



OCEAN OBSERVATORIES INITIATIVE

DATA PRODUCT SPECIFICATION FOR SPIKE TEST

Version 1-01

Document Control Number 1341-10006

2012-05-23

Consortium for Ocean Leadership
1201 New York Ave NW, 4th Floor, Washington DC 20005
www.OceanLeadership.org

in Cooperation with

University of California, San Diego
University of Washington
Woods Hole Oceanographic Institution
Oregon State University
Scripps Institution of Oceanography
Rutgers University

Document Control Sheet

Version	Date	Description	Author
0-01	2011-12-15	Initial draft	M. Lankhorst
0-02	2012-02-10	Updated to match DPS Outline.	S. Webster
0-03	2012-03-28	Added info about appropriate metadata to Output and info about irregularly spaced data to Known Limitations.	S. Webster
0-04	2012-04-16	Inserted comments from focused review. Fixed formatting error in example data table.	M. Lankhorst
0-05	2012-04-20	Updated exemplar QC look-up table and moved to Test Data section 4.6.	S. Webster
1-00	2012-05-22	Initial Release	E. Chapman
1-01	2012-05-23	Formatting, copy edits	E. Griffin

Signature Page

This document has been reviewed and approved for release to Configuration Management.

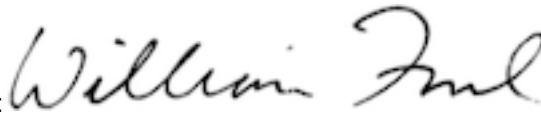
OOI Chief Systems Engineer: _____



Date: 2012-05-22

This document has been reviewed and meets the needs of the OOI Cyberinfrastructure for the purpose of coding and implementation.

OOI CI Signing Authority: _____



Date: 2012-05-23

Table of Contents

1	Abstract.....	1
2	Introduction.....	1
2.1	Author Contact Information.....	1
2.2	Metadata Information.....	1
2.3	Instruments.....	1
2.4	Literature and Reference Documents.....	2
2.5	Terminology.....	2
3	Theory.....	2
3.1	Description.....	2
3.2	Mathematical Theory.....	2
3.3	Known Theoretical Limitations.....	2
3.4	Revision History.....	2
4	Implementation.....	3
4.1	Overview.....	3
4.2	Inputs.....	3
4.3	Processing Flow.....	3
4.4	Outputs.....	3
4.5	Computational and Numerical Considerations.....	3
4.6	Code Verification and Test Data Sets.....	3
Appendix A	Example Code.....	1

1 Abstract

This document describes the OOI Spike Test (SPKETST) quality control algorithm used on various OOI data products. This algorithm generates flags for data values according to whether a single data value deviates significantly from surrounding data values. The purpose of this document is to serve as a reference in order to document which processing steps have been applied to a data product.

2 Introduction

2.1 Author Contact Information

Please contact Matthias Lankhorst (mlankhorst@ucsd.edu) or the Data Product Specification lead (DPS@lists.oceanobservatories.org) for more information concerning the algorithm and other items in this document.

2.2 Metadata Information

2.2.1 Data Product Name

n/a

2.2.2 Data Product Abstract (for Metadata)

n/a

2.2.3 Computation Name

The name for this quality control algorithm is

- Spike Test (SPKETST)

2.2.4 Computation Abstract (for Metadata)

The OOI Spike Test quality control algorithm generates a flag for individual data values that deviate significantly from surrounding data values.

2.2.5 Instrument-Specific Metadata

n/a

2.2.6 Synonyms

n/a

2.2.7 Similar Computations

A “gradient test” generates similar output, based on whether the difference between two successive data values exceeds a certain threshold. It can detect failure modes that consist of multiple successive “bad” data points, as long as the system switches from “good” to “bad” rapidly enough to trigger the threshold. In contrast, the spike test only detects individual outliers and assumes that all other surrounding values are “good.”

2.3 Instruments

n/a

2.4 Literature and Reference Documents

DCN 1342-00xxx

Instrument-specific Processing Flow Documents contain flow diagrams detailing all of the specific algorithms (product, calibration, QC) necessary to compute all data products from the instrument at all levels of QC and the order that the algorithms must be applied

2.5 Terminology

2.5.1 Definitions

n/a

2.5.2 Acronyms, Abbreviations and Notations

General OOI acronyms, abbreviations and notations are contained in the Level 2 Reference Module in the OOI requirements database (DOORS). There are no other acronyms, abbreviations, or notations for this document.

2.5.3 Variables and Symbols

See Section 4.2 and 4.4 for variable definitions.

3 Theory

3.1 Description

This is a data quality control algorithm testing a time series for spikes. It returns 1 for presumably good data and 0 for data presumed bad.

3.2 Mathematical Theory

The time series is divided into windows of length L (an odd integer number). Then, window by window, each value is compared to its $(L-1)$ neighboring values: a range R of these $(L-1)$ values is computed (max. minus min.), and replaced with the measurement accuracy ACC if $ACC > R$. A value is presumed to be good, i.e. no spike, if it deviates from the mean of the $(L-1)$ peers by less than a specified multiple of the range, $N * \max(R, ACC)$.

Further than $(L-1)/2$ values from the start or end points, the peer values are symmetrically before and after the test value. Within that range of the start and end, the peers are the first/last L values (without the test value itself).

The purpose of ACC is to restrict spike detection to deviations exceeding a minimum threshold value ($N * ACC$) even if the data have little variability. Use $ACC=0$ to disable this behavior. See example source code.

3.3 Known Theoretical Limitations

- Can only detect single outliers, not multiple outliers.
- Implicitly assumes that successive measurements follow similar statistical distribution, and that spacing of measurements (temporal or spatial) is sufficiently homogeneous to not disturb these statistics. Specifically, the theory behind this algorithm assumes that the data are collected at constant time intervals. However, OOI will run this algorithm independent of whether this is actually the case, i.e. run over differently spaced data points if there are data gaps or changes in sampling scheme. The amount of uncertain output caused by this is expected to be negligible.

3.4 Revision History

n/a

4 Implementation

4.1 Overview

See example source code.

4.2 Inputs

<code>dat</code>	Input dataset, a real numeric vector.
<code>acc</code>	Accuracy of input measurement from SPKETST look-up table
<code>N</code> (optional, defaults to 5)	Range multiplier from SPKETST look-up table
<code>L</code> (optional, defaults to 5)	Window length from SPKETST look-up table

4.3 Processing Flow

Call MatLab function “`dataqc_stuckvaluetest`”:

```
x_out = dataqc_stuckvaluetest(dat, acc, N, L)
```

See code in Appendix A.

4.4 Outputs

`out`: Array of class logical, same size as `dat`. The convention is that “good” data are flagged “1”, and those with spikes as “0”.

The metadata that must be included with the output are the input parameters to this algorithm:

- `acc`, accuracy parameter used (from SPKETST look-up table)
- `N`, range multiplier used (from SPKETST look-up table)
- `L`, window length used (from SPKETST look-up table)

4.5 Computational and Numerical Considerations

n/a

4.6 Code Verification and Test Data Sets

The algorithm code will be verified using the test data set provided, which contains inputs and their associated correct outputs. CI will verify that the algorithm code is correct by checking that the algorithm pressure output, generated using the test data inputs, is identical to the test data output.

Table 1: Test Data Set

<code>dat</code>	<code>acc</code>	<code>N</code>	<code>L</code>	<code>out</code>
-4	0.1	5	5	1
3				1
40				0
-1				1
1				1
-6				1
-6				1
1				1

In addition to the output (`qcflag`), the metadata from the QC Lookup table must be included with the output.

Table 2: Example Spike Test Lookup Table

Instrument Class	Data Product	Data Product Description	Accuracy of input (acc)	Range multiplier (N)	Window length (L)
CTDMO	CONDWAT	Conductivity	0.0003 S/m	9	9
CTDMO	TEMPWAT	Temperature	0.002 °C	9	9

Note that this table is for example purposes only and some/all values may not be correct. The official QC lookup tables are kept separately from the DPS and, at the time of writing, do not exist in their final form.

Appendix A Example Code

This Appendix contains all Matlab subroutines necessary for performing the Spike Test quality control algorithm as described herein.

A.1 Spike Test Quality Control Algorithm

```
% DATAQC_SPIKETEST Data quality control algorithm testing a time
% series for spikes. Returns 1 for presumably
% good data and 0 for data presumed bad.
%
% Time-stamp: <2010-07-28 14:25:42 mlankhorst>
%
% METHODOLOGY: The time series is divided into windows of length L
% (an odd integer number). Then, window by window, each value is
% compared to its (L-1) neighboring values: a range R of these
% (L-1) values is computed (max. minus min.), and replaced with
% the measurement accuracy ACC if ACC>R. A value is presumed to
% be good, i.e. no spike, if it deviates from the mean of the
% (L-1) peers by less than a multiple of the range, N*max(R,ACC).
%
% Further than (L-1)/2 values from the start or end points, the
% peer values are symmetrically before and after the test
% value. Within that range of the start and end, the peers are
% the first/last L values (without the test value itself).
%
% The purpose of ACC is to restrict spike detection to deviations
% exceeding a minimum threshold value (N*ACC) even if the data
% have little variability. Use ACC=0 to disable this behavior.
%
% USAGE: out=dataqc_spiketest(dat,acc,N,L);
% OR: out=dataqc_spiketest(dat,acc);
%
% out: Boolean. 0 for detected spike, else 1.
% dat: Input dataset, a real numeric vector.
% acc: Accuracy of any input measurement.
% N (optional, defaults to 5): Range multiplier, cf. above
% L (optional, defaults to 5): Window length, cf. above
%
% EXAMPLE:
%
% >> x=[-4 3 40 -1 1 -6 -6 1];
% >> dataqc_spiketest(x,.1)
%
% ans =
%
% 1 1 0 1 1 1 1 1
%
function out=dataqc_spiketest(varargin);

error(nargchk(2,4,varargin,'struct'))

dat=varargin{1};
acc=varargin{2};
N=5;
L=5;

switch nargin
case 3,
if ~isempty(varargin{3})
```

```

        N=varargin{3};
    end
    case 4,
        if ~isempty(varargin{3})
            N=varargin{3};
        end
        if ~isempty(varargin{4})
            L=varargin{4};
        end
    end
end

if ~isnumeric(dat)
    error('DAT must be numeric.')
end
if ~isvector(dat)
    error('DAT must be a vector.')
end
if ~isreal(dat)
    error('DAT must be real.')
end

if ~isnumeric(acc)
    error('ACC must be numeric.')
end
if ~isscalar(acc)
    error('ACC must be scalar.')
end
if ~isreal(acc)
    error('ACC must be real.')
end

if ~isnumeric(N)
    error('N must be numeric.')
end
if ~isscalar(N)
    error('N must be scalar.')
end
if ~isreal(N)
    error('N must be real.')
end

if ~isnumeric(L)
    error('L must be numeric.')
end
if ~isscalar(L)
    error('L must be scalar.')
end
if ~isreal(L)
    error('L must be real.')
end

L=ceil(abs(L));
if (L/2)==round(L/2)
    L=L+1;
    warning('L was even; setting L:=L+1')
end
if L<3
    L=5;
    warning('L was too small; setting L:=5')
end

ll=length(dat);
out=zeros(size(dat));

```

```

L2=(L-1)/2;
i1=1+L2;
i2=11-L2;

if ll>=L

    for ii=i1:i2
        tmpdat=dat(ii+[-L2:-1 1:L2]);
        R=max(tmpdat)-min(tmpdat);
        R=max([R acc]);
        if (N*R)>abs(dat(ii)-mean(tmpdat))
            out(ii)=1;
        end
    end

    for ii=1:L2
        tmpdat=dat([1:ii-1 ii+1:L]);
        R=max(tmpdat)-min(tmpdat);
        R=max([R acc]);
        if (N*R)>abs(dat(ii)-mean(tmpdat))
            out(ii)=1;
        end
    end

    for ii=11-L2+1:11
        tmpdat=dat([11-L+1:ii-1 ii+1:11]);
        R=max(tmpdat)-min(tmpdat);
        R=max([R acc]);
        if (N*R)>abs(dat(ii)-mean(tmpdat))
            out(ii)=1;
        end
    end

else
    warning('L was greater than length of DAT, returning zeros.')
end

```

A.2 isnumeric – Determine whether input is numeric array

Syntax

```
tf = isnumeric(A)
```

Description

`tf = isnumeric(A)` returns logical 1 (`true`) if `A` is a numeric array and logical 0 (`false`) otherwise. For example, sparse arrays and double-precision arrays are numeric, while strings, cell arrays, and structure arrays and logicals are not.

Examples

Given the following cell array,

```
C{1,1} = pi; % double
C{1,2} = 'John Doe'; % char array
C{1,3} = 2 + 4i; % complex double
C{1,4} = ispc; % logical
C{1,5} = magic(3) % double array

C =
    [3.1416] 'John Doe' [2.0000+ 4.0000i] [1][3x3 double]
isnumeric shows that all but C{1,2} and C{1,4} are numeric arrays.
for k = 1:5
x(k) = isnumeric(C{1,k});
end

x
x =
     1     0     1     0     1
```

A.3 isvector – Determine whether input is vector

Syntax

```
isvector(A)
```

Description

`isvector(A)` returns logical 1 (`true`) if `size(A)` returns `[1 n]` or `[n 1]` with a nonnegative integer value `n`, and logical 0 (`false`) otherwise.

Examples

Test matrix `A` and its row and column vectors:

```
A = rand(5);

isvector(A)
ans =
     0

isvector(A(3, :))
ans =
     1

isvector(A(:, 2))
ans =
     1
```

A.4 isscalar – Determine whether input is scalar

Syntax

```
isscalar(A)
```

Description

`isscalar(A)` returns logical 1 (`true`) if `size(A)` returns `[1 1]`, and logical 0 (`false`) otherwise.

Examples

Test matrix `A` and one element of the matrix:

```
A = rand(5);
```

```
isscalar(A)
```

```
ans =
```

```
0
```

```
isscalar(A(3,2))
```

```
ans =
```

```
1
```

A.5 isreal – Check if input is real array

Syntax

```
TF = isreal(A)
```

Description

`TF = isreal(A)` returns logical 1 (`true`) if `A` does not have an imaginary part. It returns logical 0 (`false`) otherwise. If `A` has a stored imaginary part of value 0, `isreal(A)` returns logical 0 (`false`).

Note For logical and char data classes, `isreal` always returns `true`. For numeric data types, if `A` does not have an imaginary part `isreal` returns `true`; if `A` does have an imaginary part `isreal` returns `false`. For cell, struct, function_handle, and object data types, `isreal` always returns `false`.

`~isreal(x)` returns `true` for arrays that have at least one element with an imaginary component. The value of that component can be 0.

Tips

If `A` is real, `complex(A)` returns a complex number whose imaginary component is 0, and `isreal(complex(A))` returns `false`. In contrast, the addition `A + 0i` returns the real value `A`, and `isreal(A + 0i)` returns `true`.

If `B` is real and `A = complex(B)`, then `A` is a complex matrix and `isreal(A)` returns `false`, while `A(m:n)` returns a real matrix and `isreal(A(m:n))` returns `true`.

Because MATLAB software supports complex arithmetic, certain of its functions can introduce significant imaginary components during the course of calculations that appear to be limited to real numbers. Thus, you should use `isreal` with discretion.

Example 1

If a computation results in a zero-value imaginary component, `isreal` returns `true`.

```
x=3+4i;
y=5-4i;
isreal(x+y)
```

```
ans =
```

```
1
```

Example 2

These examples use `isreal` to detect the presence or absence of imaginary numbers in an array. Let

```
x = magic(3);
y = complex(x);
isreal(x) returns true because no element of x has an imaginary component.
```

```
isreal(x)
```

```
ans =
```

```
1
```

`isreal(y)` returns `false`, because every element of `x` has an imaginary component, even though the value of the imaginary components is 0.

```
isreal(y)
```

```
ans =
```

```
0
```

This expression detects strictly real arrays, i.e., elements with 0-valued imaginary components are treated as real.

```
~any(imag(y(:)))
```

```
ans =
```

```
1
```

Example 3

Given the following cell array,

```
C{1} = pi; % double
C{2} = 'John Doe'; % char array
C{3} = 2 + 4i; % complex double
C{4} = ispc; % logical
C{5} = magic(3); % double array
C{6} = complex(5,0) % complex double
```

```
C =
```

```
[3.1416] 'John Doe' [2.0000+ 4.0000i] [1] [3x3 double] [5]
```

`isreal` shows that all but `C{1,3}` and `C{1,6}` are real arrays.

```
for k = 1:6
```

```
x(k) = isreal(C{k});
```

```
end
```

```
x
```

```
x =
```

```
1 1 0 1 1 0
```

A.6 isempty – Test if array is empty

Syntax

```
tf = isempty(A)
```

Description

`tf = isempty(A)` returns logical true (1) if A is an empty array and logical false (0) otherwise. An empty array has at least one dimension of size zero, for example, 0-by-0 or 0-by-5.

Examples

```
B = rand(2,2,2);  
B(:, :, :) = [];
```

```
isempty(B)
```

```
ans =  
     1
```