Journal of Statistical Software

October 2016, Volume 74, Issue 6. doi: 10.18637/jss.v074.i06

pyJacqQ: Python Implementation of Jacquez's Q-Statistics for Space-Time Clustering of Disease Exposure in Case-Control Studies

Saman Jirjies
Arizona State University

Garrick Wallstrom
Arizona State University

Rolf U. Halden
Arizona State University

Matthew Scotch
Arizona State University

Abstract

Jacquez's Q is a set of statistics for detecting the presence and location of space-time clusters of disease exposure. Until now, the only implementation was available in the proprietary **SpaceStat** software which is not suitable for a pipeline Linux environment. We have developed an open source implementation of Jacquez's Q statistics in Python using an object-oriented approach. The most recent source code for the implementation is available at `https://github.com/sjirjies/pyJacqQ` under the GPL-3. It has a command line interface and a Python application programming interface.

Keywords: Python, epidemiology, Jacquez's Q, clustering, public health.

1. About Jacquez's Q-statistics

Epidemiology benefits from the existence of several geo-spatial clustering methods. For example, Moran's I is capable of globally detecting spatial autocorrelation (Moran 1948). Cuzick and Edward proposed a method using k-nearest neighbors (KNN) that is capable of detecting the presence and location of clusters (Cuzick and Edwards 1990). The limitation of these methods is that they operate on a static geography and do not account for the movement of individuals. The Knox test (Knox and Bartlett 1964) and Jacquez's K (Jacquez 1996) are examples of methods that do consider individuals' movements; however, they indicate only the presence of space-time clusters and not their locations.

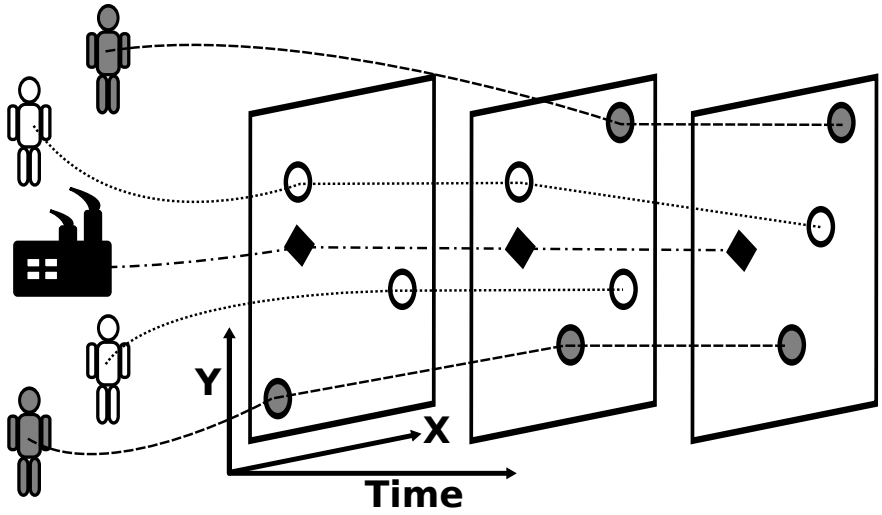

Figure 1: Relationship between study entities and time slices. The white and gray persons represent individual cases and controls, respectively. The factory represents a focus location of interest. The large rectangles represent discrete snapshots in time. The area within a particular slice of time represents the geographic area of the study at that time. Each time slice captures the location of cases, controls, and focus points of interest. Times between the discrete time slices are not considered. Here, gray points are the location of cases and white points the location of controls. Diamonds are the locations in the focus geography. The dotted lines connecting points show the relation of cases, controls, and focus entities through time. Note that individuals and focus points can enter or exit the study at different times as shown in the diagram.

Recently, Jacquez, Kaufmann, Meliker, Goovaerts, AvRuskin, and Nriagu (2005) have developed an elaborate method, Jacquez's Q, that can detect the existence of space-time clusters in addition to elucidating their locations and time ranges (Jacquez *et al.* 2005). The method shows many similarities to the ones previously mentioned, especially Cuzick and Edward's KNN and Jacquez's K. This method operates on the residential histories of cases and controls by creating a discrete time slice for every occurrence when a case or control changes residence (Jacquez *et al.* 2005). This is done in such a way that ensures consistency in the arrangement of points for all times between adjacent time slices (Jacquez *et al.* 2005). Then for every time slice, every case present at that time slice has a count of the number of other cases in its KNNs. This method also includes the ability to test for space-time clustering of cases around arbitrary geographic locations, or a 'focus' geography (Jacquez *et al.* 2005). Focus points, for example, could represent sources of contamination and can be static or mobile. Cases, controls, and focus points are capable of entering and leaving the study at any time and are only considered during times in which they are part of the study. In Figure 1, we show a diagram of the relationship between cases, controls, focus locations, and time slices.

Instead of a single statistic, this method generates a plethora of statistics that fall within certain sets. The local statistics are composed of the above mentioned counts that are performed for every case at every time slice in which the case is present. The local statistic is denoted Q_{it}. The '*it*' represents clustering around an *individual* case at a specific *time*.

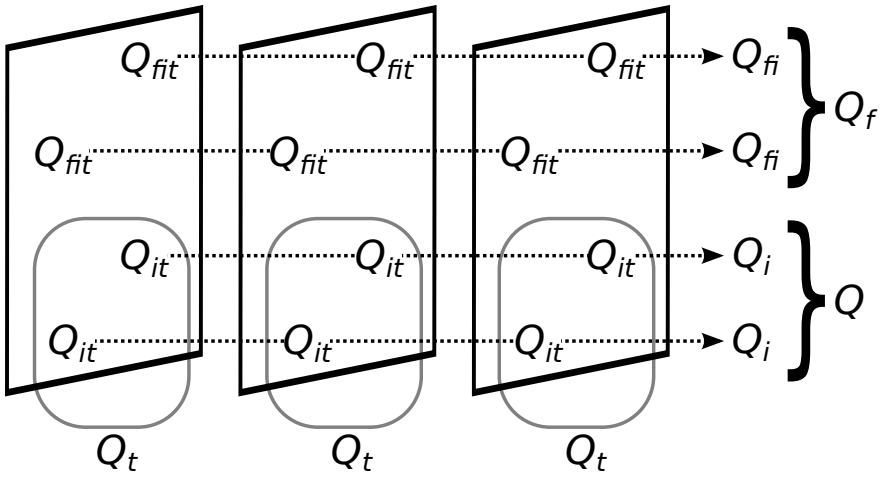

Figure 2: Relationship between Q-statistics. The skewed rectangles represent time slices, as in Figure 1. Each time slice contains a Q_{it} statistic for each case and a Q_{fit} statistic for each focus point. The sum of a local statistic for a case through time produces a Q_i statistic for that case, indicated by the dotted arrow. This also holds for focus points. The sum of all Q_i statistics across cases produces the global Q and the sum of all Q_{fi} statistics across focus entities produces the global Q_f statistic as shown by the brackets. The rounded rectangles demonstrate that time slice statistics (Q_t) are the sum of all local case statistics within the time slice.

A summation of all the local statistics within a given time slice yields that time slices's Q_t statistic (Jacquez *et al.* 2005). A Q_t statistic exists for every time slice in the study and measures global clustering at a given time (Jacquez *et al.* 2005). Yet another set of statistics exist for the summation of all Q_{it} statistics for an individual through time, denoted Q_i. Again, a Q_i statistic is calculated for every case in the study and measures the tendency for a case to be part of clusters through time (Jacquez *et al.* 2005). Finally, the method also calculates a global Q statistic which is the summation of all cases' Q_i statistics. The Q statistic measures global clustering throughout the entire study (Jacquez *et al.* 2005).

The statistics generated by the focus geography also fall into sets. Similar to Q_{it}, each focus point can have a statistic calculated for every time slice in which it exists, creating a Q_{fit} statistic. Since focus points do not have a case-control status, Q_{fit} is calculated by counting the number of cases within the focus points KNN disregarding other focus points (Jacquez *et al.* 2005). Analogous to Q_i, a focus points's Q_{fit} can be summed through time to yield a Q_{fi} statistic. There is also a global Q_f statistic which is the sum of all Q_{fi} statistics. The Q_f statistic indicates the global presence of clustering around focus points in the study, similar to Q (Jacquez *et al.* 2005). The relationships between all of these statistics are illustrated in Figure 2.

Each of the described set Q statistics also have multiple versions, depending on the desires of the investigator. Researchers can opt to employ the statistics with exposure periods, requiring them to supply latency and exposure durations for cases and controls. Here a latency duration is an estimated length of time when an individual had the disease but it remained dormant.

An exposure duration is the estimated length of time that an individual was susceptible to receive the disease. A date of diagnosis must also be given for each case and matched control allowing for a window of opportunity for disease exposure calculated from latency period and exposure duration. Jacquez *et al.* (2005) refer to this window of opportunity for disease exposure as an "exposure trace". The exposure trace spans the time between initial exposure and disease formation. The date of initial exposure can be estimated by subtracting both the latency period and exposure duration from the date of diagnosis. Similarly, the date of disease formation is found by subtracting just the latency period from the date of diagnosis. A cases is referred to as "active" at times when it falls within its exposure trace (Jacquez *et al.* 2005). In the exposure clustering setup, the local statistic is performed only for active cases and consists of the count of KNN that are also active cases (Jacquez *et al.* 2005). This version allows researchers to find clusters of actual disease exposure rather than simply detecting clusters of cases regardless of their disease status at cluster time.

Similarly, another version of all these statistics accounts for covariates in the cases and control by allowing researchers to supply the probability of each individual being a case, possibly as the result of a logistic regression (Jacquez *et al.* 2005, 2006). The use of case weights helps ensure that detected clusters share an association with their geographic location rather than containing individuals who all share similar covariates. For example, a study of lung cancer using this method could detect clusters that have the disease in association with their geographic location instead of finding clusters of smokers. It is worth noting that these versions can be combined to yield statistics that use both exposure windows and case weights.

The significance of each measure within Jacquez's Q statistics is calculated using Monte Carlo tests (Jacquez *et al.* 2005). This is performed by first shuffling the case-control status of each individual in the study, which is done in one of two manners. The simpler scheme is used under the assumption of equal disease risk for all individuals in the study. Here the total number of cases in the study is counted and individuals are then chosen with uniform randomness and assigned as a case until the original number of cases in the study is achieved.

The second method involves the use of the previously mentioned case weights and accounts for covariates. The individuals are sorted by their case weight and each weight is normalized by dividing by the sum of case weights under consideration (Jacquez *et al.* 2006). Each weight is then mapped to an interval proportional to its magnitude such that each interval is contiguous with its sorted neighbor intervals and the total range spans 0–1 (Jacquez *et al.* 2006). A uniform random number is chosen between 0–1 and the individual with an interval spanning this random number is marked as a case (Jacquez *et al.* 2006). The new case is removed and the procedure is repeated until the original number of cases have been selected (Jacquez *et al.* 2006). The remaining individuals are the controls (Jacquez *et al.* 2006).

Regardless of the method chosen, all statistics are then recalculated with the new case-control distribution. This is a single permutation of the Monte Carlo test. This entire procedure, involving redistribution of case-control status and recalculation of all statistics, is repeated for several permutations and a pseudo-p value is calculated for each statistic as $(a+1) \div (b+1)$ (Jacquez *et al.* 2005). Here a is the number of permutations where the statistic was at least as extreme as observed in the original data and b is the total number of permutations conducted (Jacquez *et al.* 2005). For example, if a local statistic for a case was observed to have 14 case neighbors out of 15 then the local statistic, Q_{it}, for this case would be 14 cases. If out of 99 Monte Carlo permutations this statistic happened to be 14 or above a total of three times, the p value for this local statistic would be calculated as $(3+1) \div (99+1) = 0.04$.

Note that the minimum p value obtainable is $1 \div (b+1)$, meaning that the p value resolution is determined by the number of permutations used during Monte Carlo testing. For example, the minimum p value possible for 99 shuffles is 0.01 and the minimum for 999 shuffles is 0.001.

Jacquez's Q is a promising method given its capacity to pinpoint the location and times of space-time clusters and could be of great potential to epidemiologists and public health informaticians, however, there have been mixed findings regarding its utility (Sloan *et al.* 2012; Nordsborg, Meliker, Ersboll, Jacquez, and Raaschou-Nielsen 2013). An analysis of the method using simulated data conducted by Sloan *et al.* (2012) found that investigating significant intersections of Q_i and Q_{it} proved the best strategy. They also found that Q statistics performs best when the population is large and mobile but recommend checking the findings with scan statistics (Sloan *et al.* 2012). According to their analysis, Jacquez's Q is expensive with a worst-case running time of $O(n^2 \cdot \log(n))$ and for this reason they recommend using other methods for populations that are not very mobile (Sloan *et al.* 2012).

A study into space-time clustering of non-Hodgkin Lymphoma in Denmark found the method unable to consistently detect any significant clusters (Nordsborg *et al.* 2013). The team used 3,210 cases and two sets of controls each with 3,210 individuals; on average an individual lived in four locations throughout the study period of 33 years (Nordsborg *et al.* 2013). They even supplied case-weights to adjust for known covariates (Nordsborg *et al.* 2013). Despite this they were unable to detect the same clusters using their two control groups or any clusters when they combined their controls; presenting largely inconsistent results (Nordsborg *et al.* 2013). This group found the biggest issue with Q-stats is the time required for analysis, which could take 8 hours with a dataset their size (Nordsborg *et al.* 2013).

In addition, there are several challenges presented in interpreting the results produced by Jacquez's Q. This is caused chiefly by the number of tests conducted. The issues can be demonstrated using the previously mentioned Danish study as a numerical guide. In a study with 3,000 cases and the same number of controls where each individual moves three times throughout the study, a conservative estimate places the number of time slices at $(3,000 + 3,000) \cdot 3 = 18,000$. Each one of these 18,000 time slices would have 3,000 local statistics, one for each case at that time. This would produce $18,000 \cdot 3,000 = 54,000,000$ local statistics. The total number of statistics would be the sum of all local, case, and time slice statistics. In this example, the total would be $54,000,000 + 3,000 + 18,000 = 54,021,000$ statistics. Given the typical 0.05 alpha used for each test, we would expect 5% or 2,701,050 of these statistics to be false positives. A correction must be applied to account for this multiple testing problem. As explained previously, the p value resolution is determined by the number of permutations conducted during Monte Carlo testing. If 99 permutations were used, each of these 54,021,000 statistics would have to be calculated under 100 conditions, once for each permutation and once for the observed data to produce a minimum possible p value of 0.01. If a family-wise correction such as the Bonferroni were applied for the multiple testing in this example, the corrected alpha would be $0.05 \div 2,701,050 = 1.85 \cdot 10^{-8}$. It is impossible for any statistic calculated using 99 permutations to fall below this significance since the minimum possible p value is magnitudes larger. In fact, achieving $1.85 \cdot 10^{-8}$ as a minimum p value would require 54,021,000 permutation shuffles. Recalling that the time complexity of Jacquez's Q is $O(n^2 \cdot \log(n))$ (Sloan *et al.* 2012), the method is time consuming even for 99 shuffles rendering family-wise corrections impractical. Sloan *et al.* (2012) found that the false discovery rate correction can be used but only with some success.

One method of dealing with this is to have each set of statistics undergo a binomial test

comparing the total number of significant statistics to the expected number of significant statistics within each set (Sloan *et al.* 2012). Here the sets are the local statistics (all Q_{it}), the time slice statistics (all Q_t), and the case statistics (all Q_i). If a focus point geography is provided this also includes the focus statistics (all Q_{fi}) and the local focus statistics (all Q_{fit}). This is done for each set with a null hypothesis that the number of significant statistics in the set is the same as the number of expected false positives and an alternative hypothesis that the number of significant statistics in the set is not the expected number of false positives. The binomial test is given in Equation 1 (Sloan *et al.* 2012):

$$\mathsf{P}(S) = \sum_{j=0}^{S} \frac{N!}{j!\,(N-j)!} \alpha^j (1-\alpha)^{N-j}. \tag{1}$$

Here S is the number of significant statistics within the set, N is the total number of statistics within the set, and α is the significance level used for each test. This gives the cumulative distribution function, denoted in the equation as $\mathsf{P}(S)$, between zero and the observed number of significant statistics. Equation 1 can be used to produce a p value for the set in question using Equation 2 given below

$$p_{\text{value}} = 1 - \mathsf{P}(S-1). \tag{2}$$

Here p_{value} is the output p value and both P and S denote the same concepts as in Equation 1. Continuing the previous example; if we observe 980 significant time slice statistics in the set of all 18,000 Q_i time slice statistics, each with an alpha of 0.05, Equation 1 would produce a p value of approximately 0.003. A separate alpha for this test would be chosen in advance to draw a conclusion regarding the significance in the number of p values at or below α within this set. Researchers could then investigate the sets that exhibit such a condition after using this above technique (Sloan *et al.* 2012).

We refer readers to the original publications for more information on Jacquez's, including the formal equations Q (Jacquez *et al.* 2005, 2006; Meliker and Jacquez 2007). Until now, the only implementation of this method was in the proprietary **SpaceStat** software owned by BiomedWare Inc. (BioMedware 2015b), a company founded by Geoffrey Jacquez of Jacquez's Q (BioMedware 2015a). A limitation of **SpaceStat** is that it is only available for the Microsoft Windows operating system (BioMedware 2015b), hence our development of **pyJacqQ**.

2. Code and its explanation

The latest version of the project can be found at `https://github.com/sjirjies/pyJacqQ`. Previous versions can be accessed at `https://github.com/sjirjies/pyJacqQ/releases`. The project is released under the GPL-3. More details can be found in the `LICENSE` file distributed with the project. The dependencies for **pyJacqQ** are Python version 3.4 or greater, **numpy** version 1.8.2 or greater (Van der Walt, Colbert, and Varoquaux 2011), and **scipy** version 0.13.3 or greater (Jones, Oliphant, Peterson, and others 2015).

We developed **pyJacqQ** using an object oriented approach. The entire analysis is encapsulated within a '`QStatsStudy`' object which contains all the global information needed to perform the method on a set of raw data. During analysis, a '`QStatsStudy`' object will have a list of '`StudyEnitity`' objects, one of '`FocusEntity`' objects, and another of '`TimeSlice`' objects. Each '`StudyEnitity`' object holds case-control information that does not change over time

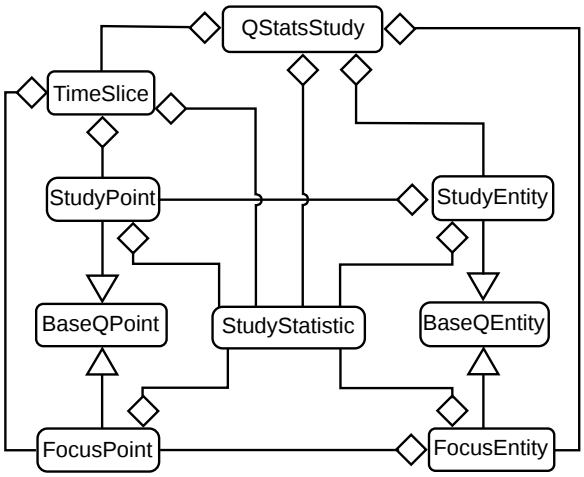

Figure 3: Simplified class diagram for interacting classes. The rectangles represent classes given by their containing label. A connecting line between classes represents a relationship between them. A diamond indicates that a class acts as a container, exhibiting a "has a" relationship. For example, an instance of '`QStatsStudy`' contains '`TimeSlice`' instances. A triangle indicates inheritance: the "is a" relationship. For example, '`StudyPoint`' is a special case of '`BaseQPoint`'. Cardinalities, attributes, and methods are not shown.

such as the individual's identifier, case-control status, case weight, exposure duration, latency period, and so forth. '`FocusEntity`' objects store similar information but for the real-world object represented by focus points. Both the '`StudyEnitity`' class and the '`FocusEntity`' class inherit from a '`BaseQEntity`' class to reduce redundancy.

A '`TimeSlice`' object is instantiated for every distinct time in which any case, control, or focus point moves location or when any case or control enters or exits its exposure trace if exposure clustering is enabled. Each '`TimeSlice`' object contains a list of '`StudyPoint`' objects and one of '`FocusPoint`' objects for every individual or focus entity present in the study at the time represented by the '`TimeSlice`' object. '`StudyPoint`' objects hold the information for a '`StudyEnitity`' object that changes over time, namely the x and y location. Likewise for '`FocusPoint`' objects and their associated '`FocusEntity`' objects. Again to reduce redundancy, the '`StudyPoint`' and '`FocusPoint`' classes inherit from a '`BaseQPoint`' class. Each of the previously mentioned '`StudyEnitity`' objects and '`FocusEntity`' objects maintains a list of their respective '`StudyPoint`' object and '`FocusPoint`' objects. In this scheme, we have an entity for each item of study linked to points representing the entity's location for each discrete time slice.

All of the objects previously mentioned contain the required information to calculate a particular statistic. This is done with the aid of a '`StudyStatistic`' object which is a container for the statistic and its p value. In Figure 3 we summarize the relationships in a simple class diagram.

The type of statistic tracked in a '`StudyStatistic`' object depends on its owner. For example, each '`StudyPoint`' object has a '`StudyStatistic`' object that records its Q_{it} statistic. Table 1 lists the relationships between the objects of the classes in the implementation and the Q-statistic tracked.

Object of class	'`StudyStatistic`'
'`QStatsStudy`'	Q and Q_f
'`TimeSlice`'	Q_t
'`StudyEnitity`'	Q_i
'`FocusEntity`'	Q_{fi}
'`StudyPoint`'	Q_{it}
'`FocusPoint`'	Q_{fit}

Table 1: Objects and their Q-statistic tracked by '`StudyStatistic`'.

As previously mentioned, the '`QStatsStudy`' class drives the analysis and is the only class the user must interact with to perform the calculations. First, a '`QStatsStudy object`' is instantiated with the locations of two files in comma-separated-values (CSV) form. The first CSV designates the identifier and case-control status of each individual in the study. Optionally it may also contain columns for the date of diagnosis, latency period, case weight, and exposure duration. The second CSV file holds the residential histories of the individuals with their identifiers, start and end dates of residence, and the x and y locations of their residence. '`QStatsStudy`' initialization also accepts the location of an optional third CSV for a focus geography. After this, a user would call the `run_analysis` method which takes study parameters such as the type of correction to apply and the number of permutation shuffles.

This method performs the majority of the work. It first loads the data from the CSVs and converts the raw CSV data to the previously discussed objects. Then it calculates the observed statistic, performs Monte Carlo testing to obtain the reference distribution, calculates the p values, applies any corrections specified for multiple testing, and then builds and returns a '`QStudyResults`' object that users can query for results. During calculation of the local statistic, nearest neighbor ties are broken arbitrarily. Multiple testing is dealt with by either applying the binomial tests previously mentioned, using a Benjamini-Yekutieli false discovery rate adjustment (Benjamini and Yekutieli 2001), or by ignoring it; as depending on user choice.

All of this previously mentioned functionality is contained within the `jacqq.py` file. This file also contains a function, `check_data_dirty`, that takes the paths to the CSVs and returns any errors present in the data; for example, missing values or incorrect data types. Users can run this function on their datasets before performing an analysis to check if they have any issues with their data. This file also contains a directive that instantiates a command line interface when it is called as the main module which by default runs `check_data_dirty` on the inputs and if any errors are found they are written to standard error, otherwise the analysis proceeds as usual. **pyJacqQ** also contains other files that are used to generate simulated data and perform unit testing using test cases to ensure the validity of the implementation.

2.1. Test cases

We created test cases for the entire implementation using test datasets. These tests and datasets are available in the `tests` directory within the project. We fed these datasets through **SpaceStat** v. 4.0.20 (BioMedware 2015b) and the results of all statistics are converted to CSV and saved with the project. This is done once for each test data set. When these tests are run, the test datasets are fed into **pyJacqQ** producing results for all statistics. The "correct"

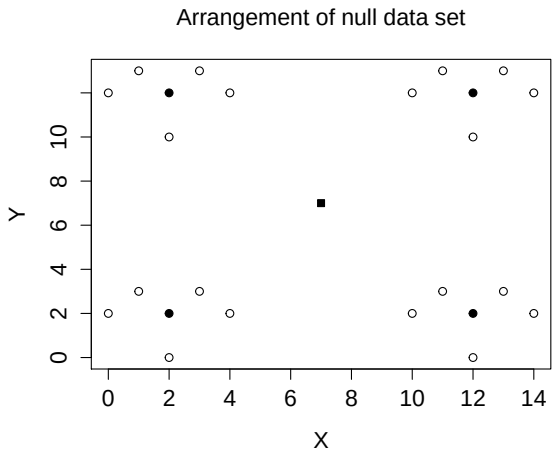

Figure 4: Null test dataset geography. The black circles indicate case locations and the white circles indicate control locations. Each case is surrounded by controls. The black square in the center represents a focus point. The example shown here is a 2×2 lattice. This pattern can be repeated to any size.

results from **SpaceStat** (BioMedware 2015b) are then loaded from file, and the two sets of results are compared for differences. If differences in the statistic are found the test fails producing a message specifying the reason for failure. Since both implementations use Monte Carlo tests to acquire the p values and therefore contain elements of randomness, the p values are not compared. These tests are available in the project and can be automatically run at any time. Test datasets are kept small enough to run quickly while still allowing for a meaningful test. This is important as they were run frequently before and after the addition of each feature to ensure that changes to the code did not invalidate the implementation.

We crafted several of these test datasets under different study conditions. We designed one of these, which we call a null data set, such that all resulting statistics are zero. We achieved this by creating a lattice where each case is surrounded by five controls in isolated pentagonal patterns so that the five nearest neighbors of each case consist entirely of controls. Similarly, focus points are placed between these clusters so that their five nearest neighbors are also composed entirely of controls. This null data set spans a single time duration and results in only one time slice. This test dataset is the output of a script we wrote that is available in the project and creates this pattern for any user specified lattice size. The purpose of this dataset is to detect any bugs that would affect the edge case of a zero statistic. In Figure 4 we diagram the null data set.

To test clustering in maximum and moderation we created a simple dataset distinguished by three primary clusters. The first is a cluster of cases, the second a mixture of cases and controls, and the third a cluster of controls. A focus point is also placed within each cluster. This simple, static dataset also spans a single time slice. In Figure 5 we show the distribution of points. This dataset tests the situation in which clustering of cases occurs at its maximum and near the center of its range when five nearest neighbors are used during calculation of the statistics. For example, using five nearest neighbors we expect the left focus point to have a Q_{fit} of five cases since it is surrounded only by cases.

We also created a dataset to test the case weights feature. This test dataset consists of four

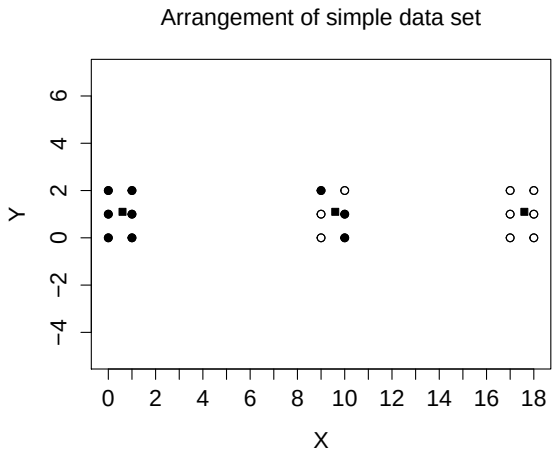

Figure 5: Simple test dataset geography. Cases and controls are shown as black and white circles respectively. The black squares represent focus points. The left cluster consists of cases and the right of controls. A mix of case and controls occupies the center of the geography.

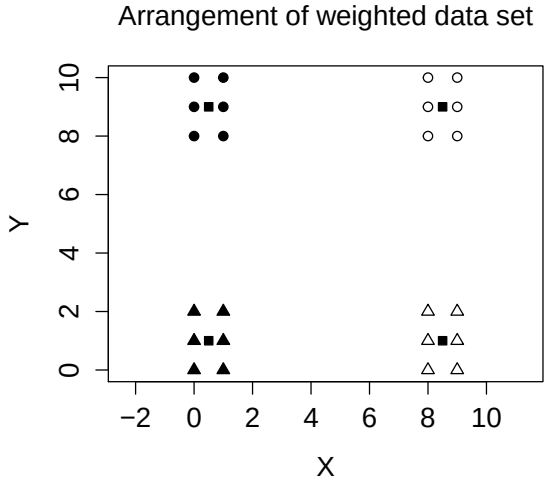

Figure 6: Weighted test dataset geography. Black triangles and black circles are heavy-weighted cases and light-weighted cases respectively. White triangles and white circles are heavy-weighted controls and light-weighted controls respectively. Black squares are focus points.

clusters. The first consists of cases with the maximum weight, the second consists of cases with the minimum weight, the third consists of controls with the maximum weight, and the fourth consists of controls with the minimum weight. Here the maximum weight is unity (1.0) and the minimum weight is zero (0.0). This dataset allows us to test the extreme situations using the case weights option. A focus point is also placed within each cluster. In Figure 6 we show the geography of this weights dataset.

We designed yet another test dataset, this time to test the exposure clustering feature. This test dataset ensures that the first and last dates of an exposure trace, or the date of initial exposure and date of disease formation, are handled correctly. The geography is static and

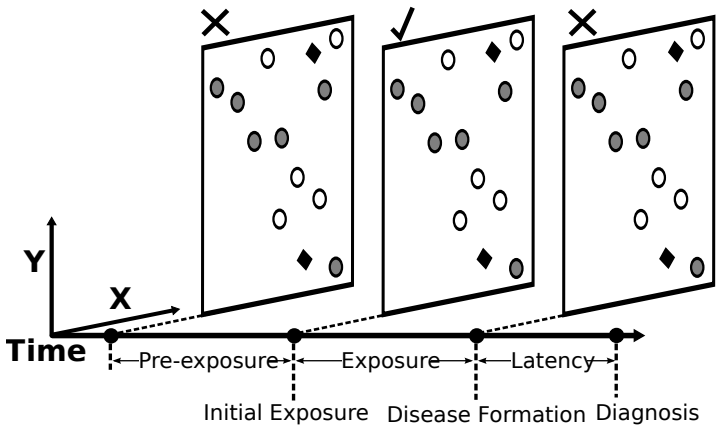

Figure 7: Milestones in the exposure test dataset with geography. Here gray circle and white circles are cases and controls respectively; diamonds are focus points. The exposure trace is the period of time between initial disease exposure and the disease formation. All cases share this same exposure trace; therefore they are eligible for disease exposure on the same date of initial exposure up to, but not including, the same date of disease formation. As shown by the 'X's above the first and last time slices, theses should not exhibit any clustering or non-zero statistics as they lie outside the exposure trace for all cases. The central time slice, indicated by the checkmark, should have non-zero statistics as it lies within the exposure trace.

consists of scattered cases, controls, and focus points. Each of these have only a single location of residence that spans a single, large date range. Everybody is given the same date of diagnoses, latency period, and exposure duration. The values for these are chosen such that all individuals have the same dates of initial exposure and disease formation. The exposure trace for each individual falls entirely within the start and end dates of their residential histories. Case points are only counted if they are active, that is, within a time slice that falls on or after the case's date of initial exposure but before the cases's date of disease formation.

Because all cases in this test dataset have the same dates of initial exposure and disease formation, they are all active within the same window of time. Proper analysis of such a dataset should produce three time durations split by the dates of initial exposure and disease formation. The first and last time duration should contain only statistics with values of zero since these durations lie outside the individuals' exposure traces. The center duration should contain non-zero clustering statistics. In Figure 7 we illustrate the concept using this exposure dataset.

Lastly, we created dirty datasets each with a unique problem, such as missing column titles or empty fields, to test that the `check_data_dirty` function detects them. This is all in addition to unit tests that ensure that the most commonly used methods of each class produce expected output under several cases. All of the test scripts are available in the `tests` folder of the project and the test datasets are available in the `datasets` directory within that folder.

3. Examples

As previously mentioned, **pyJacqQ** has a Python application programming interface (API) and a command line interface (CLI). Examples of both are given below.

3.1. Application programming interface

Here is an example use of the API as seen using Python's interactive interpreter. The first consideration is instantiating a '`QStatsStudy`' object with the file paths of the details, residential histories, and optionally the focus geography

```
>>> import jacqq
>>> details = "tests/simulation_data/input_details.csv"
>>> histories = "tests/simulation_data/input_residence_histories.csv"
>>> focus = "tests/simulation_data/input_focus.csv"
```

At this point you may elect to use the data preparser to check for errors given the parameters you plan to use:

```
>>> errors = jacqq.check_data_dirty(details, histories, focus,
...   exposure = True, weights = True)
>>> print(errors)

  []
```

This returns a list of errors present in the data that need correction, for example, missing attributes or wrong data types. If errors are present, they should be corrected and `check_data_dirty` re-run to ensure no additional errors. Next instantiate a '`QStatsStudy`' object with the location of the input files:

```
>>> study = jacqq.QStatsStudy(details, histories, focus)
```

At this point, simply call the `run_analysis` method with the parameters desired:

```
>>> r = study.run_analysis(k = 5, use_exposure = True,
...   use_weights = True, correction = 'BINOM', seed = 4077096852)
```

This returns a '`QStudyResults`' object that contains all the results for the study ran with the given options. Note that the `run_analysis` method could be re-run and the new results assigned to a different variable. In this case `r` contains the results. The results contain global study data, case data, and focus entity data. In addition it holds time slice data and all the data for every point within every time slice. To make the '`QStudyResults`' object convenient, users can query the results across either the individual or date axis. In addition it is possible to select only significant results. Below are example operations:

Find out if exposure clustering was used

```
>>> r.exposure_enabled

True
```

Get the adjusted (or final) alpha (significance level)

```
>>> r.adjusted_alpha

0.05
```

Get the number of shuffles in the Monte Carlo testing

```
>>> r.number_permutation_shuffles

99
```

Get the seed value used for random number generation

```
>>> r.seed

4077096852
```

Get global Q as (statistic case-years, p value, significance)

```
>>> r.Q_case_years

(501.26575342465753, 0.01, 1)
```

Get the Q_f statistic normalized by the number of cases

```
>>> r.normalized_Qf

1.2561643835616438
```

Get all of the time slice results

```
>>> r.time_slices

OrderedDict([(20150101,
    <jacqq.QStudyTimeSliceResult object at 0x7f276e8c1080>), ... ])
```

Get the Q_t statistic for the slice at January 3rd, 2015

```
>>> r.time_slices[20150103].stat

(87, 0.02, 1)
```

Get a list of only significant case points at that date

```
>>> r.time_slices[20150103].sig_points
```

```
OrderedDict([('BM',
    <jacqq.QStudyPointResult object at 0x7f276e880d68>), ... ])
```

Get the local Q_{it} statistic for case 'BM' on January 3rd, 2015

```
>>> r.time_slices[20150103].points['BM'].stat

(2, 0.03, 1)

>>> r.cases['BM'].points[20150103].stat

(2, 0.03, 1)
```

Get only significant Q_{it} stats for case 'BM'

```
>>> r.cases['BM'].sig_points

OrderedDict([(20150101,
    <jacqq.QStudyPointResult object at 0x7f276e849c18>), ...])
```

Find the x, y location of case 'BM' on January 2nd, 2015

```
>>> r.cases['BM'].points[20150102].loc

(59.0, 67.0)
```

Get the focus Q_{fi} results in tabular/tuple form

```
>>> r.get_tabular_focus_data()

(['id', 'Qif_case_years', 'pval', 'sig'],
    [['Away From Sources', 0.0, 1.0, 0],
    ['Large Constant', 2.106849315068493, 0.01, 1],
    ['Medium Linear', 1.2027397260273973, 0.01, 1],
    ['Small Constant', 1.715068493150685, 0.01, 1]])
```

Get the binomial test results for dates as (number significant statistics, p value, significance)

```
>>> r.binom.dates

(200, 1.1102230246251565e-16, 1)
```

Get time slices that have less than $k + 1$ points

```
>>> r.dates_lower_k_plus_one

{}
```

Write the results to files using a path and filename prefix:

```
>>> r.write_to_files_prefixed('path/to/my_results_folder', 'my_study_prefix')
```

Write the results to files but specify individual file names:

```
>>> r.write_to_files('global.csv', 'cases.csv', 'dates.csv',
...   'local_cases.csv', 'focus_results.csv', 'focus_local.csv')
```

More detail can be found using the built-in Python `help()` function on any of the objects or classes.

3.2. Command line interface

To use the CLI simply call Python with `jacqq.py` and the required parameters. These parameters are the location of input CSV files and the desired location of output CSV files. In addition, flags such as the number of permutation shuffles and significance can be set. Some options, such as the use of exposure clustering or case weights, are set by the presence of a flag without further values.

An explanation of all options and flags can be accessed by passing a `-h` or `-help` flag:

```
$ python3 jacqq.py --help

usage: jacqq.py [-h] --resident HISTORIES --details DETAILS --output_location
                OUTPUT_LOCATION --output_prefix OUTPUT_PREFIX [--exposure]
                [--weights] [--focus_data FOCUS_DATA] [--neighbors NEIGHBORS]
                [--alpha ALPHA] [--shuffles SHUFFLES]
                [--correction CORRECTION] [--no_inspect] [--seed SEED]
                [--only-cases]

Calculate the Jacquez Q-statistics.

optional arguments:
  -h, --help            show this help message and exit
  --resident HISTORIES, -r HISTORIES
                        Location of the residential histories file. (default:
                        None)
  --details DETAILS, -d DETAILS
                        Location of individuals' status dataset. Case-control
                        status must be given for all individuals. (default:
                        None)
  --output_location OUTPUT_LOCATION, -o OUTPUT_LOCATION
                        Pathway to the folder to output the results. (default:
                        None)
  --output_prefix OUTPUT_PREFIX, -p OUTPUT_PREFIX
                        The prefix to include in the file names of the output.
                        (default: None)
```

```
  --exposure, -e        If this flag is added then the dataset containing
                        case-control flags must also contain columns 'DOD' and
                        'latency' for the date of diagnosis and the number of
                        days of disease latency. These values must be
                        specified for both cases and controls. The presence of
                        this flag signals for clustering of exposure. In its
                        absence clustering of points will be done instead.
                        (default: False)
  --weights, -w         If this flag is supplied then individuals will have
                        their case-control flags shuffled in a way that
                        adjusts for co-variates. A 'weight' column must be
                        supplied in the case-control dataset with values
                        between 0 and 1. If this flag is not added then the
                        calculation will assume equal disease risk for
                        everyone. (default: False)
  --focus_data FOCUS_DATA, -f FOCUS_DATA
                        Location of the dataset containing focus points of
                        geographic interest, such as factories. The dataset
                        must be in time series format. (default: None)
  --neighbors NEIGHBORS, -k NEIGHBORS
                        Value K to use for number of nearest neighbors.
                        (default: 15)
  --alpha ALPHA, -a ALPHA
                        Value used to check for significance of test results.
                        (default: 0.05)
  --shuffles SHUFFLES, -s SHUFFLES
                        The number of case-control permutations to conduct
                        when calculating pseudo p-values. (default: 99)
  --correction CORRECTION, -c CORRECTION
                        Correction to apply for multiple testing. 'FDR'
                        applies a Benjamini-Yekutieli False Discovery Rate.
                        Note that this often requires a large number of
                        shuffles for any significance.'BINOM' applies the
                        binomial method used in [1]; this is the default. If
                        any other string such as 'NONE' is given than no
                        correction will be used. (default: BINOM)
  --no_inspect, -N      Pass this flag to prevent the program from pre-parsing
                        the data for errors. (default: False)
  --seed SEED           The seed to use with the random number generator.
                        (default: None)
  --only-cases, -O      Pass this flag to prevent output of control results.
                        (default: False)
```

[1] Sloan CD, Jacquez GM, Gallagher CM, et al. Performance of cancer cluster Q-statistics for case-control residential histories. Spatial and spatio-temporal epidemiology. 2012;3(4):297-310. doi:10.1016/j.sste.2012.09.002.

For example, say the residential histories file, case-control details file, and optional focus points file are located within the `tests/simulation_data/` directory with respective file names `input_residence_histories.csv`, `input_details.csv`, and `input_focus.csv`. The following example runs the analysis using exposure clustering and case weight and saves the output to various files prefixed with '`study1`' in a directory '`studyDir`':

```
$ python3 jacqq.py -r=tests/simulation_data/input_residence_histories.csv \
>   -d=tests/simulation_data/input_details.csv \
>   -f=tests/simulation_data/input_focus.csv \
>   -o=studyDir -p=study1 -e -w
```

When using the CLI, **pyJacqQ** will first pre-parse the datasets and write any errors to standard error. Users are expected to repair these errors and retry. An example is given below:

```
$ python3 jacqq.py \
>   -r=tests/datasets/dirty_data/histories_wrong_types.csv \
>   -d=tests/simulation_data/input_details.csv \
>   -o=studyDir -p=study1 -e -w
```

This would produce the following error:

```
File 'histories_wrong_types':
        Row 2 requires date format YYYYMMDD for attribute 'start_date'
File 'histories_wrong_types':
        Row 3 requires date format YYYYMMDD for attribute 'end_date'
File 'histories_wrong_types':
        Row 4 requires a number for attribute 'x'
```

The error pre-parser can be disabled by passing the `no_inspect` or `-N` flag.

If the `tests` folder is set as the current working directory, all the unit tests can be run automatically using **nose** (Pellerin 2009).

```
pyJacqQ/tests$ nosetests3

.....................................................................
----------------------------------------------------------------------
Ran 103 tests in 0.259s

OK
```

4. Conclusion

Jacquez's Q is a recent advancement in space-time analysis that detects clusters of disease exposure. This has significant potential for epidemiology since it detects the location and duration of clusters. Up until now, the only available implementation of this method was in **SpaceStat** (BioMedware 2015b) owned by BiomedWare. We here present the first open-source version of this method, available at `https://github.com/sjirjies/pyJacqQ`. We hope our implementation will be of use to public health researchers.

References

Benjamini Y, Yekutieli D (2001). "The Control of the False Discovery Rate in Multiple Testing under Dependency." *The Annals of Statistics*, **29**(4), 1165–1188. doi:10.1214/aos/1013699998.

BioMedware (2015a). "About." URL http://www.biomedware.com/?module=Page&sID=about.

BioMedware (2015b). "**SpaceStat**." URL http://www.biomedware.com/?module=Page&sID=spacestat.

Cuzick J, Edwards R (1990). "Spatial Clustering for Inhomogeneous Populations." *Journal of the Royal Statistical B*, **52**(1), 73–104.

Jacquez G (1996). "A k Nearest Neighbour Test for Space-Time Interaction." *Statistics in Medicine*, **15**(18), 1935–1949. doi:10.1002/(sici)1097-0258(19960930)15:18<1935::aid-sim406>3.0.co;2-i.

Jacquez G, Kaufmann A, Meliker J, Goovaerts P, AvRuskin G, Nriagu J (2005). "Global, Local and Focused Geographic Clustering for Case-Control Data With Residential Histories." *Environmental Health: A Global Access Science Source*, **4**(4). doi:10.1186/1476-069x-4-4.

Jacquez G, Meliker J, AvRuskin G, Goovaerts P, Kaufmann A, Wilson M, Nriagu J (2006). "Case-Control Geographic Clustering for Residential Histories Accounting for Risk Factors and Covariates." *International Journal of Health Geographics*, **5**(32). doi:10.1186/1476-072X-5-32.

Jones E, Oliphant T, Peterson P, others (2015). "**SciPy**: Open Source Scientific Tools for Python." [Online; accessed 2015-05-29], URL http://www.scipy.org/.

Knox E, Bartlett M (1964). "The Detection of Space-Time Interactions." *Journal of the Royal Statistical Society C*, **13**(1), 25–30. doi:10.2307/2985220.

Meliker JR, Jacquez GM (2007). "Space-Time Clustering of Case-Control Data with Residential Histories: Insights into Empirical Induction Periods, Age-Specific Susceptibility, and Calendar Year-Specific Effects." *Stochastic Environmental Research and Risk Assessment*, **21**(5), 625–634. doi:10.1007/s00477-007-0140-3.

Moran P (1948). "The Interpretation of Statistical Maps." *Journal of the Royal Statistical Society B*, **10**(2), 243–251.

Nordsborg R, Meliker J, Ersboll A, Jacquez G, Raaschou-Nielsen O (2013). "Space-Time Clustering of Non-Hodgkin Lymphoma Using Residential Histories in a Danish Case-Control Study." *PLoS ONE*, **8**(4), e60800. doi:10.1371/journal.pone.0060800.

Pellerin J (2009). "**nose** 1.3.6." [Online; accessed 2015-06-05], URL https://nose.readthedocs.org/en/latest/.

Sloan C, Jacquez G, Gallagher C, Ward M, Raaschou-Nielsen O, Nordsborg R, Meliker J (2012). "Performance of Cancer Cluster Q-Statistics for Case-Control Residential Histories." *Spatial and Spatio-Temporal Epidemiology*, **3**(4), 297–310. doi:10.1016/j.sste.2012.09.002.

Van der Walt S, Colbert SC, Varoquaux G (2011). "The **NumPy** Array: A Structure for Efficient Numerical Computation." *Computing in Science Engineering*, **13**(2), 22–30. doi:10.1109/mcse.2011.37.

Affiliation:

Matthew Scotch
Department of Biomedical Informatics
Arizona State University
Tempe, Arizona, United States of America
E-mail: matthew.scotch@asu.edu
URL: https://webapp4.asu.edu/directory/person/1615221

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
October 2016, Volume 74, Issue 6 *Submitted:* 2015-06-19
doi:10.18637/jss.v074.i06 *Accepted:* 2015-09-07