NPST: A Generalized **N**on-**P**arametric **S**easonality **T**est

Roland Rau[*]

April 11, 2006

Abstract

The present article suggests a non-parametric test for seasonality based on the work of Hewitt et al. (1971) and Rogerson (1996). The new test is a generalized version of the previous tests which relaxes their respective assumptions. An algorithm has been developed which allows to empirically determine the distribution of the test statistic using Monte Carlo simulation. This algorithm has been implemented in various computer languages and tentatively been called NPST which stands for **N**on-**P**arametric **S**easonality **T**est. The implementations have been checked for correctness and benchmarked for speed of executing the code. The software can be obtained from the author.

1 Introduction

More than 2000 years ago, Hippocrates already pointed at the effect of seasons on diseases. If diseases, and ultimately mortality, occur seasonally, "an environmental factor has to be considered in the etiology of that disease" (Marrero, 1983). Diseases like cardiovascular, cerebrovascular and respiratory diseases, which constitute more than half of all deaths in modern Western societies, all show a pronounced seasonal pattern with a peak in winter and a trough in summer (Eurowinter Group, 1997, 2000; Kunst et al., 1990; Mackenbach et al., 1992; Rau, 2005; Yen et al., 2000).

[*]Max Planck Institute for Demographic Research, Konrad Zuse Str. 1, 18057 Rostock, Germany; Phone: +49-381-2081-109; Fax: +49-381-2081-409, Email: `rau@demogr.mpg.de`

Despite the knowledge of seasonal effects on diseases for two millennia, the definition and the measurement of seasonality has not been the center of attention until Edwards (1961) developed a test based on a geometrical framework which was specifically designed for seasonality. It turned out to become "the most cited and the benchmark against which other tests are evaluated" (Wallenstein et al., 1989, p. 817). In his article, Edwards explicitly also mentions the possibility to estimate "cyclic trends" by considering "the ranking order of the events which are above or below the median number" (Edwards, 1961, p. 83). This idea has been taken up by Hewitt et al. (1971). They did not use a binary indicator as suggested by Edwards but all the ranking information.

Rogerson (1996) made a first step to generalize this test, relaxing the relatively strict assumption of Hewitt et al. (1971) that seasonality is only present if a six-month peak period is followed by a six-month trough period. Rogerson allowed that the peak period can also last three, four, or five months.

The present article with its associated software relaxes these assumptions even further and allows total flexibility for the basic time duration as well as for the length of the peak period. Hence, one is not restricted anymore to a certain structure of the data. Weekly values, for example, do not have to be converted anymore into monthly values. With the present implementation it is possible to test, for example, whether there is a peak period of twelve weeks during a year with 52 weeks.

2 Nonparametric Seasonality Tests

Based on a brief suggestion in Edwards (1961), Hewitt et al. (1971) elaborated a non-parametric test based on rank sums. While Edwards suggested "to consider the ranking order of the events which are above or below the median number" (Edwards, 1961, p. 83), Hewitt et al. propose to use "all the ranking information rather than a simple dichotomy" (Hewitt et al., 1971, p. 175). According to them, the monthly frequencies are ranked. The month with most occurrences (e.g. deaths) will have the value "12" assigned. Consequently, "1" indicates the month having the least events or the lowest rate. Keeping the original order of the months (e.g. starting with January and ending with December), we can calculate the rank-sums of six consecutive months (January–June, February–July, ..., December–May). This ranksum serves as the test statistic T to test the Null hypothesis of no seasonality against the alternative hypothesis of seasonality in the data. T is symmetrically distributed and can reach a maximum value of 57 in this example $(12 + 11 + 10 + 9 + 8 + 7 = 57)$.

This test has the advantage that one "avoids the problem of specifying a particular algebraic version" (Freedman, 1979, p. 225) of what is meant by seasonal fluctuation. This test cannot be applied — as Reijneveld (1990) points out — if there are ties. Ties, however, almost never occur in demographic applications as data typically come in large numbers that monthly rates or cases are not equivalent. One can also object — as Rogerson (1996), Wallenstein et al. (1989) and Marrero (1983) did — to "the assumption that the year is split into two equally wide intervals of 6 months each" (Rogerson, 1996, p. 644). While Wallenstein et al. and Marrero take a "one-pulse model" into consideration to overcome this drawback, Rogerson developed a generalization of Hewitt's Test for peak periods of 3, 4, and 5 months (Rogerson, 1996). Similar to take the maximum rank sum of all possible combinations of six consecutive months, Rogerson uses the maximum rank sum of any consecutive three, four, or five months period, respectively. Because of its relative simplicity to calculate, Hewitt's Test has enjoyed widespread use (Rogerson, 1996). For example in the case of seasonal mortality, Akslen and Hartveit (1988) used Hewitt's test to analyze seasonal variation in melanoma deaths.

We consider the work of Hewitt et al. (1971) and Rogerson (1996) only as a first step. Sometimes, data are not provided in the appropriate form, i.e. 12 months. For example, in the United Kingdom weekly bills of mortality are published for several centuries (see for a copy of the earliest known Weekly Bill of Mortality: Ogle, 1892). One approach would be to convert the data into an appropriate form for the test. We suggest, however, another approach: we generalize the test to allow for any form of the data. It means that it does not matter whether the data are given in a monthly, bi-weekly, weekly or even daily manner. In addition, we are not restricted to a specified peak period as it was the case in the tests by Hewitt et al. (1971) and Rogerson (1996) with peak periods of 3, 4, 5, or 6 months.

3 The Distribution of the Test Statistic T

The test statistic T, which is the maximum rank sum (of six consecutive months for the situation described by Hewitt et al. (1971)), does not follow any “standard” distribution such as the χ^2 or the t distribution. It can be obtained by either one of two approaches: A combinatorial method has been outlined by Walter (1980). But as pointed out by the original authors (Walter, 1980) as well as by Rogerson (1996), it is relatively tedious since one of the requirements of this method is to know how many different permutations are possible to obtain a fixed ranksum. It is relatively easy for the highest ranksum; with decreasing ranksum, however, it becomes increasingly difficult to calculate all possibilities for a fixed ranksum as shown by Walter (1980). Therefore we opted for the other approach: Using Monte-Carlo simulation to generate the distribution of T. This has been done by the original authors (Hewitt et al., 1971) as well as by Rogerson (1996) for their restrictive frameworks. The algorithm for our generalized these Monte-Carlo simulations is described in the next section. The software to derive the distribution of T empirically has tentatively been called NPST which stands for **N**on-**P**arametric **S**easonality **T**est.

4 The algorithm

Irrespective of the specific implementation language, the same algorithm has been used in each implementation. After the theoretical presentation of the algorithm, we will give an illustrating example using the situation Hewitt et al. had in mind with a basic period of twelve months and a peak period of six months. Please note that the Hewitt-approach represents a special case for our algorithm which allows total flexibility for the length of the base period and the length of the peak period.

4.1 Theoretical Description

The single steps in the algorithm are:

1. The user has to input four parameters:
   (a) How long is the duration of the underlying data? This parameter is denoted as `long`.
   (b) The second parameter specifies the length of the peak period of interest (`peak`).
   (c) The third parameter fixes the number of Monte Carlo samples which should be generated (`repts`).

   (d) The last parameter (`outputrows`) is not required by the algorithm itself. It sets the number of rows which should be output in the end.

2. We generate a vector with integers (`basedata`) numbered from 1 to `long`.

3. The maximum rank sum `maxranksum` is calculated by summing up the last `peak` elements of `basedata`.

4. A vector is generated of length `maxranksum` called `resultvector`. Initially, each element has the value 0.

5. We make a loop with `repts` repetitions which iterates through the following steps:

   (a) We generate a random permutation of `basedata` called `randperm`.

   (b) An array is generated with twice the length of `basedata` in which `randperm` is duplicated. Let's call this new array `doublerandperm`.

   (c) Now we calculate all rank sums of `peak` consecutive months. Using the duplicated array from the previous sub-step facilitates the work. If we had not duplicated `randperm` we would need to switch back to the first element when it is required to calculate the rank sum from December until March, for example. With `doublerandperm` we can just continue counting from element 12 until 15.

   (d) The maximum of the consecutive rank-sums is chosen (`maxvalue`). The `maxvalue`$^{\text{th}}$ position of `resultvector` is increased by a value of 1.

6. After the loop is finished, `resultvector` is reversed and the relative cumulative frequencies of each position are calculated (`reversed-ecdf`).

7. The computer is now printing the first `outputrows` elements of `reversed-ecdf` in conjunction with the corresponding rank sums.

4.2 Example

We will now present how the algorithm would work within the framework provided by Hewitt et al. (1971) with a peak period of six months within one year of twelve months.

1. The four parameters:
   (a) `long` $\leftarrow 12$
   (b) `peak` $\leftarrow 6$
   (c) `repts` $\leftarrow 1000$
   (d) `outputrows` $\leftarrow 5$

2. `basedata`

1	2	3	4	5	6	7	8	9	10	11	12

3. The maximum rank sum `maxranksum` is calculated by summing up the last `peak` elements of `basedata`.

$$\sum_{i=long-peak+1}^{long} i = \sum_{i=12-6+1=7}^{12} i = 7+8+9+10+11+12 = 57$$

4. `resultvector` (by row)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0			

5. The loop
   (a) `randperm`.

9	7	10	5	1	8	2	6	3	11	12	4

   (b) `doublerandperm`

9	7	10	5	1	8	2	6	3	11	12	4	9	7	...	11	12	4

(c) Consecutive rank sums

- $9 + 7 + 10 + 5 + 1 + 8 = 40$
- $7 + 10 + 5 + 1 + 8 + 2 = 33$
- $10 + 5 + 1 + 8 + 2 + 6 = 32$
- ...
- $4 + 9 + 7 + 10 + 5 + 1 = 36$

(d) `maxvalue`$= 53$ ($11 + 12 + 4 + 9 + 7 + 10$, October–March);

- `resultvector` after one iteration step:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0			

- `resultvector` after `repts` $= 1000$ iteration steps (by rows):

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	2	11	22	61	94
111	117	108	99	86	77	54	70	41	22	12	12			

6. `reversed-ecdf`:

0.012	0.024	0.046	0.087	0.157	0.211	0.288	0.374	0.473	0.581
0.698	0.809	0.903	0.964	0.986	0.997	0.999	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1			

7. Output of `outputrows` $= 5$ values:

Ranksum:	57	56	55	54	53
Simulated Cumulative Probability:	0.012	0.024	0.046	0.087	0.157

These empirical results are close to the exact values shown in Table 1 given by Walter (1980):

Based on this test, a ranksum of 57 is, therefore, required to reject the Null-Hypothesis of no seasonality on (approximately) the 1%-level and of 55 on (approximately) the 5%-level.

Table 1: Exact Distribution:

Ranksum:	57	56	55	54	53
Exact Values:	0.0130	0.0253	0.0483	0.0805	0.1209

Source: Walter, 1980, p. 148

5 Implementations

This algorithm has been implemented in various programming languages. The correctness of the implementations has been checked by comparing the results with the tabulated values given in Hewitt et al. (1971), Walter (1980), and Rogerson (1996). Table 2 on page 9 gives an overview in which languages this algorithm has been successfully implemented. It also shows how quickly the execution of the code finished using a benchmark case with a base-length of twelve months, a peak-period of six month and 100,000 repetitions. The results here are the outcome of tests on a computer having a 3GHz Intel Pentium 4 Processor with 512MB RAM running Microsoft Windows XP Professional SP2.[1]

As one can see, the speed of executing the benchmark case varies considerably among the languages.[2] The most important indicator for speed is the question whether there is a compiler available which translates the source code into native code or not. It takes less than one second for either C or Ocaml to perform the benchmark test. Please see the Appendix for further explanations of the tests.

One has to ask, of course, also: how many repetitions are necessary to have values which are approximately close to the theoretical values? Maybe 5,000 repetitions are enough as conducted in the original article? Our ex-

[1]If compilers/interpreters were available under a Free Software licence comparative tests have been conducted also on a computer with an AMD64 3000+ processor with 1GB of RAM running the 64bit-version of Suse Linux 9.3 Professional. These results differed only slightly from the ones conducted on the Windows computer.

[2]The chosen languages are, admittedly, highly selective and reflect my personal preference for functional approaches to programming. Please check section A on page 15 in the Appendix for the webpages of the various languages.

Table 2: Comparing the Speed of Various Implementations of the "NPST"-algorithm With a Basic Duration of 12 Month, a Peak Period of 6 months and 100,000 repetitions

Language	Version / Implementation	Mode of Code Execution	Time (in sec.)
R	2.1.0	interpreted	33
S-Plus	7.0.3	interpreted	335
Python	2.3.4	interpreted	7
Lisp-Stat	3.04	interpreted	20
	3.04	byte-compiled	10
Common-Lisp	Corman-Lisp 2.51	native-code compiled	9
	SBCL 0.9.9	native-code compiled	2
	Allegro CL 7.0 Trial	interpreted	1000
	Allegro CL 7.0 Trial	compiled‡	5
	GCL 2.6.1	interpreted	78
	GCL 2.6.1	native-code compiled	7
	Clisp 2.33.1†	interpreted	97
	Clisp 2.33.1†	byte-compiled	19
	LispWorks PE 4.3.7	interpreted	54
	LispWorks PE 4.3.7	compiled‡	17
Emacs-Lisp	GNU Emacs 21.3.1	interpreted	81
	GNU Emacs 21.3.1	byte-compiled	37
	XEmacs 21.4.13	interpreted	74
	XEmacs 21.4.13	byte-compiled	41
OCaml	3.08.3	interpreted	4
	3.08.3	byte-compiled	4
	3.08.3	native-code compiled	< 1
C	GCC 3.3.1 (Flag: -O3)	native-code compiled	< 1

† Running on Cygwin, the GNU/Linux-like environment for MS Windows

‡ Due to the remarkable speed advantage, we assume native-code compilation.

ecution time increases linearly with the number of repetitions.[3] Therefore even the slowest implementation would be finished in less than 20 seconds

[3]Thus, one can denote our algorithm in computer science terms as an $O(n)$-algorithm (see Knuth, 1998).

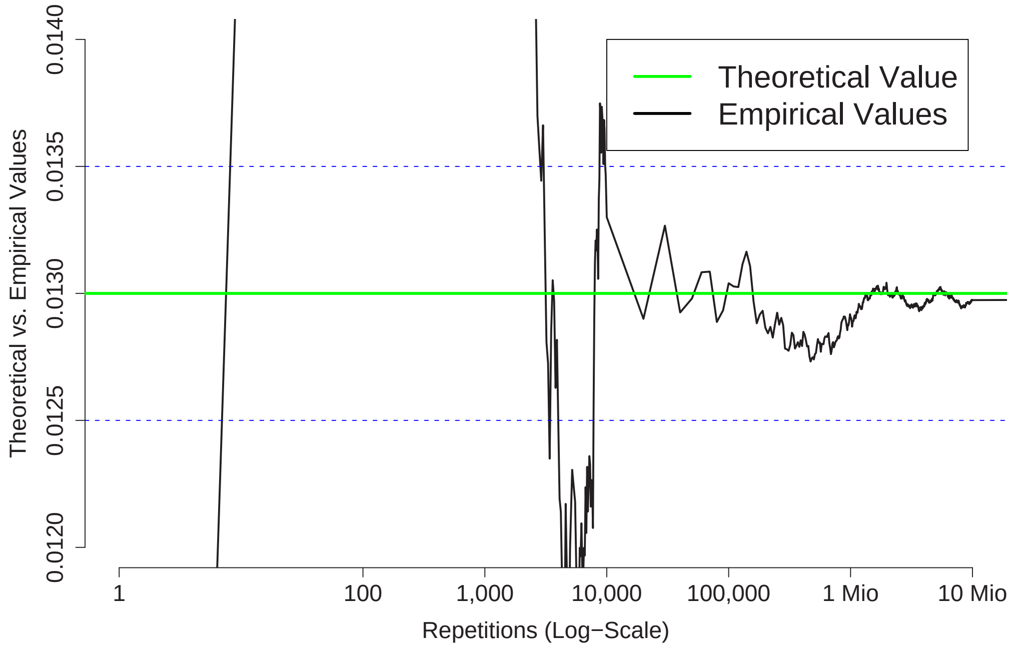
for 5,000 repetitions based on our results from Table 2.
A test has been conducted using an implementation of NPST in Python with 10 Mio. repetitions. Every 100 repetitions the current value of the empirical probability of a maximum ranksum of 57 in our benchmark case has been written to a file. After 10,000 repetitions, every 10,000 repetitions were written to that file. The contents of this file is plotted in Figure 1 on page 11. On the horizontal axis the number of repetitions are plotted on a log-scale. The green horizontal line indicates the true value for the probability that a maximum ranksum of 57 is obtained in the benchmark case ($\mathsf{P}(\mathsf{T} \geq 57) = \mathsf{P}(\mathsf{T} = 57) = 0.0130$ as given by Walter (1980)). The two blue dashed lines mark a corridor of ± 0.0005 around the true value. As indicated by the black solid line, fewer than 100,000 repetitions can yield unsatisfactory results. The results from Figure 1 suggest that one should use at least one million repetitions — preferably even more.

Due to these requirements of having at least 10^6 repetitions, a suitable implementation for distribution among interested researchers should contain a freely available compiler which translates the source code into *native*-code for the respective architecture. This leaves basically only Ocaml and C/GCC among the languages in which NPST is implemented at the moment. Both are licenced under a Free Software licence,[4] available on most architectures and operating systems and are able to develop standalone-applications easily. The language of choice is Ocaml. Although it is slightly slower than C in the current implementations, it has two major advantages: automatic memory managment and the possibility of rapid prototyping via the interpreter which shortens the typical tedious "edit-compile-test-debug cycle" of compiled languages.

The current version of the NPST-implementation in Ocaml on WinXP is shown in Figure 2 on page 12. After specifying the four required parameters on the command-line, the program prints the meaning of the input, the empirical values from the present NPST-run as well as the duration of executing the current NPST-run. Please note how close the empirical values after one million repetitions are to the theoretical values as specified by Walter (1980) and shown in Table 1 on page 8.

[4]GCC is licenced under the GPL, Ocaml under the QPL. Please see `http://www.gnu.org/philosophy/free-sw.html` for a Definition of "Free Software".

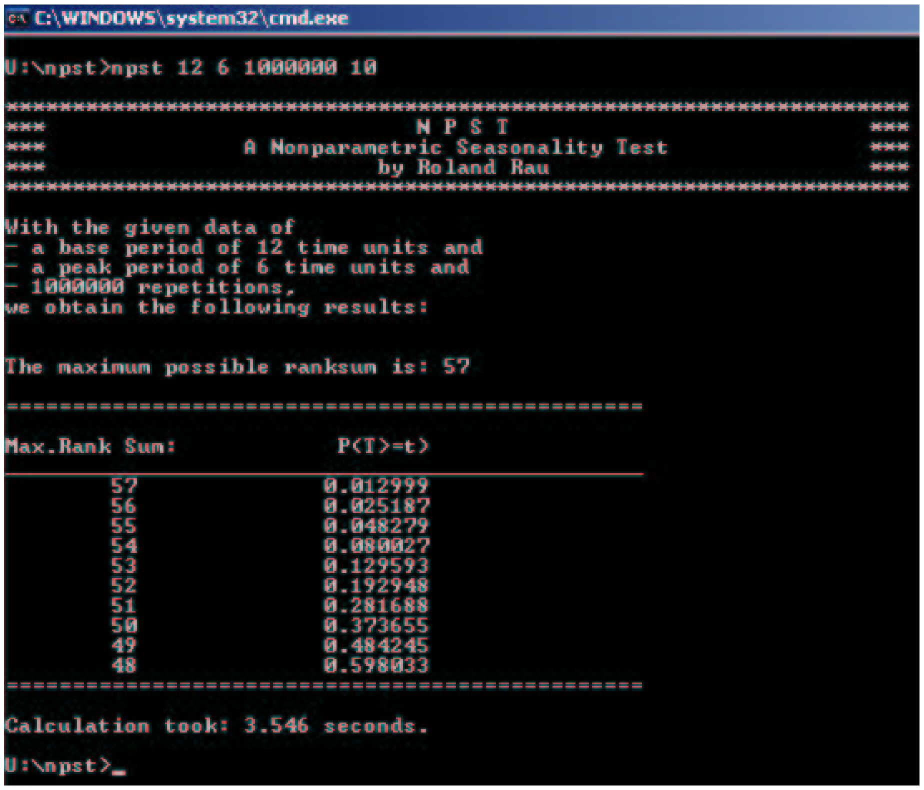
Figure 1: Theoretical Value vs. Empirical Values for an “NPST-Run” with a Basic Duration of 12 units, a Peak Period of 6 units and 10 Mio. repetitions



6 Summary

This paper described a nonparametric test for seasonality, called NPST, which generalizes the approaches of Hewitt et al. (1971) and Rogerson (1996). Whereas both previous publications used ranking information during a period of twelve months to detect seasonality with a duration of three, four or five months (Rogerson, 1996) or six months (Hewitt et al., 1971), the present does not give any restrictions — neither on the basic duration nor on the peak period. The algorithm can handle any situation regardless of the length of main duration (weeks, bi-weekly units, days) and the length of peak period of interest. Using Monte Carlo resampling techniques, the empiricial distribution of the test statistic T is estimated. This algorithm has been implemented in various computer languages. The implementation

Figure 2: A Demonstration of the Ocaml-Implementation of NPST with a Basic Duration of 12 units, a Peak Period of 6 units, 1 Mio. repetitions and 10 lines as output



```
C:\WINDOWS\system32\cmd.exe

U:\npst>npst 12 6 1000000 10

**************************************************************************
***                               N P S T                             ***
***                  A Nonparametric Seasonality Test                 ***
***                            by Roland Rau                          ***
**************************************************************************

With the given data of
- a base period of 12 time units and
- a peak period of 6 time units and
- 1000000 repetitions,
we obtain the following results:

The maximum possible ranksum is: 57

================================================

Max.Rank Sum:             P(T>=t)
________________________________________________
        57               0.012999
        56               0.025187
        55               0.048279
        54               0.080027
        53               0.129593
        52               0.192948
        51               0.281688
        50               0.373655
        49               0.484245
        48               0.598033
================================================

Calculation took: 3.546 seconds.

U:\npst>_
```

in Ocaml is the preferred platform for distribution and further modifications due to the speed of execution, the ease of development and the automatic memory management. Generating one million Monte-Carlo samples for a framework of twelve months as the basic duration and a peak period of six months took less than four seconds on a standard PC. For any precision in practical applications, the empirical results were equivalent to the theoretical values if at least one million samples were generated. The software can be obtained from the author via email (`mailto:Rau@demogr.mpg.de`) in source format as well as pre-compiled binaries for Windows XP, and GNU/Linux on i586 and AMD64.

References

Akslen, L. A. and F. Hartveit (1988). Seasonal Variation in Melanoma Deaths and the Pattern of Disease Process. A Preliminary Analysis. *Chronobiologia 15*, 257–263.

Edwards, J. H. (1961). The recognition and estimation of cyclic trends. *Annals of Human Genetics 25*, 83–86.

Eurowinter Group (1997). Cold exposure and winter mortality from ischaemic heart disease, cerebrovascular disease, respiratory disease, and all causes in warm and cold regions of Europe. *Lancet 349*, 1341–1346.

Eurowinter Group (2000). Winter mortality in relation to climate. *International Journal of Circumpolar Health 59*, 154–159.

Freedman, L. (1979). The use of a Kolmogorov-Smirnov type statistic in testing hypotheses about seasonal variation. *Journal of Epidemiology and Community Health 33*, 223–228.

Hewitt, D., J. Milner, A. Csima, and A. Pakula (1971). On Edwards' criterion of seasonality and a non-parametric alternative. *British Journal of Preventive Social Medicine 25*, 174–176.

Knuth, D. E. (1998). *Fundamental Algorithms* (Third ed.), Volume 1 of *The Art of Computer Programming.* Reading, Massachusetts: Addison-Wesley.

Kunst, A., C. Looman, and J. Mackenbach (1990). The decline in winter excess mortality in the Netherlands. *International Journal of Epidemiology 20*, 971–977.

Mackenbach, J., A. Kunst, and C. Looman (1992). Seasonal variation in mortality in the Netherlands. *Journal of Epidemiology and Community Health 46*, 261–265.

Marrero, O. (1983). The performance of several statistical tests for seasonality in monthly data. *Journal of Computational Statistics and Simulation 17*, 275–296.

Ogle, W. (1892). An Inquiry into the trustworthiness of the Old Bills of Mortality. *Journal of the Royal Statistical Society 55*, 437–460.

Rau, R. (2005). *Seasonality in Human Mortality. A Demographic Approach.* Ph. D. thesis, Universität Rostock, Rostock, Germany.

Reijneveld, S. A. (1990). The choice of a statistic for testing hypotheses regarding seasonality. *American Journal of Physical Anthropology 83*, 181–184.

Rogerson, P. A. (1996). A Generalization of Hewitt's Test for Seasonality. *International Journal of Epidemiology 25*, 644–648.

Wallenstein, S., C. R. Weinberg, and M. Gould (1989). Testing for a pulse in seasonal event data. *Biometrics 45*, 817–830.

Walter, S. (1980). Exact significance levels for Hewitt's test for seasonality. *Journal of Epidemiology and Community Health 34*, 147–149.

Yen, C.-J., C.-L. Chao, F.-C. Sung, W.-J. Chen, C.-S. Liau, and Y.-T. Lee (2000). Seasonal effects on cardiovascular mortality in older patients. *Age & Ageing 29*, 186–187.

A Appendix

Internet Adresses of the Implementation Languages

Language	URL
• R	`http://www.r-project.org/`
• S-Plus	`http://www.insightful.com/`
• Python	`http://www.python.org/`
• Lisp-Stat	`http://www.stat.uiowa.edu/~luke/xls/xlsinfo/xlsinfo.html`
• Corman-Lisp	`http://www.cormanlisp.com/index.html`
• Steel Bank Common Lisp (SBCL)	`http://sbcl.sourceforge.net/`
• GNU Common Lisp (GCL)	`http://www.gnu.org/software/gcl/gcl.html`
• Allegro Common Lisp	`http://www.franz.com/`
• Clisp	`http://clisp.cons.org/`
• Xanalys Lispworks Personal Edition	`http://www.lispworks.com/`
• GNU Emacs	`http://www.gnu.org/software/emacs/emacs.html`
• XEmacs	`http://www.xemacs.org`
• Ocaml	`http://caml.inria.fr/ocaml/`
• GCC	`http://gcc.gnu.org/`

Implementing the Benchmark Case

Benchmarks are notoriously problematic. CPU load, memory usage, and the number of running processes can hardly be controlled for to be exactly the same for each test. Also in our case it is possible that some values are biased due to the changing environment. Surprisingly, however, the timings were relatively constant even for two completely different frameworks (Operating System: Windows 32bit vs. GNU/Linux 64bit; Architecture: Intel Pentium 4, 2.4GHz vs. AMD64 3000+; RAM: 512MB vs. 1GB). Nevertheless, one should be careful to consider one language to be faster than another one since the speed of execution depends heavily on the programmer's proficiency in the various languages. In addition, one could gain further speed improvements by using compiler optimization options and/or by giving optional type information.