Data Preprocessing using MATLAB

Instructor: Dr.Eng.Rahmat Widyanto

Preprocessing Data

Data cleaning, smoothing, grouping

Data sets can require preprocessing techniques to ensure accurate, efficient, or meaningful analysis. Data cleaning refers to methods for finding, removing, and replacing bad or missing data. Smoothing and detrending are processes for removing noise and linear trends from data. Grouping and binning methods are techniques that identify relationships among the data variables.

Functions

Missing Data and Outliers

`ismissing`	Find missing values
`rmmissing`	Remove missing entries
`fillmissing`	Fill missing values
`missing`	Create missing values
`standardizeMissing`	Insert standard missing values
`isoutlier`	Find outliers in data
`filloutliers`	Detect and replace outliers in data

Smoothing and Detrending Data

`smoothdata`	Smooth noisy data
`movmean`	Moving mean
`movmedian`	Moving median
`detrend`	Remove linear trends
`filter`	1-D digital filter
`filter2`	2-D digital filter

Grouping and Binning Data

`discretize`	Group data into bins or categories
`histcounts`	Histogram bin counts
`histcounts2`	Bivariate histogram bin counts
`findgroups`	Find groups and return group numbers
`splitapply`	Split data into groups and apply function
`rowfun`	Apply function to table or timetable rows
`varfun`	Apply function to table or timetable variables
`accumarray`	Construct array with accumulation

ismissing,

Find missing values

Syntax

```
TF = ismissing(A)
TF = ismissing(A,indicator)
```

Description

`TF = ismissing(A)` returns a logical array that indicates which elements of an array or table contain missing values. The size of `TF` is the same as the size of `A`.

Standard missing values depend on the data type:

- `NaN` for `double`, `single`, `duration`, and `calendarDuration`
- `NaT` for `datetime`
- `<missing>` for `string`
- `<undefined>` for `categorical`
- `' '` for `char`
- `{''}` for `cell` of character vectors

Examples

NaN Values in Vector

Create a row vector A that contains NaN values, and identify their location in A.

```
A = [3 NaN 5 6 7 NaN NaN 9];
TF = ismissing(A)
```

```
TF = 1×8 logical array
   0   1   0   0   0   1   1   0
```

Missing Values in Table with Various Data Types

Create a table with variables of different data types and find the elements with missing values.

```
dblVar = [NaN;3;5;7;9;11;13];
singleVar = single([1;NaN;5;7;9;11;13]);
cellstrVar = {'one';'three';'';'seven';'nine';'eleven';'thirteen'};
charVar = ['A';'C';'E';' ';'I';'J';'L'];
categoryVar = categorical({'red';'yellow';'blue';'violet';'';'ultraviolet';'orange'});
dateVar = [datetime(2015,1:2:10,15) NaT datetime(2015,11,15)]';
stringVar = ["a";"b";"c";"d";"e";"f";missing];

A = table(dblVar,singleVar,cellstrVar,charVar,categoryVar,dateVar,stringVar)
```

```
A = 7×7 table
    dblVar    singleVar    cellstrVar     charVar    categoryVar      dateVar       stringVar
    ______    _________    __________     _______    ___________    ___________    _________

     NaN          1        'one'            A        red            15-Jan-2015    "a"
       3        NaN        'three'          C        yellow         15-Mar-2015    "b"
       5          5        ''               E        blue           15-May-2015    "c"
       7          7        'seven'                   violet         15-Jul-2015    "d"
       9          9        'nine'           I        <undefined>    15-Sep-2015    "e"
      11         11        'eleven'         J        ultraviolet    NaT            "f"
      13         13        'thirteen'       L        orange         15-Nov-2015    <missing>
```

`ismissing` returns 1 where the corresponding element in `A` has a missing value.

```
TF = ismissing(A)
```

```
TF = 7x7 logical array
   1   0   0   0   0   0   0
   0   1   0   0   0   0   0
   0   0   1   0   0   0   0
   0   0   0   1   0   0   0
   0   0   0   0   1   0   0
   0   0   0   0   0   1   0
   0   0   0   0   0   0   1
```

The size of `TF` is the same as the size of `A`.

Specify Indicators for Missing Values in Table

Create a table where `'NA'`, `''`, `-99`, `NaN`, and `Inf` represent missing values. Then, find the elements with missing values.

```
dblVar = [NaN;3;Inf;7;9];
int8Var = int8([1;3;5;7;-99]);
cellstrVar = {'one';'three';'';'NA';'nine'};
charVar = ['A';'C';'E';' ';'I'];

A = table(dblVar,int8Var,cellstrVar,charVar)
```

```
A = 5×4 table
    dblVar    int8Var    cellstrVar    charVar
    ______    _______    __________    _______

    NaN          1       'one'         A
      3          3       'three'       C
    Inf          5       ''            E
      7          7       'NA'
      9        -99       'nine'        I
```

smoothdata

Smooth noisy data

Syntax

```
B = smoothdata(A)
B = smoothdata(A,dim)
```

```
B = smoothdata(___,method)
B = smoothdata(___,method,window)
```

```
B = smoothdata(___,nanflag)
```

```
B = smoothdata(___,Name,Value)
```

```
[B,window] = smoothdata(___)
```

Description

`B = smoothdata(A)` returns a moving average of the elements of a vector using a fixed window length that is determined heuristically. The window slides down the length of the vector, computing an average over the elements within each window.

- If `A` is a matrix, then `smoothdata` computes the moving average down each column.
- If `A` is a multidimensional array, then `smoothdata` operates along the first dimension whose size does not equal 1.
- If `A` is a table or timetable with numeric variables, then `smoothdata` operates on each variable separately.

`B = smoothdata(A,dim)` operates along the dimension `dim` of `A`. For example, if `A` is a matrix, then `smoothdata(A,2)` smooths the data in each row of `A`.

`B = smoothdata( __ ,method)` specifies the smoothing method for either of the previous syntaxes. For example, `B = smoothdata(A,'sgolay')` uses a Savitzky-Golay filter to smooth the data in `A`.

`B = smoothdata( __ ,method,window)` specifies the length of the window used by the smoothing method. For example, `smoothdata(A,'movmedian',5)` smooths the data in `A` by taking the median over a five-element sliding window.

`B = smoothdata( __ ,nanflag)` specifies how `NaN` values are treated for any of the previous syntaxes. `'omitnan'` ignores `NaN` values and `'includenan'` includes them when computing within each window.

`B = smoothdata( __ ,Name,Value)` specifies additional parameters for smoothing using one or more name-value pair arguments. For example, if `t` is a vector of time values, then `smoothdata(A,'SamplePoints',t)` smooths the data in `A` relative to the times in `t`.

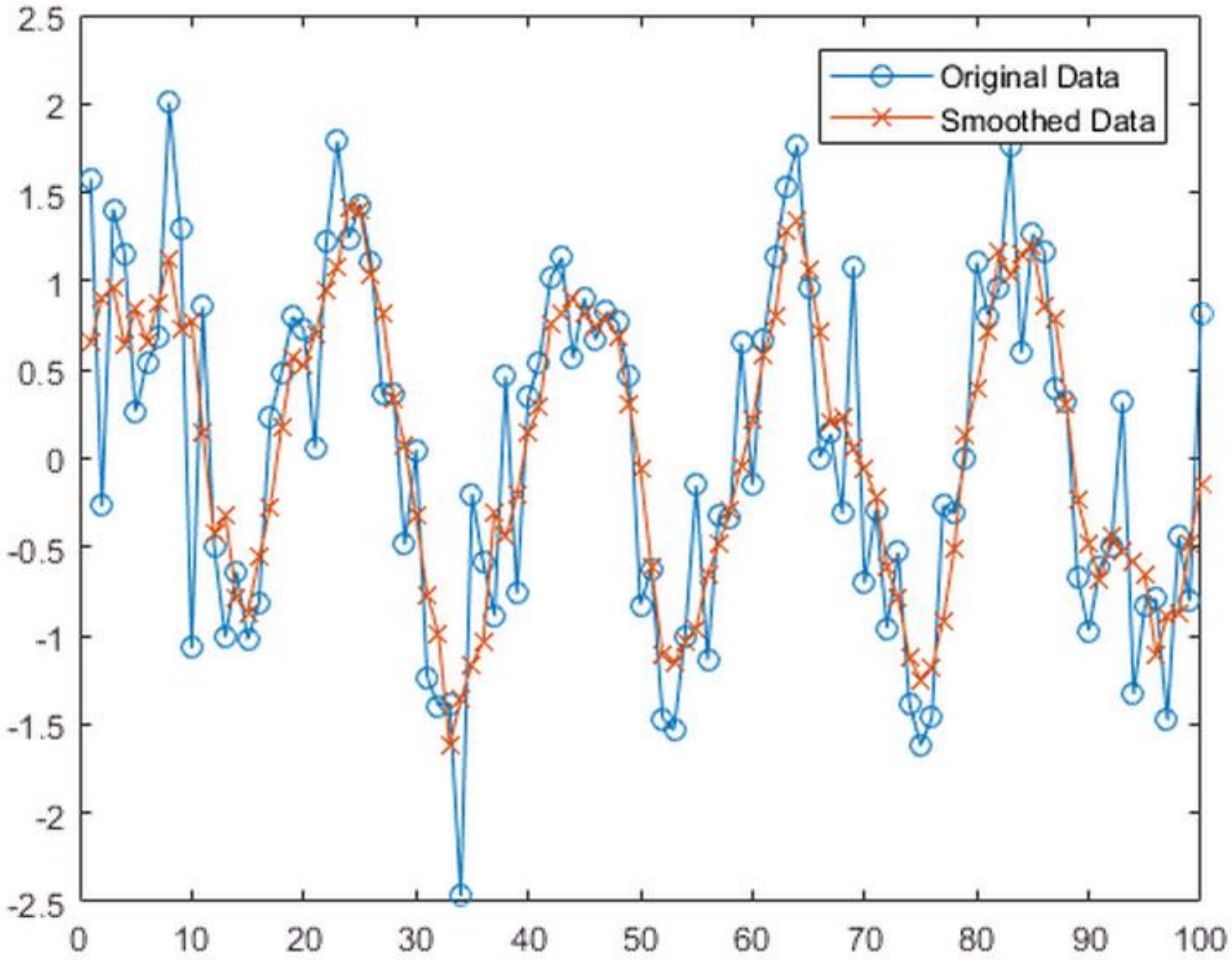
`[B,window] = smoothdata( __ )` also returns the moving window length.

Examples

Smooth Data with Moving Average

Create a vector containing noisy data, and smooth the data with a moving average. Plot the original and smoothed data.

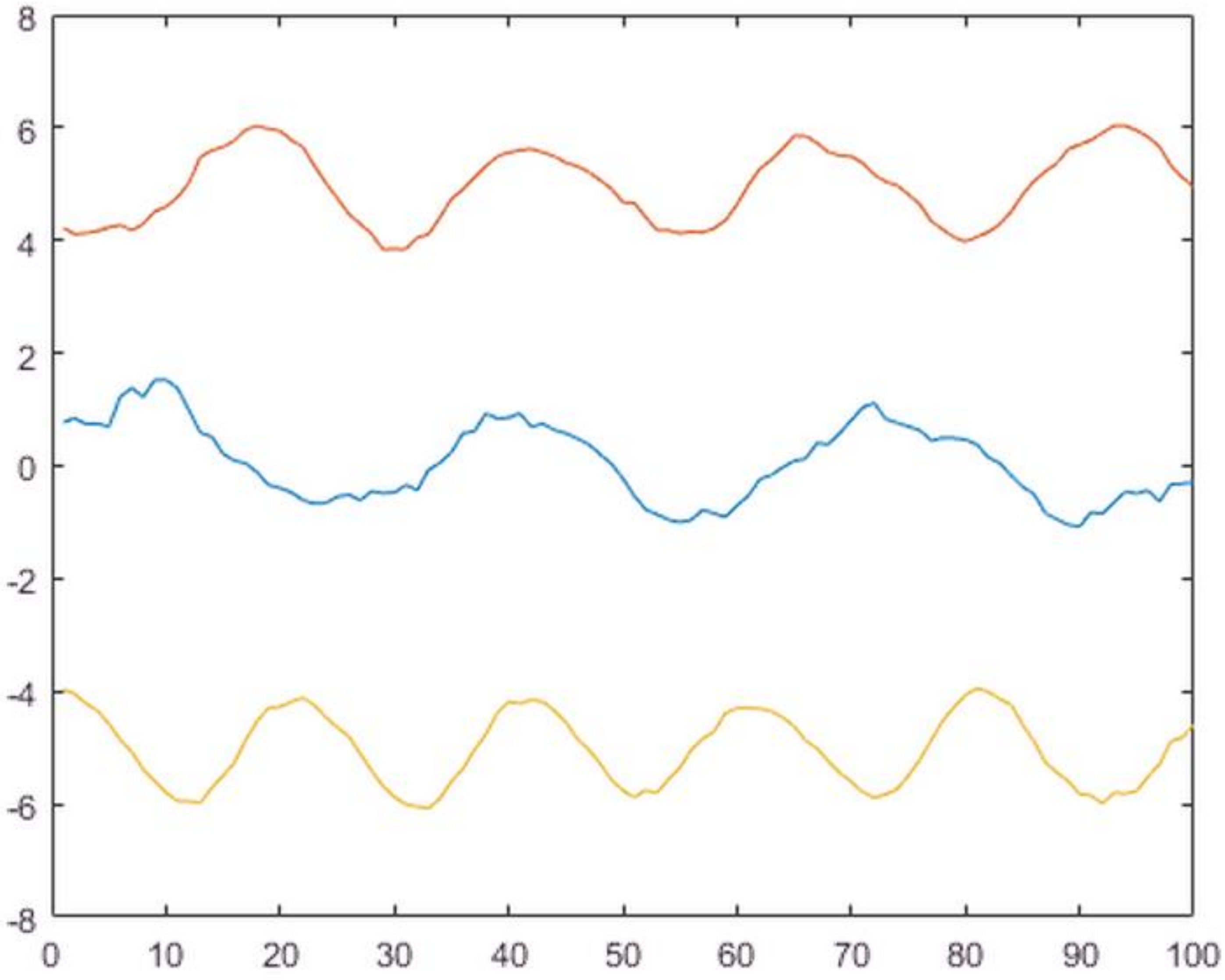
```
x = 1:100;
A = cos(2*pi*0.05*x+2*pi*rand) + 0.5*randn(1,100);
B = smoothdata(A);
plot(x,A,'-o',x,B,'-x')
legend('Original Data','Smoothed Data')
```


Original Data
Smoothed Data
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
10
20
30
40
50
60
70
80
90
100

Matrix of Noisy Data

Create a matrix whose rows represent three noisy signals. Smooth the three signals using a moving average, and plot the smoothed data.

```
x = 1:100;
s1 = cos(2*pi*0.03*x+2*pi*rand) + 0.5*randn(1,100);
s2 = cos(2*pi*0.04*x+2*pi*rand) + 0.4*randn(1,100) + 5;
s3 = cos(2*pi*0.05*x+2*pi*rand) + 0.3*randn(1,100) - 5;
A = [s1; s2; s3];
B = smoothdata(A,2);
plot(x,B(1,:),x,B(2,:),x,B(3,:))
```


8
6
4
2
0
-2
-4
-6
-8
0
10
20
30
40
50
60
70
80
90
100

Gaussian Filter

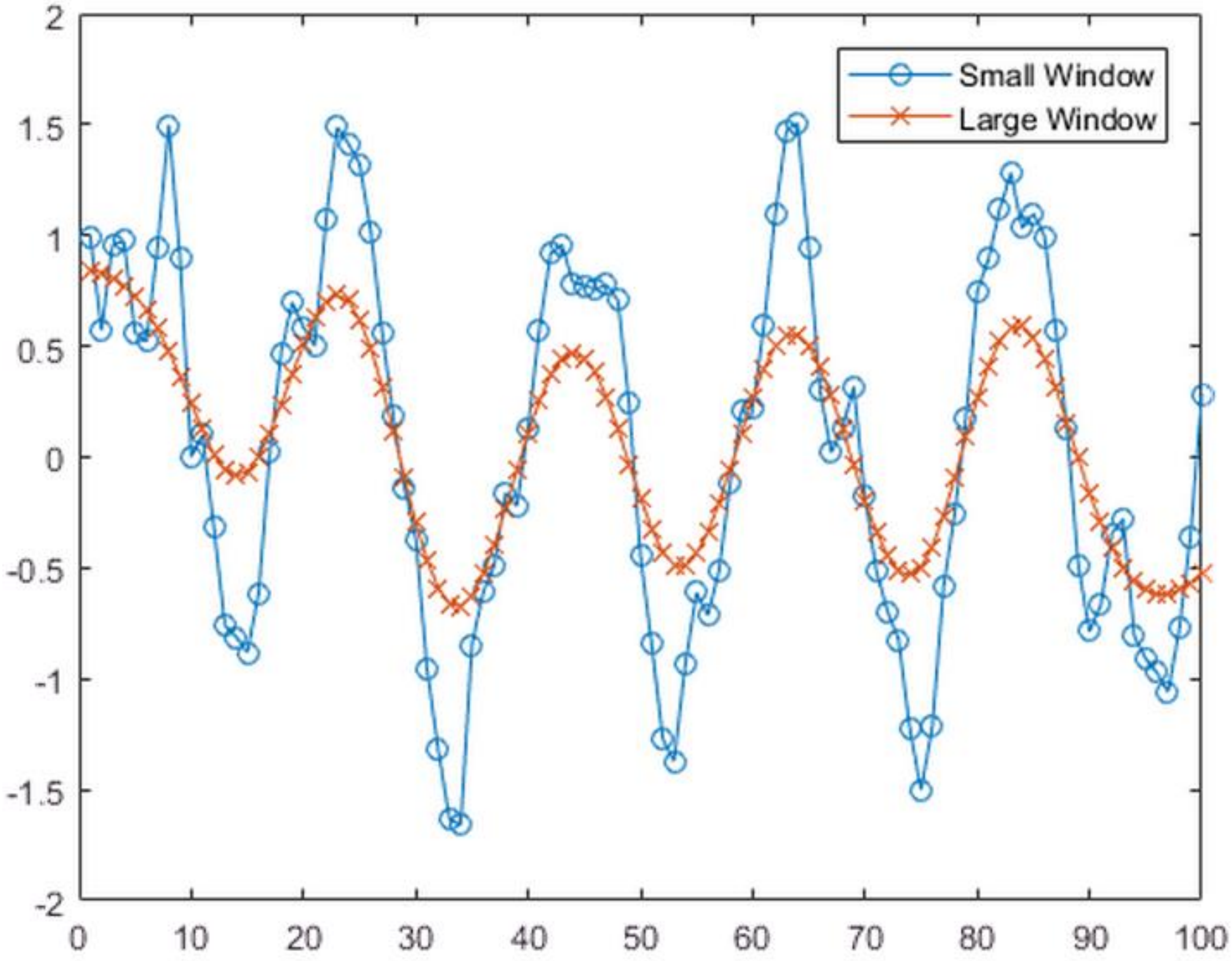
Smooth a vector of noisy data with a Gaussian-weighted moving average filter. Display the window length used by the filter.

```
x = 1:100;
A = cos(2*pi*0.05*x+2*pi*rand) + 0.5*randn(1,100);
[B, window] = smoothdata(A,'gaussian');
window
```

```
window = 4
```

Smooth the orginal data with a larger window of length 20. Plot the smoothed data for both window lengths.

```
C = smoothdata(A,'gaussian',20);
plot(x,B,'-o',x,C,'-x')
legend('Small Window','Large Window')
```


Small Window
Large Window
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
10
20
30
40
50
60
70
80
90
100

Vector with NaN

Create a noisy vector containing NaN values, and smooth the data ignoring NaN, which is the default.

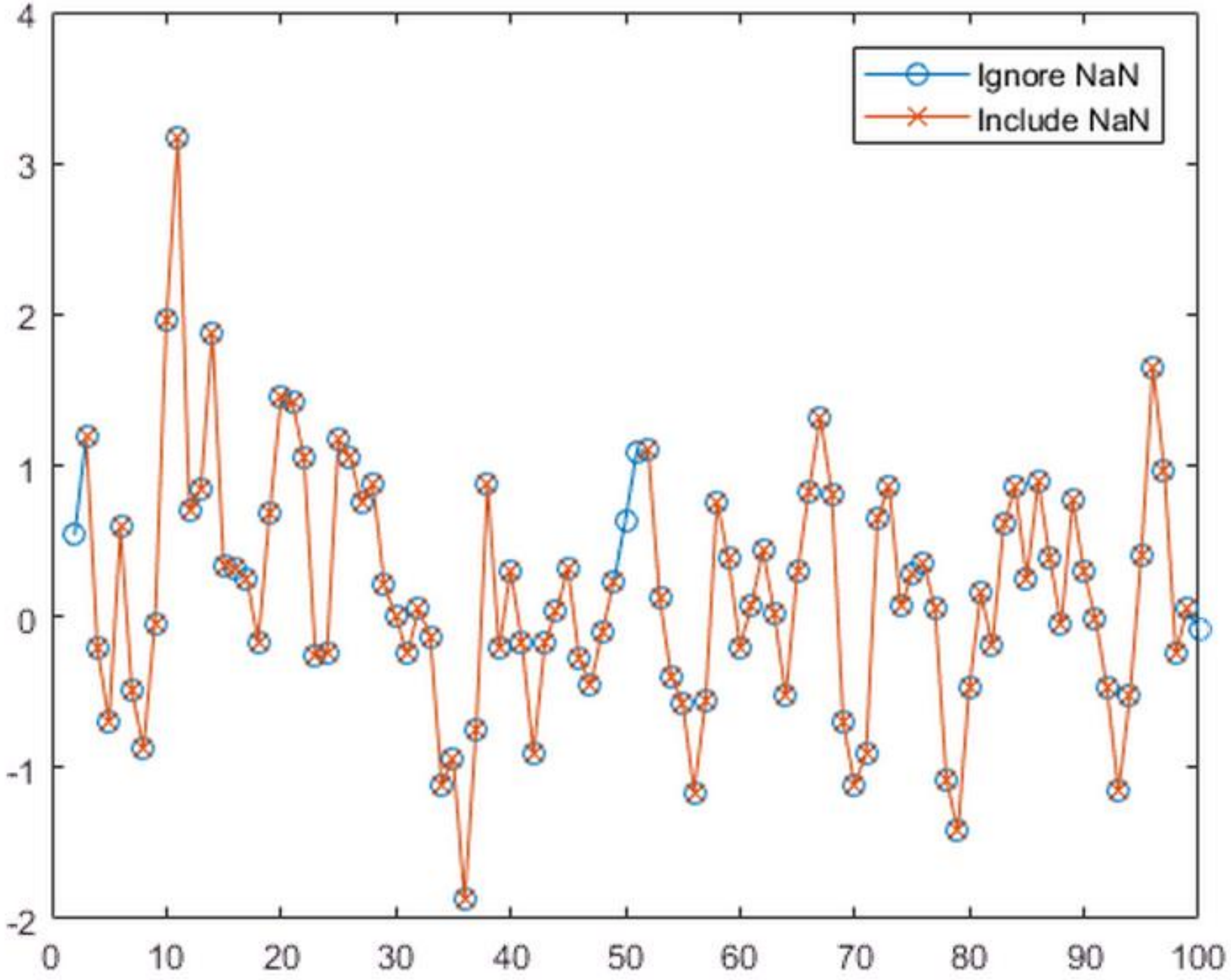
```
A = [NaN randn(1,48) NaN randn(1,49) NaN];
B = smoothdata(A);
```

Smooth the data including NaN values. The average in a window containing NaN is NaN.

```
C = smoothdata(A,'includenan');
```

Plot the smoothed data in B and C.

```
plot(1:100,B,'-o',1:100,C,'-x')
legend('Ignore NaN','Include NaN')
```



Ignore NaN
Include NaN
4
3
2
1
0
-1
-2
0
10
20
30
40
50
60
70
80
90
100

discretize

Group data into bins or categories

Syntax

```
Y = discretize(X,edges)
[Y,E] = discretize(X,N)
[Y,E] = discretize(X,dur)
[ ___ ] = discretize( ___ ,values)
```

```
[ ___ ] = discretize( ___ ,'categorical')
[ ___ ] = discretize( ___ ,'categorical',displayFormat)
[ ___ ] = discretize( ___ ,'categorical',categoryNames)
```

```
[ ___ ] = discretize( ___ ,'IncludedEdge',side)
```

Description

`Y = discretize(X,edges)` returns the indices of the bins that contain the elements of `X`. The `j`th bin contains element `X(i)` if `edges(j) <= X(i) < edges(j+1)` for `1 <= j < N`, where `N` is the number of bins and `length(edges) = N+1`. The last bin contains both edges such that `edges(N) <= X(i) <= edges(N+1)`.

`[Y,E] = discretize(X,N)` divides the range of `X` into `N` uniform bins, and also returns the bin edges `E`.

`[Y,E] = discretize(X,dur)`, where `X` is a datetime or duration array, divides `X` into uniform bins of `dur` length of time. `dur` can be a scalar `duration` or `calendarDuration`, or a unit of time. For example, `[Y,E] = discretize(X,'hour')` divides `X` into bins with a uniform duration of 1 hour.

`[ __ ] = discretize( __ ,values)` returns the corresponding element in `values` rather than the bin number, using any of the previous input or output argument combinations. For example, if `X(1)` is in bin 5, then `Y(1)` is `values(5)` rather than `5`. `values` must be a vector with length equal to the number of bins.

`[ __ ] = discretize( __ ,'categorical')` creates a categorical array where each bin is a category. In most cases, the default category names are of the form "`[A,B)`" (or "`[A,B]`" for the last bin), where `A` and `B` are consecutive bin edges. If you specify `dur` as a character vector, then the default category names might have special formats. See `Y` for a listing of the display formats.

`[ __ ] = discretize( __ ,'categorical',displayFormat)`, for datetime or duration array inputs, uses the specified datetime or duration display format in the category names of the output.

`[ __ ] = discretize( __ ,'categorical',categoryNames)` also names the categories in `Y` using the cell array of character vectors, `categoryNames`. The length of `categoryNames` must be equal to the number of bins.

`[ __ ] = discretize( __ ,'IncludedEdge',side)`, where `side` is `'left'` or `'right'`, specifies whether each bin includes its right or left bin edge. For example, if `side` is `'right'`, then each bin includes the right bin edge, except for the *first* bin which includes both edges. In this case, the `j`th bin contains an element `X(i)` if `edges(j) < X(i) <= edges(j+1)`, where `1 < j <= N` and `N` is the number of bins. The first bin includes the left edge such that it contains `edges(1) <= X(i) <= edges(2)`. The default for `side` is `'left'`.

Group Data into Bins

Use `discretize` to group numeric values into discrete bins. `edges` defines five bin edges, so there are four bins.

```
data = [1 1 2 3 6 5 8 10 4 4]
```

```
data =

     1     1     2     3     6     5     8    10     4     4
```

```
edges = 2:2:10
```

```
edges =

     2     4     6     8    10
```

```
Y = discretize(data,edges)
```

```
Y =

   NaN   NaN     1     1     3     2     4     4     2     2
```

Group Datetime Data by Month

Create a 10-by-1 datetime vector with random dates in the year 2016. Then, group the datetime values by month and return the result as a categorical array.

```
X = datetime(2016,1,randi(365,10,1))
```

```
X = 10×1 datetime array
   24-Oct-2016
   26-Nov-2016
   16-Feb-2016
   29-Nov-2016
   18-Aug-2016
   05-Feb-2016
   11-Apr-2016
   18-Jul-2016
   15-Dec-2016
   18-Dec-2016
```

```
Y = discretize(X,'month','categorical')
```

```
Y = 10×1 categorical array
     Oct-2016
     Nov-2016
     Feb-2016
     Nov-2016
     Aug-2016
     Feb-2016
     Apr-2016
     Jul-2016
     Dec-2016
     Dec-2016
```

Change Display Format of Duration Values

Group duration values by hour and return the result in a variety of display formats.

Group some random duration values by hour and return the results as a categorical array.

```
X = hours(abs(randn(1,10)))'
```

```
X = 10×1 duration array
   0.53767 hr
    1.8339 hr
    2.2588 hr
   0.86217 hr
   0.31877 hr
    1.3077 hr
   0.43359 hr
   0.34262 hr
    3.5784 hr
    2.7694 hr
```

```
Y = discretize(X,'hour','categorical')
```

```
Y = 10×1 categorical array
     [0 hr, 1 hr)
     [1 hr, 2 hr)
     [2 hr, 3 hr)
     [0 hr, 1 hr)
     [0 hr, 1 hr)
     [1 hr, 2 hr)
     [0 hr, 1 hr)
     [0 hr, 1 hr)
     [3 hr, 4 hr]
     [2 hr, 3 hr)
```