specify the flags for each end node. A '0' allows travel from any connecting segments whereas a '1' only allows travel from the other node of the same segment. Flag fields that are blank are assumed to allow travel to pass in either direction.

FromNode ID and ToNode ID

If the network is single directional, there are individual segments for each direction. Typically, two-way streets have two segments, one for each direction. On the other hand, one-way streets have only one segment. The FromNode ID and the ToNode ID identify from which end of the segment travel should occur. If no FromNode ID and ToNode ID is defined, the routine will chose the first segment of a pair that it finds, whether travel is in the right or wrong direction. To identify correctly travel direction, define the FromNode and ToNode ID fields.

Type of coordinate system

The type of coordinate system for the network file is the same as for the primary file.

Measurement unit

By default, the shortest path is in terms of distance. However, each segment can be weighted by travel time, travel speed, or travel cost.

1. For travel time, the units are minutes, hours, or unspecified cost units.

2. For speed, the units are miles per hour and kilometers per hour. In the case of speed as a weighting variable, it is automatically converted into travel time by dividing the distance of the segment by the speed, keeping units constant.

3. For travel cost, the units are undefined and the routine identifies routes by those with the smallest total cost.

Coded Functions

The CrimeStat Core Component library is composed of classes that are commonly used across various other libraries of Crimestat. These include:

1. Utilities
2. Files for Input and Output
3. Files for creating the reference grid (X/Y coordinates of lower-left and upper-right corners as well as number of columns or cell size).
4. Distance calculation and representation

The following is a description of the various public classes and enumeration types exposed by the DLL.

Library: CrimeStat Core Components.dll

Field	Description
enum DISTANCE_TYPE	Distance type. Valid Values: DISTANCE_TYPE_PROJECTED, // projected measurement. DISTANCE_TYPE_SPHERICAL, // longtitude, latitude (x,y) DISTANCE_TYPE_DIRECTIONAL, // directional (angles) DISTANCE_TYPE_UNKNOWN
enum MEASURE_TYPE	Type of distance measurement. Valid Values: MEASURE_TYPE_DIRECT, // Direct measurement. MEASURE_TYPE_MANHATTAN, // Manhattan measurement. MEASURE_TYPE_NETWORK, // Compute distance through a graph. MEASURE_TYPE_UNKNOWN
enum UNIT_TYPE	Data Unit type. Valid Values: UNIT_TYPE_DEGREES, // Decimal degress (Spherical only) UNIT_TYPE_FEET, // Feet (projected only)

	UNIT_TYPE_METERS, // Meters (projected only) UNIT_TYPE_MILES, // Miles (projected only) UNIT_TYPE_KILOMETERS, // Kilometers (projected only) UNIT_TYPE_NAUTICALMILES, // Nautical miles projected) UNIT_TYPE_UNKNOWN
enum COV_UNIT_TYPE	Coverage Unit type. Valid Values: COV_UNIT_TYPE_MILES, // (Squared) Miles COV_UNIT_TYPE_N_MILES, // (Squared) Nautical miles COV_UNIT_TYPE_FEET, // (Squared) Feet COV_UNIT_TYPE_KILOMETERS, // (Squared) Kilometers COV_UNIT_TYPE_METERS, // (Squared) Meters COV_UNIT_TYPE_UNKNOWN
enum TIME_UNIT_TYPE	Time Unit type. Valid Values: TIME_UNIT_TYPE_HOURS, // Hours TIME_UNIT_TYPE_DAYS, // Days TIME_UNIT_TYPE_WEEKS, // Weeks TIME_UNIT_TYPE_MONTHS, // Months TIME_UNIT_TYPE_YEARS, // Years TIME_UNIT_TYPE_UNKNOWN
Enum INPUT_FILE_TYPE	Valid Values: INPUT_FILE_TYPE_PRIMARY, INPUT_FILE_TYPE_SECONDARY, INPUT_FILE_TYPE_REFERENCE, INPUT_FILE_TYPE_JTC_CALIBRATE_INPUT, INPUT_FILE_TYPE_UNKNOWN
Enum INPUT_FILTER_TYPE	Valid Values: INPUT_FILTER_TYPE_BLANK, INPUT_FILTER_TYPE_NONE, INPUT_FILTER_TYPE_ZERO, INPUT_FILTER_TYPE_NEG_ONE, INPUT_FILTER_TYPE_ZERO_NEG_ONE, INPUT_FILTER_TYPE_ZERO_NEG_ONE_9999, INPUT_FILTER_TYPE_NULL

Primary and Secondary File Input

Notes: Included wrapper classes for secondary pointset data

Class: CInputParam

Synopsis

CInputParam is an object used to provide input parameters, routines for primary, secondary, reference point sets and routines for specifying JTC Calibration function generation input point sets.

Description

CInputParam has to be instantiated to create a CPointSet object for the input point set/s that is passed as an input argument to other library routines.

Fields

Field	Description
System::String^ m_sFileName	Name of the file for the Primary File. Primary filename should be specified using function AddPrimaryFile() as described below.
CPointSet m_pPrimaryPointset	CPointSet object representing data from primary file.
FILE_TYPE `m_fileType`	File type for the Primary File. Valid Values: FILE_TYPE_ASCII, // Normal, delimited files FILE_TYPE_XBASE, // xBase files FILE_TYPE_SHAPE, // Shape files
Utilities::DISTANCE_TYPE m_iDistanceType	Distance Type for the Primary File
Utilities::UNIT_TYPE m_iDataUnit	Data Unit Type for the Primary File
Utilities::TIME_UNIT_TYPE	Time Unit Type for the Primary File

m_iTimeUnit	
int m_nColumns	Number of columns in the Primary File. Only valid if value of `m_fileType` is FILE_TYPE_ASCII
System::String^ m_sDelimiter	Delimiter for the Primary File. Only valid if value of `m_fileType` is FILE_TYPE_ASCII
Bool m_bHasHeader;	Indicates if the Primary File has a header row. Only valid if value of `m_fileType` is FILE_TYPE_ASCII
double m_dCovArea	Coverage Area
Utilities::COV_UNIT_TYPE m_iCovAreaUnit	Coverage Area Unit
double m_dCovLength	Coverage Length
Utilities::COV_UNIT_TYPE m_iCovLengthUnit	Coverage Length Unit
Utilities::MEASURE_TYPE m_iMeasureType	Type of distance measurement
System::String^ m_sec_FileName	Name of the file for the Secondary File. Secondary filename should be specified using function AddSecondaryFile() as described below.
CPointSet m_pSecondaryPointSet	CPointSet object representing data from secondary file.
FILETYPE m_sfileType	FILETYPE for the Secondary File
System::String^ m_sec_ASCIIDelimiter	Delimiter for the Secondary File. Only valid if value of m_sfileType is FILE_TYPE_ASCII
int m_nSecColumns	Number of columns in the Secondary File. Only valid if value of `m_sfileType` is FILE_TYPE_ASCII
Bool m_bSecHasHeader;	Indicates if the Secondary File has a header row. Only valid if value of `m_sfileType` is FILE_TYPE_ASCII
System::String^ m_rFileName	Name of the file for the Reference File. Reference filename should be specified using function AddReferenceFile() as described

	below.
System::String^ m_sJTCCalibrationInputFileName	Name of the input file for the JTC Distance Calibration routine. Filename should be specified using function AddJTCCalibrationInputfile() as described below.
FILE_TYPE m_iJTCCalibrationInputFileType	FILETYPE for the input file for the JTC Distance Calibration routine
Utilities::DISTANCE_TYPE m_iJTCCalibrationInputDistanceType	Distance Type for the JTC Calibration input file
Utilities::UNIT_TYPE m_iJTCCalibrationInputDataUnit	Unit Type for JTC Distance Calibration input file
System::String^ m_sJTCCalibrationInputDelimiter	Delimiter for the JTC Distance Calibration input file. Valid only if value of m_ iJTCCalibrationInputFileType is FILE_TYPE_ASCII
Int m_nJTCCalibrationInputColumns	Number of columns in the JTC Distance Calibration input file. Only valid if value of m_ iJTCCalibrationInputFileType is FILE_TYPE_ASCII
Bool m_bJTCCalibrationInputHasHeader	Indicates if the JTC Distance Calibration input file has a header row. Only valid if value of m_ iJTCCalibrationInputFileType is FILE_TYPE_ASCII
CPointSet^ m_pJTCCalibrationInputPointSet	CPointSet object representing data for JTC Distance Calibration input file.

Methods

Method	Description
int AddPrimaryFile(String file_name)	Adds *file_name* file to represent PrimaryPointSet. Returns a *file id*. Files supported currently includes .dbf and .csv.
bool GetColumnName(int column_number)	Returns the Column Name of the column at valid position *column_numbe*ri. Returns null if *column_number* is greater than *m_nColumns* or less than 1.(Column numbers start from 1)
void SetX(int file_id, int col_index)	Maps *col_index* to X in primary pointset of file *file_id*.
void SetY(int file_id, int col_index)	Maps *col_index* to Y in primary pointset of file *file_id*.
void SetIntensity(int file_id, int col_index)	Maps *col_index* to Intensity in primary pointset of file *file_id* .
void SetWeight(int file_id, int col_index)	Maps *col_index* to Weight in primary pointset of file *file_id* .
void SetDistance(int file_id, int col_index)	Maps *col_index* to Distance in primary pointset of file *file_id* .
void SetDirection(int file_id, int col_index)	Maps *col_index* to Direction in primary pointset of file *file_id* .
void SetTime(int file_id, int col_index)	Maps *col_index* to Time in primary pointset of file *file_id* .
void UpdatePrimaryData()	Updates the Primary pointset with the columns currently mapped.
int AddSecondaryFile(String file_name)	Adds *file_name* file to represent SecondaryPointSet. Returns a *file id*. Files supported currently includes .dbf and .csv.
void SetX(int file_id, int col_index, INPUT_FILE_TYPE file_type,INPUT_FILTER_TYPE filter_type)	Maps *col_index* to X in pointset defined by *file_type* (INPUT_FILE_TYPE_PRIMARY for Primary and INPUT_FILE_TYPE_SECONDARY for secondary pointsets) of file *file_id* and a FILTER for missing values as filter_type.

void SetY(int file_id, int col_index, INPUT_FILE_TYPE file_type,INPUT_FILTER_TYPE filter_type)	Maps *col_index* to Y in pointset defined by *file_type* of file *file_id* and applies a filter *filter_type*.
void SetIntensity(int file_id, int col_index, INPUT_FILE_TYPE file_type,INPUT_FILTER_TYPE filter_type)	Maps *col_index* to Intensity in pointset defined by *file_type* of file *file_id* and applies a filter *filter_type*.
void SetWeight(int file_id, int col_index, INPUT_FILE_TYPE file_type,INPUT_FILTER_TYPE filter_type)	Maps *col_index* to Weight in pointset defined by *file_type* of file *file_id* and applies a filter *filter_type*.
void SetDistance(int file_id, int col_index, INPUT_FILE_TYPE file_type,INPUT_FILTER_TYPE filter_type)	Maps *col_index* to Distance in pointset defined by *file_type* of file *file_id* and applies a filter *filter_type*.
void SetTime(int file_id, int col_index, INPUT_FILE_TYPE file_type,INPUT_FILTER_TYPE filter_type)	Maps *col_index* to Time in pointset defined by *file_type* of file *file_id* and applies a filter *filter_type*.
void SetDirection(int file_id, int col_index, INPUT_FILE_TYPE file_type,INPUT_FILTER_TYPE filter_type)	Maps *col_index* to Direction in pointset defined by *file_type* of file *file_id* and applies a filter *filter_type*.
void UpdateSecondaryData()	Updates the Secondary pointset with the columns currently mapped.
int AddReferenceFile(System::String^ file_name)	Adds *file_name* file to represent Reference pointset. Returns a *file id*. Files supported currently includes .dbf and .csv
void CreateReferenceFile(double m_dGridLLX, double m_dGridLLY, double m_dGridURX, double m_dGridURY)	Used to create a Reference points dataset given the X and Y Co-ordinates of the Lower Left corner and the upper right corner of the grid. If this is specified, the input file if specified for the reference pointset by AddReferenceFile() will not be valid.
void SetNoOfColumns(unsigned int m_nNumOfColumn)	Used to specify the number of columns for generating the reference pointset
void SetCellSpacing(double m_dCellSpacing)	Used to specify the cell spacing for generating the reference pointset. If the cell spacing is set, it will be used to generate the reference pointset irrespective of whether number of

	columns is specified i.e., the value of number of columns will not be valid and considered during generation.
void UpdateReferenceData()	Updates the reference pointset with the columns currently mapped or generates the reference pointset if specified. The pointset data can be accessed using m_pReferencePointSet
Void AddJTCCalibrationInputfile(System::String^ file_name)	Adds *file_name* as the input file for the JTC Calibrate Distance routine. Files supported currently includes .dbf and .csv
void SetOriginX(int col_index, INPUT_FILTER_TYPE filter_type)	Maps column specified by *col_index* to X-Coordinate of the Origin in JTC Distance Calibration input file and applies a filter *filter_type*.
void SetOriginY(int col_index, INPUT_FILTER_TYPE filter_type)	Maps column specified by *col_index* to Y-Coordinate of the Origin in JTC Distance Calibration input file and applies a filter *filter_type*.
void SetDestinationX(int col_index, INPUT_FILTER_TYPE filter_type)	Maps column specified by *col_index* to X-Coordinate of the Destination in JTC Distance Calibration input file and applies a filter *filter_type*.
void SetDestinationY(int col_index, INPUT_FILTER_TYPE filter_type)	Maps column specified by *col_index* to Y-Coordinate of the Destination in JTC Distance Calibration input file and applies a filter *filter_type*.

Example 1 in Visual Basic: Defining the Primary File and Parameters

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
```

```
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

'Number of columns in the file
MsgBox(file.m_nColumns)

'Map columns
file.SetX(fileid, 10)
file.SetY(fileid, 9 )
file.SetIntensty(fileid, 8)
file.SetWeight(fileid, 7)
file.SetTime(fileid, 6)
file.SetDistance(fileid, 5)
file.SetDirection(fileid, 4)
'Update Data after mapping columns
file.UpdatePrimaryData()

'Use
'Use file.m_pPrimaryPointSet as argument to different modules.
'-- Example: Median Center
Dim mcmd As New CCalcMcmd
Dim savefilename As String
mcmd.PerformCalculation(file.m_pPrimaryPointset, 2, savefilename)
MsgBox(mcmd.GetResult())
```

Example 2 in Visual Basic: Defining the Primary and Secondary Files

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
```

```
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Secondary File
file.m_sfileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer
fileid = file.AddSecondaryFile("E:/crime.dbf")

'Number of columns in the file
MsgBox(file.m_nColumns)

'Map columns
file.SetX(fileid, 10, INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)
file.SetY(fileid, 9, INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL )
file.SetIntensty(fileid, 8, INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)
file.SetWeight(fileid, 7, INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)
file.SetTime(fileid, 6, INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)
file.SetDistance(fileid, 5, INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)
file.SetDirection(fileid, 4, INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)
'Update Data after mapping columns
file.UpdateSecondaryData()

'Use
'Use file.m_pSecondaryPointSet as argument to different modules.
'-- Example: Median Center
Dim mcmd As New CCalcMcmd
Dim savefilename As String
mcmd.PerformCalculation(file.m_pSecondaryPointSet, 8, savefilename)
MsgBox(mcmd.GetResult())
```

Example 3 in Visual Basic: Defining a Reference File

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE.FILE_TYPE_XBASE
fileid = file.AddPrimaryFile(file.m_sFileName)

file.m_sfileType = FILE_TYPE.FILE_TYPE_XBASE
fileid_s = file.AddSecondaryFile(file.m_sec_FileName)

file.CreateReferenceFile(-76.9, 39.2, -76.32, 39.73)
file.SetCellSpacing(0.01)

file.SetY(fileid, cboxLat.SelectedIndex)
file.SetX(fileid, cboxLon.SelectedIndex)

file.SetY(fileid_s, cBoxLatSec.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)
file.SetX(fileid_s, cBoxLonSec.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)

'Update Data after mapping columns
file.UpdatePrimaryData()
file.UpdateSecondaryData()
file.UpdateReferenceData()

Dim kernelDenParam As New CCalcKernelDensityParams
```

```
Dim RNNHParam As New CRnnhParams
Dim calcRNNH As New CCalcClusterRNNH

kernelDenParam.setKernelSingleUnit(Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_
MILES)
'kernelDenParam.setKernelSingleFile(True)

kernelDenParam.setKernelSingleMethod(Interpolation.KERNEL_DENSITY_CALC.KE
RNEL_DENSITY_CALC_NORMAL)

kernelDenParam.setKernelSingleBandwidth(Interpolation.KERNEL_BANDWIDTH.KE
RNEL_BANDWIDTH_ADAPTIVE)
kernelDenParam.setKernelSingleMinSample(100)
kernelDenParam.setKernelSingleInterval(1)

kernelDenParam.setCalculationMethod(Interpolation.KERNEL_SINGLE_CALC.KERN
EL_SINGLE_CALC_REL_DENSITY)
kernelDenParam.setKernelSingleWeight(False)
kernelDenParam.setKernelSingleIntensity(False)

kernelDenParam.setKernelSingleSource(Interpolation.KERNEL_SOURCE.KERNEL_S
OURCE_SECONDARY)
RNNHParam.m_nKernelSensitivity = 50

calcRNNH.initialize(file.m_pPrimaryPointSet, 10,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES, 0.5, 1,
file.m_pSecondaryPointSet, RNNHParam, kernelDenParam)
System.Console.WriteLine("Done Initialization!!")
calcRNNH.PerformCalculations()
System.Console.WriteLine("Done calculations!! {0:D}", calcRNNH.GetRowCount())
```

Example 4 in Visual Basic: Defining an Input file for JTC Distance Calibration Routine

---

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_sJTCCalibrationInputFileName = Me.txtFileName.Text
```

```
file.m_iJTCCalibrationInputFileType = FILE_TYPE.FILE_TYPE_XBASE
file.m_iJTCCalibrationInputDistanceType =
Utilities.DISTANCE_TYPE.DISTANCE_TYPE_SPHERICAL
file.m_iJTCCalibrationInputDataUnit = Utilities.UNIT_TYPE.UNIT_TYPE_DEGREES

file.AddJTCCalibrationInputfile(Me.txtFileName.Text)
file.SetOriginX(cboxX.SelectedIndex,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
file.SetOriginY(cboxY.SelectedIndex,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
file.SetDestinationX(cboxDX.SelectedIndex,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
file.SetDestinationY(cboxDY.SelectedIndex,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

'Update Data after mapping columns
file.UpdateJTCCalibrationInputData()
Dim jtc As New CJtcCalibrateFunction
Dim kernelDen As New
CCalcKernelDensityParamskernelDen.setKernelSingleMethod(Interpolation.KERNEL_
DENSITY_CALC.KERNEL_DENSITY_CALC_NORMAL)kernelDen.setKernelSingle
Bandwidth(Interpolation.KERNEL_BANDWIDTH.KERNEL_BANDWIDTH_FIXED_I
NTERVAL)
kernelDen.setKernelSingleMinSample(100)
kernelDen.setKernelSingleInterval(0.25)

kernelDen.setKernelSingleUnit(Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILE
S)
kernelDen.setKernelIntervalBinsNumber(100)
kernelDen.setKernelSingleOutUnit(Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MI
LES)kernelDen.setCalculationMethod(Interpolation.KERNEL_SINGLE_CALC.KERNE
L_SINGLE_CALC_REL_DENSITY)

jtc.Initialize(file.m_pJTCCalibrationInputPointSet, kernelDen)
```

Example 5 in Visual Basic: Reading a Shape file

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_DEGREE
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_SHAPE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/ theftfromautos.shp")

'Number of columns in the file
MsgBox(file.m_nColumns)

'Map columns
file.SetX(fileid, 0)
file.SetY(fileid, 1 )
file.UpdatePrimaryData()
```

Example 6 in Visual Basic: Reading an ASCII file

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_DEGREE
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT
'Add Primary File
```

```
file.m_fileType = FILE_TYPE::FILE_TYPE_ASCII
file.m_nColumns = 5

file.m_sDelimiter = vbTab

file.m_bHasHeader = True

Dim fileid As Integer fileid = file.AddPrimaryFile("E:/ Asciifile.txt")

'Map columns
file.SetX(fileid, 0)
file.SetY(fileid, 1 )
file.UpdatePrimaryData()
```

File Output

For file output, the library provides access to the following:

1. Global Variables that determine the type of output file parameter to be passed.
2. File Utils class
3. dBase File class

Fields

Field	Value	Description
SAVEAS_TYPE_NONE	0	None
SAVEAS_TYPE_ASCII	1	Save as an ASCII file
SAVEAS_TYPE_SHP	2	Save as a Shape File
SAVEAS_TYPE_MIF	8	Save as MapInfo (.mif) File
SAVEAS_TYPE_KML	10	Save as a Google Earth (.kml) file
SAVEAS_TYPE_DBF	64	Save as dBase (.dbf) File
ENUM XB_FT		XB_FT_STRING, XB_FT_INTEGER, XB_FT_DOUBLE, XB_FT_INVALID

Class: CFileUtilsPub

Synopsis

CFileUtilsPub is an object used to provide functionalities for processing File paths. The most commonly used functionality is the Split Path functionality.

Description

CFileUtilsPub has to be instantiated to access functions such as split path that can used insert prefix , add extensions to file names etc.

Fields

Field	Description
CFileUtils* m_pFileutils	Pointer to the underlying native FileUtils class
System::String ^sDir;	The directory name of the path that is specified as an input.
System::String ^sName;	The actual name of the file.
System::String ^sExt;	The file extension.

Class: CFileXBaseWrap

Synopsis

CFileXBaseWrap is an object used to create an instance of a DBF file and is passed as a parameter to any of the Calling Routines. This class is useful for producing DBF file outputs.

Description

CFileXBaseWrap has to be instantiated to access functions such as Create to create an instance of a DBF file.

Fields

Field	Description
CFileXBase *xbase	Pointer to the underlying native CFileXBase class
System::String ^m_sfieldname	The name of the field that is written to the DBF file

Spatial Distribution Statistics Library

The spatial distribution library provides statistics that describe the overall spatial distribution. These are sometimes called centrographic, global, or first-order spatial statistics. There are five routines for describing the spatial distribution and six routines for describing spatial autocorrelation. An intensity variable and a weighting variable can be used for the first five routines, though it is not required. An intensity variable is required for the six spatial autocorrelation routines; a weighting variable can also be used for the spatial autocorrelation indices. See chapter 4 in the CrimeStat manual for more detail.

The Spatial Distribution library is composed of classes for the following modules

1. Mean Center and Standard Distance Deviation
2. Center of Minimum Distance
3. Median Center
4. Directional Mean and Variance
5. Convex Hull
6. Spatial Autocorrelation
    a. Moran's I Statistic
    b. Geary's C Statistic
    c. Getis-Ord General G Statistic
    d. Moran Correlogram
    e. Geary Correlogram
    f. Getis-Ord Correlogram

The following is a brief description of the public functions and members of the Spatial Distribution Statistic library.

Library: Spatial Description.dll

Prerequisites: Crimestat Core Components.dll

Mean Center and Standard Distance Deviation

Class: CCalcMcsd

Synopsis

The mean center and standard distance define the arithmetic mean location and the degree of dispersion of the distribution. Estimates of the geometric and harmonic means are also produced.

Description

The Mcsd routine calculates 9 statistics:

1. The sample size
2. The minimum X and Y values
3. The maximum X and Y values
4. The X and Y coordinates of the mean center
5. The standard deviation of the X and Y coordinates
6. The X and Y coordinates of the geometric mean
7. The X and Y coordinates of the harmonic mean
8. The standard distance deviation, in meters, feet and miles. This is the standard deviation of the distance of each point from the mean center.
9. The circle area defined by the standard distance deviation, in square meters, square feet and square miles.

The routine can output three different means and two measures of dispersion in *ArcGIS/ArcView* shape or *MapInfo* MIF file formats. The user must provide a file name. The five graphical objects are saved are distinguished by a prefix placed before the file name – Mean center (Mc), Geometric mean (Gm), Harmonic mean (Hm), standard deviation of X and Y coordinates (Xyd), and Standard distance deviation (Sdd). See Chapter 4 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
bool PerformCalculations(CPointSet^ m_pPointSet,int m_iFileType, System::String^ path)	Calculates the mean center and standard distance. Returns 'true' on success, 'false' on failure. Parameters m_pPointSet – Input File m_iFileType –None(0) or Shape(2) or MIF(8) or KML(10) path – file path for the output file.
String GetResult()	Returns the calculated result.
Double GetMeanX()	Returns the mean of X coordinate.
Double GetMeanY()	Returns the mean of Y coordinate.
Double GetMinX()	Returns the minimum of X coordinate.
Double GetMinY()	Returns the minimum of Y coordinate.
Double GetMaxX()	Returns the maximum of X coordinate.
Double GetMaxY()	Returns the maximum of Y coordinate.
Double GetStdDevX()	Returns the standard deviation of X coordinate.
Double GetStdDevY()	Returns the standard deviation of Y coordinate.
Double GetGeometricMeanX()	Returns the geometric mean of X coordinate.
Double GetGeometricMeanY()	Returns the geometric mean of Y coordinate.
Double GetHarmonicMeanX()	Returns the harmonic mean of X coordinate.
Double GetHarmonicMeanY()	Returns the harmonic mean of Y coordinate.
Double GetStdDis()	Returns the standard distance deviation. This is the standard deviation of the distance of each point from the mean center.
void setMIFHeader (System::String ^header)	Set the header if output file type is MIF

Example 1 in Visual Basic

```
'Create mcsd object
Dim mcsd As New CCalcMcsd
Dim savefilename As String
'Calculate mcsd
mcsd.setMIFHeader("CoordSys Earth Projection 8, 79, ""m"", -2, 49, 0.9996012717,
400000, -100000)
mcsd.PerformCalculations(file.m_pPrimaryPointSet,8,savefilename)
'Get Result
mcsd.GetResult()
```

Example 2 in Visual Basic

```
Public Class CrimeStat

    Private Sub btnLoadFile_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnLoadFile.Click

        file.m_sFileName = Me.txtFileName.Text
        file.m_iDistanceType =
Utilities.DISTANCE_TYPE.DISTANCE_TYPE_SPHERICAL
        file.m_iDataUnit = Utilities.UNIT_TYPE.UNIT_TYPE_DEGREES

        file.m_iTimeUnit = Utilities.TIME_UNIT_TYPE.TIME_UNIT_TYPE_DAYS
        file.m_iMeasureType = Utilities.MEASURE_TYPE.MEASURE_TYPE_DIRECT

        file.m_fileType = FILE_TYPE.FILE_TYPE_XBASE
        fileid = file.AddPrimaryFile(file.m_sFileName)


        file.m_iMiscellaneousInputFileType = FILE_TYPE.FILE_TYPE_XBASE

        Dim i As Integer
```

```
        Dim columns As String
        columns = "Column Index and Name:" + vbCr
        For i = 0 To file.m_nColumns - 1
            columns = columns + i.ToString + ". " + file.GetColumnName(i) + vbCr
            Me.cboxLat.Items.Add(file.GetColumnName(i))
            Me.cboxLon.Items.Add(file.GetColumnName(i))
            Me.cboxInt.Items.Add(file.GetColumnName(i))
        Next
    End Sub

    Private Sub btnCalc_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCalc.Click

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
System.Console.WriteLine("{0:D},{1:D}", cboxLat.SelectedIndex,
cboxLon.SelectedIndex)

file.UpdatePrimaryData()

Dim mcsd As New CCalcMcsd

If (mcsd.PerformCalculations(file.m_pPrimaryPointSet, 2, Me.Text_Path.Text)) Then
            System.Console.WriteLine("Execution starts")
End If
System.Console.WriteLine("Execution COMPLETED")
        Me.Close()
    End Sub

Private Sub Label6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Label6.Click

End Sub

Private Sub Label12_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
```

End Sub

Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)

End Sub

End Class

Standard Deviational Ellipse

Class: CCalcSDE

Synopsis

The standard deviational ellipse defines both the dispersion and the direction (orientation) of the distribution.

Description

The standard deviational ellipse routine calculates a rotated axis through the points and then calculates two standard deviations along the new axes. The Sde routine calculates 9 statistics:

1. The sample size
2. The clockwise angle of Y-axis rotation in degrees
3. The ratio of the long to the short axis after rotation
4. The standard deviation along the new X and Y axes in meters, feet and miles
5. The X and Y axes lengths in meters, feet and miles
6. The area of the ellipse defined by these axes in square meters, square feet and square miles
7. The standard deviation along the X and Y axes in meters, feet and miles for a 2X standard deviational ellipse
8. The X and Y axes lengths in meters, feet and miles for a 2X standard deviational ellipse
9. The area of the 2X ellipse defined by these axes in square meters, square feet and square miles.

The routine can output the standard deviational ellipse in *ArcGIS/ArcView* shape or *MapInfo* MIF file formats. The user must provide a file name. Two graphical objects are saved and are distinguished by a prefix placed before the file name: a one standard deviational ellipse (SDE) and a two standard deviational ellipse (2SDE). See Chapter 4 of the CrimeStat manual for more information.

Fields

Field	Description
Double m_dEllipsePointArray[]	Single array of successive points on the periphery of the ellipse with one-standard deviation. Each point is stored in two consecutive locations. For example, (m_dEllipsePointArray[0],m_dEllipsePointArray[1]) refers to the (X,Y) pair of one point.
Double m_dEllipse2PointArray[]	single array of successive points on the periphery of the ellipse with two-standard deviation. ///Each point is stored in two consecutive locations. ///For example, (m_dEllipse2PointArray[0],m_dEllipse2PointArray[1]) refers to the (X,Y) pair of one point.

Methods

Method	Description
bool PerformCalculations(CPointSet^ m_pPointSet,int m_iFileType, System::String^ path t)	Calculate standard deviational ellipse and 2X standard deviational ellipse. Returns 'true' on success, 'false' on failure. Parameters m_pPointSet – Input File m_iFileType –None(0) or Shape(2) or MIF(8) or KML(10) path – file path for the output file.
String GetResult()	Returns calculated result.
Double GetMeanX()	Returns mean of X coordinates.
Double GetMeanY()	Returns mean of Y coordinates.
Double GetStdXAxis()	Returns standard deviation along the new X axis for the standard deviational ellipse.
Double GetStdYAxis()	Returns standard deviation along the new Y axis for the standard deviational ellipse.
Double GetStd2XAxis()	Returns standard deviation along the new X axis of the 2X standard deviational ellipse.
Double GetStd2YAxis()	Returns standard deviation along the new Y

	axis of the 2Y standard deviational ellipse.
Double GetRotateAngle()	Returns clockwise angle of Y-axis rotation in degrees.
Double GetAxesRatio()	Returns ratio of the long to the short axis after rotation.
Double GetEllipseArea()	Returns the area of the standard deviational ellipse.
Double GetEllipse2Area()	Returns the area of the 2X standard deviational ellipse.

Example 1 in Visual Basic

```
'Create SDE object
Dim sde As New CCalcSde
Dim savefilename As String
'Calculate SDE
sde.PerformCalculations(file.m_pPrimaryPointSet, 10, savefilname)
'Get result
MsgBox(sde.GetResult())
```

Graphics Output

Methods Modified

Method	Description
void Compute(CPointSet^ m_pPointSet,int m_iFileType, System::String^ path)	Parameters m_pPointSet – Input File m_iFileType – Shape or MIF path – file path for the target shape file.

void setMIFHeader (System::String ^header)	Set the header if output file type is MIF

Example 2 in Visual Basic

Public Class CrimeStat

Private Sub btnLoadFile_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnLoadFile.Click

```
file.m_sFileName = Me.txtFileName.Text
file.m_iDistanceType = Utilities.DISTANCE_TYPE.DISTANCE_TYPE_SPHERICAL
file.m_iDataUnit = Utilities.UNIT_TYPE.UNIT_TYPE_DEGREES
file.m_iTimeUnit = Utilities.TIME_UNIT_TYPE.TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities.MEASURE_TYPE.MEASURE_TYPE_DIRECT
file.m_fileType = FILE_TYPE.FILE_TYPE_XBASE
fileid = file.AddPrimaryFile(file.m_sFileName)
file.m_iMiscellaneousInputFileType = FILE_TYPE.FILE_TYPE_XBASE
Dim i As Integer
Dim columns As String
columns = "Column Index and Name:" + vbCr
For i = 0 To file.m_nColumns - 1
        columns = columns + i.ToString + ". " + file.GetColumnName(i) + vbCr
        Me.cboxLat.Items.Add(file.GetColumnName(i))
        Me.cboxLon.Items.Add(file.GetColumnName(i))
        Me.cboxInt.Items.Add(file.GetColumnName(i))
Next
```

End Sub

Private Sub btnCalc_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCalc.Click

```
file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
System.Console.WriteLine("{0:D},{1:D}", cboxLat.SelectedIndex,
cboxLon.SelectedIndex)
file.UpdatePrimaryData()
Dim sde As New CCalcSde
If (sde.PerformCalculations(file.m_pPrimaryPointSet, 2, Me.Text_Path.Text)) Then
        System.Console.WriteLine("Execution starts")
End If
System.Console.WriteLine("Execution COMPLETED")
Me.Close()
End Sub

End Class
```

Center of Minimum Distance

Class: CCalcMcmd

Synopsis

Calculates the Center of Minimum Distance, which is the point at which the sum of the distances to all measured points is minimized.

Description

The center of minimum distance defines the point at which the distance to all other points is at a minimum. Unfortunately, it is sometimes also called the 'median center', but not to be confused with the median center that is the intersection of the median of X and the median of Y.

The Mcmd routine outputs 5 statistics:

1. The sample size
2. The mean of the X and Y coordinates
3. The number of iterations required to identify a center of minimum distance
4. The degree of error (tolerance) for stopping the iterations
5. The X and Y coordinates which define the center of minimum distance

The routine can output the center of minimum distance in *ArcGIS/ArcView* shape or *MapInfo* MIF file formats. The user must provide a file name. A prefix (Mcmd) is placed before the file name. See Chapter 4 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
bool PerformCalculations(CPointSet^ m_pPointSet,int m_iFileType, System::String^ path)	Calculates the center of minimum distance in CPointSet. Returns 'true' on success, 'false' on failure Parameters m_pPointSet – Input File m_iFileType –None(0) or Shape(2) or MIF(8) or KML(10) path – file path for the output file.
String GetResult()	Returns the calculated result.
Double GetMeanX()	Returns the mean of X coordinates.
Double GetMeanY()	Returns the mean of Y coordinates.
Double GetTolerance()	Returns the degree of error (tolerance) for stopping the iterations.
Double GetDeltaX()	Returns the delta of X coordinates.
Double GetDeltaY()	Returns the delta of Y coordinates.
Double GetMinDistanceCenterX()	Returns X coordinate which defines the center of minimum distance.
Double GetMinDistanceCenterY()	Returns Y coordinate which defines the center of minimum distance.
void setMIFHeader (System::String ^header)	Set the header if output file type is MIF

Example1 in Visual Basic

```
'Create object
Dim mcmd As New CCalcMcmd
Dim savefilename As String
'Calcualte MCMD
mcmd.PerformCalculations(file.m_pPrimaryPointSet, 8, savefilename)
'Retrieve result
```

```
MsgBox(mcmd.GetResult())
MsgBox("Median Center X: " + mcmd.GetMedianCenterX.ToString() + " " + "Median
Center Y: " + mcmd.GetMedianCenterY.ToString())
```

Example 2 in Visual Basic

```
Public Class CrimeStat
Private Sub btnLoadFile_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnLoadFile.Click
file.m_sFileName = Me.txtFileName.Text
file.m_iDistanceType = Utilities.DISTANCE_TYPE.DISTANCE_TYPE_SPHERICAL
file.m_iDataUnit = Utilities.UNIT_TYPE.UNIT_TYPE_DEGREES
file.m_iTimeUnit = Utilities.TIME_UNIT_TYPE.TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities.MEASURE_TYPE.MEASURE_TYPE_DIRECT
file.m_fileType = FILE_TYPE.FILE_TYPE_XBASE
fileid = file.AddPrimaryFile(file.m_sFileName)
file.m_iMiscellaneousInputFileType = FILE_TYPE.FILE_TYPE_XBASE
Dim i As Integer
Dim columns As String
columns = "Column Index and Name:" + vbCr
For i = 0 To file.m_nColumns - 1
        columns = columns + i.ToString + ". " + file.GetColumnName(i) + vbCr
        Me.cboxLat.Items.Add(file.GetColumnName(i))
        Me.cboxLon.Items.Add(file.GetColumnName(i))
        Me.cboxInt.Items.Add(file.GetColumnName(i))
Next
End Sub

Private Sub btnCalc_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCalc.Click
file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
System.Console.WriteLine("{0:D},{1:D}", cboxLat.SelectedIndex,
cboxLon.SelectedIndex)
```

```
file.UpdatePrimaryData()
Dim mcmd As New CCalcMcmd
If (mcmd.PerformCalculations(file.m_pPrimaryPointSet, 2, Me.Text_Path.Text)) Then
        System.Console.WriteLine("Execution starts")
End If
System.Console.WriteLine("Execution COMPLETED")
Me.Close()
End Sub
End Class
```

Median Center

Class: CCalcMedianCenter

Synopsis

Calculates the point at which the median of the X coordinates intersects the median of the Y coordinates.

Description

The median center is the point at which the median of the X coordinates intersects the median of the Y coordinates. The MdnCntr routine outputs 3 statistics:

1. The sample size
2. The median value of the X coordinate
3. The median value of the Y coordinate

The routine can output the median center in *ArcGIS/ArcView* shape or *MapInfo* MIF file formats. The user must provide a file name. A prefix (MdnCntr) is placed before the file name. See Chapter 4 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
bool PerformCalculations(CPointSet^ m_pPointSet,int m_iFileType, System::String^ path)	Calculates the median center. Returns 'true' on success, 'false' on failure. Parameters

	m_pPointSet – Input File m_iFileType –None(0) or Shape(2) or MIF(8) or KML(10) path – file path for the output file.
String GetFormattedResultAsText()	Returns the calculated result.
Double GetMedianX()	Returns the median of X coordinates.
Double GetMedianY()	Returns the median of Y coordinates.
void setMIFHeader (System::String ^header)	Set the header if output file type is MIF

Example in Visual Basic

```
'Create mdn_cntr object
Dim mdn_cntr As New CCalcMedianCenter
Dim savefilename As String
'Calculate Median Center
mdn_cntr.PerformCalculations(file.m_pPrimaryPointSet, 10, savefilename)
'Get Result
MsgBox(mdn_cntr.GetResult())
```

Directional Mean

Class: CCalcDMean

Synopsis

The angular mean and variance are properties of angular measurements. The angular mean is an angle defined as a bearing from true North: 0 degrees. The directional variance is a relative indicator varying from 0 (no variance) to 1 (maximal variance.) Both the angular mean and the directional variance can be calculated either through angular (directional) coordinates or through X and Y coordinates. The triangulated mean is the intersection of the two mean angles, one from the lower-left corner of the study area (the minimum X and Y values) and the other from the upper-right corner of the study area (the maximum X and Y values).

Description

The directional mean routine will output five statistics:

1. The sample size
2. The unweighted mean angle
3. The weighted mean angle
4. The unweighted circular variance
5. The weighted circular variance.

The routine can output the directional mean and triangulated mean in *ArcGIS/ArcView* shape or *MapInfo* MIF file formats. The user must provide a file name. Three graphical objects are saved and distinguished by a prefix placed before the file name – the unweighted directional mean - the intersection of the mean angle and the mean distance (DM), the unweighted triangulated mean (TM), and the weighted triangulated mean (TMWT). See Chapter 4 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
void Initialize(CPointSet^ primary, CPointSet^ secondary)	Initialize with primary and secondary pointsets.
bool PerformCalculations(CPointSet^ m_pPointSet,int m_iFileType, System::String^ path)	Calculates the angular mean and variance are properties of angular measurements. Returns 'true' on success, 'false' on failure Parameters m_pPointSet – Input File m_iFileType –None(0) or Shape(2) or MIF(8) or KML(10) path – file path for the output file.
String GetResult()	Returns the calculated result.
Double GetMean()	Returns unweighted mean angle.
Double GetMeanWeighted()	Returns weighted mean angle.
Double GetVariance()	Returns unweighted circular variance.
Double GetVarianceWeighted()	Returns weighted circular variance.
Double GetMeanRadius()	Returns mean distance.
Double GetMeanX()	Returns the mean of X coordinate.
Double GetMeanY()	Returns the mean of Y coordinate.
Double GetMeanXWeighted()	Returns weighted mean of X coordinate.
Double GetMeanYWeighted()	Returns weighted mean of Y coordinate.
Double GetMeanXTriangulated()	Returns triangulated mean of X coordinate.
Double GetMeanYTriangulated()	Returns triangulated mean of Y coordinate.
Double GetMeanXWeightedTriangulated()	Returns weighted triangulated mean of X coordinate.
Double GetMeanYWeightedTriangulated()	Returns weighted triangulated mean of

	Y coordinate.
void setMIFHeader (System::String ^header)	Set the header if output file type is MIF

Example 1 in Visual Basic

```
'Declare dmean
Dim dmean as New CCalcDMean
Dim savefilename As String
'Initialize
dmean.Initialize(CPointSet^ primary, CPointSet^ secondary)
'Perform Calculations
dmean.PerformCalculations(dmean.m_pOwnPointSet, 2, savefilename)
'Get result
dmean.GetResult()
```

Example 2 in Visual Basic

```
Public Class CrimeStat
Private Sub btnLoadFile_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnLoadFile.Click
file.m_sFileName = Me.txtFileName.Text
file.m_sec_FileName = Me.txtSecFileName.Text
file.m_iDistanceType = Utilities.DISTANCE_TYPE.DISTANCE_TYPE_SPHERICAL
file.m_iDataUnit = Utilities.UNIT_TYPE.UNIT_TYPE_DEGREES
file.m_iTimeUnit = Utilities.TIME_UNIT_TYPE.TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities.MEASURE_TYPE.MEASURE_TYPE_DIRECT
file.m_fileType = FILE_TYPE.FILE_TYPE_XBASE
fileid = file.AddPrimaryFile(file.m_sFileName)
file.m_sfileType = FILE_TYPE.FILE_TYPE_XBASE
fileid_s = file.AddSecondaryFile(file.m_sec_FileName)
file.m_iMiscellaneousInputFileType = FILE_TYPE.FILE_TYPE_XBASE
Dim i As Integer
Dim columns As String
columns = "Column Index and Name:" + vbCr
```

```
For i = 0 To file.m_nColumns - 1
        columns = columns + i.ToString + ". " + file.GetColumnName(i) + vbCr
        Me.cboxLat.Items.Add(file.GetColumnName(i))
        Me.cboxLon.Items.Add(file.GetColumnName(i))
        Me.cboxInt.Items.Add(file.GetColumnName(i))
Next
For i = 0 To file.m_snColumns - 1
        columns = columns + i.ToString + ". " + file.GetColumnName(i,
        INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY) + vbCr
        Me.cBoxLatSec.Items.Add(file.GetColumnName(i,
        INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY))
        Me.cBoxLonSec.Items.Add(file.GetColumnName(i,
        INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY))
        Me.cboxIntSec.Items.Add(file.GetColumnName(i,
        INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY))
Next
End Sub

Private Sub btnCalc_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCalc.Click
file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
System.Console.WriteLine("{0:D},{1:D}", cboxLat.SelectedIndex,
cboxLon.SelectedIndex)
file.UpdatePrimaryData()
Dim dmean As New CCalcDMean
dmean.Initialize(file.m_pPrimaryPointSet, file.m_pSecondaryPointSet)
If (dmean.PerformCalculations(dmean.m_pOwnPointSet, 2, Me.Text_Path.Text)) Then
        System.Console.WriteLine("Execution starts")
End If
System.Console.WriteLine("Execution COMPLETED")
Me.Close()
End Sub

Private Sub Label6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Label6.Click
```

End Sub

Private Sub Label12_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
End Sub

Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
End Sub

Private Sub CrimeStat_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
End Sub

Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cBoxLatSec.SelectedIndexChanged
End Sub
End Class

Convex Hull

Class: CCalcConvexHull

Synopsis

The convex hull draws a polygon around the outer points of the distribution. It is useful for viewing the shape of the distribution.

Description

The routine outputs three statistics:

1. The sample size
2. The number of points in the convex hull
3. The X and Y coordinates for each of the points in the convex hull

The routine can output the convex hull in *ArcGIS/ArcView* shape or *MapInfo* MIF file formats. The user must provide a file name. A prefix (CHull) is placed before the file name. See Chapter 4 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
Initialize()	Initialize Convex Hull
CConvexHullResult PerformCalculations(bool% error, CPointSet^ m_pPointSet,int m_iFileType, System::String^ path))	Calculates the convex hull. Returns points on the convex hull into CConvexHullResult. Success message is returned in error.

	Parameters m_pPointSet – Input File m_iFileType –None(0) or Shape(2) or MIF(8) or KML(10) path – file path for the output file.
String GetResult()	Returns the calculated result.
void setMIFHeader (System::String ^header)	Set the header if output file type is MIF

Example 1 in Visual Basic

```
'Create convexhull result object
Dim chresult As New CConvexHullResult()
'Create convexhull object
Dim chull As New CCalcConvexHull
Dim savefilename As String
chull.initialize()
'Calculate ConvexHull
chresult = chull.PerformCalculations(False, file.m_pPrimaryPointSet, 10, savefilename)
'Get result
MsgBox(chull.GetResult())
```

Example 2 in Visual Basic

```
Public Class CrimeStat

Private Sub btnLoadFile_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnLoadFile.Click
file.m_sFileName = Me.txtFileName.Text
file.m_iDistanceType = Utilities.DISTANCE_TYPE.DISTANCE_TYPE_SPHERICAL
file.m_iDataUnit = Utilities.UNIT_TYPE.UNIT_TYPE_DEGREES
file.m_iTimeUnit = Utilities.TIME_UNIT_TYPE.TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities.MEASURE_TYPE.MEASURE_TYPE_DIRECT
file.m_fileType = FILE_TYPE.FILE_TYPE_XBASE
fileid = file.AddPrimaryFile(file.m_sFileName)
file.m_iMiscellaneousInputFileType = FILE_TYPE.FILE_TYPE_XBASE
Dim i As Integer
Dim columns As String
columns = "Column Index and Name:" + vbCr
For i = 0 To file.m_nColumns - 1
        columns = columns + i.ToString + ". " + file.GetColumnName(i) + vbCr
        Me.cboxLat.Items.Add(file.GetColumnName(i))
        Me.cboxLon.Items.Add(file.GetColumnName(i))
        Me.cboxInt.Items.Add(file.GetColumnName(i))
```

```
Next
End Sub

Private Sub btnCalc_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCalc.Click
file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
System.Console.WriteLine("{0:D},{1:D}", cboxLat.SelectedIndex,
cboxLon.SelectedIndex)
file.UpdatePrimaryData()
Dim chull As New CCalcConvexHull
chull.initialize()
chull.PerformCalculations(False, file.m_pPrimaryPointSet, 2, Me.Text_Path.Text)
System.Console.WriteLine("Execution starts")
System.Console.WriteLine("Execution COMPLETED")
Me.Close()
End Sub

Private Sub Label6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Label6.Click
End Sub

End Class
```

Moran's “I”

Class: CCalcSapd

Synopsis

Moran's "I" statistic is the classic indicator of spatial autocorrelation. It is an index of covariation between different point locations and is similar to a product moment correlation coefficient, typically varying from –1 to +1.

Description

The Moran’s “I” routine calculates 6 statistics:

1. The sample size
2. Moran's "I"
3. The spatially random (expected) "I"
4. The standard deviation of "I"
5. A significance test of "I" under the assumption of normality (Z-test)
6. A significance test of "I" under the assumption of randomization (Z-test)
   Values of I greater than the expected I indicate clustering while values of I less than the expected I indicate dispersion. The significance test indicates whether these differences are greater than what would be expected by chance.

If checked, small distances are adjusted so that the maximum weighting is 1 (see documentation for details.) This ensures that “I” won't become excessively large for points that are close together. The default value is no adjustment. The statistic requires an intensity variable in the primary file.
The tabular results can be saved to a '.dbf' file. The user must provide a file name. See Chapter 4 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
void Initialize (CPointSet^ pPointSet, CPointSet^ pSecPointSet, bool isadjsmall)	Initializes the object for Moran's I calculation. This method takes in a primary point set , a secondary pointset as parameters and a boolean flag which is set to true for small distance adjustment. This method must be called before invoking any other methods of this class.
bool PerformCalculations ()	This method is the main method that computes the statistics. This method should be invoked only after a call to initialize method above.
System::String^ GetResult()	Returns formatted Moran's I statistics as a String
double GetMoran()	Returns the Moran's I statistic value
double GetExpected()	Returns the value of the spatially expected(random) "I"
double GetNormSig()	Returns the value of the Standard error of "I"
double GetNormZScore()	Returns the value of the Normality Significance or Z Score
double GetNormZPValueOnetail ()	Returns the Normality significance p-value for one tail
double GetNormZPValueTwotail ()	Returns the Normality significance p-value for two tail
double GetRandZScore()	Returns the value for Randomization significance
double GetRandZPValueOnetail ()	Returns the randomization significance p-value for one tail
double GetRandZPValueTwotail ()	Returns the randomization significance p-value for two tail

Example in Visual Basic

```
'Create object
Dim cmoran As New CCalcSapd
'Initialize with primary and secondary dataset
cmoran.Initialize(CPointSet^ primary,CPointSet^ secondary, True)
'Calculate Moran's I statistics
cmoran.PerformCalculations()
'Get Result
’MsgBox(cmoran.GetResult())
```

Geary's “C”

Class: CCalcGearyC

Synopsis

Geary's "C" statistic is an alternative indicator of spatial autocorrelation. It is an index of paired comparisons between different point locations and typically varies from 0 (similar values) to 2 (dissimilar values) with 1 indicating no spatial autocorrelation.

Description

The Geary “C” routine calculates 5 statistics:

1. The sample size
2. Geary's "C'"
3. The spatial random (expected) "C"
4. The standard deviation of "C"
5. A significance test of "C" under the assumption of normality (Z-test)
   Values of C less than the expected C indicate clustering while values of C greater than the expected C indicate dispersion. The significance test indicates whether these differences are greater than what would be expected by chance.

If checked, small distances are adjusted so that the maximum weighting is 1 (see documentation for details.) This ensures that C won't become excessively large or excessively small for points that are close together. The default value is no adjustment.
The statistic requires an intensity variable in the primary file.
The tabular results can be saved to a '.dbf' file. The user must provide a file name. See Chapter 4 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
void Initialize (CPointSet^ pPointSet, CPointSet^ pSecPointSet, bool `isAdjsmallDist`)	Initializes the object for Geary calculation. This method takes in a primary point set, a secondary pointset, a boolean flag which is set to true for small distance adjustment as parameters. This method must be called before invoking any other methods of this class.
bool PerformCalculations ()	This method is the main method that computes the statistics. This method should be invoked only after a call to initialize method above.
System::String^ GetResult()	Returns formatted Geary “C” statistics as a String
double GetGearyC ()	Returns the Geary’s “C” statistic value
double GetGearyExpected ()	Returns the value of the spatially expected(random) “C”
double GetStdDev()	Returns the value of the Standard error of “C”
Double GetNormSig()	Returns the value of the Normality Significance or Z Score
double GetOnetail ()	Returns the Normality significance p-value for one tail
double GetTwotail ()	Returns the Normality significance p-value for two tail

---

Example in Visual Basic

```
'Create object
Dim gearyc As New CCalcGearyC
'Initialize Geary's C with primary and secondary pointset
gearyc.Initialize(CPointSet^ primary,CPointSet^ secondary, False)
'Calculate Geary's C statistics
gearyc.PerformCalculations()
'Get Result
MsgBox(cgeary.GetResult())
```

Getis-Ord General “G”

Class: CCalcGetisOrdGG

Synopsis

The Getis-Ord General “G” routine calculates the G statistic for a fixed search distance. The statistics test whether, in general, high values are spatially near other high values (hot spots) or whether low values are spatially near other low values (cold spots). The statistic typically varies from -1 to 1.

Description

The Getis-Ord “G” statistic is an index of spatial autocorrelation for values that fall within a specified distance of each other. It has the advantage over the other global spatial autocorrelation measures (Moran, Geary) in that it can distinguish between ‘hot spots’ and ‘cold spots’. The “G” value is calculated with respect to a specified search distance (defined by the user). The “G” value typically varies from -1 to 1 though those are not absolute limits. A positive z-value indicates spatial clustering of high values (‘hot spots’) while a negative z-value indicates spatial clustering of low values (‘cold spots’). A “G” value around 0 typically indicates no spatial autocorrelation. The statistic requires an intensity variable in the primary file.

The Getis-Ord General “G” routine calculates five statistics:

1. The sample size
2. The G
3. The spatial random (expected) G
4. The standard deviation of G
5. A significance test of G under the assumption of normality (Z-test)
   Values of G less than the expected G indicate dispersion while values of G greater than the expected G indicate clustering. The significance test indicates whether these differences are greater than what would be expected by chance.

A Monte Carlo simulation can be run to estimate approximate confidence intervals around the G values. Specify the number of simulations to be run (e.g., 100, 1000,

10000). For the simulation, there are six additional statistics output for the G statistics respectively:

6. The minimum I value for the distance bin
7. The maximum I value for the distance bin
8. The 0.5 percentile for the distance bin
9. The 2.5 percentile for the distance bin
10. The 97.5 percentile for the distance bin
11. The 99.5 percentile for the distance bin.

The tabular results can be saved to a '.dbf' file. The user must provide a file name. See Chapter 4 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
initialize(CPointSet^ pPointSet, double srchDistIpD, int noOfSimIpD, COV_UNIT_TYPE nOutUnit)	Initializes the object for Getis-Ord G calculation. This method takes in a primary point set (pPointSet), the search distance (srchDistIpD), number of monte carlo simulation runs (noOfSimIpD) and unit for output as parameters. This method must be called before invoking any other methods of this class.
void PerformInitialCalculations ()	This method performs the initial calculations to compute the statistics. This method should be invoked only after a call to initialize method above.
void PerformFinalCalculations ()	This method performs the final calculations to compute the statistics. This method should be invoked only after a call to initialize method above.
System::String^ GetResult()	Returns formatted Getis-Ord G statistics as a String
unsigned int getSampleSize ()	Returns the sample size used to compute the statistics
String^ getMeasurementType()	Returns the measurement type used to compute the

	statistics
double getGetisOrdGeneralG()	Returns the value of the Getis-Ord G spatial autocorrelation statistic
double getSpatialRandomG()	Returns the spatially expected (random) value for Getis-Ord G
double getStandardErrorG()	Returns the standard error for Getis-Ord G
double getNormalitySignificanceG()	Returns the normality significance of Getis-Ord G
double getPValueOneTailG()	Returns the Normality significance p-value for one tail of Getis-Ord G
double getPValueTwoTailG()	Returns the Normality significance p-value for two tail of Getis-Ord G
array<double>^ getSimulationG()	Returns an array of simulation data for Getis-Ord G. The array is one dimensional with values corresponding to the values returned by getPercentileValues() method described below.
array<double>^ getPercentileValues()	Returns an array of percentile values used during simulation

---

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

'Number of columns in the file
MsgBox(file.m_nColumns)
```

```
'Map columns
file.SetX(fileid, 10 ,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetY(fileid, 9 ,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetIntensty(fileid, 8 , INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
'Update Data after mapping columns
file.UpdatePrimaryData()

'Use file.m_pPrimaryPointSet as argument to different modules.
'-- Example: Getis-Ord G
Dim getisordgg As New CCalcGetisOrdGG
getisordgg.initialize(file.m_pPrimaryPointSet, 1, 2,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES)
getisordgg.PerformInitialCalculations()
getisordgg.PerformFinalCalculations()
MsgBox(getisordgg.GetResult())
```

Moran Correlogram

Class: CCalcMoranCorrelogram

Synopsis

The Moran Correlogram calculates the Moran's "I" index for different distance intervals/bins. The user can select any number of distance intervals. The default is 10 distance intervals.

Description

The Moran Correlogram applies Moran's "I" statistic to pairs varying by different distances. The user defines the number of distance intervals (the default is 10) and the routine calculates the "I" value for each interval. The output includes five statistics:

1. The sample size
2. The maximum distance
3. The bin (interval) number
4. The midpoint of the distance bin
5. The I value for the distance bin (I[B])

If the item is checked, small distances are adjusted so that the maximum weighting is 1 (see chapter 4 in the CrimeStat manual for details.) This ensures that the I values for individual distances won't become excessively large or excessively small for points that are close together. The default value is no adjustment. The statistic requires an intensity variable in the primary file

A Monte Carlo simulation can be run to estimate approximate confidence intervals around the I value. Specify the number of simulations to be run (e.g., 100, 1000, 10000). For the simulation, there are six additional statistics output for each distance interval:

6. The minimum I value for the distance bin
7. The maximum I value for the distance bin
8. The 0.5 percentile for the distance bin
9. The 2.5 percentile for the distance bin
10. The 97.5 percentile for the distance bin
11. The 99.5 percentile for the distance bin.

Fields

None

Methods

Method	Description
void initialize(CPointSet^ pPointSet, unsigned int pNoDistIntervals, int pNoSimulations, COV_UNIT_TYPE nOutUnit, bool pCalcIndIntervals, bool pIsAdjSmall)	Initializes the object for Moran's I Correlogram calculation. Parameters for this method are 1. primary point set (pPointSet) 2. Search distance (pNoDistIntervals) 3. number of monte carlo simulation runs (pNoSimulations) 4. Unit of distance for output 5. Calculation required for individual interval or not. pCalcIndIntervals = true will calculate the statistics for individual intervals only 6. pIsAdjSmall – Boolean parameter to specify if adjustment is required for small distances. pIsAdjSmall = true will adjust the calculations for small distances. This method must be called

	before invoking any other methods of this class.
void PerformCalculations ()	This method computes the statistics. This method should be invoked only after a call to initialize method above.
System::String^ GetResult()	Returns formatted Moran Correlogram statistics as a String
unsigned int getSampleSize ()	Returns the sample size used to compute the statistics
String^ getMeasurementType()	Returns the measurement type used to compute the statistics
UNIT_TYPE getUnitType()	Returns the unit type used in calculations
array<double>^ getDistances()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Moran's Correlogram distance for the distance intervals specified.
array<double>^ getMoransIB()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Moran's Correlogram I(B) value for the distance intervals specified.
array<double>^ getMoransIBMin()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Moran's Correlogram I(B) Min value for the distance intervals

	specified. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getMoransIBMax()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Moran's Correlogram I(B) Max value for the distance intervals specified. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getMoransIB99PctLow()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Moran's Correlogram I(B) 99th percentile low indices value. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getMoransIB95PctLow()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Moran's Correlogram I(B) 95th percentile low indices value. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getMoransIB95PctHigh()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The

	array has the Moran's Correlogram I(B) 95th percentile high indices value. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getMoransIB99PctHigh()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Moran's Correlogram I(B) 99th percentile high indices value. The values are valid only if Number of Monte Carlo simulations is > 0.
Void CCalcMoranCorrelogram::saveResult(CFileXBaseWrap ^pFile)	The function is invoked to save the results to a DBF file(tabular DBF). The parameters are : 1. The Xbase file to write the output.
unsigned int getNumberOfDistanceIntervals()	Returns the number of distance intervals used for the calculation of correlogram statistics.

Example 1 in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

'Number of columns in the file
MsgBox(file.m_nColumns)

'Map columns
file.SetX(fileid, 10 ,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetY(fileid, 9 ,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetIntensty(fileid, 8 , INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
'Update Data after mapping columns
file.UpdatePrimaryData()

'Use file.m_pPrimaryPointSet as argument to different modules.
Dim corr As New CCalcMoranCorrelogram
Dim noOfSimulationRuns As Integer
noOfSimulationRuns = 2

corr.initialize(file.m_pPrimaryPointSet, 10, noOfSimulationRuns,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES, False, False)
```

```
corr.PerformCalculations()

System.Console.WriteLine("SampleSize {0:D}", corr.getSampleSize())
System.Console.WriteLine("maximum Distance {0:F6}", corr.getMaximumDistance())

Dim distances() As Double
Dim moranIB() As Double
Dim moranIBMin() As Double
Dim moranIBMax() As Double
Dim moranIB99PctLow() As Double
Dim moranIB95PctLow() As Double
Dim moranIB95PctHigh() As Double
Dim moranIB99PctHigh() As Double

distances = corr.getDistances()
moranIB = corr.getMoransIB()
moranIBMin = corr.getMoransIBMin()
moranIBMax = corr.getMoransIBMax()
moranIB99PctLow = corr.getMoransIB99PctLow()
moranIB95PctLow = corr.getMoransIB95PctLow()
moranIB95PctHigh = corr.getMoransIB95PctHigh()
moranIB99PctHigh = corr.getMoransIB99PctHigh()

Dim i As Integer
For i = 0 To corr.getNumberOfDistanceIntervals() - 1 Step 1
   System.Console.WriteLine("Distance for Bin {0:D} {1:F6}", i, distances(i))
   System.Console.WriteLine("Morans IB value for Bin {0:D} {1:F6}", i, moranIB(i))
   If noOfSimulationRuns > 0 Then
      System.Console.WriteLine("Morans IB Min value for Bin {0:D} {1:F6}", i,
moranIBMin(i))
      System.Console.WriteLine("Morans IB Max value for Bin {0:D} {1:F6}", i,
moranIBMax(i))
      System.Console.WriteLine("Morans IB 99th Percentile Low value for Bin {0:D}
{1:F6}", i, moranIB99PctLow(i))
      System.Console.WriteLine("Morans IB 95th Percentile Low  value for Bin {0:D}
{1:F6}", i, moranIB95PctLow(i))
      System.Console.WriteLine("Morans IB 95th Percentile High value for Bin {0:D}
{1:F6}", i, moranIB95PctHigh(i))
```

```
      System.Console.WriteLine("Morans IB 99th Percentile High value for Bin {0:D}
{1:F6}", i, moranIB99PctHigh(i))
   End If
Next i

MsgBox(corr.GetResult())
```

Example 2: Writing DBF Files in Visual Basic:

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

'Number of columns in the file
MsgBox(file.m_nColumns)

'Map columns
file.SetX(fileid, 10 ,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetY(fileid, 9 ,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetIntensty(fileid, 8 , INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
'Update Data after mapping columns
file.UpdatePrimaryData()

Dim morancorr_dbf As New CFileXBaseWrap
```

```
Dim dirname As String
Dim sname As String
Dim filename As String
Dim fileutils As New CFileUtilsPub
fileutils.SplitPath(Me.Text_Path.Text, dirname, sname)
 dirname = fileutils.sDir
 sname = fileutils.sName
 sname = "MoranCorr" + sname
 filename = dirname + sname
 Dim success As Boolean
 success = morancorr_dbf.Create(filename)

'Use
'Use file.m_pPrimaryPointSet as argument to different modules.
        Dim morancorr As New CCalcMoranCorrelogram
         morancorr.initialize(file.m_pPrimaryPointSet, 20, 10,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES, True, True)
        morancorr.PerformCalculations()

        success = success And morancorr.saveResult(morancorr_dbf)
        If (success) Then
           System.Console.WriteLine("Writing to DBF Success")
        End If
        If (success <> True) Then
           System.Console.WriteLine("Writing to DBF Failure")
        End If
        morancorr_dbf.Close()
        System.Console.WriteLine("Executon COMPLETED")
        Me.Close()
```

Geary Correlogram

Class: CCalcGearyCCorr

Synopsis

The Geary Correlogram calculates the Geary's "C" index for different distance intervals/bins. For each distance interval, the "C" value typically varies between 0 (similar values) to 2 (dissimilar values.). The user can select any number of distance intervals. The default is 10 distance intervals.

Description

The Geary Correlogram applies Geary's "C" statistic to pairs varying by different distances. The user defines the number of distance intervals (the default is 10) and the routine calculates the "I" value for each interval. The output includes five statistics:

1. The sample size
2. The maximum distance
3. The bin (interval) number
4. The midpoint of the distance bin
5. The C value for the distance bin (C[B])

If the item is checked, small distances are adjusted so that the maximum weighting is 1 (see chapter 4 in the CrimeStat manual for details.) This ensures that the C values for individual distances won't become excessively large or excessively small for points that are close together. The default value is no adjustment. The statistic requires an intensity variable in the primary file

A Monte Carlo simulation can be run to estimate approximate confidence intervals around the C value. Specify the number of simulations to be run (e.g., 100, 1000, 10000). For the simulation, there are six additional statistics output for each distance interval:

6. The minimum C value for the distance bin
7. The maximum C value for the distance bin
8. The 0.5 percentile for the distance bin

9. The 2.5 percentile for the distance bin
10. The 97.5 percentile for the distance bin
11. The 99.5 percentile for the distance bin.

Fields

None

Methods

Method	Description
initialize(CPointSet^ pPointSet, unsigned int pNoDistIntervals, int pNoSimulations, COV_UNIT_TYPE nOutUnit, bool pCalcIndIntervals, bool pIsAdjSmall)	Initializes the object for Geary C Correlogram calculation. Parameters for this method are 1. primary point set (pPointSet) 2. Search distance (pNoDistIntervals) 3. number of monte carlo simulation runs (pNoSimulations) 4. Unit of distance for output 5. Calculation required for individual interval or not. pCalcIndIntervals = true will calculate the statistics for individual intervals only 6. pIsAdjSmall – Boolean parameter to specify if adjustment is required for small distances. pIsAdjSmall = true will adjust the calculations for small distances. This method must be called before invoking any other methods of this class.
void PerformCalculations ()	This method computes the statistics. This method should be invoked only after a call to initialize method above.
System::String^ GetResult()	Returns formatted Moran Correlogram statistics as a String
unsigned int getSampleSize ()	Returns the sample size used to compute the statistics
String^ getMeasurementType()	Returns the measurement type used to

	compute the statistics
UNIT_TYPE getUnitType()	Returns the unit type used in calculations
array<double>^ getDistances()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Moran's Correlogram distance for the distance intervals specified.
array<double>^ getGearyCB ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Geary's Correlogram C(B) value for the distance intervals specified.
array<double>^ getGearyCBMin ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Geary's Correlogram C(B) Min value for the distance intervals specified. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getGearyCBMax ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Geary's Correlogram C(B) Max value for the distance intervals specified. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getGearyCB99PctLow ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Geary's Correlogram C(B) 99th percentile low indices value. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getGearyCB95PctLow ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Geary's Correlogram C(B) 95th percentile low indices value. The values are valid only

	if Number of Monte Carlo simulations is > 0.
array<double>^ getGearyCB95PctHigh ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Geary's Correlogram C(B) 95th percentile high indices value. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getGearyCB99PctHigh ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Geary's Correlogram C(B) 99th percentile high indices value. The values are valid only if Number of Monte Carlo simulations is > 0.
unsigned int getNumberOfDistanceIntervals()	Returns the number of distance intervals used for the calculation of correlogram statistics.

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
```

```
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

'Number of columns in the file
MsgBox(file.m_nColumns)

'Map columns
file.SetX(fileid, 10 ,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetY(fileid, 9 ,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetIntensty(fileid, 8 , INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
'Update Data after mapping columns
file.UpdatePrimaryData()

'Use file.m_pPrimaryPointSet as argument to different modules.
Dim corr As New CCalcGearyCCorr
Dim noOfSimulationRuns As Integer
noOfSimulationRuns = 2
corr.initialize(file.m_pPrimaryPointSet, 10, noOfSimulationRuns,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES, False, False)

corr.PerformCalculations()

System.Console.WriteLine("SampleSize {0:D}", corr.getSampleSize())
System.Console.WriteLine("maximum Distance {0:F6}", corr.getMaximumDistance())

Dim distances() As Double
Dim gearyCB() As Double
Dim gearyCBMin() As Double
Dim gearyCBMax() As Double
Dim gearyCB99PctLow() As Double
Dim gearyCB95PctLow() As Double
Dim gearyCB95PctHigh() As Double
Dim gearyCB99PctHigh() As Double

distances = corr.getDistances()
gearyCB = corr.getGearyCB()
```

```
gearyCBMin = corr.getGearyCBMin()
gearyCBMax = corr.getGearyCBMax()
gearyCB99PctLow = corr.getGearyCB99PctLow()
gearyCB95PctLow = corr.getGearyCB95PctLow()
gearyCB95PctHigh = corr.getGearyCB95PctHigh()
gearyCB99PctHigh = corr.getGearyCB99PctHigh()

Dim i As Integer
For i = 0 To corr.getNumberOfDistanceIntervals() - 1 Step 1
  System.Console.WriteLine("Distance for Bin {0:D} {1:F6}", i, distances(i))
  System.Console.WriteLine("Geary's CB value for Bin {0:D} {1:F6}", i, gearyCB(i))
  If noOfSimulationRuns > 0 Then
    System.Console.WriteLine("Geary's CB Min value for Bin {0:D} {1:F6}", i,
gearyCBMin(i))
    System.Console.WriteLine("Geary's CB Max value for Bin {0:D} {1:F6}", i,
gearyCBMax(i))
    System.Console.WriteLine("Geary's CB 99th Percentile Low value for Bin {0:D}
{1:F6}", i, gearyCB99PctLow(i))
    System.Console.WriteLine("Geary's CB 95th Percentile Low  value for Bin {0:D}
{1:F6}", i, gearyCB95PctLow(i))
    System.Console.WriteLine("Geary's CB 95th Percentile High value for Bin {0:D}
{1:F6}", i, gearyCB95PctHigh(i))
    System.Console.WriteLine("Geary's CB 99th Percentile High value for Bin {0:D}
{1:F6}", i, gearyCB99PctHigh(i))
  End If
Next i

MsgBox(corr.GetResult())
```

Graphics output for CCalcGearyCCor

bool CCalcGearyCCorr::saveResult(CFileXBaseWrap ^file)	The function is invoked to save the results to a DBF file(tabular DBF). The parameters are : 1.The Xbase file to write the output. It returns true if the file has been successfully written.

Getis-Ord Correlogram

Class: CCalcGetisOrdGGCorr

Synopsis

The Getis-Ord Correlogram calculates the Getis-Ord General "G" statistics for different distance intervals/bins. The "G" values typically vary from -1 to 1 for each distance interval.

Description

The Getis-Ord Correlogram calculates the Getis-Ord "G" index for different distance intervals/bins. The user can select any number of distance intervals. The default is 10 distance intervals. The statistic requires an intensity variable in the primary file. A Monte Carlo simulation can be run to estimate approximate confidence intervals around the "G" value. For each distance interval, the G values typically vary between -1 to 1. A value of -1 indicates the clustering of observations with low values (a 'cold spot') while a value of 1 indicates the clustering of observations with high values (a 'hot spot'). A value of 0 indicates no clustering. However, with increasing distances, the values of G will approach 1. The user can select any number of distance intervals. The default is 10 distance intervals.

For each distance interval, the Geary Correlogram calculates five statistics:

1. The sample size
2. The maximum distance
3. The bin (interval) number
4. The midpoint of the distance bin
5. The G value for the distance bin (G[B])

A Monte Carlo simulation can be run to estimate approximate confidence intervals around the G values. Specify the number of simulations to be run (e.g., 100, 1000, 10000). For the simulation, there are six additional statistics output for the G for each distance interval:

6. The minimum G value for the distance bin
7. The maximum G value for the distance bin

8. The 0.5 percentile for the distance bin
9. The 2.5 percentile for the distance bin
10. The 97.5 percentile for the distance bin
11. The 99.5 percentile for the distance bin.

The tabular results can be saved to a '.dbf' file. The user must provide a file name. See the update chapter for CrimeStat version 3.2 for more information.

Fields

None

Methods

Method	Description
initialize(CPointSet^ pPointSet, unsigned int pNoDistIntervals, int pNoSimulations, COV_UNIT_TYPE nOutUnit)	Initializes the object for Getis-Ord G Corellogram calculation. Parameters for this method are 1. primary point set (pPointSet) 2. Search distance (pNoDistIntervals) 3. number of monte carlo simulation runs (pNoSimulations) 4. Unit of distance for output This method must be called before invoking any other methods of this class.
void PerformCalculations ()	This method computes the statistics. This method should be invoked only after a call to initialize method above.
System::String^ GetResult()	Returns formatted Moran Correlogram statistics as a String
unsigned int getSampleSize ()	Returns the sample size used to compute the statistics
String^ getMeasurementType()	Returns the measurement type used to compute the statistics
UNIT_TYPE getUnitType()	Returns the unit type used in calculations
array<double>^ getDistances()	Returns a 1-D array of size equal to the number of distance intervals specified during

	initialization. The array has the Moran's Correlogram distance for the distance intervals specified.
array<double>^ getGetisOrdGB ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Getis-Ord G(B) Corellogram value for the distance intervals specified.
array<double>^ getGetisOrdGBMin()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Getis-Ord G(B) Min value for the distance intervals specified. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getGetisOrdGBMax ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Getis-Ord G(B) Max value for the distance intervals specified. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getGetisOrdGB99PctLow ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Getis-Ord G(B) 99th percentile low indices value. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getGetisOrdGB95PctLow ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Getis-Ord G(B) 95th percentile low indices value. The values are valid only if Number of Monte Carlo simulations is > 0.
array<double>^ getGetisOrdGB95PctHigh ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Getis-Ord G(B) 95th percentile high indices value. The values are valid only if Number of Monte Carlo simulations is > 0.

array<double>^ getGetisOrdGB99PctHigh ()	Returns a 1-D array of size equal to the number of distance intervals specified during initialization. The array has the Getis-Ord G(B) 99th percentile high indices value. The values are valid only if Number of Monte Carlo simulations is > 0.
unsigned int getNumberOfDistanceIntervals()	Returns the number of distance intervals used for the calculation of correlogram statistics.

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

'Number of columns in the file
MsgBox(file.m_nColumns)

'Map columns
file.SetX(fileid, 10 ,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetY(fileid, 9 ,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetIntensty(fileid, 8 , INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
```

```
'Update Data after mapping columns
file.UpdatePrimaryData()

'Use
'Use file.m_pPrimaryPointSet as argument to different modules.
'-- Example: Getis-Ord G
Dim corr As New CCalcGetisOrdGGCorr
Dim noOfSimulationRuns As Integer
noOfSimulationRuns = 2
corr.initialize(file.m_pPrimaryPointSet, 10, noOfSimulationRuns,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES)
corr.PerformCalculations()

System.Console.WriteLine("SampleSize {0:D}", corr.getSampleSize())
System.Console.WriteLine("maximum Distance {0:F6}", corr.getMaximumDistance())

Dim distances() As Double
Dim getisordGB() As Double
Dim getisordGBMin() As Double
Dim getisordGBMax() As Double
Dim getisordGB99PctLow() As Double
Dim getisordGB95PctLow() As Double
Dim getisordGB95PctHigh() As Double
Dim getisordGB99PctHigh() As Double

distances = corr.getDistances()
getisordGB = corr.getGetisOrdGB()
getisordGBMin = corr.getGetisOrdGBMin()
getisordGBMax = corr.getGetisOrdGBMax()
getisordGB99PctLow = corr.getGetisOrdGB99PctLow()
getisordGB95PctLow = corr.getGetisOrdGB95PctLow()
getisordGB95PctHigh = corr.getGetisOrdGB95PctHigh()
getisordGB99PctHigh = corr.getGetisOrdGB99PctHigh()

Dim i As Integer
For i = 0 To corr.getNumberOfDistanceIntervals() - 1 Step 1
   System.Console.WriteLine("Distance for Bin {0:D} {1:F6}", i, distances(i))
   System.Console.WriteLine("Getis-Ord G B value for Bin {0:D} {1:F6}", i,
getisordGB(i))
   If noOfSimulationRuns > 0 Then
```

```
        System.Console.WriteLine("Getis-Ord G B Min value for Bin {0:D} {1:F6}", i,
getisordGBMin(i))
        System.Console.WriteLine("Getis-Ord G B Max value for Bin {0:D} {1:F6}", i,
getisordGBMax(i))
        System.Console.WriteLine("Getis-Ord G B 99th Percentile Low value for Bin
{0:D} {1:F6}", i, getisordGB99PctLow(i))
        System.Console.WriteLine("Getis-Ord G B 95th Percentile Low  value for Bin
{0:D} {1:F6}", i, getisordGB95PctLow(i))
        System.Console.WriteLine("Getis-Ord G B 95th Percentile High value for Bin
{0:D} {1:F6}", i, getisordGB95PctHigh(i))
        System.Console.WriteLine("Getis-Ord G B 99th Percentile High value for Bin
{0:D} {1:F6}", i, getisordGB99PctHigh(i))

    End If
Next i

MsgBox(corr.GetResult())
```

Distance Analysis Library

Distance analysis provides statistics about the distances between point locations. It is useful for identifying the degree of clustering of points. It is sometimes called second-order analysis. There are four routines for describing properties of the distances.

1. Nearest neighbor analysis
2. Ripley's K
3. Assign primary points to secondary points
4. Distance matrices.

See chapter 6 in the CrimeStat manual for more information. The following is a description of the functions conducted by the libraries.

Namespace: Distance Analysis

Field	Description
BORDER_CORRECTION_TYPE	Enum type to specify the method for border correction in NNA. Valid values are BORDER_CORRECTION_TYPE_NONE – No border correction BORDER_CORRECTION_TYPE_RECTANGULAR – Rectangular border correction BORDER_CORRECTION_TYPE_CIRCULAR – Circular border correction

Library: Distance Analysis.dll

Prerequisites: **Crimestat Core Components.dll**
Spatial Description.dll

Nearest Neighbor Analysis

Class: CCalcNna

Synopsis

This routine calculates the areal and linear nearest neighbor index. CCalcNna calculates the algorithm.

Description

The nearest neighbor index provides an approximation about whether points are more clustered or dispersed than would be expected on the basis of chance. It compares the average distance of the nearest other point (nearest neighbor) with a spatially random expected distance by dividing the empirical average nearest neighbor distance by the expected random distance (the nearest neighbor index.) The nearest neighbor routine requires that the geographical area be entered. If the distance units used are direct, then the ordinary nearest neighbor index is conducted. If the distance units are indirect, then the linear nearest neighbor index is conducted. The routine can output the statistics to a dBase (.dbf) file format. The user must provide a file name. See chapter 6 in the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
void Initialize (CPointSet ^pPointSet)	Initializes the object for NNA calculation. This method takes in the following parameters pPriPointSet – object of type CPointSet corresponding to the primary point set This method must be called before invoking any other methods of this class.
void SetBorderCorrection(	An optional method to set the type of border

BORDER_CORRECTION_TYPE nNnaBorderCorrection)	correction. By default the no border correction would be used. The valid values for the parameter are defined by the enum type BORDER_CORRECTION_TYPE. This method should be called before PerformCalculations() is called.
void SetNumberOfNearestNeighbors(unsigned int m_nMaxPoints)	An optional method to specify the number of nearest neighbors to be used in the calculation. The default value is 1. This method should be called before PerformCalculations() is called.
void SetArea(double dArea, COV_UNIT_TYPE dAreaType)	An optional method to specify the area to be used in the calculation of mean random distance. In the previous version, the area used to be defined in the measurement parameters form. The default value will be the study area of the pointset specified. This method should be called before PerformCalculations() is called.
Void PerformCalculations()	This method is the main method that computes the NNA statistics. This method should be invoked only after a call to initialize method above.
unsigned int GetNumberOfNearestNeighbors()	Returns the value of number of nearest neighbors used in calculating the statistic. This is the same as that specified by the method SetNumberOfNearestNeighbors if invoked before the call to PerformCalculations.
double GetMeanNNDistance()	The mean nearest neighbor distance
double GetLinearMeanNNDistance()	The mean nearest neighbor distance when the distance measurement is Manhattan
double GetStandardDeviationNNDistance()	The standard deviation of the nearest neighbor distance
double GetLinearStandardDeviationOnUserInput()	The standard deviation of the nearest neighbor distance for the user input length if provided when the distance measurement is Manhattan.
double GetMinimumNNDistance()	The minimum distance between any points in

	the dataset
double GetLinearMinimumNNDistance()	The minimum distance between any points in the dataset when the distance measurement is Manhattan
double GetMaximumNNDistance()	The maximum distance between any points in the dataset
double GetLinearMaximumNNDistance()	The maximum distance between any points in the dataset when the distance measurement is Manhattan
double GetBoundingRectangleArea()	Area of the bounding rectangle for the data set
double GetUserInputArea()	Area input by the user if provided
Double GetMeanRandomDistanceOnBoundingRectangle()	The mean random distance for the bounding rectangle area of the dataset
double GetMeanRandomDistanceOnUserInputArea()	The mean random distance for the user input area if provided
Double GetLinearMeanRandomDistanceOnUserInput()	The mean random distance for the user input area if provided when the distance measurement is Manhattan
Double GetMeanDispersedDistanceOnBoundingRectangle()	The mean dispersed distance for the bounding rectangle area of the dataset
double GetMeanDispersedDistanceOnUserInputArea()	The mean dispersed distance for the user input area if provided
double GetNNIndexOnBoundingRectangle()	The nearest neighbor index for the area of the bounding rectangle of the dataset specified
double GetNNIndexOnUserInputArea()	The nearest neighbor index for the user input area if provided
double GetLinearNNIndexOnUserInput()	The nearest neighbor index for the user input length the user input area if provided when the distance measurement is Manhattan
double GetStandardErrorOnBoundingRectangle()	The standard error of the nearest neighbor index for the area of the bounding rectangle of the dataset specified
double GetStandardErrorOnUserInputArea()	The standard error of the nearest neighbor index for the area of the the user input area if provided

double GetLinearStandardErrorOnUserInput()	The standard error of the nearest neighbor index for the area of the the user input area if provided when the distance measurement is Manhattan
double GetZTestStatisticOnBoundingRectangle()	Value of a significance test of the nearest neighbor index (Z-test) for the area of the bounding rectangle of the dataset specified
double GetZTestStatisticOnUserInputArea()	Value of a significance test of the nearest neighbor index (Z-test) for the area of the the user input area if provided
double GetPValueOneTailOnBoundingRectangle()	The p-values associated with one tail significance test for the area of the bounding rectangle of the dataset specified
double GetPValueTwoTailOnBoundingRectangle()	The p-values associated with two tail significance test for the area of the bounding rectangle of the dataset specified
double GetPValueOneTailOnUserInputArea()	The p-values associated with one tail significance test for the area of the the user input area if provided
double GetPValueTwoTailOnUserInputArea()	The p-values associated with two tail significance test for the area of the the user input area if provided
double GetLinearTTestStatistic()	Value of a significance test of the nearest neighbor index when the distance measurement is Manhattan
double GetLinearPValueTwoTailOnUserInput()	The p-values associated with two tail significance test for the area of the the user input area if provided when the distance measurement is Manhattan
array<double>^ GetMeanNNDistanceArray()	Array of size equal to the number of nearest neighbors corresponding to the Nearest Neighbor distance
array<double>^ GetExpectedNNDistanceArray()	Array of size equal to the number of nearest neighbors corresponding to the expected Nearest Neighbor distance
bool SaveToDbf(CFileXBaseWrap ^pFile)	Pass an object of Type CFileXBaseWrap as a parameter and a return of True implies the

	successful creation of a Dbf File.
array<double>^ GetNNIndexArray()	Array of size equal to the number of nearest neighbors corresponding to the Nearest Neighbor indices

Example in Visual Basic

```
'Create an InputParam object and specify Primary point set

Dim nna As New CCalcNna
nna.Initialize(file.m_pPrimaryPointSet)

nna.SetBorderCorrection(DistanceAnalysis.BORDER_CORRECTION_TYPE.BORDER
_CORRECTION_TYPE_NONE)
nna.SetNumberOfNearestNeighbors(10)
nna.SetArea(100, Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES)
nna.PerformCalculations()
Dim mode_dbf As New CFileXBaseWrap
      Dim dirname As String
      Dim sname As String
      Dim filename As String
      Dim fileutils As New CFileUtilsPub
      fileutils.SplitPath(savefilename, dirname, sname)
      dirname = fileutils.sDir
      sname = fileutils.sName
      sname = "NNA" + sname
      filename = dirname + sname
      Dim success As Boolean
      success = mode_dbf.Create(filename)
      If (nna.SaveToDbf(mode_dbf)) Then
         System.Console.WriteLine("Writing to Dbf Success")
         mode_dbf.Close()
      End If
      System.Console.WriteLine("**********Printing
Results for NNA**********")
```

```
If file.m_pPrimaryPointSet.GetMeasureType <>
Utilities.MEASURE_TYPE.MEASURE_TYPE_MANHATTAN Then
        System.Console.WriteLine("    Mean Nearest Neighbor Distance ..{0:F6}",
nna.GetMeanNNDistance())
        System.Console.WriteLine("    Standard Dev of Nearest")
        System.Console.WriteLine("    Neighbor Distance .............{0:F6}",
nna.GetStandardDeviationNNDistance())
        System.Console.WriteLine("    Minimum Distance .............{0:F6}",
nna.GetMinimumNNDistance())
        System.Console.WriteLine("    Maximum Distance .............{0:F6}",
nna.GetMaximumNNDistance())
        System.Console.WriteLine("    Based on Bounding Rectangle:")
        System.Console.WriteLine("    Area ..........................{0:F6}",
nna.GetBoundingRectangleArea())
        System.Console.WriteLine("    Mean Random Distance ............{0:F6}",
nna.GetMeanRandomDistanceOnBoundingRectangle())
        System.Console.WriteLine("    Mean Dispersed Distance .........{0:F6}",
nna.GetMeanDispersedDistanceOnBoundingRectangle())
        System.Console.WriteLine("    Nearest Neighbor Index ..........{0:F6}",
nna.GetNNIndexOnBoundingRectangle())
        System.Console.WriteLine("    Standard Error ...................{0:F6}",
nna.GetStandardErrorOnBoundingRectangle())
        System.Console.WriteLine("    Test Statistic (Z) ..............{0:F6}",
nna.GetZTestStatisticOnBoundingRectangle())

        If (nna.GetPValueOneTailOnBoundingRectangle() > 0) Then
           System.Console.WriteLine("    p-value (one tail) ..............{0:F6}",
nna.GetPValueOneTailOnBoundingRectangle())
        Else
           System.Console.WriteLine("    p-value (one tail) ..............:  n.s.")
        End If

        If (nna.GetPValueTwoTailOnBoundingRectangle() > 0) Then
           System.Console.WriteLine("    p-value (two tail) ..............{0:F6}",
nna.GetPValueTwoTailOnBoundingRectangle())
        Else
           System.Console.WriteLine("    p-value (two tail) ..............:  n.s.")
        End If

        If (nna.GetUserInputArea() > 0.0) Then
```

```
            System.Console.WriteLine("    Based on User Input Area:")
            System.Console.WriteLine("    Area ..........................{0:F6}",
nna.GetUserInputArea())
            System.Console.WriteLine("    Mean Random Distance ............{0:F6}",
nna.GetMeanRandomDistanceOnUserInputArea())
            System.Console.WriteLine("    Mean Dispersed Distance .........{0:F6}",
nna.GetMeanDispersedDistanceOnUserInputArea())
            System.Console.WriteLine("    Nearest Neighbor Index ..........{0:F6}",
nna.GetNNIndexOnUserInputArea())
            System.Console.WriteLine("    Standard Error ..................{0:F6}",
nna.GetStandardErrorOnUserInputArea())
            System.Console.WriteLine("    Test Statistic (Z) ..............{0:F6}",
nna.GetZTestStatisticOnUserInputArea())

            If (nna.GetPValueOneTailOnUserInputArea() > 0) Then
               System.Console.WriteLine("    p-value (one tail) ..............{0:F6}",
nna.GetPValueOneTailOnUserInputArea())
            Else
               System.Console.WriteLine("    p-value (one tail) ..............: n.s.")
            End If
            If (nna.GetPValueTwoTailOnUserInputArea() > 0) Then
               System.Console.WriteLine("    p-value (two tail) ..............{0:F6}",
nna.GetPValueTwoTailOnUserInputArea())
            Else
               System.Console.WriteLine("    p-value (two tail) ..............:  n.s.")
            End If
         End If
Else
         If (nna.GetLinearUserInputLength() <> 0.0) Then
            System.Console.WriteLine("    Mean Linear Nearest Neighbor Distance
...{0:F6}", nna.GetLinearMeanNNDistance())
            System.Console.WriteLine("    Minimum Distance ........................{0:F6}",
nna.GetMinimumNNDistance())
            System.Console.WriteLine("    Maximum Distance ........................{0:F6}",
nna.GetMaximumNNDistance())
            System.Console.WriteLine("    Based on User Input length:")
            System.Console.WriteLine("    Length ..................................{0:F6}",
nna.GetLinearUserInputLength())
            System.Console.WriteLine("    Mean Random Linear Distance
.............{0:F6}", nna.GetLinearMeanRandomDistanceOnUserInput())
```

```
            System.Console.WriteLine("    Nearest Neighbor Index .....{0:F6}",
nna.GetLinearNNIndexOnUserInput())
            System.Console.WriteLine("    Standard Deviation .....{0:F6}",
nna.GetLinearStandardDeviationOnUserInput())
            System.Console.WriteLine("    Standard Error .....{0:F6}",
nna.GetLinearStandardErrorOnUserInput())
            System.Console.WriteLine("    Test Statistic (t) .....{0:F6}",
nna.GetLinearTTestStatistic())
            System.Console.WriteLine("    Degrees of Freedom .....{0:D}",
file.m_pPrimaryPointSet.GetPointCount() - 1)
            If (nna.GetLinearPValueTwoTailOnUserInput() > 0.05) Then
               System.Console.WriteLine("    p-value (two tail) .....:  n.s.")
            ElseIf (nna.GetLinearPValueTwoTailOnUserInput() < 0.000001) Then
               System.Console.WriteLine("    p-value (two tail) .....:  0.000001")
            Else
               System.Console.WriteLine("    p-value (two tail) .....:  {0:F6}",
nna.GetLinearPValueTwoTailOnUserInput())
            End If
         End If
End If

Dim nnaMeanLinearNN As Double()
Dim nnaExpectedNN As Double()
Dim nnaLinearNNIndex As Double()

nnaMeanLinearNN = nna.GetMeanNNDistanceArray()
nnaExpectedNN = nna.GetExpectedNNDistanceArray()
nnaLinearNNIndex = nna.GetNNIndexArray()

Dim i As Integer
System.Console.WriteLine(" Order      Mean Linear Nearest Neighbor(m)      Expected
Nearest Neighbor Distance(m)      Linear Nearest Neighbor Index")
For i = 0 To nna.GetNumberOfNearestNeighbors() - 1 Step 1
         System.Console.WriteLine(" {0:D}      {1:F6}      {2:F6}      {3:F6} ", i + 1,
nnaMeanLinearNN(i), nnaExpectedNN(i), nnaLinearNNIndex(i))
Next i

System.Console.WriteLine("**********************************End Printing Results
for NNA**********************************")
```

Ripley’s K

Class: CCalcRipleyK

Synopsis

This routine computes Ripley’s”K” statistic. CCalcRipleyK calculates the algorithm.

Description

Ripley's "K" statistic compares the number of points within any distance to an expected number for a spatially random distribution. The empirical count is transformed into a square root function, called L. A Monte Carlo simulation can be run to evaluate an approximate confidence interval around the L statistic. The user specifies the number of simulation runs and the L statistic is calculated for randomly assigned data. The random output is sorted and percentiles are calculated. Values of L that are greater than any particular percentile indicate more concentration while values of L less than any particular percentile indicate more dispersion. L is calculated for each of 100 distance intervals (bins.) Eight percentiles are identified for these statistics:

1. The minimum for the spatially random L value
2. The maximum for the spatially random L value
3. The 0.5 percentile for the spatially random L value
4. The 2.5 percentile for the spatially random L value
5. The 95 percentile for the spatially random L value
6. The 97.5 percentile for the spatially random L value
7. The 99 percentile for the spatially random L value
8. The 99.5 percentile for the spatially random L value

This routine can output the Ripley's K statistic to the dBase (.dbf) file format. The user must provide a file name. See chapter 6 in the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
void Initialize (CPointSet ^pPointSet,COV_UNIT_TYPE nRipleyKUnit)	Initializes the object for Ripley's K calculation. This method takes in a primary point set, and the output unit type. This method must be called before invoking any other methods of this class.
void UseWeightVariable(bool bUseWeight)	An optional routine that specifies to use the weight variable defined in the pointset. By default the weights are not used in the calculations. This method needs to be invoked before PerformCalculations is invoked in order to use the weights
void UseIntensityVariable(bool bUseWeight)	An optional routine that specifies to use the intensity variable defined in the pointset. By default the intensities are not used in the calculations. This method needs to be invoked before PerformCalculations is invoked in order to use the weights
void SetBorderCorrection(BORDER_CORRECTION_TYPE mBCType)	An optional method to set the type of border correction. By default the no border correction would be used. The valid values for the parameter are defined by the enum type BORDER_CORRECTION_TYPE. This method should be called before PerformCalculations() is called.
virtual void setSimulationRuns(unsigned int nSimRuns)	An optional method to set the number of simulation runs. By default the number of simulation runs is 0. This method should be called before PerformCalculations() is called.
Void PerformCalculations(CFileXBaseWrap ^pFile)	This method is the main method that computes the Ripley's K. This method should be invoked only after a call to initialize method above. This method requires a parameter of type CFileXBaseWrap which is correponds to an object for a Dbf file.
array<double>^ GetDistances()	Returns an array of size 100(Number of intervals used in Crimestat for Ripley's K)

	corresponding to the distance for each interval
unsigned int GetNumberOfBins()	Returns the number of intervals, equal to 100
array<double>^ GetLT()	Returns an array of size 100(Number of intervals used in Crimestat for Ripley's K) corresponding to the LT for each interval when the number of simulation runs specified is greater than 0
array<double>^ GetLCSR()	Returns an array of size 100(Number of intervals used in Crimestat for Ripley's K) corresponding to the LCSR for each interval when the number of simulation runs specified is greater than 0
array<double>^ GetLTMin()	Returns an array of size 100(Number of intervals used in Crimestat for Ripley's K) corresponding to the Minimum value of LT for each interval when the number of simulation runs specified is greater than 0
array<double>^ GetLTMax()	Returns an array of size 100(Number of intervals used in Crimestat for Ripley's K) corresponding to the maximum value of LT for each interval when the number of simulation runs specified is greater than 0
array<double>^ GetLT0_5Percentile()	Returns an array of size 100(Number of intervals used in Crimestat for Ripley's K) corresponding to the 0.5 percentile value of LT for each interval during monte carlo simulations
array<double>^ GetLT2_5Percentile()	Returns an array of size 100(Number of intervals used in Crimestat for Ripley's K) corresponding to the 2.5 percentile value of LT for each interval during monte carlo simulations
array<double>^ GetLT97_5Percentile()	Returns an array of size 100(Number of intervals used in Crimestat for Ripley's K) corresponding to the 97.5 percentile value of LT for each interval during monte carlo simulations

array<double>^ GetLT99_5Percentile()	Returns an array of size 100(Number of intervals used in Crimestat for Ripley's K) corresponding to the 99.5 percentile value of LT for each interval during monte carlo simulations

---

Example in Visual Basic

```
    Dim ripK As New CCalcRipleyK
    ripK.Initialize(file.m_pPrimaryPointSet,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES)

ripK.SetBorderCorrection(DistanceAnalysis.BORDER_CORRECTION_TYPE.BORDE
R_CORRECTION_TYPE_CIRCULAR)
    ripK.SetSimulationRuns(5)
    Dim mode_dbf As New CFileXBaseWrap
    Dim dirname As String
    Dim sname As String
    Dim filename As String
    Dim fileutils As New CFileUtilsPub
    fileutils.SplitPath(savefilename, dirname, sname)
    dirname = fileutils.sDir
    sname = fileutils.sName
    sname = "ripleyK" + sname
    filename = dirname + sname
    Dim success As Boolean
    success = mode_dbf.Create(filename)
    ripK.PerformCalculations(mode_dbf)
    mode_dbf.Close()
    System.Console.WriteLine("**********************************Printing
Results for Ripley's K**********************************")
    Dim ripKDist() As Double
    Dim ripKLt() As Double
    Dim ripKLcsr() As Double
    Dim ripKLtmin() As Double
    Dim ripKLtmax() As Double
    Dim ripKLt1() As Double
```

```
        Dim ripKLt2() As Double
        Dim ripKLt3() As Double
        Dim ripKLt4() As Double

        ripKDist = ripK.GetDistances()
        ripKLt = ripK.GetLT()
        ripKLcsr = ripK.GetLCSR()
        ripKLtmin = ripK.GetLTMin()
        ripKLtmax = ripK.GetLTMax()
        ripKLt1 = ripK.GetLT0_5Percentile()
        ripKLt2 = ripK.GetLT2_5Percentile()
        ripKLt3 = ripK.GetLT97_5Percentile()
        ripKLt4 = ripK.GetLT99_5Percentile()

        Dim i As Integer
        System.Console.WriteLine(" Bin      Distance      L(t)      L(csr)      L(t)min
L(t)max      L(t)0.5      L(t)2.5      L(t)97.5      L(t)99.5")
        For i = 0 To ripK.GetNumberOfBins() - 1 Step 1
            System.Console.WriteLine(" {0:D}      {1:F6}      {2:F6}      {3:F6}      {4:F6}
{5:F6}      {6:F6}      {7:F6}      {8:F6}      {9:F6}", i + 1, ripKDist(i), ripKLt(i),
ripKLcsr(i), ripKLtmin(i), ripKLtmax(i), ripKLt1(i), ripKLt2(i), ripKLt3(i), ripKLt4(i))
        Next i
        System.Console.WriteLine("**********************************End Printing
Results for Ripley's K**********************************")
```

Assign Primary Points to Secondary Points

Class: CCalcAssignPrim

Synopsis

This routine assigns each primary point to a secondary point and will then sum the number of primary points assigned to each secondary point. There are two types of assignment, one based on the nearest neighbor and the other based on point-in-polygon for a specific zone file. CCalcAssignPrim calculates the algorithm.

Description

This routine will assign each primary point to a secondary point and then will sum by the number of primary points assigned to each secondary point. It is useful for adding up the number of primary points that are close to each secondary point. For example, in the crime travel demand module, this routine can assign incidents to zones as the module uses zonal totals. The result is a count of primary points associated with each secondary point. It is also possible to sum different variables sequentially. For example, in the crime travel demand module, both the number of crimes originating in each zone and the number of crimes occurring in each zone are needed. This can be accomplished in two runs. First, sum the incidents defined by the origin coordinates to each zone (secondary file). Second, sum the incidents defined by the destination coordinates to each zone (also secondary file). The result would be two columns, one showing the number of origins in each secondary file zone and the second showing the number of destinations in each secondary file zone. There are two methods for assigning the primary points to the secondary. First, there is a nearest neighbor assignment where each primary point is assigned to the secondary point to which it is closest. If there are two or more secondary points that are exactly equal, the assignment goes to the first one on the list. Second, there is point-in-polygon assignment where each primary point is assigned to the secondary point for which it falls within its polygon (zone). A zone (polygon) shape file must be provided and the routine checks which secondary zone each primary point falls within. The routine outputs all the fields of the secondary file along with one extra field (FREQ) if no weighting option is selected. If a weighting option is selected, one more extra field (WEIGHT) is added. In the case of the point-in-polygon assignment, an ID field is also added along with the above fields. This routine can output to dBase (.dbf) file format. The user must provide a file name.

Fields

None

Methods

Method	Description
void Initialize (CPointSet^ pPriPointSet,CPointSet^ pSecPointSet)	Initializes the object with the primary and secondary pointset. This function should be called before invoking any other functions of the object.
void SetWeight (array<double>^ adWeights)	An optional function to specify the weights to be used by the assign primary points to secondary points routine. API's from CInputParam Class of the Crimestat CoreComponents library can be used to read the weights from either the secondary or miscellaneous file. By default no weights are used in the assignment. This function needs to be invoked before PerformCalculations()
void SetZoneFile (System::String^ zoneFileName)	An optional function to specify the zones to be used in calculations. By default the nearest neighbor assignment will be used. If a zone file is specified, the point in polygon assignment will be used. This function needs to be invoked before PerformCalculations().
bool PerformCalculations(CFileXBaseWrap ^presultfile,System::String) ^secfilename)	Performs the regression calculation. This function should be invoked only after initialize function has been invoked. Returns true on successful computation. This function takes two parameters namely the object of the result Dbf file and the name of the secondary file.Returns true on success.
array<int>^ GetPointCount()	Returns an array of integer of size equal to the number of points in the secondary data set. Each value in the array corresponds to the number of primary points assigned to a particular point in the secondary data set.
array<double>^ GetWeightedPointCount()	Returns an array of double of size equal to the number of points in the secondary data set. Each value in the array corresponds to the weighted number of primary points assigned to a particular point in the secondary

	data set.
bool IsUseWeight()	Returns true if weights are specified
int GetNumberOfSecondaryPoints()	Returns the number of points in the secondary dataset.

Example in Visual Basic

```
'Create an InputParam object and specify Primary point set as well as secondary point set

 Dim assignPriSec As New CCalcAssignPrim
      assignPriSec.Initialize(file.m_pPrimaryPointSet, file.m_pSecondaryPointSet)
      assignPriSec.SetWeight(file.getMiscInputDataColumn())
      If Me.zoneFileTxtBox.Text.Length > 0 Then
         assignPriSec.SetZoneFile(zoneFileTxtBox.Text)
      End If
      Dim mode_dbf As New CFileXBaseWrap
      Dim dirname As String
      Dim sname As String
      Dim filename As String
      Dim fileutils As New CFileUtilsPub
      fileutils.SplitPath(savefilename, dirname, sname)
      dirname = fileutils.sDir
      sname = fileutils.sName
      sname = "Primsec" + sname
      filename = dirname + sname
      Dim success As Boolean
      success = mode_dbf.Create(filename)

      assignPriSec.PerformCalculations(mode_dbf, file.m_sec_FileName)
      mode_dbf.Close()
      System.Console.WriteLine("**********************************Printing
Results for Assign Primary to Secondary
Points**********************************")
      If (assignPriSec.IsUseWeight()) Then
         System.Console.WriteLine("Secondary Point #          # of Assigned Primary
Points          Weighted # assigned primary points")
      Else
```

```
        System.Console.WriteLine("Secondary Point #          # of Assigned Primary
Points")
    End If

    Dim pointCnt() As Integer
    Dim weightCnt() As Double
    pointCnt = assignPriSec.GetPointCount()
    'If (assignPriSec.IsUseWeight()) Then
    weightCnt = assignPriSec.GetWeightedPointCount()
    'End If
    Dim i As Integer

    For i = 0 To assignPriSec.GetNumberOfSecondaryPoints() - 1 Step 1
      If (assignPriSec.IsUseWeight()) Then
        System.Console.WriteLine("{0:D}                {1:D}                {2:F6}", i +
1, pointCnt(i), weightCnt(i))
      Else
        System.Console.WriteLine("{0:D}                {1:D}", i + 1, pointCnt(i))
      End If
    Next i
    System.Console.WriteLine("*********************************End Printing
Results for Primary to Secondary assignment*********************************")
```

Distance Matrices

Description

The next four routines output one of four different distance matrices:

1. From each primary point to every other primary point
2. From each primary point to each secondary point
3. From each primary point to the centroid of each reference file grid cell. This requires a reference file to be defined or used.
4. From each secondary point to the centroid of each reference file grid cell. This requires a reference file to be defined or used.

The routines can calculate the distances between points for a single file or the distances between points for two different files. These matrices can be useful for examining the frequency of different distances or for providing distances for another program. Because the output files are usually very large, only text output is allowed. This can then be read into a database or large statistical program for processing. Keep in mind that there may be storage problems for large matrices.

Within File Point to Point

Class: CCalcDisMatrix

Synopsis

This routine outputs the distance from each primary point to every other primary point CCalcDisMatrix calculates the algorithm.

Fields

None

Method

Method	Description
void Initialize(CPointSet^	Initializes the object with the primary pointset and the distance

pPriPointSet, COV_UNIT_TYPE covUnitType)	unit type for the output. This function should be called before invoking any other functions of the object.
bool PerformCalculations (unsigned int nRowStart, unsigned int nColStart, unsigned int nRowEnd, unsigned int nColEnd)	Performs the distance computation. The function takes in the start row, start column as well the end row, end column and calculates the matrix between the points in the rectangle specified. For Eg, specifying (1,1,5,6) will compute the distances between the points from row 1 to row 5 and the points between column 1 to column 6. This function should be invoked only after initialize function has been invoked. Returns true on success.
array <double,2>^ GetMatrix()	Returns back a 2 D array of double corresponding to the distances between the pair of points specified.

Example in Visual Basic

```
'Create an InputParam object and specify Primary point set

Dim disMatrix As New CCalcDisMatrix
disMatrix.Initialize (file.m_pPrimaryPointSet,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES)
Dim dMResult As Double(,)

Dim rowStart As Integer
Dim rowEnd As Integer
Dim colStart As Integer
Dim colEnd As Integer

rowStart = 1
rowEnd = 45
colStart = 1
colEnd = 45

If disMatrix.PerformCalculations(rowStart, colStart, rowEnd, colEnd) Then
        dMResult = disMatrix.GetMatrix()
        Dim i As Integer
        For i = 0 To rowEnd - rowStart Step 1
           System.Console.Write("{0:D}    ", i + 1)
```

```
            Dim j As Integer
            For j = 0 To colEnd - colStart Step 1
               System.Console.Write("{0:F6}    ", dMResult(i, j))
            Next j
            System.Console.WriteLine()
         Next i
End If
```

From Primary File Points to Secondary File Points

Class: CalcInterMatrix

Synopsis

This routine outputs the distance from each primary point to each secondary point. CalcInterMatrix calculates the algorithm.

Fields

None

Method

Method	Description
void Initialize(CPointSet^ pPriPointSet, CPointSet^ pSecPointSet,COV_UNIT_TYPE covUnitType)	Initializes the object with the primary pointset, secondary pointset and the distance unit type for the output. This function should be called before invoking any other functions of the object.
bool PerformCalculations (unsigned int nRowStart, unsigned int nColStart, unsigned int nRowEnd, unsigned int nColEnd)	Performs the distance computation. The function takes in the start row, start column as well the end row, end column and calculates the matrix between the points in the rectangle specified. For Eg, specifying (1,1,5,6) will compute the distances between the points from row 1 to row 5 and the points between column 1 to column 6. This function should be invoked only after initialize function has been invoked. Returns true on success.
array <double,2>^ GetMatrix()	Returns back a 2 D array of double corresponding to the distances between the pair of points specified.

From Primary File Points to Reference Grid

Class: CCalcPrimGridMatrix

Synopsis

This routine calculates the distance from each primary point to the centroid of each reference file grid cell. This requires a reference file to be defined or used. CCalcPrimGridMatrix calculates the algorithm.

Fields

None

Method

Method	Description
void Initialize(CPointSet^ pPriPointSet, CPointSet^ pRefPointSet, COV_UNIT_TYPE covUnitType)	Initializes the object with the primary pointset, reference or grid pointset and the distance unit type for the output. This function should be called before invoking any other functions of the object.
bool PerformCalculations (unsigned int nRowStart, unsigned int nColStart, unsigned int nRowEnd, unsigned int nColEnd)	Performs the distance computation. The function takes in the start row, start column as well the end row, end column and calculates the matrix between the points in the rectangle specified. For Eg, specifying (1,1,5,6) will compute the distances between the points from row 1 to row 5 and the points between column 1 to column 6. This function should be invoked only after initialize function has been invoked. Returns true on success.
array <double,2>^ GetMatrix()	Returns back a 2 D array of double corresponding to the distances between the pair of points specified.

From Secondary File Points to Reference Grid

Class: CCalcSecondGridMatrix

Synopsis

This routine calculates the distance from each secondary point to the centroid of each reference file grid cell. This requires a reference file to be defined or used. CCalcSecondGridMatrix calculates the algorithm.

Fields

None

Method

Method	Description
void Initialize(CPointSet^ pPriPointSet, CPointSet^ pRefPointSet, COV_UNIT_TYPE covUnitType)	Initializes the object with the secondary pointset, reference or grid pointset and the distance unit type for the output. This function should be called before invoking any other functions of the object.
bool PerformCalculations (unsigned int nRowStart, unsigned int nColStart, unsigned int nRowEnd, unsigned int nColEnd)	Performs the distance computation. The function takes in the start row, start column as well the end row, end column and calculates the matrix between the points in the rectangle specified. For Eg, specifying (1,1,5,6) will compute the distances between the points from row 1 to row 5 and the points between column 1 to column 6. This function should be invoked only after initialize function has been invoked. Returns true on success.
array <double,2>^ GetMatrix()	Returns back a 2 D array of double corresponding to the distances between the pair of points specified.

Hot Spot Analysis Library

The Hot Spot Analysis library provides two groups of classes of various statistical functions:

1. Hot Spot Analysis: There are six statistical functions, each of which may have supplementary class functions:

   A. Mode & Fuzzy Mode
   B. Nearest Neighbor Hierarchical Clustering
   C. Risk-adjusted Nearest Neighbor Hierarchical Clustering
   D. Spatial & Temporal Analysis of Crime (STAC)
   E. K-Means
   F. Anselin's Local Moran

2. Interpolation: There are two statistical functions under this category:

   A. Single-kernel Density Interpolation
   B. Duel-kernel Density Interpolation

See chapters 6, 7 and 8 in the CrimeStat manual for more information. The following is a description of the functions conducted by the libraries.

Library: Hot Spot Analysis.dll

Prerequisites: **Crimestat Core Components.dll**
Spatial Description.dll

Mode

Class: CCalcMode

Synopsis

The mode is the number of incidents at each location, ranked from the highest to the lowest. CCalcMode runs the algorithm.

Description

The mode is the most intuitive type of hot spot. It is the location with the largest number of incidents. The *CrimeStat* Mode routine calculates the frequency of incidents occurring at each unique location (a point with a unique X and Y coordinate), sorts the list, and outputs the results in rank order from the most frequent to the least frequent.

The tabular results can be printed, or output as a '.dbf' file. The user must provide a file name. See Chapter 6 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
void Initialize(CPointSet^ pPointSet)	Initializes the object for Mode calculation. This method takes in a primary point set. This method must be called before invoking any other methods of this class.
Void PerformCalculations()	This method is the main method that computes the mode. This method should be invoked only after a call to initialize method above.
bool SaveToDbf(CFileXBaseWrap ^pFile)	Returns a True value if writing to DBF files was a success.

unsigned int GetRowCount()	Returns the number of modes in the data set. This number is the same as the number of distinct points in the dataset.
array<CModeResult ^> ^ GetResult()	Returns the modes for the point set specified. The return value is an array of references to CModeResult class. The details for each mode can be referenced by using the routines of CModeResult.

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

'Update Data after mapping columns
file.UpdatePrimaryData()

Dim mode As New CCalcMode
mode.Initialize(file.m_pPrimaryPointSet)
mode.PerformCalculations()
```

```
Dim modeResult() As CModeResult
modeResult = mode.GetResult()
System.Console.WriteLine("**********************************Printing Results for
Mode**********************************")
Dim i As Integer
For i = 0 To mode.GetRowCount() - 1 Step 1
        Dim cm As CModeResult
        cm = modeResult(i)
    System.Console.WriteLine("{0:F} {1:F6} {2:F6}", cm.getFrequency(),
cm.getXCoordinate(), cm.getYCoordinate())
Next i
System.Console.WriteLine("**********************************End Printing
Results for Mode**********************************")
```

File Output Example in Visual Basic

```
Dim mode As New CCalcMode
      mode.Initialize(file.m_pPrimaryPointSet)
      mode.PerformCalculations()

      Dim modeResult() As CModeResult
      modeResult = mode.GetResult()
      System.Console.WriteLine("**********************************Printing
Results for Mode**********************************")
      Dim i As Integer
      For i = 0 To mode.GetRowCount() - 1 Step 1
         Dim cm As CModeResult
         cm = modeResult(i)
         System.Console.WriteLine("{0:F} {1:F6} {2:F6}", cm.getFrequency(),
cm.getXCoordinate(), cm.getYCoordinate())
      Next i
      System.Console.WriteLine("**********************************End Printing
Results for Mode**********************************")
      'Write results to DBF File
      Dim mode_dbf As New CFileXBaseWrap
      Dim dirname As String
      Dim sname As String
      Dim filename As String
```

```
        Dim fileutils As New CFileUtilsPub
        fileutils.SplitPath(savefilename, dirname, sname)
        dirname = fileutils.sDir
        sname = fileutils.sName
        sname = "Mode " + sname
        filename = dirname + sname
        Dim success As Boolean
        success = mode_dbf.Create(filename)
        success = success And mode.SaveToDbf(mode_dbf)
        If (success) Then
           mode_dbf.Close()
           System.Console.WriteLine("Writing to DBF Success")
        End If
        If (success <> True) Then
           System.Console.WriteLine("Writing to DBF Failure")
        End If
        Dim fmode As New CCalcFuzzyMode
        fmode.Initialize(file.m_pPrimaryPointSet, 1,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES)
        fmode.PerformCalculations()
        Dim fmodeResult() As CModeResult
        fmodeResult = fmode.GetResult()
        System.Console.WriteLine("**********************************Printing
Results for Fuzzy Mode**********************************")
        Dim j As Integer
        For j = 0 To fmode.GetRowCount() - 1 Step 1
           Dim cm As CModeResult
           cm = fmodeResult(j)
           System.Console.WriteLine("{0:F} {1:F6} {2:F6}", cm.getFrequency(),
cm.getXCoordinate(), cm.getYCoordinate())
        Next j
        System.Console.WriteLine("**********************************End Printing
Results for Fuzzy Mode**********************************")
        'Write results to DBF File
        Dim fmode_dbf As New CFileXBaseWrap
        Dim fdirname As String
        Dim fsname As String
        Dim ffilename As String
        fileutils.SplitPath(savefilename, dirname, sname)
        fdirname = fileutils.sDir
```

```
fsname = fileutils.sName
fsname = "FuzzyMode " + fsname
ffilename = dirname + fsname
Dim fsuccess As Boolean
fsuccess = fmode_dbf.Create(ffilename)
fsuccess = fsuccess And fmode.SaveToDbf(fmode_dbf)
If (fsuccess) Then
   System.Console.WriteLine("Writing to DBF Success")
   fmode_dbf.Close()
End If
If (fsuccess <> True) Then
   System.Console.WriteLine("Writing to DBF Failure")
End If
```

Fuzzy Mode

Class: CCalcFuzzyMode

This class inherits from CCalcMode.

Synopsis

The fuzzy mode is the number of incidents within a specified distance of each event. CCalcFuzzyMode runs the algorithm.

Description

The fuzzy mode calculates the frequency of incidents for each unique location within a user-specified distance. The user must specify the search radius and the units for the radius (miles, nautical miles, feet, kilometers, meters). Distances should be small (e.g., less than 0.25 miles). The routine will identify each unique location, defined by its X and Y coordinates, and will calculate the number of incidents that fall within the search radius. It will output a list of all unique locations and their X and Y coordinates and the number of incidents occurring at each, ranked in decreasing order from most frequent to least frequent. Because each circle can include multiple incidents, many incidents are counted multiple times.

The tabular results can be printed, saved to a text file, or output as a '.dbf' file. The user must provide a file name. See Chapter 6 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
Initialize(CPointSet^ pPointSet, double radius, COV_UNIT_TYPE nOutUnit)	Initializes the object for Mode calculation. This method takes in a primary point set, the radius for search and the Distance type for the radius. This method must be called before invoking any other methods of this class.
Void PerformCalculations()	This method is the main method that computes the mode. This method should be invoked only after a call to initialize method above.
unsigned int GetRowCount()	Returns the number of fuzzy modes in the data set.
bool SaveToDbf(CFileXBaseWrap ^pFile)	Returns a True value if writing to DBF files was a success.
array<CModeResult ^> ^ GetResult()	Returns the fuzzy mode for the point set specified. The return value is an array of references to CModeResult class.

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")
```

```
file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

'Update Data after mapping columns
file.UpdatePrimaryData()

Dim mode As New CCalcFuzzyMode
mode.Initialize(file.m_pPrimaryPointSet, 1,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES)
mode.PerformCalculations()

Dim modeResult() As CModeResult
modeResult = mode.GetResult()
System.Console.WriteLine("**********************************Printing Results for
Fuzzy Mode**********************************")
Dim i As Integer
For i = 0 To mode.GetRowCount() - 1 Step 1
        Dim cm As CModeResult
        cm = modeResult(i)
    System.Console.WriteLine("{0:F} {1:F6} {2:F6}", cm.getFrequency(),
cm.getXCoordinate(), cm.getYCoordinate())
Next i
System.Console.WriteLine("**********************************End Printing
Results for Fuzzy Mode**********************************")
```

File Output Example in Visual Basic

```
Dim mode As New CCalcMode
      mode.Initialize(file.m_pPrimaryPointSet)
      mode.PerformCalculations()

      Dim modeResult() As CModeResult
      modeResult = mode.GetResult()
```

```
        System.Console.WriteLine("**********************************Printing
Results for Mode**********************************")
        Dim i As Integer
        For i = 0 To mode.GetRowCount() - 1 Step 1
            Dim cm As CModeResult
            cm = modeResult(i)
            System.Console.WriteLine("{0:F} {1:F6} {2:F6}", cm.getFrequency(),
cm.getXCoordinate(), cm.getYCoordinate())
        Next i
        System.Console.WriteLine("**********************************End Printing
Results for Mode**********************************")
        'Write results to DBF File
        Dim mode_dbf As New CFileXBaseWrap
        Dim dirname As String
        Dim sname As String
        Dim filename As String
        Dim fileutils As New CFileUtilsPub
        fileutils.SplitPath(savefilename, dirname, sname)
        dirname = fileutils.sDir
        sname = fileutils.sName
        sname = "Mode " + sname
        filename = dirname + sname
        Dim success As Boolean
        success = mode_dbf.Create(filename)
        success = success And mode.SaveToDbf(mode_dbf)
        If (success) Then
            System.Console.WriteLine("Writing to DBF Success")
        End If
        If (success <> True) Then
            System.Console.WriteLine("Writing to DBF Failure")
        End If
        Dim fmode As New CCalcFuzzyMode
        fmode.Initialize(file.m_pPrimaryPointSet, 1,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES)
        fmode.PerformCalculations()
        Dim fmodeResult() As CModeResult
        fmodeResult = fmode.GetResult()
        System.Console.WriteLine("**********************************Printing
Results for Fuzzy Mode**********************************")
        Dim j As Integer
```

```
        For j = 0 To fmode.GetRowCount() - 1 Step 1
            Dim cm As CModeResult
            cm = fmodeResult(j)
            System.Console.WriteLine("{0:F} {1:F6} {2:F6}", cm.getFrequency(),
cm.getXCoordinate(), cm.getYCoordinate())
        Next j
        System.Console.WriteLine("**********************************End Printing
Results for Fuzzy Mode**********************************")
        'Write results to DBF File
        Dim fmode_dbf As New CFileXBaseWrap
        Dim fdirname As String
        Dim fsname As String
        Dim ffilename As String
        fileutils.SplitPath(savefilename, dirname, sname)
        fdirname = fileutils.sDir
        fsname = fileutils.sName
        fsname = "FuzzyMode " + fsname
        ffilename = dirname + fsname
        Dim fsuccess As Boolean
        fsuccess = fmode_dbf.Create(ffilename)
        fsuccess = fsuccess And fmode.SaveToDbf(fmode_dbf)
        If (fsuccess) Then
            System.Console.WriteLine("Writing to DBF Success")
        End If
        If (fsuccess <> True) Then
            System.Console.WriteLine("Writing to DBF Failure")
        End If
```

Mode & Fuzzy Mode Utility

Class: CModeResult

Synopsis

CModeResult provides a template to hold the results of the Mode or Fuzzy Mode routines.

Description

This routine is used in conjunction with the Mode or Fuzzy Mode classes.

Fields

None

Methods

Method	Description
double getKey()	Returns back the frequency count for the particular spatial point as the key.
double getFrequency()	Returns back the frequency count for the particular spatial point.
double getXCoordinate()	Returns back the X-Cordinate of the spatial point
double getYCoordinate()	Returns back the X-Cordinate of the spatial point

Hot Spot Analysis Results

Class: ResultCluster

Synopsis

ResultCluster provides a template to hold the resulting clusters from the different hot spot routines.

Description

This routine is used in conjunction with the Nnh, Rnnh, STAC, or KMeans classes.

Fields

Method	Description
unsigned int nOrder	
unsigned int nCluster	Cluster Number or Cluster Identifier
double dMeanX	Mean value of the X Co-ordinate of all the points in a particular cluster
double dMeanY	Mean value of the Y Co-ordinate of all the points in a particular cluster
double dAngle	Angle of rotation of the ellipse of a particular cluster
double dXAxis	X Axis in Miles for a particular cluster
double dYAxis	Y Axis in Miles for a particular cluster
double dArea	Area of the ellipse for a particular cluster
double dDensity	
unsigned int nPoints	No of points in a particular cluster

Methods

None

Nearest Neighbor Hierarchical Clustering

Class: CCalcClusterNNH

Synopsis

The nearest neighbor hierarchical clustering (NNH) groups incidents into clusters using the nearest neighbor criterion. CCalcClusterNNH runs the algorithm.

Description

The nearest neighbor hierarchical spatial clustering routine is a constant-distance clustering routine that groups points together on the basis of spatial proximity. The user defines a threshold distance and the minimum number of points that are required for each cluster, and an output size for displaying the clusters with ellipses. The routine identifies first-order clusters, representing groups of points that are closer together than the threshold distance and in which there is at least the minimum number of points specified by the user. Clustering is hierarchical in that the first-order clusters are treated as separate points to be clustered into second-order clusters, and the second-order clusters are treated as separate points to be clustered into third-order clusters, and so on. Higher-order clusters will be identified only if the distances between their centers are closer than the new threshold distance.

The routine outputs six results for each cluster that is calculated:

1. The hierarchical order and the cluster number
2. The mean center of the cluster (Mean X and Mean Y)
3. The standard deviational ellipse of the cluster (the rotation and the lengths of the X and Y axes
4. The number of points in the cluster
5. The area of the cluster
6. The density of the cluster (points divided by area)

The tabular results can be printed, saved to a text file, or output as a '.dbf' file. The user must provide a file name. The graphical results can be output as either ellipses or as

convex hulls (or both) to *ArcView* '.shp' or *MapInfo* '.mif' files. The user must provide a file name. The routine outputs separate graphical objects for the ellipses (NNH) or convex hulls (CNNH) and are distinguished by a prefix placed before the file name. Different orders are distinguished by a number, starting with 1 for first-order clusters. For example, the prefix NNH1 refers to the first-order ellipses while NNH2 refers to the second-order ellipsess (if any). Similarly, CNNH1 refers to the first-order convex hulls while CNNH2 refers to the second-order convex hulls. See Chapter 6 in the *CrimeStat* manual for more information.

Fields

None

Methods

Method	Description
void initialize(CPointSet^ priPointSet, unsigned int minPtsPerCluster, COV_UNIT_TYPE nOutputUnit, double srchRadiusSignificance, double noOfClusterMultiples);	Initializes for NNH calculation. priPointSet - primary point set minPtsPerCluster – Desired minimum Number of Points per cluster nOutputUnit – Unit type for output of Type enumeration Utilities. COV_UNIT_TYPE srchRadiusSignificance – Search Radius significance noOfClusterMultiples – Number of Std. Dev. For ellipses This method must be called before invoking any other methods of this class.
Void PerformCalculations()	This method is the main method that computes the NNH Clusters. This method should be invoked only after a call to initialize method above.
void setFixedDistance(double dFixedDistValue, COV_UNIT_TYPE nFixedDistUnit)	This method is used to set a desired fixed distance for the radius. It takes as parameter the value of the desired radius and the Unit for the value
void setSimulationRuns(unsigned int noOfSimulationRuns)	This method is used to set the number of Monte Carlo simulation runs for NNH routine. It takes as input an unsigned integer which is the number of simulation runs desired.

unsigned int GetClusterCount( )	Returns the number of clusters in the resulting cluster set
Queue^ GetClusters()	Returns a Queue of Clusters. Each object in the Queue is of the class ResultCluster. Please see the documentation for ResultCluster for further details to access its members.
array<double>^ GetSimulationArea()	Returns an array with each element being a double for the area of a particular confidence under Monte Carlo simulation
array<int>^ GetSimulationClusters()	Returns an array with each element being a int for the number of ellipses of a particular confidence under Monte Carlo simulation
array<int>^ GetSimulationPoints()	Returns an array with each element being a int for the number of points in the cluster of a particular confidence under Monte Carlo simulation
array<double>^ GetSimulationDensity()	Returns an array with each element being a double for the density of clusters of a particular confidence under Monte Carlo simulation
void SaveEllipses(System::String ^m_sfilename, int m_iFileType, System::String ^projname, int projnum, int datum)	Intializes the Graphical FileOutput output object to ellipses. It takes the filename, the file type, the projection, projection number and datum. m_iFileType –Shape(2) or MIF(8) or KML(10)
void SaveHull(System::String ^m_sfilename, int m_iFileType, System::String ^projname, int projnum, int datum)	Intializes the Graphical FileOutput output object to convex hulls. It takes the filename, the file type , the projection, projection number and datum. m_iFileType –Shape(2) or MIF(8) or KML(10)
bool SaveToDbf(CFileXBaseWrap ^pFile)	Returns a True value if writing to DBF files was a success, this method writes only the fields.

void setMIFHeader (System::String ^header)	Set the header if output file type is MIF
bool IsClusterFoundInSimulations()	Returns true if clusters are found from simulations, otherwise returns false.

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

'Update Data after mapping columns
file.UpdatePrimaryData()

Dim nnh As New CCalcClusterNNH
nnh.initialize(file.m_pPrimaryPointSet, 4,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES, 0.5, 1)
Dim nSimRuns As Integer
nSimRuns = 3
Dim nnh_dbf As New CFileXBaseWrap
      Dim dirname As String
```

```
        Dim sname As String
        Dim filename As String
        Dim fileutils As New CFileUtilsPub
        fileutils.SplitPath(sdbfname, dirname, sname)
        dirname = fileutils.sDir
        sname = fileutils.sName
        sname = "NNH" + sname
        filename = dirname + sname
        nnh_dbf.Create(filename)
        nnh.SaveToDbf(nnh_dbf)
        nnh.setMIFHeader("CoordSys Earth Projection 8, 79, ""m"", -2, 49, 0.9996012717,
                         400000, -100000)
        nnh.SaveEllipses(smifname, 8, "Earth Projection", 1, 33)
        'nnh.SaveHull(smifname, 2, "Earth Projection", 1, 33)
nnh.setSimulationRuns(nSimRuns)
nnh.PerformCalculations()
nnh_dbf.Close()

System.Console.WriteLine("**********************************Printing Results for
NNH**********************************")
System.Console.WriteLine("No of Ellipses {0:D}", nnh.GetClusterCount())
Dim clusterQ As New Queue
clusterQ = nnh.GetClusters()
System.Console.WriteLine(" Order " & vbTab & " Cluster " & vbTab & " Mean X " &
vbTab & " Mean Y " & vbTab & " Rotation " & vbTab & " X-Axis " & vbTab & " Y-
Axis " & vbTab & " Area " & vbTab & " Points " & vbTab & " Density ")
Dim i As Integer
For i = 0 To nnh.GetClusterCount() - 1 Step 1
    Dim rc As ResultCluster
    rc = clusterQ.Dequeue()
    System.Console.WriteLine(" {0}" & vbTab & "{1}" & vbTab & "{2:F6}" & vbTab
& "{3:F6}" & vbTab & "{4:F6}" & vbTab & "{5:F6}" & vbTab & "{6:F6}" & vbTab &
"{7:F6}" & vbTab & "{8}" & vbTab & "{9:F6}", rc.nOrder, rc.nCluster, rc.dMeanX,
rc.dMeanY, rc.dAngle, rc.dXAxis, rc.dYAxis, rc.dArea, rc.nPoints, rc.dDensity)
Next i

If nSimRuns > 0 Then
    Dim simArea As Double()
    Dim simClust As Integer()
    Dim simPoints As Integer()
```

```
    Dim simDen As Double()
    simArea = nnh.GetSimulationArea()
    simClust = nnh.GetSimulationClusters()
    simPoints = nnh.GetSimulationPoints()
    simDen = nnh.GetSimulationDensity()

    System.Console.WriteLine(" Clusters " & vbTab & " Area " & vbTab & " Points " &
vbTab & " Density ")

    For i = 0 To 10 Step 1
       System.Console.WriteLine(" {0}" & vbTab & "{1:F6}" & vbTab & "{2}" &
vbTab & "{3:F6}", simClust(i), simArea(i), simPoints(i), simDen(i))
     Next i
End If

System.Console.WriteLine("**********************************End Printing
Results for NNH**********************************")
```

Risk-adjusted Nearest Neighbor Hierarchical Clustering

Class: CCalcClusterRNNH

This class inherits from CCalcClusterNNH class.

Synopsis

The risk-adjusted nearest neighbor hierarchical clustering (RNNH) groups incidents into clusters using the nearest neighbor criterion relative to the likelihood of a baseline variable. It is a risk measure. CCalcClusterRNNH runs the algorithm.

Description

The risk-adjusted nearest neighbor hierarchical spatial clustering routine clusters points together on the basis of spatial proximity, but the grouping is adjusted according to the distribution of a baseline variable. The routine requires both a primary file (e.g., robberies) and a secondary file (e.g., population). For the secondary variable, if an intensity or weight variable is to be used, it should be specified.

The user selects a threshold probability for grouping a *pair* of points together by chance and the minimum number of points that are required for each cluster, and an output size for displaying the clusters with ellipses. In addition, a kernel density model for the secondary variable must be specified. The threshold distance is determined by the threshold probability and the grid cell density produced by the kernel density estimate of the secondary variable. Thus, in areas with high density of the secondary variable, the threshold distance is smaller than in areas with low density of the secondary variable.

The routine identifies first-order clusters, representing groups of points that are closer together than the threshold distance and in which there is at least the minimum number of points specified by the user. Clustering is hierarchical in that the first-order clusters are treated as separate points to be clustered into second-order clusters, and the second-order clusters are treated as separate points to be clustered into third-order clusters, and so on. Higher-order clusters will be identified only if the distance between their clustercenters are closer than the new threshold distance.

The routine outputs six results for each cluster that is calculated:

1. The hierarchical order and the cluster number
2. The mean center of the cluster (Mean X and Mean Y)
3. The standard deviational ellipse of the cluster (the rotation and the lengths of the X and Y axes
4. The number of points in the cluster
5. The area of the cluster
6. The density of the cluster (points divided by area)

The tabular results can be printed, saved to a text file, or output as a '.dbf' file. The user must provide a file name. The graphical results can be output as either ellipses or as convex hulls (or both) to *ArcView* '.shp' or *MapInfo* '.mif' files. The user must provide a file name. The routine outputs separate graphical objects for the ellipses (RNNH) or convex hulls (CRNNH) and are distinguished by a prefix placed before the file name. Different orders are distinguished by a number, starting with 1 for first-order clusters. For example, the prefix RNNH1 refers to the first-order ellipses while RNNH2 refers to the second-order ellipsess (if any). Similarly, CRNNH1 refers to the first-order convex hulls while CRNNH2 refers to the second-order convex hulls. See Chapter 6 in the *CrimeStat* manual for more information.

Fields

None

Methods

Method	Description
virtual void initialize(CPointSet^ priPointSet, unsigned int minPtsPerCluster, COV_UNIT_TYPE nOutputUnit, double srchRadiusSignificance, double noOfClusterMultiples, CPointSet^ secPointSet, CRnnhParams^ params, CCalcKernelDensityParams^ kDenParam)	Initializes for NNH calculation. priPointSet - primary point set minPtsPerCluster – Desired minimum Number of Points per cluster nOutputUnit – Unit type for output of Type enumeration Utilities. COV_UNIT_TYPE srchRadiusSignificance – Search Radius significance noOfClusterMultiples – Number of Std. Dev. For ellipses secPointSet – Secondary Point set params – RNNH parameters passed as a CRnnhParams object kDenParam – Parameters for Kernel Density estimation routine passed as a CCalcKernelDensityParams This method must be called before invoking any other methods of this class.
unsigned int GetClusterCount( )	Returns the number of clusters in the resulting cluster set
void SaveEllipses(System::String ^m_sfilename, int m_iFileType, System::String ^projname, int projnum, int datum)	Intializes the Graphical FileOutput object to output ellipses. It takes the filename, the file type , the projection, projection number and datum. m_iFileType –Shape(2) or MIF(8) or KML(10)
void SaveHull(System::String ^m_sfilename, int m_iFileType, System::String ^projname, int projnum, int datum)	Intializes the Graphical FileOutput object to output convex hulls. It takes the filename, the file type, the projection, projection number and datum. m_iFileType –Shape(2) or MIF(8) or KML(10)
bool SaveToDbf(CFileXBaseWrap ^pFile)	Returns a True value if writing to

	DBF files was a success, this method writes only the fields.
void setMIFHeader (System::String ^header)	Set the header if output file type is MIF

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

Dim fileid_s As Integer fileid_s = file.AddSecondaryFile("E:/baltpop.dbf")

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,INPUT_FILTER_TYPE.INP
UT_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
```

```
file.CreateReferenceFile(-76.9, 39.2, -76.32, 39.73)
file.SetCellSpacing(0.01)

'Update Data after mapping columns
file.UpdatePrimaryData()
file.UpdateSecondaryData()
file.UpdateReferenceData()

Dim kernelDenParam As New CCalcKernelDensityParams
Dim RNNHParam As New CRnnhParams
Dim calcRNNH As New CCalcClusterRNNH

kernelDenParam.setKernelSingleUnit(Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_
MILES)
kernelDenParam.setKernelSingleMethod(Interpolation.KERNEL_DENSITY_CALC.KE
RNEL_DENSITY_CALC_NORMAL)
kernelDenParam.setKernelSingleBandwidth(Interpolation.KERNEL_BANDWIDTH.KE
RNEL_BANDWIDTH_ADAPTIVE)
kernelDenParam.setKernelSingleMinSample(100)
kernelDenParam.setKernelSingleInterval(1)
kernelDenParam.setCalculationMethod(Interpolation.KERNEL_SINGLE_CALC.KERN
EL_SINGLE_CALC_REL_DENSITY)
kernelDenParam.setKernelSingleWeight(False)
kernelDenParam.setKernelSingleIntensity(False)
kernelDenParam.setKernelSingleSource(Interpolation.KERNEL_SOURCE.KERNEL_S
OURCE_SECONDARY)
RNNHParam.m_nKernelSensitivity = 50

calcRNNH.initialize(file.m_pPrimaryPointSet, 10,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES, 0.5, 1,
file.m_pSecondaryPointSet, RNNHParam, kernelDenParam)
Dim rnnh_dbf As New CFileXBaseWrap
Dim dirname As String
Dim sname As String
Dim filename As String
Dim fileutils As New CFileUtilsPub
fileutils.SplitPath(sdbfname, dirname, sname)
dirname = fileutils.sDir
sname = fileutils.sName
sname = "RNNH" + sname
```

```
filename = dirname + sname
calcRNNH _dbf.Create(filename)
calcRNNH.SaveToDbf(rnnh_dbf)
calcRNNH.SaveEllipses(smifname, 2, "Earth Projection", 1, 33)
calcRNNH.SaveHull(smifname, 2, "Earth Projection", 1, 33)

calcRNNH.PerformCalculations()
rnnh_dbf.Close()

System.Console.WriteLine("**********Printing Results for RNNH**********")
System.Console.WriteLine("No of Ellipses {0:D}", nnh.GetClusterCount())
Dim clusterQ As New Queue
clusterQ = calcRNNH.GetClusters()
System.Console.WriteLine(" Order " & vbTab & " Cluster " & vbTab & " Mean X " & vbTab & " Mean Y " & vbTab & " Rotation " & vbTab & " X-Axis " & vbTab & " Y-Axis " & vbTab & " Area " & vbTab & " Points " & vbTab & " Density ")
Dim i As Integer
For i = 0 To nnh.GetClusterCount() - 1 Step 1
    Dim rc As ResultCluster
    rc = clusterQ.Dequeue()
    System.Console.WriteLine(" {0}" & vbTab & "{1}" & vbTab & "{2:F6}" & vbTab & "{3:F6}" & vbTab & "{4:F6}" & vbTab & "{5:F6}" & vbTab & "{6:F6}" & vbTab & "{7:F6}" & vbTab & "{8}" & vbTab & "{9:F6}", rc.nOrder, rc.nCluster, rc.dMeanX, rc.dMeanY, rc.dAngle, rc.dXAxis, rc.dYAxis, rc.dArea, rc.nPoints, rc.dDensity)
Next i

If nSimRuns > 0 Then
    Dim simArea As Double()
    Dim simClust As Integer()
    Dim simPoints As Integer()
    Dim simDen As Double()
    simArea = calcRNNH.GetSimulationArea()
    simClust = calcRNNH.GetSimulationClusters()
    simPoints = calcRNNH.GetSimulationPoints()
    simDen = calcRNNH.GetSimulationDensity()

    System.Console.WriteLine(" Clusters " & vbTab & " Area " & vbTab & " Points " & vbTab & " Density ")
```

```
    For i = 0 To 10 Step 1
       System.Console.WriteLine(" {0}" & vbTab & "{1:F6}" & vbTab & "{2}" &
vbTab & "{3:F6}", simClust(i), simArea(i), simPoints(i), simDen(i))
     Next i
End If

System.Console.WriteLine("**********************************End Printing
Results for RNNH**********************************")
```

Parameters Template for Risk-adjusted Nearest Neighbor Hierarchical Clustering

Class: CRnnhParams

Synopsis

CRnnhParams provides a template to pass the parameters for the Risk Adjusted Nearest Neighbor Hierarchical Clustering algorithm.

Description

This class is used in conjunction with the Risk-adjusted Nearest Neighbor Hierarchical clustering class.

Fields

Field	Description
unsigned int m_nKernelSensitivity	Sensitivity for the kernel used to determine the reference grid

Methods

None

Spatial and Temporal Analysis of Crime (STAC) Clustering

Class: CCalcStac

Synopsis

The Spatial and Temporal Analysis of Crime (STAC) routine groups incidents together based on a search circle placed over the nodes of a grid. CCalcStac runs the algorithm.

Description

The Spatial and Temporal Analysis of Crime (STAC) routine is a variable-distance clustering routine. It initially groups points together on the basis of a constant search radius, but then combines clusters that overlap. On the STAC Parameters tab, define a search radius, the minimum number of points that are required for each cluster, and an output size for displaying the clusters with ellipses.

The tabular results can be printed, saved to a text file, or output as a '.dbf' file. The graphical results can be output as either ellipses or as convex hulls (or both). Separate file names must be selected for the ellipse output and for the convex hull output.

The routine outputs eight results for each cluster that is calculated:

1 The hierarchical order and the cluster number
2. The mean center of the cluster (Mean X and Mean Y)
3. The standard deviational ellipse of the cluster (the rotation and the lengths of the X and Y axes
4. The number of points in the cluster
5. The area of the cluster
6. The density of the cluster (cluster points divided by area)
7. The number of points in the ellipse
8. The density of the ellipse (ellipse points divided by area)

The graphical results can be output as either ellipses or as convex hulls (or both) to *ArcView* '.shp' or *MapInfo* '.mif' files. The user must provide a file name. The routine outputs separate graphical objects for the ellipses (ST) or convex hulls (CST) and are

distinguished by a prefix placed before the file name. See Chapter 7 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
void Initialize(CPointSet^ pPointSet, CPointSet^ pSecPointSet,CStacParams^ pStacParams)	Initializes for NNH calculation. priPointSet – Primary point set pSecPointSet – Secondary point set pStacParams – An object of type CStacParams, this is used to pass various parameters for the STAC routine. Please see the detailed documentation below for a description of the fields and methods of this class. This method must be called before invoking any other methods of this class.
void setSimulationRuns(unsigned int noOfSimulationRuns)	Optional method to set the number of monte carlo simulation runs.
bool PerformCalculations()	This method is the main method that computes STAC. Returns false if any error is encountered during execution. This method should be invoked only after a call to initialize method above.
array<ResultCluster^>^ GetResultClusters()	Returns an array of ResultCluster objects corresponding to the clusters computed by the STAC routine.
array<CResultSimulationClusters^>^ GetSimulationPercentiles()	Returns an array of CResultSimulationClusters objects corresponding to the clusters at various percentile values computed by Monte carlo simulation for the STAC routine
void SaveEllipses(System::String ^m_sfilename, int m_iFileType, System::String ^projname, int projnum, int datum)	Intializes the Graphical FileOutput object to output ellipses. It takes the filename, the file type , the projection, projection number and datum. m_iFileType –Shape(2) or MIF(8) or

	KML(10)
void SaveHull(System::String ^m_sfilename, int m_iFileType, System::String ^projname, int projnum, int datum)	Intializes the Graphical FileOutput object to output convex hulls. It takes the filename, the file type, the projection, projection number and datum. m_iFileType –Shape(2) or MIF(8) or KML(10)
bool SaveToDbf(CFileXBaseWrap ^pFile)	Returns a True value if writing to DBF files was a success, this method writes only the fields.
void setMIFHeader (System::String ^header)	Set the header if output file type is MIF
double GetBoundaryMinX()	Returns the lower left X of the boundary.
double GetBoundaryMinY()	Returns the lower left Y of the boundary.
double GetBoundaryMaxX()	Returns the upper left X of the boundary.
double GetBoundaryMaxY()	Returns the upper left Y of the boundary.
int GetPointInsideBoundary()	Returns the nimber of points inside the boundary.

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")
```

```
file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

'Update Data after mapping columns
file.UpdatePrimaryData()
Dim stacParam As New CStacParams
Dim stac As New CCalcStac
Dim stac_dbf As New CFileXBaseWrap
Dim dirname As String
Dim sname As String
Dim filename As String
Dim fileutils As New CFileUtilsPub
fileutils.SplitPath(sdbfname, dirname, sname)
dirname = fileutils.sDir
sname = fileutils.sName
sname = "ST"+sname
filename = dirname + sname
stac_dbf.Create(filename)
stac.SaveToDbf(stac_dbf)
stac.SaveEllipses(smifname, 10, "Earth Projection", 1, 33)
stac.SaveHull(smifname, 10, "Earth Projection", 1, 33)
Dim clusters() As ResultCluster
stacParam.m_dScanRadius = 0.5
stacParam.m_nMinPointPerCluster = 5
stacParam.m_nBoundarySource =
STAC_BOUNDARY_SOURCE.STAC_BOUNDARY_SOURCE_DATA_SET
stacParam.m_nEllipseStdDev = 1
stacParam.m_nScanType =
STAC_SCAN_TYPE.STAC_SCAN_TYPE_RECTANGULAR
stacParam.m_nSearchRadiusUnit =
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES
stacParam.m_nStacOutputUnit =
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES

stac.Initialize(file.m_pPrimaryPointSet, file.m_pPrimaryPointSet, stacParam)
stac.setSimulationRuns(5)
```

```
stac.PerformCalculations()
stac_dbf.Close()
clusters = stac.GetResultClusters
Dim StrArray() As String = {"Min", "0.5", "1.0", "2.5", "5.0", "10.0", "90.0", "95.0",
"97.5", "99.0", "99.5", "Max"}
Dim i As Integer
For i = 0 To clusters.Length - 1 Step 1
    System.Console.WriteLine("Printing Cluster {0}", i + 1)
    System.Console.WriteLine("Mean X {0:F6}", clusters(i).dMeanX)
    System.Console.WriteLine("Mean Y {0:F6}", clusters(i).dMeanY)
    System.Console.WriteLine("Rotation {0:F6}", clusters(i).dAngle)
    System.Console.WriteLine("X-Axis {0:F6}", clusters(i).dXAxis)
    System.Console.WriteLine("Y-Axis {0:F6}", clusters(i).dYAxis)
    System.Console.WriteLine("Points {0:F6}", clusters(i).nPoints)
    System.Console.WriteLine("Area {0:F6}", clusters(i).dArea)
    System.Console.WriteLine("Density {0:F6}", clusters(i).dDensity)
    System.Console.WriteLine("----------------------------------")
Next i

Dim simclusters() As CResultSimulationClusters
simclusters = stac.GetSimulationPercentiles()
For i = 0 To 11 Step 1
    System.Console.WriteLine("Percentile {0}", StrArray(i))
    System.Console.WriteLine("Clusters {0:D}", simclusters(i).nClusters)
    System.Console.WriteLine("Area {0:F6}", simclusters(i).dArea)
    System.Console.WriteLine("Points {0:D}", simclusters(i).nPoints)
    System.Console.WriteLine("Density {0:F6}", simclusters(i).dDensity)
    System.Console.WriteLine("----------------------------------")
Next i
```

Parameters Template for STAC Clustering

Class: CStacParams

Synopsis

CStacParams provides a template to pass the parameters for the Spatial and Temporal Analysis of Crime (STAC) clustering algorithm.

Description

This class is used in conjunction with the STAC routine.

Fields

Field	Description
double m_dScanRadius	Search Radius for the STAC routine
COV_UNIT_TYPE m_nSearchRadiusUnit	Unit of measurement for the search radius
STAC_SCAN_TYPE m_nScanType	Scan type for STAC. Takes in values STAC_SCAN_TYPE. STAC_SCAN_TYPE_RECTANGULAR or STAC_SCAN_TYPE. STAC_SCAN_TYPE_TRIANGULAR
STAC_BOUNDARY_SOURCE m_nBoundarySource;	Source for the boundary. Takes in values: STAC_BOUNDARY_SOURCE. STAC_BOUNDARY_SOURCE_DATA_SET or STAC_BOUNDARY_SOURCE. STAC_BOUNDARY_SOURCE_REFERENCE
unsigned int m_nMinPointPerCluster	Used to specify the minimum number of points per cluster for the STAC routine
Int m_nEllipseStdDev	Used to specify the number of standard deviation for the ellipse

COV_UNIT_TYPE m_nStacOutputUnit	Output unit type
double m_dGridLLX,m_dGridLLY,m_dGridURX,m_dGridURY	Used to specify the lower left and upper right X and Y co-ordinates respectively for the reference grid

Methods

None

STAC Simulation

Class: CResultSimulationClusters

Synopsis

CResultSimulationClusters provides a template to hold the resulting clusters of a STAC simulation routine in Hotspot Analysis.

Description

This class is used in conjunction with the STAC routine.

Fields

Method	Description
unsigned int nClusters	Cluster Number or Cluster Identifier
double dArea	Area of the cluster
double dDensity	Density of the cluster
unsigned int nPoints	Number of points in the cluster

Methods

None

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT
```

```
'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

'Update Data after mapping columns
file.UpdatePrimaryData()

Dim kMeans As New CCalcClusterKMean
Dim kmeans_dbf As New CFileXBaseWrap
Dim dirname As String
Dim sname As String
Dim filename As String
Dim fileutils As New CFileUtilsPub
fileutils.SplitPath(sdbfname, dirname, sname)
dirname = fileutils.sDir
sname = fileutils.sName
sname = "KM"+sname
filename = dirname + sname
kmeans_dbf.Create(filename)
kMeans.SaveToDbf(kmeans_dbf)
kMeans.SaveEllipses(smifname, 2, "Earth Projection", 1, 33)
kMeans.SaveHull(smifname, 2, "Earth Projection", 1, 33)
kMeans.Initialize(file.m_pPrimaryPointSet, 10, 1.0, 1.0)
kMeans.PerformCalculations()
Dim clusters() As ResultKMeansCluster
clusters = kMeans.GetResultClusters()

Dim i As Integer
For i = 0 To clusters.Length - 1 Step 1
   System.Console.WriteLine("Printing Cluster {0}", i + 1)
   System.Console.WriteLine("Mean X {0:F6}", clusters(i).dMeanX)
   System.Console.WriteLine("Mean Y {0:F6}", clusters(i).dMeanY)
```

```
    System.Console.WriteLine("Rotation {0:F6}", clusters(i).dAngle)
    System.Console.WriteLine("X-Axis {0:F6}", clusters(i).dXAxis)
    System.Console.WriteLine("Y-Axis {0:F6}", clusters(i).dYAxis)
    System.Console.WriteLine("Points {0:F6}", clusters(i).nPoints)
    System.Console.WriteLine("Area {0:F6}", clusters(i).dArea)
    System.Console.WriteLine("Sum Of Square {0:F6}", clusters(i).dSumOfSquare)
    System.Console.WriteLine("Mean Square Error {0:F6}",
clusters(i).dMeanSquareError)
    System.Console.WriteLine("----------------------------------")
Next i

System.Console.WriteLine("Total sum of squares {0:F6}",
kMeans.GetTotalSumOfSquares())
System.Console.WriteLine("Total mean squared error {0:F6}",
kMeans.GetTotalMeanSquaredError())
```

K-Means Clustering

Class: CCalcClusterKMean

Synopsis

K-means clustering groups incidents into a user-specified number of clusters, K. CCalcClusterKMean runs the algorithm.

Description

The *K-means* clustering routine (Kmeans) is a partitioning procedure where the data are grouped into *K* groups based on a specified number of seed locations or number of clusters, *K*, defined by the user. The routine tries to find the best positioning of the *K* centers and then assigns each point to the center that is nearest. See chapter 7 for details.

The tabular results can be printed, saved to a text file, or output as a '.dbf' file. The user must provide a file name. The graphical results can be output as either ellipses or as convex hulls (or both) to *ArcView* '.shp' or *MapInfo* '.mif' files. The user must provide a file name. The routine outputs separate graphical objects for the ellipses (KM) or convex hulls (CKM) and are distinguished by a prefix placed before the file name. See Chapter 7 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
Initialize(CPointSet^ pPointSet, unsigned int nClusters, double dClusterSeparation, double m_nKMeanClusterMult)	Initializes for k-means calculation. This method takes in a primary point set, the number of clusters to be found, the cluster separation and the number of multiples for the standard deviations for the ellipses. This method must be called before invoking any

	other methods of this class.
Void PerformCalculations()	This method is the main method that computes the K-Means Clusters. This method should be invoked only after a call to initialize method above.
void UseSecondaryFileForInitialSeeds(CPointSet^ pSecPointSet)	This method sets the point set specified as the initial seeds for the k-Means calculation. This method is optional and is used only when the user needs to specify the initial seeds for the k-Means routine.
array<ResultKMeansCluster^>^ GetResultClusters()	Returns the results of the k-Means hot spot analysis module for the point set specified. The return value is an array of references to ResultKMeansCluster class. The details for each k-Means Cluster can be referenced by using the routines of ResultKMeansCluster.
double GetTotalSumOfSquares()	Returns the total sum of squares of all clusters generated by k-Means. This method should be invoked only after the execution of PerformCalculations method.
double GetTotalMeanSquaredError()	Returns the value of the Total mean squared error of all the clusters. This method should be invoked only after the execution of PerformCalculations method.
String^ GetError()	Returns the error string if the routine terminates due to an error.
void SaveEllipses(System::String ^m_sfilename, int m_iFileType, System::String ^projname, int projnum, int datum)	Intializes the Graphical FileOutput object to output ellipses. It takes the filename, the file type, the projection, projection number and datum. m_iFileType –Shape(2) or MIF(8) or KML(10)
void SaveHull(System::String ^m_sfilename, int m_iFileType, System::String ^projname, int projnum, int datum)	Intializes the Graphical FileOutput object to output convex hulls. It takes the filename, the file type, the projection, projection number and datum. m_iFileType –Shape(2) or MIF(8) or KML(10)

bool SaveToDbf(CFileXBaseWrap ^pFile)	Returns a True value if writing to DBF files was a success, this method writes only the fields.
void setMIFHeader (System::String ^header)	Set the header if output file type is MIF

K-Means Cluster Analysis Results

Class: ResultKMeansCluster

This class inherits from ResultCluster class.

Synopsis

ResultKMeansCluster provides a template to hold the resulting clusters of K-Means rourine in Hotspot Analysis.

Description

This routine is used in conjunction with the KMeans class.

Fields

Field	Description
double dSumOfSquare	Value of sum of squares for a particular cluster
double dMeanSquareError	Value of Mean Square error for a particular cluster

Methods

None

Anselin’s Local Moran

Class: CCalcLocalMoran

Synopsis

Anselin’s Local Moran tests for spatial autocorrelation in individual zones with the Moran’s “I” statistic. CCalcLocalMoran runs the algorithm.

Description

Anselin's Local Moran statistic applies the Moran's "I" statistic to individual points (or zones) to assess whether particular points/zones are spatially related to the nearby points (or zones.) The statistic requires an intensity variable in the primary file. The individual “I” values typically vary from -1 to +1 with 0 indicating no clusterings, though the individual range varies. Unlike the global Moran's "I" statistic, the local Moran is applied to individual zones. The index points to clustering or dispersion relative to the local neighborhood. Zones with high "I" values have an intensity value that is higher than their neighbors while zones with low "I" values have intensity values lower than their neighbors.
The tabular results can be printed, saved to a text file, or output as a '.dbf' file. The user must provide a file name. See Chapter 7 for more information.

Fields

None

Methods

Method	Description
void Initialize(CPointSet^ pPointSet, bool m_bLocalMoranVariance,bool m_bAdjLocalMoran, int nSimulations)	Initializes for Anselin Local Moran calculation. priPointSet - primary point set m_bLocalMoranVariance – Boolean parameter to adjust for variance m_bAdjLocalMoran – Boolean parameter to adjust for small distances. True indicates that the adjustment has to be done for small distances. nSimulations- Number of simulations. This method must be called before invoking any other methods of this class.
Void PerformCalculations()	This method is the main method that computes the Anselin Local Morans' Statistic. This method should be invoked only after a call to initialize method above.
unsigned int GetRowCount( )	Returns the number of rows in the result of the calculation
double GetX(int nRow)	Get the value of X Co-ordinate for row specified by nRow
double GetY(int nRow)	Get the value of Y Co-ordinate for row specified by nRow
double GetI(int nRow)	Get the value of Anselin Local Moran's I for row specified by nRow
double GetExpectedI(int nRow)	Get the value of Expected Anselin Local Moran's I for row specified by nRow
double GetVariance(int nRow)	Get the value of variance of Local Moran's for row specified by nRow when m_bLocalMoranVariance is set true in the Initialize function.
bool SaveToDbf(CFileXBaseWrap ^pFile)	Returns a True value if writing to DBF files was a success, this method writes only the fields.
double GetStandardizedI(int nRow)	Get the value of Standardized Local Moran I's for row specified by nRow when

	m_bLocalMoranVariance is set true in the Initialize function.
double GetLocalMoranIB99PctLow(int nRow)	Get the 99th percentile low indices value for row specified by nRow . The values are valid only if Number of simulations is > 0.
double GetLocalMoranIB95PctLow(int nRow)	Get the 95th percentile low indices value for row specified by nRow . The values are valid only if Number of simulations is > 0.
double GetLocalMoranIB95PctHigh(int nRow)	Get the 95th percentile high indices value for row specified by nRow . The values are valid only if Number of simulations is > 0.
double GetLocalMoranIB99PctHigh(int nRow)	Get the 99th percentile high indices value for row specified by nRow . The values are valid only if Number of simulations is > 0.
System::String^ GetResult()	Returns formatted Moran'I statistics as a String

---

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
```

```
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
file.SetIntensity(fileid, cboxInt.SelectedIndex)

'Update Data after mapping columns
file.UpdatePrimaryData()

Dim kLocalMoran As New CCalcLocalMoran

kLocalMoran.Initialize(file.m_pPrimaryPointSet, True, True)
Dim lm_dbf As New CFileXBaseWrap
Dim dirname As String
Dim sname As String
Dim filename As String
Dim fileutils As New CFileUtilsPub
fileutils.SplitPath(savefilename, dirname, sname)
dirname = fileutils.sDir
sname = fileutils.sName
sname = "LM " + sname
filename = dirname + sname
Dim success As Boolean
success = lm_dbf.Create(filename)
success = success And kLocalMoran.SaveToDbf(lm_dbf)

kLocalMoran.PerformCalculations()
lm_dbf.Close()

Dim i As Integer
System.Console.WriteLine(" Cluster " & vbTab & " X Co-ordinate" & vbTab & " Y Co-
ordinate " & vbTab & " I " & vbTab & " Expected I " & vbTab & " Variance of I " &
vbTab & "Standardized I" )
For i = 0 To kLocalMoran.GetRowCount() - 1 Step 1
    System.Console.WriteLine(" {0}" & vbTab & "{1:F6}" & vbTab & "{2:F6}" &
vbTab & "{3:F6}" & vbTab & "{4:F6}" & vbTab & "{5:F6}" & vbTab & "{6:F6}", i +
1, kLocalMoran.GetX(i), kLocalMoran.GetY(i), kLocalMoran.GetI(i),
kLocalMoran.GetExpectedI(i), kLocalMoran.GetVariance(i),
kLocalMoran.GetStandardizedI(i))
Next i
```

Getis-Ord Local “G”

Class: CCalcGetisOrdLocalZone

Synopsis

Getis-Ord Local “G” tests for spatial autocorrelation in individual zones with the Getis-Ord “G” statistic. CCalcGetisOrdLocalZone runs the algorithm.

Description

The Getis-Ord Local G statistic applies the Getis-Ord "G" statistic to individual zones to assess whether particular points are spatially related to the nearby points. Unlike the global Getis-Ord “G”, the Getis-Ord Local “G” is applied to each individual zone. The statistic requires an intensity variable in the primary file.
The tabular results can be printed, saved to a text file, or output as a '.dbf' file. The user must provide a file name. See Chapter 7 for more information.

Fields

None

Methods

Method	Description
void Initialize(CPointSet^ pPointSet, double dSearchDistance ,COV_UNIT_TYPE nOutputUnit, int nSimulations)	Initializes for Getis-Ord Local “G” calculation. pPointSet - primary point set. dSearchDistance –the value of search distance. nOutputUnit –the unit of output. This method must be called before invoking any other methods of this class. nSimulations-number of simulations.
Void PerformCalculations()	This method is the main method that computes the Getis-Ord Local “G” Statistic. This method should be invoked only after a call to initialize method above.
unsigned int GetRowCount( )	Returns the number of rows in the result of the calculation
double GetX(int nRow)	Get the value of X Co-ordinate for row specified by nRow
double GetY(int nRow)	Get the value of Y Co-ordinate for row specified by nRow
double GetG(int nRow)	Get the value of Getis-Ord Local “G” for row specified by nRow
double GetExpectedG(int nRow)	Get the value of Expected Getis-Ord Local “G” for row specified by nRow
double GetVariance(int nRow)	Get the value of variance of Getis-Ord Local “G” for row specified by nRow
bool SaveToDbf(CFileXBaseWrap ^pFile)	Returns a True value if writing to DBF files was a success, this method writes only the fields.
double GetStd(int nRow)	Get the value of Standard deviation of Getis-Ord Local “G”’s for row

	specified by nRow
double GetZ(int nRow)	Get the value of Z of Getis-Ord Local “G” for row specified by nRow
double GetGetisOrdLocalZoneGB99PctLow(int nRow)	Get the 99th percentile low indices value for row specified by nRow . The values are valid only if Number of simulations is > 0.
double GetGetisOrdLocalZoneGB95PctLow(int nRow)	Get the 95th percentile low indices value for row specified by nRow . The values are valid only if Number of simulations is > 0.
double GetGetisOrdLocalZoneGB95PctHigh(int nRow)	Get the 95th percentile high indices value for row specified by nRow . The values are valid only if Number of simulations is > 0.
double GetGetisOrdLocalZoneGB99PctHigh(int nRow)	Get the 99th percentile high indices value for row specified by nRow . The values are valid only if Number of simulations is > 0.
System::String^ GetResult()	Returns formatted Getis-Ord “G” statistics as a String

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT
```

```
'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
file.SetIntensity(fileid, cboxInt.SelectedIndex)

'Update Data after mapping columns
file.UpdatePrimaryData()

Dim localGetisG As New CCalcGetisOrdLocalZone
localGetisG.Initialize(file.m_pPrimaryPointSet, 2,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES, 5)
localGetisG.PerformCalculations()
MsgBox(localGetisG.GetResult())
```

Single-kernel Density Interpolation

Class: CCalcKernelDensity

Synopsis

The single-kernel density interpolation routine interpolates data from individual locations to grid cells using a mathematical interpolator. The result is either an estimate of density in each grid cell or an estimate of probability. CCalcKernelDensity runs the algorithm.

Description

Interpolation allows estimates of point density using the kernel density smoothing method. There are two types of kernel density smoothing, one applied to a single distribution of points and the other that compares two different distributions. Each type has variations on the method that can be selected. Both types require a reference file that is overlaid on the study area. The kernels are placed over each point and the distance between each reference cell and each point are evaluated by the kernel function. The individual kernel estimates for each cell are summed to produce an overall estimate of density for that cell. The intensity and weighting variables can be used in the kernel estimate. The densities can be converted into probabilities.

The single kernel density routine estimates the density of points for a single distribution by overlaying a symmetrical surface over each point, evaluating the distance from the point to each reference cell by the kernel function, and summing the evaluations at each reference cell.

The graphical results can be output to *ArcView* '.shp' or *MapInfo* '.mif' files. The user must provide a file name. The routine places a prefix (K) before the file name. See Chapter 8 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
Initialize(CPointSet ^pPointSet, CPointSet ^pSecPointSet, CPointSet ^pRefPointSet, CCalcKernelDensityParams^ kDenParam)	Initializes for NNH calculation. pPointSet – Primary point set pSecPointSet – Secondary point set, used only if the source for the single kernel is Secondary. pRefPointSet – Reference Point set, mandatory parameter kDenParam – An object of type CCalcKernelDensityParams, this is used to pass various parameters for the kernel density estimation routine. For more details on passing parameters for kernel density estimation please refer to the documentation of CCalcKernelDensityParams. This method must be called before invoking any other methods of this class.
void PerformCalculations(int m_filetype,System::String ^filename)	This method is the main method that computes Single Kernel Density. Returns false if any error is encountered during execution. This method should be invoked only after a call to initialize method above. Parameters m_filetype – Shape(2) or MIF(8) or KML(10) filename – file name for the target shape/MIF file.
unsigned int getNumberOfDensityPoints()	Returns the number of density points in the kernel density estimation.
array<double>^ getDensityValues()	Returns an array of double corresponding to the density values at various points of the dataset.

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
```

```
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

'Update Data after mapping columns
file.UpdatePrimaryData()
file.UpdateReferenceData()

Dim kernelDenParam As New CCalcKernelDensityParams
Dim kernelden As New CCalcKernelDensity

kernelDenParam.setKernelSingleUnit(Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_
MILES)
kernelDenParam.setKernelSingleSource(Interpolation.KERNEL_SOURCE.KERNEL_S
OURCE_PRIMARY)
kernelDenParam.setKernelSingleMethod(Interpolation.KERNEL_DENSITY_CALC.KE
RNEL_DENSITY_CALC_NORMAL)
kernelDenParam.setKernelSingleBandwidth(Interpolation.KERNEL_BANDWIDTH.KE
RNEL_BANDWIDTH_ADAPTIVE)
kernelDenParam.setKernelSingleMinSample(100)
kernelDenParam.setKernelSingleInterval(1)
kernelDenParam.setCalculationMethod(Interpolation.KERNEL_SINGLE_CALC.KERN
EL_SINGLE_CALC_REL_DENSITY)
kernelDenParam.setKernelSingleWeight(False)
kernelDenParam.setKernelSingleIntensity(False)
```

```
kernelden.Initialize(file.m_pPrimaryPointSet, file.m_pPrimaryPointSet,
file.m_pReferencePointSet, kernelDenParam)
savefilename = "K"+savefilename
kernelden.PerformCalculations(10, savefilename)
Dim kernelDensityRes() As Double
kernelDensityRes = kernelden.getDensityValues()

System.Console.WriteLine("**********************************Printing Results for
Kernel Density Estimation**********************************")
Dim i As Integer
For i = 0 To kernelden.getNumberOfDensityPoints() - 1 Step 1
   System.Console.WriteLine("Density for Location {0:D} {1:E}", i,
kernelDensityRes(i))
Next i
System.Console.WriteLine("********************************End Printing Results
for Kernel Density Estimation**********************************")
```

Duel-kernel Density Interpolation

Class: CCalcKernelRatio

This class inherits from CCalcKernelDensity, hence all public member of CCalcKernelDensity are also accessible through this class. For more details on the public members in CCalcKernelDensity, please refer to the documentation of CCalcKernelDensity above.

Synopsis

The duel-kernel density interpolation routine interpolates incident data from individual locations to grid cells using a mathematical interpolator and interpolates baseline data to the same grid cells. It then compares the two distributions, either as ratio, differences or sums. CCalcKernelRatio runs the algorithm.

Description

The dual kernel density routine compares two different distributions involving the primary and secondary files. A 'first' file and 'second' file need to be defined. The comparison allows the ratio of the first file divided by the second file, the logarithm of the ratio of the first file divided by the second file, the difference between the first file and second file (i.e., first file – second file), or the sum of the first file and the second file.

The graphical results can be output to *ArcView* '.shp' or *MapInfo* '.mif' files. The user must provide a file name. The routine places a prefix (DK) before the file name. See Chapter 8 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
Initialize(CPointSet ^pPointSet, CPointSet ^pSecPointSet, CPointSet ^pRefPointSet, CCalcKernelDensityParams^ kDenParam)	Initializes for Kernel density ratio calculation. This method overrides the initialize method of CCalcKernelDensity class. pPointSet – Primary point set pSecPointSet – Secondary point set, used only if the source for the single kernel is Secondary. pRefPointSet – Reference Point set, mandatory parameter kDenParam – An object of type CCalcKernelDensityParams, this is used to pass various parameters for the kernel density estimation routine. For more details on passing parameters for kernel density estimation please refer to the documentation of CCalcKernelDensityParams. This method must be called before invoking any other methods of this class.

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")
```

```
file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

'Update Data after mapping columns
file.UpdatePrimaryData()

file.SetY(fileid_s, cBoxLatSec.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)
file.SetX(fileid_s, cBoxLonSec.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)
file.SetIntensity(fileid_s, cboxIntSec.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_SECONDARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_NULL)

file.UpdateSecondaryData()
file.UpdateReferenceData()

Dim kernelDenParam As New CCalcKernelDensityParams
Dim kernelden As New CCalcKernelRatio

kernelDenParam.setKernelSingleUnit(Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_
MILES)
kernelDenParam.setKernelDualUnit(Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_M
ILES)

kernelDenParam.setKernelSingleSource(Interpolation.KERNEL_SOURCE.KERNEL_S
OURCE_PRIMARY)

kernelDenParam.setKernelDualSource(Interpolation.KERNEL_SOURCE.KERNEL_SO
URCE_PRIMARY)

kernelDenParam.setKernelSingleMethod(Interpolation.KERNEL_DENSITY_CALC.KE
RNEL_DENSITY_CALC_NORMAL)
```

```
kernelDenParam.setKernelSingleBandwidth(Interpolation.KERNEL_BANDWIDTH.KE
RNEL_BANDWIDTH_ADAPTIVE)
kernelDenParam.setKernelSingleMinSample(100)
kernelDenParam.setKernelSingleInterval(1)
kernelDenParam.setKernelDualInterval(1)

kernelDenParam.setDualCalculationMethod(Interpolation.KERNEL_DUAL_CALC.KE
RNEL_DUAL_CALC_RATIO)
kernelDenParam.setKernelSingleWeight(False)
kernelDenParam.setKernelSingleIntensity(False)
kernelDenParam.setKernelDualWeight(False)
kernelDenParam.setKernelDualIntensity(True)

kernelDenParam.setKernelSingleOutUnit(Utilities.COV_UNIT_TYPE.COV_UNIT_TY
PE_MILES)

kernelden.Initialize(file.m_pPrimaryPointSet, file.m_pSecondaryPointSet,
file.m_pReferencePointSet, kernelDenParam)
savefilename = "DK"+savefilename
kernelden.PerformCalculations(2, savefilename)

Dim kernelDensityRes() As Double
kernelDensityRes = kernelden.getDensityValues()

System.Console.WriteLine("**********************************Printing Results for
Kernel Density Estimation**********************************")
Dim i As Integer

For i = 0 To kernelden.getNumberOfDensityPoints() - 1 Step 1
    System.Console.WriteLine("Density for Location {0:D} {1:E}", i,
kernelDensityRes(i))
Next i

System.Console.WriteLine("********************************End Printing Results
for Kernel Density Estimation**********************************")
```

Space-time Analysis Library

The Space-time Analysis library is a set of routines for examining the interaction between spatial location and time. The library provides two groups of classes of various statistical operations as given below:

1. Spatial-temporal autocorrelation – these test whether there is an interaction between spatial location and time. There are two statistical functions, each of which may have supplementary class functions:

   A. Knox Index
   B. Mantel Index

2. Correlated Walk Analysis (CWA) – these examine the spatial and temporal behavior of a serial offender. There are three statistical functions that make up this model:

   A. CWA – Correlogram: Examines whether there is any repeatability in the actions of a serial offender by distance traveled, direction, and time interval
   B. CWA – Regression: Allows the estimation of repeating behavior by a serial offender for distance traveled, direction, and time interval
   C. CWA – Prediction: Allows the prediction of the next crime event committed by a serial offender for distance traveled, direction, and time interval using either a regression model, the mean or median.

3. Spatial-temporal Moving Average – "I need your help here"

See Chapter 9 in the CrimeStat manual for more information.

Library: **Space-Time Analysis.dll**

Prerequisites: **Crimestat Core Components.dll**
Spatial Description.dll

Namespace: Space-Time Analysis

The Space-Time analysis namespace provides access to the Knox Index, Mantel Index and Correlated Walk Analysis routines.

Field	Description
enum KNOX_INDEX_CLOSENESS_METHOD	Method for Closeness in Knox Index Calculation. Valid Values: KNOX_INDEX_CLOSENESS_METHOD_MEAN, // Mean. KNOX_INDEX_CLOSENESS_METHOD_MEDIAN, // Median KNOX_INDEX_CLOSENESS_METHOD_CUSTOM, // Custom Method
enum REGRESSION_VARIABLE	Regression variable for Correlated Walk Analysis REGRESSION_VARIABLE_DISTANCE, REGRESSION_VARIABLE_TIME, REGRESSION_VARIABLE_BEARING

Field	Description
enum PREDICTION_METHOD	Method for prediction in CWA. Valid Values: PREDICTION_METHOD_MEAN, // Mean. PREDICTION_METHOD_MEDIAN, // Median PREDICTION_METHOD_REGRESSION,

Knox Index

Class: CCalcKnoxIndex

Synopsis

The Knox Index tests whether there is a relationship between closeness in time and closeness in spatial location. CCalcKnoxIndex calculates the algorithm.

Description

The Knox index is an index showing the relationship between 'closeness in time' and 'closeness in distance'. Pairs of events are compared in distance and in time and are represented as a 2 x 2 table. If there is a relationship, it would normally be positive, that is events that are close together in space (i.e., in distance) are also occurring in a short time span. There are three methods for defining closeness in time or in distance:

1. Mean. That is, events that are closer together than the mean time interval or are closer together than the mean distance are defined as 'Close' whereas events that are father together than mean time interval or are farther together than the mean distance are defined as 'Not close'. This is the default.

2. Median. That is, events that are closer together than the median time interval or are closer together than the median distance are defined as 'Close' whereas events that are father together than median time interval or are farther together than the median distance are defined as 'Not close'.

3. User defined. The user can specify any value for distinguishing 'Close' and 'Not close' for either time or distance.

The output includes a 2 x 2 table of the distribution of pairs categorized as 'Close' or 'Not close' in time and in distance. The output also includes a table of the expected of the distribution of pairs on the assumption that events in time are space are independent of each other. Finally, the output includes a chi-square test of the differences between the observed and expected distributions.

Note, that since pairs are being compared, independence of observations is not true and a usual p-value associated with the chi-square test cannot be properly calculated. A Monte

Carlo simulation can be run to estimate the approximate Type I error probability levels for the Knox index. The user specifies the number of simulation runs. Data are randomly assigned and the chi-square value for the Knox index is calculated for each run. The random output is sorted and percentiles are calculated. Twelve percentiles are identified for this index:

1. The minimum for the spatially random Knox chi-square
2. The maximum for the spatially random Knox chi-square
3. The 0.5 percentile for the spatially random Knox chi-square
4. The 1 percentiles for the spatially random Knox chi-square
5. The 2.5 percentile for the spatially random Knox chi-square
6. The 5 percentile for the spatially random Knox chi-square
7. The 10 percentile for the spatially random Knox chi-square
8. The 90 percentile for the spatially random Knox chi-square
9. The 95 percentile for the spatially random Knox chi-square
10. The 97.5 percentile for the spatially random Knox chi-square
11. The 99 percentile for the spatially random Knox chi-square
12. The 99.5 percentile for the spatially random Knox chi-square

The tabular results can be printed. The user must provide a file name. See Chapter 9 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
initialize(CPointSet^ pPriPointSet,KNOX_INDEX_CLOSENESS_METHOD nKnoxCloseMethod,double dKnoxCloseDist, COV_UNIT_TYPE nKnoxCloseDistUnit,double dKnoxCloseTime, TIME_UNIT_TYPE nKnoxCloseTimeUnit)	Initializes the object for Knox Index calculation. This method takes in the following parameters pPriPointSet – object of type CPointSet corresponding to the primary point set nKnoxCloseMethod – Closeness method for the Knox Index calculation. For valid values please refer to the documentation of enum class KNOX_INDEX_CLOSENESS_METHOD dKnoxCloseDist – value of the “close” distance nKnoxCloseDistUnit – Unit type for the “close distance” dKnoxCloseTime – value of the “close” time nKnoxCloseTimeUnit – Unit type for the “close” time. This method must be called before invoking any other methods of this class.
setSimulationRuns(unsigned int nSimRuns)	An optional method to set the number of simulation runs. By default the number of simulation runs is 0. This method should be called before PerformCalculations() is called.
Void PerformCalculations()	This method is the main method that computes the Knox Index. This method should be invoked only after a call to initialize method above.
double getCloseDistance()	Returns the value of the “close” Distance after PerformCalculations is invoked to calculate the Knox Index
double getCloseTime()	Returns the value of the “close” time after PerformCalculations is invoked to calculate the Knox Index
double getChiSquare()	Returns an array of double, the values of

	which correspond to the close distance time pairs as follows Index 0 – Close in Space and Time Index 1 – Not close in Space but close in Time Index 2 – Close in Space but not close in time Index 3 – Not close in space and time
array<double>^ getKnoxIndexCloseDistTimePairs()	Returns an array of double, the values of which correspond to the expected close distance time pairs as follows Index 0 – Close in Space and Time Index 1 – Not close in Space but close in Time Index 2 – Close in Space but not close in time Index 3 – Not close in space and time
array<double>^ getKnoxIndexExpectedCloseDistTimePairs()	Returns the Chi-Square values for the simulation runs as an array in the order for min, 0.5, 1.0, 2.5, 5.0, 10.0, 90.0, 95.0, 97.5, 99.0, 99.5 and max
array<double>^ getSimulationPercentiles()	

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
```

```
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

'Update Data after mapping columns
file.UpdatePrimaryData()

Dim knoxInd As New CCalcKnoxIndex
 knoxInd.initialize(file.m_pPrimaryPointSet,
SpaceTimeAnalysis.KNOX_INDEX_CLOSENESS_METHOD.KNOX_INDEX_CLOS
ENESS_METHOD_MEAN, 1,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES, 1,
Utilities.TIME_UNIT_TYPE.TIME_UNIT_TYPE_DAYS)
 knoxInd.setSimulationRuns(5)
 knoxInd.PerformCalculations()

Dim timeDistPairs() As Double
Dim expTimeDistPairs() As Double
timeDistPairs = knoxInd.getKnoxIndexCloseDistTimePairs()
expTimeDistPairs = knoxInd.getKnoxIndexExpectedCloseDistTimePairs()

System.Console.WriteLine("**********************************Printing Results for
Knox Index**********************************")
System.Console.WriteLine("Close time: {0:F6}", knoxInd.getCloseTime())
System.Console.WriteLine("Close Distance: {0:F6}", knoxInd.getCloseDistance())
System.Console.WriteLine("Close in Time and Close in Space: {0:F6}", timeDistPairs(0)
)
System.Console.WriteLine("Close in Time and Not Close in Space: {0:F6}",
timeDistPairs(1) )
System.Console.WriteLine("Not close in Time and Close in Space: {0:F6}",
timeDistPairs(2) )
System.Console.WriteLine("Not close in Time and Close in Space: {0:F6}",
timeDistPairs(3) )
System.Console.WriteLine("Expected Values")
```

```
System.Console.WriteLine("Close in Time and Close in Space: {0:F6}",
expTimeDistPairs (0) )
System.Console.WriteLine("Close in Time and Not Close in Space: {0:F6}",
expTimeDistPairs (1) )
System.Console.WriteLine("Not close in Time and Close in Space: {0:F6}",
expTimeDistPairs (2) )
System.Console.WriteLine("Not close in Time and Close in Space: {0:F6}",
expTimeDistPairs (3) )

Dim knoxIndSimPerc() As Double
Dim StrArray() As String = {"Min", "0.5", "1.0", "2.5", "5.0", "10.0", "90.0", "95.0",
"97.5", "99.0", "99.5", "Max"}
knoxIndSimPerc = knoxInd.getSimulationPercentiles()
Dim i As Integer
For i = 0 To 11 Step 1
        System.Console.WriteLine("Percentile {0}", StrArray(i))
        System.Console.WriteLine("Value {0:F6}", knoxIndSimPerc (i))
        System.Console.WriteLine("----------------------------------")
Next i
System.Console.WriteLine("**********************************End Printing Results
for Knox Index**********************************")
```

Mantel Index

Class: CCalcMantelIndex

Synopsis

The Mantel Index tests whether there is a correlation between closeness in time and closeness in spatial location. CCalcMantelIndex calculates the algorithm.

Description

The Mantel index is the correlation between closeness in time and closeness in distance across pairs. Each pair of events is compared for the time interval and the distance between them. If there is a positive relationship between closeness in time and closeness in space (distance), then there should be a sizeable positive correlation between the two measures.

A Monte Carlo simulation can be run to estimate the approximate confidence intervals around the Mantel correlation. The user specifies the number of simulation runs and the Mantel index is calculated for randomly assigned data. The random output is sorted and percentiles are calculated. Twelve percentiles are identified for this index:

1. The minimum for the spatially random Mantel index
2. The maximum for the spatially random Mantel index
3. The 0.5 percentile for the spatially random Mantel index
4. The 1 percentiles for the spatially random Mantel index
5. The 2.5 percentile for the spatially random Mantel index
6. The 5 percentile for the spatially random Mantel index
7. The 10 percentile for the spatially random Mantel index
8. The 90 percentile for the spatially random Mantel index
9. The 95 percentile for the spatially random Mantel index
10. The 97.5 percentile for the spatially random Mantel index
11. The 99 percentile for the spatially random Mantel index
12. The 99.5 percentile for the spatially random Mantel index

The tabular results can be printed. The user must provide a file name. See Chapter 9 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
virtual void initialize(CPointSet^ pPriPointSet, bool pAdjustMantelDistances,Utilities::TIME_UNIT_TYPE pTimeUnit)	Initializes the object for Mantel Index calculation. This method takes in a primary point set, a Boolean indicating whether the Distances need to be adjusted and the time unit type. This method must be called before invoking any other methods of this class.
Void PerformCalculations()	This method is the main method that computes the mantel index. This method should be invoked only after a call to initialize method above.
virtual void setSimulationRuns(unsigned int nSimRuns)	An optional method to set the number of simulation runs. By default the number of simulation runs is 0. This method should be called before PerformCalculations() is called.
double getMantelIndexValue()	Returns the value of Mantel Index for the point set specified.
array<double>^ getSimulationPercentiles()	Returns the Mantel Index values for the simulation runs as an array in the order for min, 0.5, 1.0, 2.5, 5.0, 10.0, 90.0, 95.0, 97.5, 99.0, 99.5 and max

Example in Visual Basic

```
'Create Object
Dim file As New CInputParam
'Initialize values
file.m_iDistanceType =
Utilities::DISTANCE_TYPE::DISTANCE_TYPE_PROJECTED
file.m_iDataUnit = Utilities::UNIT_TYPE::UNIT_TYPE_FEET
file.m_iTimeUnit = Utilities::TIME_UNIT_TYPE::TIME_UNIT_TYPE_DAYS
file.m_iMeasureType = Utilities::MEASURE_TYPE::MEASURE_TYPE_DIRECT

'Add Primary File
file.m_fileType = FILE_TYPE::FILE_TYPE_XBASE
Dim fileid As Integer fileid = file.AddPrimaryFile("E:/crime.dbf")

file.SetY(fileid, cboxLat.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,INPUT_FILTER_TYPE.INPUT
_FILTER_TYPE_BLANK)
file.SetX(fileid, cboxLon.SelectedIndex,
INPUT_FILE_TYPE.INPUT_FILE_TYPE_PRIMARY,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)

'Update Data after mapping columns
file.UpdatePrimaryData()

Dim mantelInd As New CCalcMantelIndex
mantelInd.initialize(file.m_pPrimaryPointSet, False, file.m_iTimeUnit)
mantelInd.setSimulationRuns(5)
mantelInd.PerformCalculations()

System.Console.WriteLine("**********************************Printing Results for
Mantel Index**********************************")
System.Console.WriteLine("Mantel Index {0:F6} ", mantelInd.getMantelIndexValue())
Dim mantelSimPerc() As Double
Dim StrArray() As String = {"Min", "0.5", "1.0", "2.5", "5.0", "10.0", "90.0", "95.0",
"97.5", "99.0", "99.5", "Max"}
mantelSimPerc = mantelInd.getSimulationPercentiles()
```

```
Dim i As Integer
For i = 0 To 11 Step 1
        System.Console.WriteLine("Percentile {0}", StrArray(i))
        System.Console.WriteLine("Clusters {0:F6}", mantelSimPerc(i))
        System.Console.WriteLine("---------------------------------")
Next i
System.Console.WriteLine("**********************************End Printing Results
for Mantel Index**********************************")
```

Correlated Walk Analysis

Correlated Walk Analysis (CWA) analyzes the sequential movements of a serial offender and makes predictions about the time and location of the next event. Sequential movements are analyzed in terms of three parameters: Time difference between events (e.g., the number of days between two consecutive events), Distance between events – the distance between two consecutive events, and Bearing (direction) between events – the angular direction between two consecutive events in degrees (from 0 to 360).

There are three CWA routines for analyzing sequential events:

1. Correlogram
2. Regression diagnostics
3. Prediction

See Chapter 9 of the CrimeStat manual for more information.

Correlated Walk Analysis Correlogram

Class: CCalcCorrelatedWalkCgram

Synopsis

The Correlated Walk Analysis Correlogram examines whether there are any repetitions in the sequential crimes/events committed by a serial offender in terms of distance, direction, and time interval. CCalcCorrelatedWalkCgram provides the object for the correlogram statistics of Correlated Walk analysis. This class derives from CCalcCorrelatedWalk.

Description

The CWA Correlogram presents the lagged correlations between events for time difference, distance traveled, and bearing (direction). The lags are the sequential comparisons. A lag of 0 is the sequence compared with itself; by definition, the correlation is 1.0. A lag of 1 is the sequence compared with the previous sequence. A lag of 2 is the sequence compared with two previous sequences. A lag of 3 is the sequence compared with three previous sequences, and so forth. In total, comparisons are made up to seven previous sequences (a lag of 7).

Typically, for time difference, distance and location separately, the lag with the highest correlation is the strongest. However, with each consecutive lag, the sample size decreases by one and a high correlation associated with a high lag comparison can be unreliable if the sample size is small. Consequently, the adjusted correlogram discounts the correlations by the number of lags.

The tabular results can be printed. The user must provide a file name. See Chapter 9 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
initialize(CPointSet^ pPriPointSet,)	Initializes the object with the primary pointset. This function should be called before invoking any other functions of the object.
bool PerformCalculations()	Performs the correlelogram calculation. This function should be invoked only after initialize function has been invoked. Returns true on successful computation.
int getMaximumLag()	Returns back the value of maximum lag
double getFinalDistanceToOrigin()	Returns back the value of final distance to origin in meters (distance between last and first event)
double getRandomWalkDistance()	Expected random walk distance from origin (if sequence was strictly random)
double getDrift()	Drift (the ratio of actual distance from origin to expected random walk distance)
double getFinalBearingToOrigin()	Final bearing from origin (direction between last event and first event)
double getExpectedRandomWalkBearing()	Expected random walk bearing. Defined as 0 because there is no expected direction.

array<double>^ getTime()	Returns an array of double of size equal to the return value of getMaximumLag() corresponding to correlations by lag for time.
array<double>^ getCorrelationDistance()	Returns an array of double of size equal to the return value of getMaximumLag() corresponding to correlations by lag for distance.
array<double>^ getBearing()	Returns an array of double of size equal to the return value of getMaximumLag() corresponding to correlations by lag for bearing.
array<double>^ getAdjustedTime ()	Returns an array of double of size equal to the return value of getMaximumLag() corresponding to adjusted correlations by lag for time.
array<double>^ getAdjustedCorrelationDistance ()	Returns an array of double of size equal to the return value of getMaximumLag() corresponding to adjusted correlations by lag for distance.
bool SaveToDbf(CFileXBaseWrap ^pFile)	Returns a True if writing to a dBase(.dbf) file was a success. Requires a parameter of object type CFileXBaseWrap.
array<double>^ getAdjustedBearing ()	Returns an array of double of size equal to the return value of getMaximumLag() corresponding to adjusted correlations by lag for bearing.

Example in Visual Basic

```
'Create an InputParam object and specify Primary point set

Dim cwaCGram As New CCalcCorrelatedWalkCgram
cwaCGram.initialize(file.m_pPrimaryPointSet)
cwaCGram.PerformCalculations()

System.Console.WriteLine("**********************************Printing Results for
CWA CGram**********************************")
System.Console.WriteLine("Final Distance to Origin {0:F6} ",
cwaCGram.getFinalDistanceToOrigin())
```

```
System.Console.WriteLine("Random Walk Distance {0:F6} ",
cwaCGram.getRandomWalkDistance())
System.Console.WriteLine("Drift {0:F6} ", cwaCGram.getDrift())
System.Console.WriteLine("Final bearing to origin {0:F6} ",
cwaCGram.getFinalBearingToOrigin())
System.Console.WriteLine("Expected random walk bearing {0:F6} ",
cwaCGram.getExpectedRandomWalkBearing())

Dim cwaTime() As Double
Dim cwaDist() As Double
Dim cwaBearing() As Double

Dim cwaAdjTime() As Double
Dim cwaAdjDist() As Double
Dim cwaAdjBearing() As Double
Dim cgram_dbf As New CFileXBaseWrap
Dim dirname As String
Dim sname As String
Dim filename As String
Dim fileutils As New CFileUtilsPub
fileutils.SplitPath(sdbfname, dirname, sname)
dirname = fileutils.sDir
sname = fileutils.sName
sname = "CWgram" + sname
filename = dirname + sname
cgram_dbf.Create(filename)
cwaCGram.SaveToDbf(cgram_dbf)
cgram_dbf.Close()
cwaTime = cwaCGram.getTime()
cwaDist = cwaCGram.getCorrelationDistance()
cwaBearing = cwaCGram.getBearing()

cwaAdjTime = cwaCGram.getAdjustedTime()
cwaAdjDist = cwaCGram.getAdjustedCorrelationDistance()
cwaAdjBearing = cwaCGram.getAdjustedBearing()

Dim i As Integer
System.Console.WriteLine("Lag\tTime\tCorrelation Distance\tBearing")
For i = 0 To cwaCGram.getMaximumLag() Step 1
```

```
        System.Console.WriteLine("{0:F6}    {1:F6}    {2:F6}    {3:F6}", i, cwaTime(i),
cwaDist(i), cwaBearing(i))
        System.Console.WriteLine("-----------------------------------")
Next i

System.Console.WriteLine("Adjusted:")
System.Console.WriteLine("Lag\tTime\tCorrelation Distance\tBearing")
For i = 0 To cwaCGram.getMaximumLag() Step 1
        System.Console.WriteLine("{0:F6}    {1:F6}    {2:F6}    {3:F6}", i,
cwaAdjTime(i), cwaAdjDist(i), cwaAdjBearing(i))
        System.Console.WriteLine("-----------------------------------")
Next i
System.Console.WriteLine("**********************************End Printing Results
for CWA CGram**********************************")
```

Correlated Walk Analysis Regression

Class: CCalcCorrelatedWalk

Synopsis

The Correlated Walk Analysis Regression routine estimates repeatability of actions in terms of distance between the events, direction, and time interval. CCalcCorrelatedWalk provides the routine for Correlated Walk Analysis Regression diagnostics.

Description

The Correlated Walk Analysis Regression diagnostics presents the regression statistics for different lag models. The lag must be specified; the default is a lag of 1 (the sequential events compared with the previous events). Three regression models are run for time difference, direction, and bearing. The output includes statistics for:

1. The sample size
2. The distance and time units
3. The lag of the model (from 1 to 7)
4. The multiple R (correlation) between the lags
5. The squared multiple R (i.e., R-squared)
6. The standard error of estimate for the regression
7. The coefficient, standard error, t-value, and probability value (two-tail) for the constant.
8. The coefficient, standard error, t-value, and probability value (two-tail) for the coefficient.
9. The analysis of variance for the regression model, including the sum-of-squares and the mean-square error for the regression model and the residual (error), the F-test of the regression mean-square error divided by the residual mean-square error, and the probability level for the F-test.

In general, the model with the lowest standard error of estimate (and, consequently, highest multiple R) is best. However, with a small sample size, the model can be unreliable. Further, with each consecutive lag, the sample size decreases by one and a high multiple R associated with a high lag comparison can be unreliable if the sample size is small.

The tabular results can be printed. The user must provide a file name. See Chapter 9 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
initialize(CPointSet^ pPriPointSet, int lagLevel)	Initializes the object with the primary pointset and the lag level. This function should be called before invoking any other functions of the object.
bool PerformCalculations()	Performs the regression calculation. This function should be invoked only after initialize function has been invoked. Returns true on successful computation.
int getLag()	Returns back the value of lag
int getDegreeOfFreedom()	Returns back the value of degree of freedom for the dataset and the value of lag
double getStandardErrorEstimate(REGRESSION_VARIABLE var)	The overall standard error of estimate for the passed value of regression variable. For E.g, getStandardErrorEstimate (REGRESSION_VARIABLE. REGRESSION_VARIABLE_DISTANCE) provides the value of overall standard error of estimate for distance.
double getMultipleR(REGRESSION_VARIABLE var)	Returns the value for multiple correlation coefficient for the value of regression variable var passed as described previously.
double getSquaredMultipleR(REGRESSION_VARIABLE var)	Returns the value for the squared multiple correlation coefficient (i.e., R^2) for the regression variable var passed.
double getCoefficientOfConstant(REGRESSION_VARIABLE var)	Returns the value of the regression coefficient for the constant for a particular value of regression variable var passed.
double getStandardErrorOfConstant (REGRESSION_VARIABLE var)	Returns the value of the standard error of the regression coefficient for the constant for a particular value of regression variable var passed.

double getTValueOfConstant(REGRESSION_VARIABLE var)	Returns the value of the t-values for the regression coefficient for the constant for a particular value of regression variable var passed.
double getPValueOfConstant(REGRESSION_VARIABLE var)	Returns the value of the p-value (two tail) for the regression coefficients for a particular value of regression variable var passed.
double getCoefficientOfLaggedVariable (REGRESSION_VARIABLE var)	Returns the value of the regression coefficient for the lagged variable for a particular value for a particular value of regression variable var passed.
double getStandardErrorOfLaggedVariable (REGRESSION_VARIABLE var)	Returns the value of the standard error of the regression coefficient for the lagged variable for a particular value of regression variable var passed.
double getTValueOfLaggedVariable (REGRESSION_VARIABLE var)	Returns the value of the t-values for the regression coefficient for the lagged variable for a particular value of regression variable var passed.
double getPValueOfLaggedVariable(REGRESSION_VARIABLE var)	Returns the value of the p-value for the regression coefficient for the lagged variable for a particular value of regression variable var passed.
double getRegressionSumOfSquares(REGRESSION_VARIABLE var)	Returns the value of the sum of squares for the regression term for a particular value of regression variable var passed.
int getRegressionDFValue(REGRESSION_VARIABLE var)	Returns the DF value for the regression term for a particular value of regression variable var passed.
double getRegressionMeanSquare(REGRESSION_VARIABLE var)	Returns the value of the mean square for the regression term for a particular value of regression variable var passed.
double getRegressionFRatio(REGRESSION_VARIABLE var)	Returns the value of F-ratio (ratio of the regression sum of squares to the residual sum of squares) for a particular value of regression variable var passed.
double getRegressionPValue(REGRESSION_VARIABLE var)	Returns the P-value associated with the F-ratio for a particular value of regression variable var passed.
double getResidualSumOfSquares(REGRESSION_VARIABLE var)	Returns the value of the sum of squares for the residual term for a particular value of regression variable var passed.
int getResidualDFValue(REGRESSION_VARIABLE var)	Returns the DF value for the residual term for a particular value of regression variable var passed

double getResidualMeanSquare(REGRESSION_VARIABLE var)	Returns the value of the mean square for the residual term for a particular value of regression variable var passed.
double getTotalSumOfSquares(REGRESSION_VARIABLE var)	Returns the value of total sum of squares of the regression and residual terms.
double getTotalDFValue(REGRESSION_VARIABLE var)	Returns the DF value of the total of the regression and residual terms.

Example in Visual Basic

```
'Create an InputParam object and specify Primary point set

Dim cwa As New CCalcCorrelatedWalk
cwa.initialize(file.m_pPrimaryPointSet, 3)
cwa.PerformCalculations()

System.Console.WriteLine("**********Printing Results for CWA Regression**********")
System.Console.WriteLine("Lag {0:D} ", cwa.getLag())
System.Console.WriteLine("Degree of freedoms {0:D} ", cwa.getDegreeOfFreedom())

System.Console.WriteLine("Variable: Time ")
System.Console.WriteLine("Standard error of estimate: {0:F6} ", cwa.getStandardErrorEstimate(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("Multiple R: {0:F6} ", cwa.getMultipleR(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("Squared multiple R: {0:F6} ", cwa.getSquaredMultipleR(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))

System.Console.WriteLine("Co-efficient of Constant: {0:F6} ", cwa.getCoefficientOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("Standard Error of Constant: {0:F6} ", cwa.getStandardErrorOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
```

```
System.Console.WriteLine("T Value of Constant: {0:F6} ",
cwa.getTValueOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("P Value of Constant: {0:F6} ",
cwa.getPValueOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))

System.Console.WriteLine("Co-efficient of Lagged Variable: {0:F6} ",
cwa.getCoefficientOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("Standard Error of Lagged Variable: {0:F6} ",
cwa.getStandardErrorOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("T Value of Lagged Variable: {0:F6} ",
cwa.getTValueOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("P Value of Lagged Variable: {0:F6} ",
cwa.getPValueOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))

System.Console.WriteLine("Analysis of Variance")
System.Console.WriteLine("Sum of Squares of Regression: {0:F6} ",
cwa.getRegressionSumOfSquares(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("DF of Regression: {0:F6} ",
cwa.getRegressionDFValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("Mean Square of Regression: {0:F6} ",
cwa.getRegressionMeanSquare(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("F-Ratio of Regression: {0:F6} ",
cwa.getRegressionFRatio(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
System.Console.WriteLine("P of Regression: {0:F6} ",
cwa.getRegressionPValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))

System.Console.WriteLine("Residual Sum of Squares: {0:F6} ",
cwa.getResidualSumOfSquares(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME))
```

```
System.Console.WriteLine("Residual DF : {0:F6} ",
cwa.getResidualDFValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TI
ME))
System.Console.WriteLine("Residual Mean Square : {0:F6} ",
cwa.getResidualMeanSquare(REGRESSION_VARIABLE.REGRESSION_VARIABLE
_TIME))

System.Console.WriteLine("Total Sum of Squares: {0:F6} ",
cwa.getTotalSumOfSquares(REGRESSION_VARIABLE.REGRESSION_VARIABLE_
TIME))
System.Console.WriteLine("Total DF : {0:F6} ",
cwa.getTotalDFValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_TIME
))

System.Console.WriteLine("Variable: Distance ")
System.Console.WriteLine("Standard error of estimate: {0:F6} ",
cwa.getStandardErrorEstimate(REGRESSION_VARIABLE.REGRESSION_VARIABL
E_DISTANCE))
System.Console.WriteLine("Multiple R: {0:F6} ",
cwa.getMultipleR(REGRESSION_VARIABLE.REGRESSION_VARIABLE_DISTAN
CE))
System.Console.WriteLine("Squared multiple R: {0:F6} ",
cwa.getSquaredMultipleR(REGRESSION_VARIABLE.REGRESSION_VARIABLE_D
ISTANCE))

System.Console.WriteLine("Co-efficient of Constant: {0:F6} ",
cwa.getCoefficientOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIABL
E_DISTANCE))
System.Console.WriteLine("Standard Error of Constant: {0:F6} ",
cwa.getStandardErrorOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIA
BLE_DISTANCE))
System.Console.WriteLine("T Value of Constant: {0:F6} ",
cwa.getTValueOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIABLE_D
ISTANCE))
System.Console.WriteLine("P Value of Constant: {0:F6} ",
cwa.getPValueOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIABLE_D
ISTANCE))
```

```
System.Console.WriteLine("Co-efficient of Lagged Variable: {0:F6} ",
cwa.getCoefficientOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_VA
RIABLE_DISTANCE))
System.Console.WriteLine("Standard Error of Lagged Variable: {0:F6} ",
cwa.getStandardErrorOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_V
ARIABLE_DISTANCE))
System.Console.WriteLine("T Value of Lagged Variable: {0:F6} ",
cwa.getTValueOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_VARIA
BLE_DISTANCE))
System.Console.WriteLine("P Value of Lagged Variable: {0:F6} ",
cwa.getPValueOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_VARIA
BLE_DISTANCE))

System.Console.WriteLine("Analysis of Variance")
System.Console.WriteLine("Sum of Squares of Regression: {0:F6} ",
cwa.getRegressionSumOfSquares(REGRESSION_VARIABLE.REGRESSION_VARIA
BLE_DISTANCE))
System.Console.WriteLine("DF of Regression: {0:F6} ",
cwa.getRegressionDFValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_
DISTANCE))
System.Console.WriteLine("Mean Square of Regression: {0:F6} ",
cwa.getRegressionMeanSquare(REGRESSION_VARIABLE.REGRESSION_VARIABL
E_DISTANCE))
System.Console.WriteLine("F-Ratio of Regression: {0:F6} ",
cwa.getRegressionFRatio(REGRESSION_VARIABLE.REGRESSION_VARIABLE_DI
STANCE))
System.Console.WriteLine("P of Regression: {0:F6} ",
cwa.getRegressionPValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_D
ISTANCE))

System.Console.WriteLine("Residual Sum of Squares: {0:F6} ",
cwa.getResidualSumOfSquares(REGRESSION_VARIABLE.REGRESSION_VARIAB
LE_DISTANCE))
System.Console.WriteLine("Residual DF : {0:F6} ",
cwa.getResidualDFValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_DI
STANCE))
System.Console.WriteLine("Residual Mean Square : {0:F6} ",
cwa.getResidualMeanSquare(REGRESSION_VARIABLE.REGRESSION_VARIABLE
_DISTANCE))
```

```
System.Console.WriteLine("Total Sum of Squares: {0:F6} ",
cwa.getTotalSumOfSquares(REGRESSION_VARIABLE.REGRESSION_VARIABLE_
DISTANCE))
System.Console.WriteLine("Total DF : {0:F6} ",
cwa.getTotalDFValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_DIST
ANCE))

System.Console.WriteLine("Variable: Bearing ")
System.Console.WriteLine("Standard error of estimate: {0:F6} ",
cwa.getStandardErrorEstimate(REGRESSION_VARIABLE.REGRESSION_VARIABL
E_BEARING))
System.Console.WriteLine("Multiple R: {0:F6} ",
cwa.getMultipleR(REGRESSION_VARIABLE.REGRESSION_VARIABLE_BEARIN
G))
System.Console.WriteLine("Squared multiple R: {0:F6} ",
cwa.getSquaredMultipleR(REGRESSION_VARIABLE.REGRESSION_VARIABLE_B
EARING))

System.Console.WriteLine("Co-efficient of Constant: {0:F6} ",
cwa.getCoefficientOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIABL
E_BEARING))
System.Console.WriteLine("Standard Error of Constant: {0:F6} ",
cwa.getStandardErrorOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIA
BLE_BEARING))
System.Console.WriteLine("T Value of Constant: {0:F6} ",
cwa.getTValueOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIABLE_B
EARING))
System.Console.WriteLine("P Value of Constant: {0:F6} ",
cwa.getPValueOfConstant(REGRESSION_VARIABLE.REGRESSION_VARIABLE_B
EARING))

System.Console.WriteLine("Co-efficient of Lagged Variable: {0:F6} ",
cwa.getCoefficientOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_VA
RIABLE_BEARING))
System.Console.WriteLine("Standard Error of Lagged Variable: {0:F6} ",
cwa.getStandardErrorOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_V
ARIABLE_BEARING))
System.Console.WriteLine("T Value of Lagged Variable: {0:F6} ",
cwa.getTValueOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_VARIA
BLE_BEARING))
```

```
System.Console.WriteLine("P Value of Lagged Variable: {0:F6} ",
cwa.getPValueOfLaggedVariable(REGRESSION_VARIABLE.REGRESSION_VARIA
BLE_BEARING))

System.Console.WriteLine("Analysis of Variance")
System.Console.WriteLine("Sum of Squares of Regression: {0:F6} ",
cwa.getRegressionSumOfSquares(REGRESSION_VARIABLE.REGRESSION_VARIA
BLE_BEARING))
System.Console.WriteLine("DF of Regression: {0:F6} ",
cwa.getRegressionDFValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_
BEARING))
System.Console.WriteLine("Mean Square of Regression: {0:F6} ",
cwa.getRegressionMeanSquare(REGRESSION_VARIABLE.REGRESSION_VARIABL
E_BEARING))
System.Console.WriteLine("F-Ratio of Regression: {0:F6} ",
cwa.getRegressionFRatio(REGRESSION_VARIABLE.REGRESSION_VARIABLE_B
EARING))
System.Console.WriteLine("P of Regression: {0:F6} ",
cwa.getRegressionPValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_B
EARING))

System.Console.WriteLine("Residual Sum of Squares: {0:F6} ",
cwa.getResidualSumOfSquares(REGRESSION_VARIABLE.REGRESSION_VARIAB
LE_BEARING))
System.Console.WriteLine("Residual DF : {0:F6} ",
cwa.getResidualDFValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_B
EARING))
System.Console.WriteLine("Residual Mean Square : {0:F6} ",
cwa.getResidualMeanSquare(REGRESSION_VARIABLE.REGRESSION_VARIABLE
_BEARING))

System.Console.WriteLine("Total Sum of Squares: {0:F6} ",
cwa.getTotalSumOfSquares(REGRESSION_VARIABLE.REGRESSION_VARIABLE_
BEARING))
System.Console.WriteLine("Total DF : {0:F6} ",
cwa.getTotalDFValue(REGRESSION_VARIABLE.REGRESSION_VARIABLE_BEA
RING))

System.Console.WriteLine("**********************************End Printing Results
for CWA Regression**********************************")
```

Correlated Walk Analysis Prediction

Class: CCalcCorrelatedWalkPred

Synopsis

CCalcCorrelatedWalkPred provides routines for prediction of Correlated Walk analysis. This class derives from CCalcCorrelatedWalk.

Description

The Correlated Walk Analysis Prediction routine allows the prediction of a next event, in time, distance, and direction. For each parameter – time difference, distance, and bearing, there are three models that can be used:

1. The mean difference (i.e., use mean time difference, mean distance, mean bearing)
2. The median difference (i.e., use median time difference, median distance, median bearing)
3. The regression model (i.e., use the estimated regression coefficient and intercept)

For each of these, a different lag comparison can be used, from 1 to 7. The lag defines the sequence from which the prediction is made. Thus, for a lag of 1, the interval from the next-to-last to the last event is used as a reference (i.e., between events N-1 and N); for a lag of 2, the interval from the third-to-last to the next-to-last event is used as a reference (i.e.,between events N-2 and N-1); and so forth. The particular model selected is then added to the reference sequence.

The output includes:

1. The method used for time, distance, and bearing
2. The lag used for time, distance, and bearing
3. The predicted time difference
4. The predicted distance
5. The predicted bearing
6. The final predicted time
7. The X-coordinate of the final predicted location

8. The Y-coordinate of the final predicted location

The tabular results can be printed, saved to a text file, or output as a '.dbf' file. The user must provide a file name. The graphical results can be output to *ArcView* '.shp' or *MapInfo* '.mif' files. The user must provide a file name. The output includes five graphical objects:

1. The sequence of incidents from the first to the last. This object has a prefix of 'Events' before the user-defined name provided by the user.

2. The predicted location of the next event. This is the event after the last in the input sequence. This object has a prefix of 'Preddest' before the user-defined name.

3. The predicted path between the last event in the sequence and the expected next event. This object has a prefix of 'Pw' before the user-defined name.

4. The center of minimum distance for the sequence of events. This is the single best measure of the likely origin location of the offender. This object has a prefix of 'POrigL' before the user-defined name.

5. The expected path between the center of minimum distance and the predicted location of the next event. This is a guess about the likely origin and likely destination for a next event by the offender. This object has a prefix of 'Patj' before the user-defined name.

See Chapter 9 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
void initialize(CPointSet^ pPriPointSet, PREDICTION_METHOD nCwaDistMethod, PREDICTION_METHOD nCwaTimeMethod, PREDICTION_METHOD nCwaBearingMethod, int nDistLevel, int nCwaPTimeLevel, int nCwaPBearingLevel)	Initializes for CWA Prediction calculation. priPointSet - primary point set nCwaDistMethod – Method of prediction for Distance nCwaTimeMethod – Method of prediction for Time. nCwaBearingMethod – Method of prediction for Bearing nDistLevel – Value of Lag for distance nCwaPTimeLevel – Value of lag for time nCwaPBearingLevel – Value of lag for bearing This method must be called before invoking any other methods of this class.
bool PerformCalculations()	This method is the main method that computes CWA prediction statistics. This method should be invoked only after a call to initialize method above.
double getPredictedTimeInterval()	Predicted value for the time interval
double getPredictedDistanceInterval()	Predicted value for the distance interval
double getPredictedBearingInterval()	Predicted value for the bearing interval
unsigned int getEventForPredictedTimeInterval()	Event from which the prediction is done for the time variable
unsigned int getEventForPredictedDistanceInterval()	Event from which the prediction is done for the distance variable

unsigned int getEventForPredictedBearingInterval()	Event from which the prediction is done for the bearing variable
double getPredictedTime()	Value of the predicted time
double getPredictedX()	X-Coordinate of the predicted location
void saveResult(System::String ^m_sfilename, int m_iFileType, System::String ^projname, int projnum, int datum)	Saves the result to appropriate graphics output format. The filename, file type, projection name and datum are passed as input parameters. m_iFileType – Shape(2) or MIF(8) or KML(10)
void setMIFHeader (System::String ^header)	Set the header if output file type is MIF
double getPredictedY()	Y-Coordinate of the predicted location

Example in Visual Basic

```
'Create an InputParam object and specify Primary point set

Dim cwaPred As New CCalcCorrelatedWalkPred
cwaPred.initialize(file.m_pPrimaryPointSet,
PREDICTION_METHOD.PREDICTION_METHOD_REGRESSION,
PREDICTION_METHOD.PREDICTION_METHOD_REGRESSION,
PREDICTION_METHOD.PREDICTION_METHOD_REGRESSION, 2, 2, 2)
cwaPred.saveResult(sfilename,2, "Earth Projection", 1, 33)
cwaPred.PerformCalculations()

System.Console.WriteLine("**********************************Printing Results for
CWA Prediction**********************************")
```

```
System.Console.WriteLine("Time Interval:: Predicted Value = {0:F6} : From Event =
{1:D}", cwaPred.getPredictedTimeInterval(),
cwaPred.getEventForPredictedTimeInterval())
System.Console.WriteLine("Distance Interval:: Predicted Value = {0:F6} : From Event =
{1:D}", cwaPred.getPredictedDistanceInterval(),
cwaPred.getEventForPredictedDistanceInterval())
System.Console.WriteLine("Bearing Interval:: Predicted Value = {0:F6} : From Event =
{1:D}", cwaPred.getPredictedBearingInterval(),
cwaPred.getEventForPredictedBearingInterval())

System.Console.WriteLine("Predicted Time = {0:F6} ", cwaPred.getPredictedTime())
System.Console.WriteLine("Predicted X Co-ordinate = {0:F6} : Predicted Y Co-ordinate
= {1:F6}", cwaPred.getPredictedX(), cwaPred.getPredictedY())

System.Console.WriteLine("**********************************End Printing Results
for CWA Prediction**********************************")
```

Spatial–temporal Moving Average

Class: CCalcSpatialTemporalAverage

Synopsis

This routine calculates the mean center as it changes over a sequence of events. CCalcSpatialTemporalAverage calculates the algorithm.

Description

This routine calculates the mean center as it changes over a sequence of the events. A dataset of incidents are inputted with each incident having X and Y coordinates and an ordered time of occurrence (e.g., days, weeks, hours). The routine sorts the incidents in the time order in which they occur. The user defines a *span* of sequential incidents (the default is five observations) and the routines places a window covering the span over the incidents. It calculates the mean center (the mean X coordinate and the mean Y coordinate) of the spanned incidents. It then moves the window one observation and re-calculates the mean center. Approximations are made at the beginning and end observations for the sequence. The result is a set of mean centers ordered from the first through last observations. This statistic is useful for identifying whether the central location for a set of incidents (perhaps committed by a serial offender) has moved over time. There are four outputs for this routine:

1. The sample size
2. The number of observations making up the span
3. The span number
4. The X and Y coordinates of the mean center for each span window.

See Chapter 9 of the CrimeStat manual for more information.

Fields

None

Methods

Method	Description
void Initialize(CPointSet^ pPointSet, int nSpanNum)	Intializes for Spatio-temporal Moving Average and requires two parameters namely the Primary Pointset and the span number.
bool PerformCalculations()	This method is the main method that computes the Spatio-Temporal moving average. This method should be invoked only after a call to initialize method above.
double getMeanX()	Returns the moving average, mean X.
double getMeanY()	Returns the moving average, mean Y.
getMeanCount()	Returns the number of moving average(s).
bool saveGraph(System::String ^m_sfilename,int m_iFileType,System::String^ projname,int projnum,int datum)	Saves the graphical output of Spatio-temporal moving average. Requires the filename, file type, projection name, projection number and datum. m_iFileType – Shape(2) or MIF(8) or KML(10)
void setMIFHeader (System::String ^header)	Set the header if output file type is MIF
bool SaveToDbf(CFileXBaseWrap ^pFile)	Saves the dbf output for Spatio-temporal moving average. This function requires a parameter of type CFileXBaseWrap as an input.
double getPredictedY()	Y-Coordinate of the predicted location

Example in Visual Basic

```
'Create an InputParam object and specify Primary point set
Dim movAvg As New CCalcSpatialTemporalAverage
 movAvg.Initialize(file.m_pPrimaryPointSet, 10)
Dim movavg_dbf As New CFileXBaseWrap
Dim dirname As String
Dim sname As String
```

```
Dim filename As String
Dim fileutils As New CFileUtilsPub
fileutils.SplitPath(sdbfname, dirname, sname)
dirname = fileutils.sDir
sname = fileutils.sName
sname = "STMA" + sname
filename = dirname + sname
movavg_dbf.Create(filename)
movAvg.SaveToDbf(movavg_dbf)
movAvg.PerformCalculations()
movavg_dbf.Close()
System.Console.WriteLine("**********Printing Results for
STMA**********")
      System.Console.WriteLine("Spatial-Temporal Moving Average")
      Dim movAvgMeanX() As Double
      Dim movAvgMeanY() As Double
      movAvgMeanX = movAvg.getMeanX()
      movAvgMeanY = movAvg.getMeanY()
      'Dim StrArray() As String = {"Min", "0.5", "1.0", "2.5", "5.0", "10.0", "90.0",
"95.0", "97.5", "99.0", "99.5", "Max"}
      'mantelSimPerc = knoxInd.getSimulationPercentiles()
      Dim i As Integer
      For i = 0 To movAvg.getMeanCount() - 1 Step 1
         System.Console.WriteLine("Span {0:D} MeanX {1:F6}, MeanY {2:F6}", i + 1,
movAvgMeanX(i), movAvgMeanY(i))
         System.Console.WriteLine("----------")
      Next i
      System.Console.WriteLine("**********End Printing
Results for STMA**********")
   End Sub
```

Journey-to-crime Library

The Journey-to-crime library is a set of two routines for modeling the likely origin (usually residence) location of a serial offender. The first routine calibrates a likelihood function for travel distance based on a sample of offenders for whom both their origin and crime locations is known. The second routine allows the user to estimate a likelihood surface for where a particular serial offender lives based on either the calibration function of an assumed mathematical model. See chapter 10 in the CrimeStat manual for more information.

Library: **JourneyToCrime.dll**

Prerequisites: **Crimestat Core Components.dll**
Hot Spot Analysis.dll

Calibrate Distance Function for JTC

Class: CJtcCalibrateFunction

Synopsis

CJtcCalibrateFunction generates the calibrated distances that can be used for JTC.

Description

Fields

None

Methods

Method	Description
void Initialize(CPointSet ^m_pPointSet, CCalcKernelDensityParams^ pParams)	Initializes the object for Ripley's K calculation. This method takes in a point set, and an object of CCalcKernelDensityParams that specifies the parameters for kernel density. Please refer to the documentation on Hot Spot Analysis for more details about CCalcKernelDensityParams. This method must be called before invoking any other methods of this class.
void PerformCalculations(System::String ^filename,int filetype)	This method is the main method that computes the Calibrated distances. This method should be invoked only after a call to initialize method above. The function also writes the calibarated distance function to the corresponding file name and file type that are specified as parameters. Currently two file types are supported, namely ASCII (1) and DBF (64)
array<double>^ GetCalibratedDistances()	Returns an array of size equal to the number of bins(value returned by GetNumberOfBins()) with each value corresponding to the calibrated distance
unsigned int GetNumberOfBins()	Returns the number of bins.

Example in Visual Basic

```
'Specify an Origin Destination file or Primary file as an Input that contains origins and destinations

file.SetOriginX(cboxX.SelectedIndex,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
    file.SetOriginY(cboxY.SelectedIndex,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
    file.SetDestinationX(cboxDX.SelectedIndex,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
    file.SetDestinationY(cboxDY.SelectedIndex,
INPUT_FILTER_TYPE.INPUT_FILTER_TYPE_BLANK)
'Update Data after mapping columns
file.UpdateJTCCalibrationInputData()
Dim jtc As New CJtcCalibrateFunction
Dim kernelDen As New CCalcKernelDensityParams
kernelDen.setKernelSingleMethod(Interpolation.KERNEL_DENSITY_CALC.KERNEL
_DENSITY_CALC_NORMAL)
kernelDen.setKernelSingleBandwidth(Interpolation.KERNEL_BANDWIDTH.KERNEL
_BANDWIDTH_FIXED_INTERVAL)
kernelDen.setKernelSingleMinSample(100)
kernelDen.setKernelSingleInterval(0.25)
kernelDen.setKernelSingleUnit(Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILE
S)
kernelDen.setKernelIntervalBinsNumber(100)
kernelDen.setKernelSingleOutUnit(Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MI
LES)
kernelDen.setCalculationMethod(Interpolation.KERNEL_SINGLE_CALC.KERNEL_SI
NGLE_CALC_REL_DENSITY)
jtc.Initialize(file.m_pJTCCalibrationInputPointSet, kernelDen)
jtc.PerformCalculations(savefilename, 64)
System.Console.WriteLine("**********Printing Results for
JTC**********")
System.Console.WriteLine("**********End Printing Results
for JTC**********")
```

Journey-to-crime Estimation (JTC)

Class: CCalcJtc

Synopsis

CCalcJtc calculates the Journey to Crime Estimates.

Description

Fields

None

Methods

Method	Description
void Initialize (CPointSet ^pPointSet,CPointSet ^refPointSet,COV_UNIT_TYPE m_nJtcUnit)	Initializes the object for JTC calculation. This method takes in the following parameters pPriPointSet – object of type CPointSet corresponding to the primary point set refPointSet - object of type CPointSet corresponding to the reference point set m_nJtcUnit – Unit of measurement of the output This method must be called before invoking any other methods of this class.
void SetNegativeExponentialDistribution(double dCoefficient, double dExponent)	An optional method to set the distribution to be used during computation. This method specifies to use Negative Exponential Distribution and the parameters are co-efficient and exponent for the distribution. This method should be called before PerformCalculations() is called.
void SetNormalDistribution(double dMeanDistance,double dStandardDev, double dCoefficient)	An optional method to set the distribution to be used during computation. This method specifies to use Normal Distribution and the parameters mean, standard deviation and co-efficient for the distribution are specified. This

	method should be called before PerformCalculations() is called.
void SetLogNormalDistribution(double dMeanDistance, double dStandardDev, double dCoefficient)	An optional method to set the distribution to be used during computation. This method specifies to use Log Normal Distribution during the calculation and the parameters mean, standard deviation and co-efficient for the distribution are required to be specified. This method should be called before PerformCalculations() is called.
void SetLinearDistribution(double dIntercept, double dSlope)	An optional method to set the distribution to be used during computation. This method specifies to use Log Linear Distribution during the calculation and the parameters Intercept and Slope for the distribution is required to be specified. This method should be called before PerformCalculations() is called.
void SetTruncatedNegativeExponentialDistribution(double dPeakLikelihood, double dPeakDistance, double dExponent)	An optional method to set the distribution to be used during computation. This method specifies the use of Truncated Negative Exponential Distribution during the calculation and the parameters Peak Likelihood, Peak Distance and Exponent for the distribution is required to be specified. This method should be called before PerformCalculations() is called.
void UseCalibratedDistance(System::String ^m_sCalibrationInputFile)	An optional method that specifies the use of a calibrated distance from a file during the calculation. This method should be called before PerformCalculations() is called.
Void PerformCalculations (int m_nFileType, System::String ^m_sFileName)	This method is the main method that computes the JTC statistics. This method should be invoked only after a call to initialize method above and one of the setter methods above. The method takes in two arguments, namely the type of the graphics output file and the name of the output file. m_nFileType – Shape(2) or MIF(8) or KML(10)
array<double>^ getLikelihoodValuesforReferencePoints()	Returns an array of double of size equal to the number of points in the reference data set where each entry in the array is the likelihood estimate for a particular cell of the reference dataset.

unsigned int getNumberOfReferencePoints()	Returns the number of points in the reference data set
unsigned int getNumberOfLocationPoints()	Returns the number of location points, i.e., number of points in the primary data set
unsigned int getPeakLikelihoodLocation()	Returns the index of the grid cell which has the peak likelihood estimate
double getPeakLikelihoodValue()	Returns the value of the peak likelihood estimate
double getPeakLikelihoodLocationXCoordinate()	Returns the X-Coordinate of the grid cell which has the peak likelihood estimate
double getPeakLikelihoodLocationYCoordinate()	Returns the Y-Coordinate of the grid cell which has the peak likelihood estimate

---

Example in Visual Basic

```
'Create an InputParam object and specify Primary and reference point set

Dim jtc As New CCalcJtc
jtc.Initialize(file.m_pPrimaryPointSet, file.m_pReferencePointSet,
Utilities.COV_UNIT_TYPE.COV_UNIT_TYPE_MILES)
'jtc.SetNegativeExponentialDistribution(1.89, -0.06)
'jtc.SetNormalDistribution(4.2, 4.6, 29.5)
'jtc.SetLogNormalDistribution(4.2, 4.6, 8.6)
'jtc.SetLinearDistribution(1.9, -0.06)
'jtc.SetTruncatedNegativeExponentialDistribution(13.8, 0.4, -0.2)
jtc.UseCalibratedDistance(calibfile)
PerformCalculations(8, savefilename)
System.Console.WriteLine("**********************************Printing Results for
JTC**********************************")
System.Console.WriteLine("Reference Points: {0}", jtc.getNumberOfReferencePoints())
System.Console.WriteLine("Location Points: {0}", jtc.getNumberOfLocationPoints())
System.Console.WriteLine("Peak Likelihood Location: {0}",
jtc.getPeakLikelihoodLocation())
System.Console.WriteLine("Peak Likelihood Value: {0:F6}",
jtc.getPeakLikelihoodValue())
System.Console.WriteLine("Peak Likelihood Location X Co-ordinate: {0:F6}",
jtc.getPeakLikelihoodLocationXCoordinate())
System.Console.WriteLine("Peak Likelihood Location Y Co-ordinate: {0:F6}",
jtc.getPeakLikelihoodLocationYCoordinate())

Dim i As Integer
Dim likelihoodArr As Double()
```

```
likelihoodArr = jtc.getLikelihoodValuesforReferencePoints()
System.Console.WriteLine("Reference Point, Likelihood")
For i = 0 To jtc.getNumberOfReferencePoints() - 1 Step 1
        System.Console.WriteLine("{0}, {1:F6}", i + 1, likelihoodArr(i))
Next i

System.Console.WriteLine("**********************************End Printing Results
for JTC**********************************")
```