



DATA PRODUCT SPECIFICATION FOR TREND TEST

Version 1-01
Document Control Number 1341-10007
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	2011-12-21	Initial draft	M. Lankhorst, S.H. Nam
0-02	2012-03-02	Modified to match DPS Outline.	S. Webster
0-03	2012-03-28	Added info about appropriate metadata to Output.	S. Webster
0-04	2012-04-16	Incorporated comments from focused review	M. Lankhorst
0-05	2012-05-16	Updated exemplar QC lookup 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.

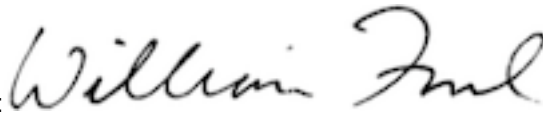
OOI Chief Systems Engineer: _____



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: 2012-05-22

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..... 1

 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 2

 4.1 Overview..... 2

 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 Trend Test (TRNDTST) quality control algorithm is used to test time series for whether the data contain a significant portion of a polynomial. The purpose of this test is to check if a significant fraction of the variability in a time series can be explained by a drift, possibly interpreted as a sensor drift. This drift is assumed to be a polynomial of specified order, e.g., 1 for linear drift.

2 Introduction

2.1 Author Contact Information

Please contact Matthias Lankhorst (mlankhorst@ucsd.edu) or the Algorithms Group (ATBD@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

- Trend Test (TRNDTST)

2.2.4 Computation Abstract (for Metadata)

The OOI Trend Test quality control algorithm tests time series for whether the data contain a significant portion of a polynomial. The algorithm is used to check if a significant fraction of the variability in a time series can be explained by a drift, where the drift is assumed to be a polynomial of specified order.

2.2.5 Instrument-Specific Metadata

n/a

2.2.6 Synonyms

n/a

2.2.7 Similar Computations

n/a

2.3 Instruments

n/a

2.4 Literature and Reference Documents

DCN 1342-000xx

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

2.5 Terminology

2.5.1 Definitions

Trend General course or prevailing tendency which time series data may or may not have

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

The following variables and symbols are defined here for use throughout this document.

$x_{in}(t)$	input time series vector
$x_{out}(t)$	output Boolean vector
$P(t)$	polynomial fitted to the input time series in a least squares sense
t	time
n	order of polynomial
$nstd$	threshold value; if the standard deviation of a time series is reduced by more than this factor after a polynomial (e.g. linear trend) is subtracted from the values, the time series will be interpreted as having a trend

3 Theory

3.1 Description

A polynomial of specified order is fitted to the time series. If the standard deviation of the difference between the original and fitted data is less than that of original data by a specified threshold factor, the data are considered as containing a significant trend, else not.

3.2 Mathematical Theory

$$P = p_1 * X^n + p_2 * X^{n-1} + \dots + p_n * X + p_{n+1}$$

If $nstd * Std. Dev. (P(t) - X(t)) < Std. Dev. (X(t))$ then $X(t), X(t+1), \dots, X(t+N)$ are taken as containing a significant trend.

3.3 Known Theoretical Limitations

n/a

3.4 Revision History

n/a

4 Implementation

4.1 Overview

This code returns "0" for trend detected and "1" for not. The code uses MatLab functions "isnumeric", "isvector", "isscalar", "isempty", and "isreal" to check the inputs prior to applying the algorithm. It uses the MatLab functions "polyfit" to find the coefficients of a polynomial P of degree n that fits the data, then uses the MatLab function "polyval" to evaluate the polynomial P at the data points t . If the standard deviation of a time series is reduced by more than the factor

nstd after the result of polyfit is subtracted from the values, the time series will be interpreted as having a trend and flagged accordingly.

4.2 Inputs

- x_in(t)* input vector, a numeric real vector.
- t* time associated with input vector
- n* polynomial order from TRNDTST look-up table (optional, defaults to 1)
- nstd* threshold value from TRNDTST look-up table; If the standard deviation of a time series is reduced by more than this factor after a polynomial (e.g. linear trend) is subtracted from the values, the time series will be interpreted as having a trend (optional, defaults to 3)

Inputs *x_in* and *t* must be aggregated over a measurement interval of the duration indicated in the lookup table, i.e. given as vectors of a certain length.

4.3 Processing Flow

Call MatLab function “dataqc_polytrendtest”:

```
x_out = dataqc_polytrendtest(x_in,n,nstd)
```

See code in Appendix A.

4.4 Outputs

x_out Boolean output: 0 where a trend is detected, 1 elsewhere.

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

- *n*, polynomial value used (from TRNDTST look-up table)
- *nstd*, threshold value used (from TRNDTST look-up table)
- start and end dates of the measurement interval over which the test was run

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.

n = 1
nstd = 3

x_in											x_out
t											[t_min t_max]
0.8147	0.9058	0.1270	0.9134	0.6324	0.0975	0.2785	0.5469	0.9575	0.9649		1
1	2	3	4	5	6	7	8	9	10		[1 10]
0.6557	0.2357	1.2491	1.5340	1.4787	1.7577	1.9431	1.7922	2.2555	1.9712		1
1	2	3	4	5	6	7	8	9	10		[1 10]
0.7060	0.5318	1.2769	1.5462	2.0971	3.3235	3.6948	3.8171	4.9502	4.5344		0
1	2	3	4	5	6	7	8	9	10		[1 10]
0	0	0	0	0	0	0	0	0	0		1
1	2	3	4	5	6	7	8	9	10		[1 10]
1	1	1	1	1	1	1	1	1	1		1
1	2	3	4	5	6	7	8	9	10		[1 10]
0.4387	-0.1184	-0.2345	-0.7048	-1.8131	-2.0102	-2.5544	-2.8537	-3.2906	-3.7453		0
1	2	3	4	5	6	7	8	9	10		[1 10]

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

Table 2: Example Trent Test Lookup Table

Instrument Class	Data Product	Data Product Description	Time interval length in days	Polynomial order (n)	Standard deviation reduction factor (nstd)
HPIES	IESPRES	Bottom pressure	90	1	5

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 the code for the Matlab function `dataqc_polytrendtest.mat`, as well as descriptions of the standard Matlab functions used within this function (according to the R2012a Documentation from Matlab), necessary for performing the Spike Test quality control algorithm as described herein.

A.1 Polynomial Trend Test Algorithm

Note: This exemplar algorithm does not utilize the time axis, but instead assumes that the input data are equally spaced in time. The actual implementation should incorporate the time axis for cases where input data is sampled at non-equal time steps.

```
% DATAQC_POLYTRENDTEST  Data quality control algorithm testing
%   if measurements contain a significant portion of a polynomial.
%   Returns 1 if this is not the case, else 0.
%
% Time-stamp: <2010-10-29 13:56:46 mlankhorst>
%
% RATIONALE: The purpose of this test is to check if a significant
%   fraction of the variability in a time series can be explained
%   by a drift, possibly interpreted as a sensor drift. This drift
%   is assumed to be a polynomial of order ORD. Use ORD=1 to
%   consider a linear drift
%
% METHODOLOGY: The time series DAT is passed to MatLab's POLYFIT
%   routine to obtain a polynomial fit PP to DAT, and the
%   difference DAT-PP is compared to the original DAT. If the
%   standard deviation of (DAT-PP) is less than that of DAT by a
%   factor of NSTD, the time series is assumed to contain a
%   significant trend (output will be 0), else not (output will be
%   1).
%
% USAGE:  OUT=dataqc_polytrendtest(DAT,ORD,NSTD);
%
%   OUT: Boolean scalar, 0 if trend is detected, 1 if not.
%
%   DAT: Input dataset, a numeric real vector.
%   ORD (optional, defaults to 1): Polynomial order.
%   NSTD (optional, defaults to 3): Factor by how much the
%   standard deviation must be reduced before OUT
%   switches from 1 to 0
%
```

```
function out=dataqc_polytrendtest(varargin);
```

```
    error(nargchk(1,3,nargin,'struct'))
```

```
    dat=varargin{1};
```

```
    if ~isnumeric(dat)
        error('DAT must be numeric.')
    end
```

```
    if ~isvector(dat)
        error('DAT must be vector.')
    end
```

```
    if ~isreal(dat)
```

```
    error('DAT must be real.')
end

ord=1;
nstd=3;

if nargin==2
    if ~isempty(varargin{2})
        ord=varargin{2};
    end
end
if nargin==3
    if ~isempty(varargin{2})
        ord=varargin{2};
    end
    if ~isempty(varargin{3})
        nstd=varargin{3};
    end
end

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

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

ord=round(abs(ord));
nstd=abs(nstd);

ll=length(dat);
x=[1:ll];

pp=polyfit(x,dat,ord);
datpp=polyval(pp,x);

if (nstd*std(dat-datpp))<std(dat)
    out=0;
else
    out=1;
end
```

A.2 isnumeric – Determine whether input is numeric array (from polyfun toolbox)

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 (from polyfun toolbox)

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
```

A.7 polyfit – Polynomial curve fitting

Syntax

```
p = polyfit(x, y, n)
[p, S] = polyfit(x, y, n)
[p, S, mu] = polyfit(x, y, n)
```

Description

`p = polyfit(x, y, n)` finds the coefficients of a polynomial $p(x)$ of degree n that fits the data, $p(x(i))$ to $y(i)$, in a least squares sense. The result `p` is a row vector of length $n+1$ containing the polynomial coefficients in descending powers:

$$p(x) = p_1x^n + p_2x^{n-1} + \dots + p_nx + p_{n+1}.$$

`[p, S] = polyfit(x, y, n)` returns the polynomial coefficients `p` and a structure `S` for use with `polyval` to obtain error estimates or predictions. Structure `S` contains fields `R`, `df`, and `normr`, for the triangular factor from a QR decomposition of the Vandermonde matrix of `x`, the degrees of freedom, and the norm of the residuals, respectively. If the data `y` are random, an estimate of the covariance matrix of `p` is $(R_{inv} * R_{inv}') * normr^2 / df$, where `Rinv` is the inverse of `R`. If the errors in the data `y` are independent normal with constant variance, `polyval` produces error bounds that contain at least 50% of the predictions.

`[p, S, mu] = polyfit(x, y, n)` finds the coefficients of a polynomial in

$$\hat{x} = \frac{x - \mu_1}{\mu_2}$$

where $\mu_1 = \text{mean}(x)$ and $\mu_2 = \text{std}(x)$. `mu` is the two-element vector $[\mu_1, \mu_2]$. This centering and scaling transformation improves the numerical properties of both the polynomial and the fitting algorithm.

Examples

This example involves fitting the error function, `erf(x)`, by a polynomial in `x`. This is a risky project because `erf(x)` is a bounded function, while polynomials are unbounded, so the fit might not be very good.

First generate a vector of `x` points, equally spaced in the interval `[0, 2.5]`; then evaluate `erf(x)` at those points.

```
x = (0: 0.1: 2.5)';
y = erf(x);
```

The coefficients in the approximating polynomial of degree 6 are

```
p = polyfit(x, y, 6)
```

```
p =
```

```
0.0084  -0.0983   0.4217  -0.7435   0.1471   1.1064   0.0004
```

A.8 polyval – Polynomial evaluation

Syntax

```
y = polyval(p,x)
[y,delta] = polyval(p,x,S)
y = polyval(p,x,[],mu)
[y,delta] = polyval(p,x,S,mu)
```

Description

`y = polyval(p,x)` returns the value of a polynomial of degree n evaluated at x . The input argument p is a vector of length $n+1$ whose elements are the coefficients in descending powers of the polynomial to be evaluated.

$$y = p_1x^n + p_2x^{n-1} + \dots + p_nx + p_{n+1}$$

x can be a matrix or a vector. In either case, `polyval` evaluates p at each element of x .

`[y,delta] = polyval(p,x,S)` uses the optional output structure S generated by `polyfit` to generate error estimates δ . δ is an estimate of the standard deviation of the error in predicting a future observation at x by $p(x)$. If the coefficients in p are least squares estimates computed by `polyfit`, and the errors in the data input to `polyfit` are independent, normal, and have constant variance, then $y \pm \delta$ contains at least 50% of the predictions of future observations at x .

`y = polyval(p,x,[],mu)` or `[y,delta] = polyval(p,x,S,mu)` use $\hat{x} = (x - \mu_1) / \mu_2$ in place of x . In this equation, $\mu_1 = \text{mean}(x)$ and $\mu_2 = \text{std}(x)$. The centering and scaling parameters $\text{mu} = [\mu_1, \mu_2]$ are optional output computed by `polyfit`.

Tips

The `polyvalm(p,x)` function, with x a matrix, evaluates the polynomial in a matrix sense. See `polyvalm` for more information.

Examples

The polynomial $p(x) = 3x^2 + 2x + 1$ is evaluated at $x = 5, 7,$ and 9 with

```
p = [3 2 1];
polyval(p,[5 7 9])
which results in
ans =
```

```
86    162    262
```