Manual of Time Scales Toolbox for MATLAB

BAYLOR UNIVERSITY

WACO, TX 76798

User’s Manual:

Time Scales Toolbox for MatLab

Written By:
Brian Ballard
Bumni Otegbade

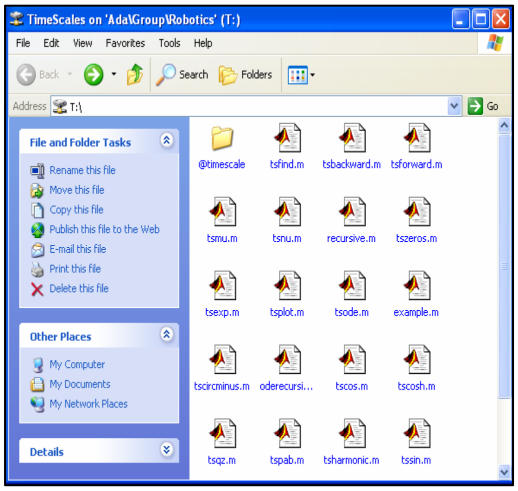
This project funded in part by NSF Grant CNS-0410685

Table of Contents

Chapter 1

Installing the Time Scale Toolbox

This chapter is an introduction to the Time Scale Toolbox explaining how to download the MatLab toolbox from the internet and implement the software on your computer.

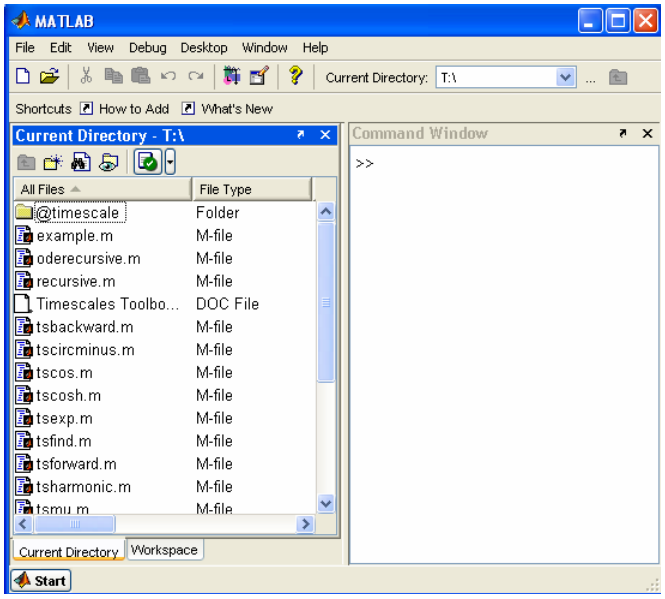


The Time Scale Toolbox for MatLab is downloaded into a zip folder. Inside this folder contains all of the functions and overloads necessary to successfully use time scales with the MatLab software. It is important to understand the structure of the **tstoolbox.zip** folder to be sure that the functions are properly used.
The **tstoolbox.zip** folder contains utility functions that are typed or called by the user, as well as overloaded functions that are regular MatLab operators but modified to perform on time scales. The overloaded functions are all located in the **@timescale** folder and are not designed to be called by their function names, but to simply be used as operators.

Once MatLab is opened, open the **tstoolbox** folder to access the time scale functions and operators.

It is important that the **tstoolbox** directory is the space in which you are working. If the user is working inside the **@timescale** directory most of the functions will not work.

The correct implementation is displayed to the right.


MATLAB
File Edit View Debug Desktop Window Help
Current Directory: T:\
Shortcuts How to Add What's New
Current Directory - T:\
Command Window
>>
Current Directory Workspace
Start

All Files	File Type
@timescale	Folder
example.m	M-file
oderecursive.m	M-file
recursive.m	M-file
Timescales Toolbo...	DOC File
tsbackward.m	M-file
tscircminus.m	M-file
tscos.m	M-file
tscosh.m	M-file
tsexp.m	M-file
tsfind.m	M-file
tsforward.m	M-file
tsharmonic.m	M-file
tsmu.m	M-file

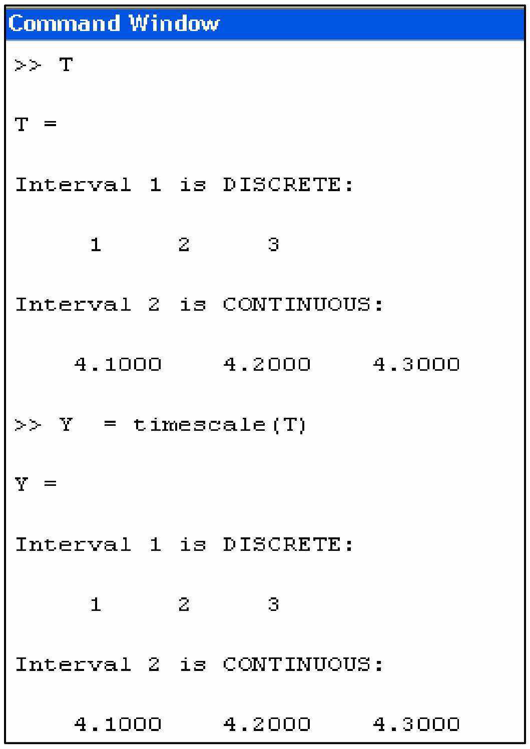
Chapter

2

Time Scale Constructor

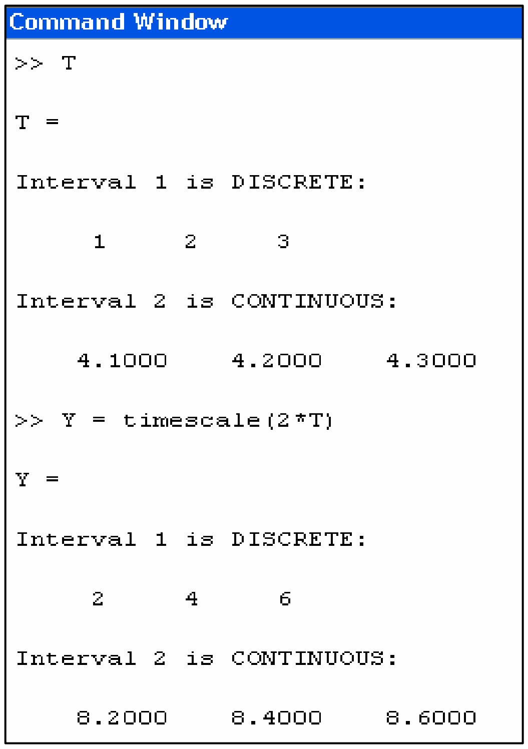
This chapter examines how the time scale constructor is set up, possible ways to define a time scale object, assignment of time scale objects and operations on these time scale objects. This time scale toolbox makes it easy to call the time scale constructor and offers a substantial amount of flexibility.

The time scale constructor is located in the file *timescale.m* in the class methods folder. The time scale object accepts zero, one or an even number of inputs, as is obvious from its calling syntax below. The constructor is made such that when there are no inputs, an empty time scale object is created. When there is only one input, it **must** be another time scale. In that case, the input time scale is simply copied into the resulting time scale, or an operation performed on the input is reported to the result. Time scale objects will accept other time scale objects.



Command Window

```
>> T

T =

Interval 1 is DISCRETE:

     1     2     3

Interval 2 is CONTINUOUS:

    4.1000    4.2000    4.3000

>> Y  = timescale(T)

Y =

Interval 1 is DISCRETE:

     1     2     3

Interval 2 is CONTINUOUS:

    4.1000    4.2000    4.3000
```



Command Window

```
>> T

T =

Interval 1 is DISCRETE:

     1     2     3

Interval 2 is CONTINUOUS:

    4.1000    4.2000    4.3000

>> Y = timescale(2*T)

Y =

Interval 1 is DISCRETE:

     2     4     6

Interval 2 is CONTINUOUS:

    8.2000    8.4000    8.6000
```

FORMAT OF A TIME SCALE OBJECT

T = TIMESCALE(DATA1,FLAG1,DATA2,FLAG2,...)

Inputs to the Time Scale Object:

DATAn:	An array of numbers that represents a time scale interval. The DATA arrays must be 1 x n "row vectors."
FLAGn:	A flag, 'c' or 'd' must be appended to each interval of the time scale to represent whether the associated interval is continuous or discrete. If flag = 'c', then the numbers in DATA are a discretization of a continuous interval. If something other than 'c' or 'd' is input, the default is 'c' (continuous).

Outputs of the Time Scale Object:

T:	A time scale object containing a concentration of the DATA intervals along with the type (continuous or discrete) of the intervals.

An alternative calling syntax is

T = TIMESCALE(DATA, S)

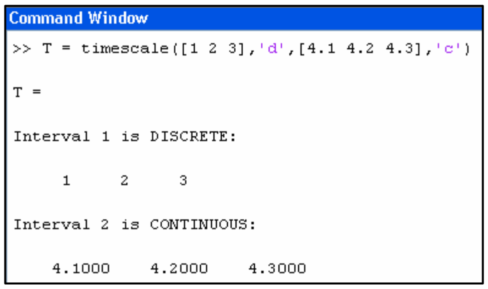
where DATA is a vector of data points and S is a time scale size matrix (cf. *Size* in Chapter 2)

Ways to Enter a Time Scale Object

The time scale constructor can be called in various ways. Examples are given below. All of the definitions below yield the same time scale object.

1. Enter each number individually inside a set of brackets with a space in between each object.

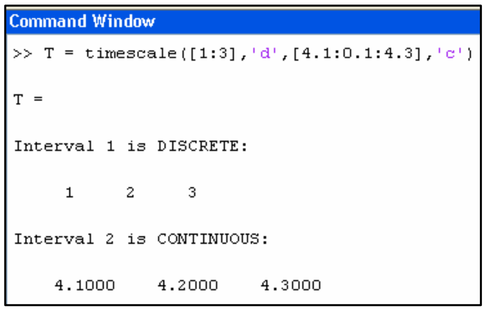
   T = timescale ([1 2 3] , 'd' , [4.1 4.2 4.3] , 'c')



```
>> T = timescale([1 2 3],'d',[4.1 4.2 4.3],'c')

T =

Interval 1 is DISCRETE:

     1     2     3

Interval 2 is CONTINUOUS:

    4.1000    4.2000    4.3000
```

2. Enter a range of numbers inside brackets using the colon operator:

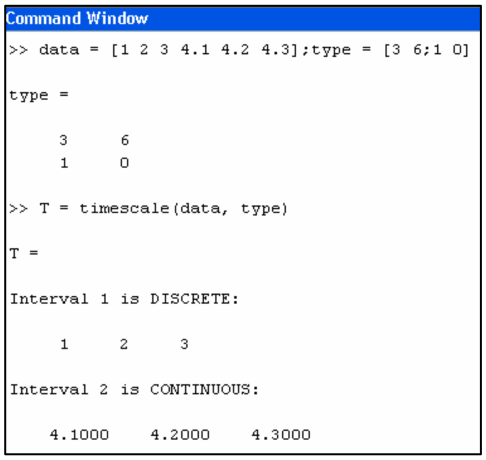
   T = timescale ([1:3], 'd', [4.1:0.1:4.3], 'c')



```
>> T = timescale([1:3],'d',[4.1:0.1:4.3],'c')

T =

Interval 1 is DISCRETE:

     1     2     3

Interval 2 is CONTINUOUS:

    4.1000    4.2000    4.3000
```

3. Create a data vector and a type matrix:

   T = timescale (data, type)
      where data = [1 2 3 4.1 4.2 4.3]
      and type = [3 6;1 0]



Command Window

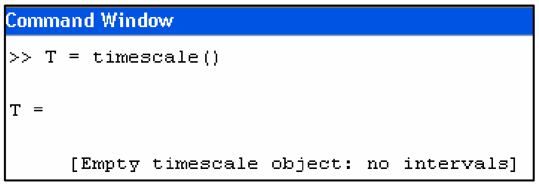
```
>> data = [1 2 3 4.1 4.2 4.3];type = [3 6;1 0]

type =

     3     6
     1     0

>> T = timescale(data, type)

T =

Interval 1 is DISCRETE:

     1     2     3

Interval 2 is CONTINUOUS:

    4.1000    4.2000    4.3000
```

Notice that each different syntax produces the same result. There are multiple ways to enter time scale objects.

Empty Time Scales

Empty sets or empty interval time scales can also be created.

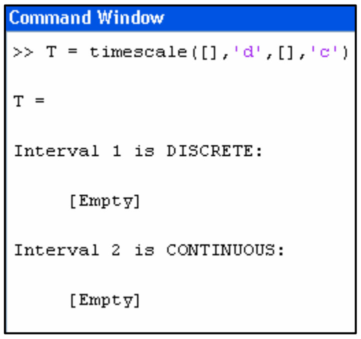
Here is an example of an empty set time scale.



Command Window

```
>> T = timescale()

T =

     [Empty timescale object: no intervals]
```

Here is an example of an empty interval time scale.



Command Window

```
>> T = timescale([],'d',[],'c')

T =

Interval 1 is DISCRETE:

     [Empty]

Interval 2 is CONTINUOUS:

     [Empty]
```

How Time Scale Objects Are Stored and Handled

When created, a time scale object T is a class object of type ‘timescale’ which has two class fields.

DATA Field

The DATA field is several arrays of the numbers that make up each interval of the object. These N DATA arrays must be 1 x n or row vectors, where n is the length (number of data points) of each interval and N is the number of intervals in the object.

TYPE Field

The second piece of the time scale object is the TYPE field which is a 2 x N matrix where N is the number of intervals. It contains very vital and useful information about the time scale object that has been created.

First Row: indices of the points in the DATA array. Each index in the TYPE matrix is the index of the last number in each interval. If an interval is EMPTY, its index is the same as the index of the previous interval.

Second Row: a binary representation of the flags 'c' (represented as 0) and 'd' (represented as 1). Again, there are N columns of these flags.

<u>Example</u>

Consider the time scale

T = timescale ([1 2 3], 'd' , [4.1 4.2 4.3] , 'c').

As shown in the figure below, the type matrix is

$$TYPE = \begin{bmatrix} 3 & 6 \\ 1 & 0 \end{bmatrix}.$$

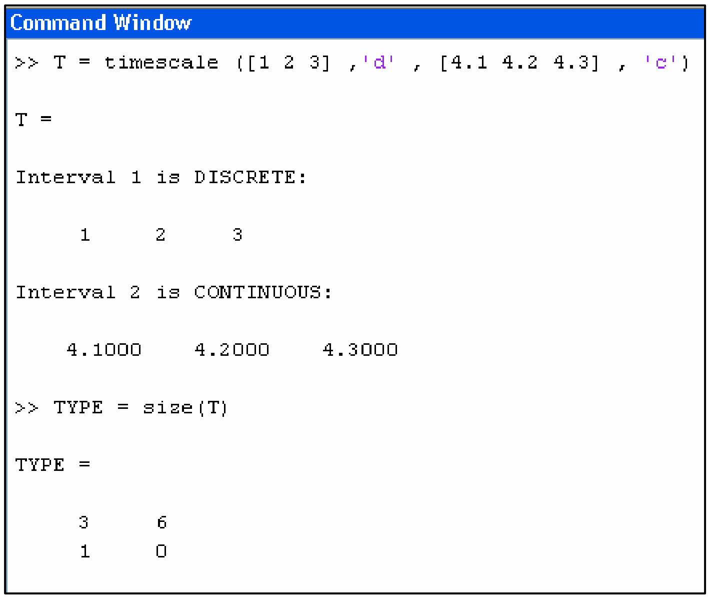
From this matrix, we can reach several conclusions about the time scale object:

i) there are only two intervals in the time scale object.

ii) the last piece of data in the first interval is the third in the time scale and the last number in the second interval is the sixth in the time scale.

iii) there are three data points in both intervals.

iv) the first interval is discrete and the second is continuous



```
>> T = timescale ([1 2 3] ,'d' , [4.1 4.2 4.3] , 'c')

T =

Interval 1 is DISCRETE:

     1     2     3

Interval 2 is CONTINUOUS:

    4.1000    4.2000    4.3000

>> TYPE = size(T)

TYPE =

     3     6
     1     0
```

Chapter
3

Overloaded Function Operations

In creating the Time Scales toolbox many of the usual operations must be overloaded to be defined for the time scale class. Some new operations have been added.

Many mathematical functions which can be performed on real numbers cannot be performed the same way on time scales. MATLAB already provides these functions for real numbers, but this toolbox allows users to perform these operations on time scales, using time scale calculus. These functions are contained inside a folder called "@timescale" and do not always require a call of their function name to operate.

abs **Calling Syntax:** A = abs (B)

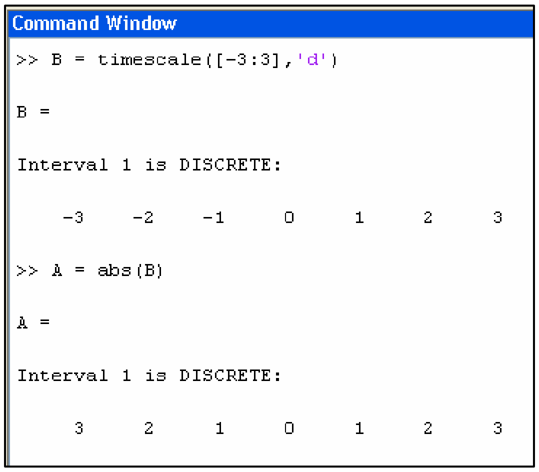
Inputs:

A: time scale object

Output:

B: time scale object containing the absolute value of the data entered

This is the absolute value of the elements of the time scale object. When any of the data in the time scale is complex, this function yields the complex modulus (magnitude) of those elements of the time scale.



```
>> B = timescale([-3:3],'d')

B =

Interval 1 is DISCRETE:

    -3    -2    -1     0     1     2     3

>> A = abs(B)

A =

Interval 1 is DISCRETE:

     3     2     1     0     1     2     3
```

and **Calling Syntax:** C = and (A, B)

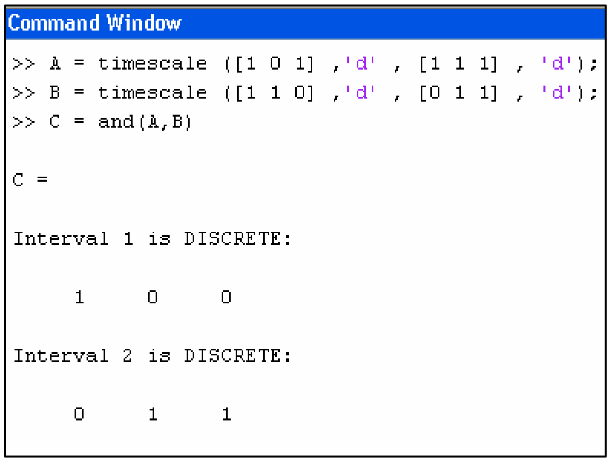
Inputs:

A, B: time scale objects

Output:

C: time scale object containing the usual logical AND of the data in time scales A and B

The AND function implements the time scale equivalent of MATLAB's usual AND logical operator, C = A & B. The output holds a time scale object of 1's and 0's.


Command Window

```
>> A = timescale ([1 0 1] ,'d' , [1 1 1] , 'd');
>> B = timescale ([1 1 0] ,'d' , [0 1 1] , 'd');
>> C = and(A,B)

C =

Interval 1 is DISCRETE:

     1     0     0

Interval 2 is DISCRETE:

     0     1     1
```

angle **Calling Syntax:** A = angle (B)

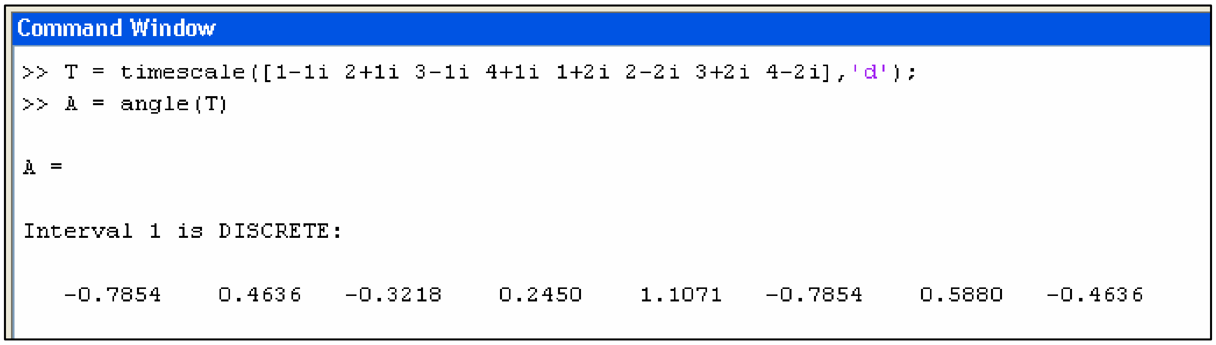
Inputs:

B: time scale object

Output:

A: time scale object containing the complex plane angles of the complex numbers in input B

ANGLE is a time scale overload of the regular MATLAB angle function. A time scale containing the complex plane angles of the complex numbers in the input is returned. The solotion is given in radians.


Command Window

```
>> T = timescale([1-1i 2+1i 3-1i 4+1i 1+2i 2-2i 3+2i 4-2i],'d');
>> A = angle(T)

A =

Interval 1 is DISCRETE:

   -0.7854    0.4636   -0.3218    0.2450    1.1071   -0.7854    0.5880   -0.4636
```

display **Calling Syntax:** display (A)

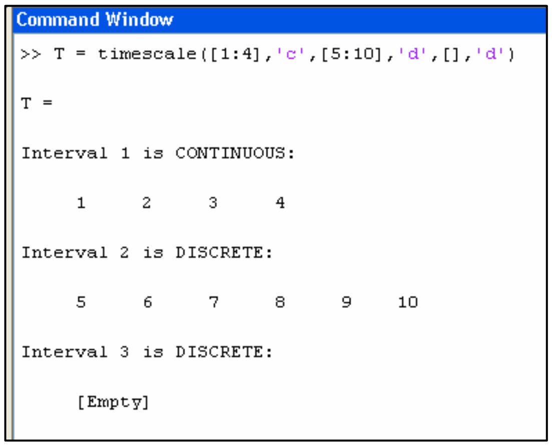
Inputs:

A: time scale object

Output:

None. Operation is performed.

The DISPLAY function is implicitly called when a MATLAB command is issued and not terminated with a semi-colon (see image on the right). For time scale objects, it is called by simply evaluating a time scale object without a semi-colon so that the user can see the contents of the time scale. It specifies whether the interval is discrete, continuous or empty. Unlike the regular MATLAB display function, this function displays "empty" when there is an empty interval.



```
>> T = timescale([1:4],'c',[5:10],'d',[],'d')

T =

Interval 1 is CONTINUOUS:

     1     2     3     4

Interval 2 is DISCRETE:

     5     6     7     8     9    10

Interval 3 is DISCRETE:

     [Empty]
```

find **Calling Syntax:** [I, (flag)] = find (T)

Inputs:

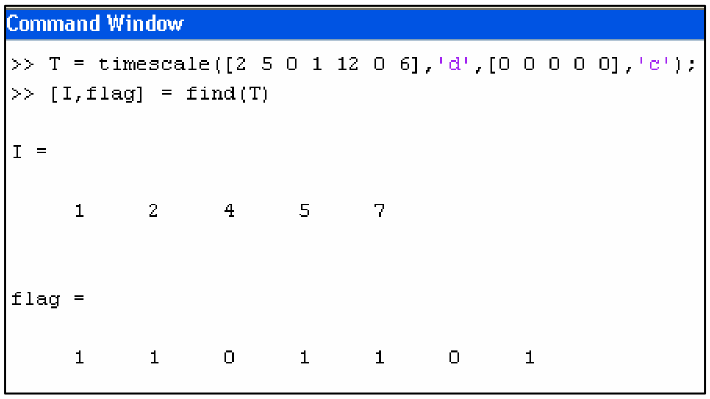
T: time scale object

Outputs:

I: vector

flag: Optional vector

The FIND function works similarly to the standard MATLAB FIND function. It returns the instances of nonzero entries in the time scale object T, such that T(I) returns the nonzero values. The optional FLAG output indicates whether the corresponding indices point to a continuous (FLAG = 0) or discrete (FLAG = 1) interval. FIND returns empty if there are no nonzero entries in T.



```
>> T = timescale([2 5 0 1 12 0 6],'d',[0 0 0 0 0],'c');
>> [I,flag] = find(T)

I =

     1     2     4     5     7


flag =

     1     1     0     1     1     0     1
```

incompatible **Calling Syntax:** Bool = incompatible (A, B)

Inputs:

A, B: time scale objects

Output:

Bool: scalar

This function determines whether time scale objects A and B are compatible. "Compatible" is defined as both time scales having:

(i) the same number of intervals

(ii) the same interval type (continuous or discrete) with each interval

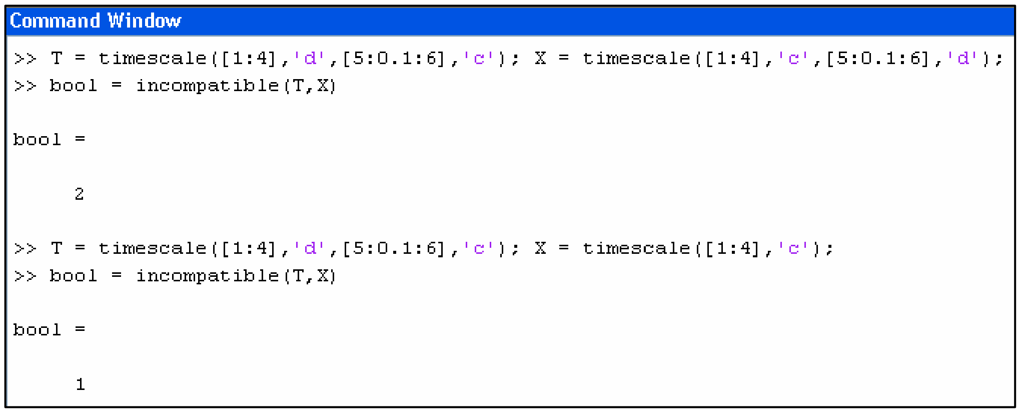
(iii) the same number of data points.

The value of BOOL is determined based on the following:

- **0** The time scales are compatible
- **1** The time scales have a differing number of intervals
- **2** The time scales have differing interval types
- **3** At least one interval in A has a different number of data points than its companion in B.
- **4** A or B is not a time scale.

As seen in the example below, when given two time scales, one with a discrete and continuous interval, the other with identical intervals differing in type, the function returns a 2.

In the second example below, one time scale contains 2 intervals while another time scale contains 1 interval, causing the function to return a 1.



```
>> T = timescale([1:4],'d',[5:0.1:6],'c'); X = timescale([1:4],'c',[5:0.1:6],'d');
>> bool = incompatible(T,X)

bool =

     2

>> T = timescale([1:4],'d',[5:0.1:6],'c'); X = timescale([1:4],'c');
>> bool = incompatible(T,X)

bool =

     1
```

isdiscrete **Calling Syntax:** D = isdiscrete (T)

Inputs:

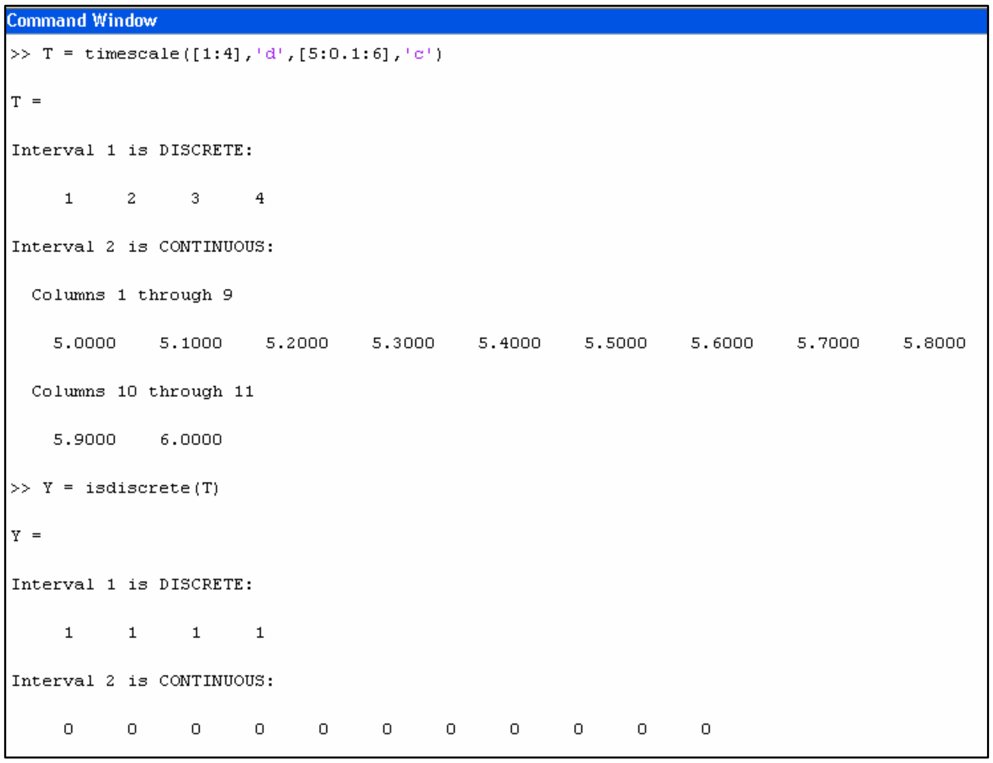
T: time scale object

Output:

D: time scale object containing 1's for discrete intervals

ISDISCRETE returns a time scale object D of the same size and type as input T, with 1's in the discrete intervals and zeros in the continuous intervals.

An example using *isdiscrete* is given:



Command Window

```
>> T = timescale([1:4],'d',[5:0.1:6],'c')

T =

Interval 1 is DISCRETE:

     1     2     3     4

Interval 2 is CONTINUOUS:

  Columns 1 through 9

    5.0000    5.1000    5.2000    5.3000    5.4000    5.5000    5.6000    5.7000    5.8000

  Columns 10 through 11

    5.9000    6.0000

>> Y = isdiscrete(T)

Y =

Interval 1 is DISCRETE:

     1     1     1     1

Interval 2 is CONTINUOUS:

     0     0     0     0     0     0     0     0     0     0     0
```

mtimes **Calling Syntax:** S = mtimes (A, B)

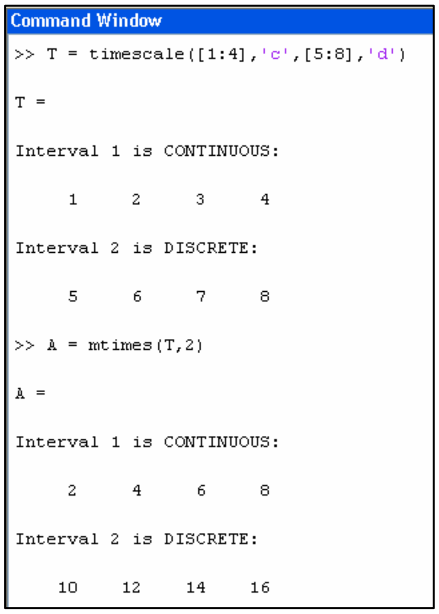
Inputs:

A, B: one time scale object, one scalar

Output:

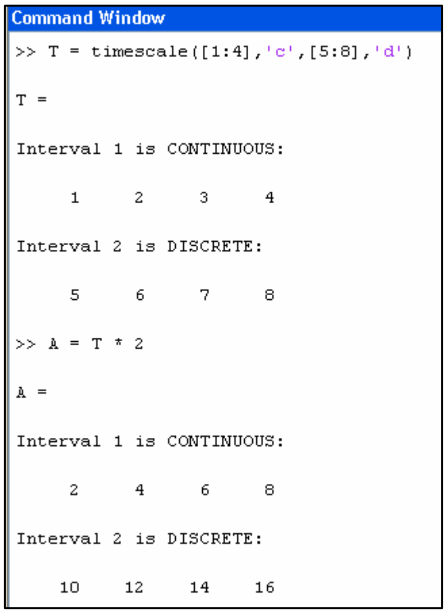
S: time scale object holding the product

MTIMES multiplies data in a time scale object with a scalar and S is a time scale object containing the product. This can be performed by using either of the two examples to the right.



Command Window

```
>> T = timescale([1:4],'c',[5:8],'d')

T =

Interval 1 is CONTINUOUS:

     1     2     3     4

Interval 2 is DISCRETE:

     5     6     7     8

>> A = mtimes(T,2)

A =

Interval 1 is CONTINUOUS:

     2     4     6     8

Interval 2 is DISCRETE:

    10    12    14    16
```



Command Window

```
>> T = timescale([1:4],'c',[5:8],'d')

T =

Interval 1 is CONTINUOUS:

     1     2     3     4

Interval 2 is DISCRETE:

     5     6     7     8

>> A = T * 2

A =

Interval 1 is CONTINUOUS:

     2     4     6     8

Interval 2 is DISCRETE:

    10    12    14    16
```

size **Calling Syntax:** S = size (A)

Inputs:

A, B: time scale object

Output:

S: 2 x m array

SIZE returns the size of the timescale object T. S is a (2 x m) array, where m specifies the number of (continuous and/or discrete) intervals in the timescale. The first row contains the cumulative lengths of the intervals, and the second row indicates whether each interval is continuous or discrete. For example,

$$S = \begin{pmatrix} 2 & 5 & 10 \\ 0 & 1 & 1 \end{pmatrix}$$

implies that interval 1 is of length 2, representing continuous data. Interval 2 is of length 5-2 = 3, representing discrete data. And interval 3 is of length 10-5 = 5, also representing discrete data. Users may construct a size matrix and associated with a vector of data points using the timescale constructor.

subsasgn **Calling Syntax:** A = subsasgn (A, S, B)

Inputs:		**Output:**
A, S, B:	timescale object, syntax structure, reference structure	S: timescale object

SUBSASGN is the left to right time scale overload of the parenthesis operators in MATLAB. This function enables the user to assign values to time scale objects using indexes similar to vector and matrix assignment.

A (n) = 2	Assigns 2 to the nth element of the time scale's data object, where n must be a scalar within the length of the time scale data array. The time scale may not shrink or expand under this operation.
A (2:4) = [3:5]	Returns an error.
A (2:4, k) = [3:5]	Assigns the values [3,4,5] to elements 2 through 4 of interval k. The interval may shrink or expand under this operation, but more intervals may not be added. Assignment of the empty interval A (:,n) = [ ] will not delete the interval but will simply leave its contents empty.
A {n} = T	Embeds time scale T as interval n in time scale A. The assignment requires T to have only one interval. Intervals may be deleted by using the syntax A{n} = [ ]. If n is greater than the number of intervals in A, then A will expand to have n intervals by filling in any missing intervals with empty intervals of type 'd'.

subsref **Calling Syntax:** B = subsref (A, S)

Inputs:		**Output:**
A, S:	one time scale object, one syntax structure	B: time scale object holding the product

SUBSREF is the right to left time scale overload of the parenthesis operator in MATLAB. It returns the operation A (1) or A (2:4) where A is a time scale object. The output is a vector the same length as the input. Also, this function returns the time scale data and type arrays as A.data and A.type.

A (m:n)	Returns a row vector containing the data in A from index m to n regardless of interval boundaries
A (m:n, k)	Returns a row vector containing the data in the k-th interval of A.
A (m:n , p:q)	Returns an error.
A {n}	Returns a time scale object containing the data of interval n. This is a shortcut for S = size (A); T = timescale (A(:,n), 'd' or 'c')

Other Overloaded Operations

As it would be somewhat unnecessary to document how every overloaded function works, each of the following overloaded functions can be understood intuitively and are briefly described below.

NAME	CALLING SYNTAX	DESCRIPTION
conj	**A = conj (B)** A, B are time scale objects	Timescale overload of conj function. A is a timescale containing the conjugates of the complex numbers in timescale B
cos	**A = cos (B)** A, B are time scale objects	Timescale overload of the co-sine function. A contains *point-wise* cos of timescale B. Note that this is different from *tscos*, which is part of actual time scales trigonometry.
eq	**C = eq (A, B)** A, B, C are time scale objects	Implements the timescale equivalent of MATLAB's usual == logical function, C = A==B. Output C holds a timescale object of 1's and 0's.
exp	**A = exp (B)** A, B, C are time scale objects	Timescale overload of exp function. A is a timescale containing the point-wise exponential of timescale B. Note that this is not to be confused with *tsexp* which is part of actual time scales calculus.
ge	**C = ge (A, B)** A, B, C are time scale objects	Implements the timescale equivalent of MATLAB's usual >= logical function, C = A>=B. Output C holds a timescale object of 1's and 0's.
gt	**C = gt (A, B)** A, B, C are time scale objects	Implements the timescale equivalent of MATLAB's usual greater-than logical function, C = A>B. Output C holds a timescale object of 1's and 0's.
horzcat	**T = horzcat (T1, T2, T3...)** **T = [T1 T2 T3]** T, T1, T2…are time scale objects	Concatenates many timescales into one. Inputs must be timescale objects. Overloads the bracket operator so that entering T = [T1 T2 T3] returns one time scale containing each of the time scales.
imag	**A = imag (B)** A, B, C are time scale objects	Timescale overload of imag function. A is a timescale containing the imaginary part of timescale B.
le	**C = le (A, B)** A, B, C are time scale objects	Implements the timescale equivalent of MATLAB's usual less-than-or-equal-to (<=) logical function, C = A<=B. Output C holds a timescale object of 1's and 0's.

length	**L = length (T)** L is a scalar, T is a time scale object	Returns the length of a timescale object, meaning the total number of data points in all intervals of T.
log	**A = log (B)** A, B are time scale objects	Timescale overload of log function. A contains point-wise natural logarithm of timescale B.
lt	**C = lt (A, B)** A, B, C are time scale objects	Implements the timescale equivalent of MATLAB's usual less-than (<) logical function, C = A<B. Output C holds a timescale object of 1's and 0's.
min	**A = min (B)** A is a scalar; B is a time scale object.	Overloaded min function. Returns the minimum (smallest number) in timescale B.
minus	**S = minus (A, B)** A, B are either time scale objects or scalars. S is a time scale object.	Subtracts data in time scale objects and produces a new time scale object holding the difference. A or B may be scalar. Output S is a time scale object holding the difference (A – B).
mpower	**S = mpower (A, B)** S, A, B are time scale objects.	Identical to A .^B - allows users to -- see power(A,B)
mrdivide	**C = mrdivide (A, B)** A, C are time scale objects. B is a scalar.	Called to overload the A/B syntax for timescale objects. B must be a scalar.
ne	**C = ne (A, B)** A, B, C are time scale objects.	Implements the timescale equivalent of MATLAB's usual not equal to (~=) logical function, C = A~=B. Output C holds a timescale object of 1's and 0's.
not	**B = not (A)** A, B are time scale objects.	Implements the time scale equivalent of MATLAB's usual NOT logical operator, B = ~A. Output B holds a time scale object of 1's and 0's.
or	**C = or (A, B)** A, B, C are time scale objects.	Implements the time scale equivalent of MATLAB's usual OR logical operator, C = A|B. Output C holds a timescale object of 1's and 0's.

plus	**S = plus (A, B)** A, B are either time scale objects or scalars. S is a time scale object.	Adds data in time scale objects and produces a new time scale object holding the sum. A or B can be a scalar as well.
power	**S = power (A, B)** S, A, B are time scale objects.	Element-wise power of one timescale to another. Works if either argument is a scalar also.
real	**A = real (B)** A, B are time scale objects.	Time scale overload of real function. Returns a timescale object A containing the real part of timescale object B.
rdivide	**Y = rdivide (A / B)** A is a scalar. B, Y are time scale objects.	Time scale overload of rdivide function. Returns a timescale object Y containing the element to element division of a scalar divided by a time scale.
sin	**A = sin (B)** A, B are time scale objects.	Time scale overload of sin function. A contains point-wise sine of time scale B.
sqrt	**Y = sqrt (X)** X, Y are time scale objects.	Time scale overload of square root function. Executes a point-wise square root on the data in time scale object X and returns the result in Y.
tan	**A = tan (B)** A, B are time scale objects.	Timescale overload of tangent (tan) function. A contains point-wise tangent of timescale B.
times	**S = times (A, B)** A, B are either time scale objects or scalars. S is a time scale object.	Element-wise multiply of two timescale objects to produce another. Either argument can be a scalar also. S is the output time scale object holding the product.
uminus	**A = uminus (B)** A, B are timescale objects.	Time scale overload of unary negation. A contains the negative of B.
vertcat	**T = vertcat (varargin)**	Vertical concatenation is not defined for time scales. Hence this function always yields an error.

Chapter

4

Utility Functions

The Time Scales toolbox in MATLAB is furnished with several functions which are not only useful, but also provide a means of performing regular mathematical operations on time scales.

Many mathematical functions which can be evaluated using real numbers cannot be evaluated the same way on time scales. MATLAB already provides these functions for real numbers, but this toolbox allows users to perform these operations on time scales and also do time scale calculus.

Basic Time Scale Operations

In order to do anything with time scales there are some basic operations including calculating the graininess, the forward jump operator, the backward jump operator, and the backward graininess, that are essential to performing more complex time scale operations.

tsbackward **Calling Syntax:** rho = tsbackward (T, [n])

Inputs:

T: time scale object

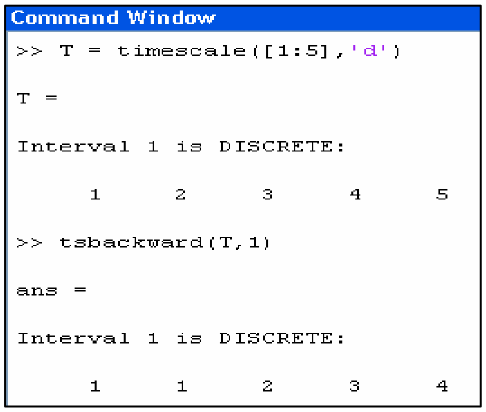
n: (optional) jumps backward n number of points

Output:

rho: time scale object containing the backward jump solution

This function calculates the backward jump operator, $\rho(t)$, which is the current point in a time scale minus the distance to the previous point. There is an optional input, n, that allows the user to return the nth backward jump. For continuous time scales, $t = \rho(t)$.

Example: Entering a discrete time scale at the right and performing the tsbackward operation generates a solution time scale of the same size and type.



```
>> T = timescale([1:5],'d')

T =

Interval 1 is DISCRETE:

     1     2     3     4     5

>> tsbackward(T,1)

ans =

Interval 1 is DISCRETE:

     1     1     2     3     4
```

tsforward **Calling Syntax:** sigma = tsforward (T, [n])

Inputs:

T: time scale object

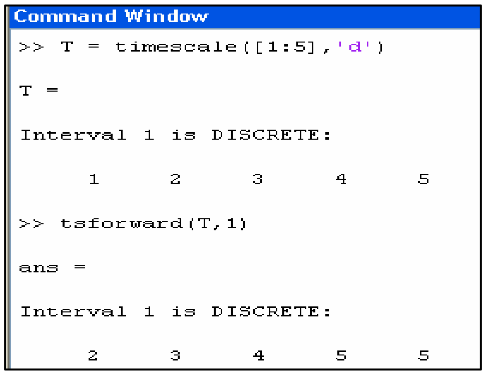
n: (optional) jumps forward n number of points

Output:

sigma: time scale object containing the forward jump solution

This function calculates the forward jump operator, σ(t), which is the current point in a time scale plus the distance to the next point. There is an optional input, n, that allows the user to return the nth forward jump. For continuous time scales, t = σ(t).

Example: Entering a discrete time scale at the right and performing the tsforward operation generates a solution time scale of the same size and type.



Command Window

```
>> T = timescale([1:5],'d')

T =

Interval 1 is DISCRETE:

     1     2     3     4     5

>> tsforward(T,1)

ans =

Interval 1 is DISCRETE:

     2     3     4     5     5
```

tsmu **Calling Syntax:** mu = tsmu(T)

Inputs:

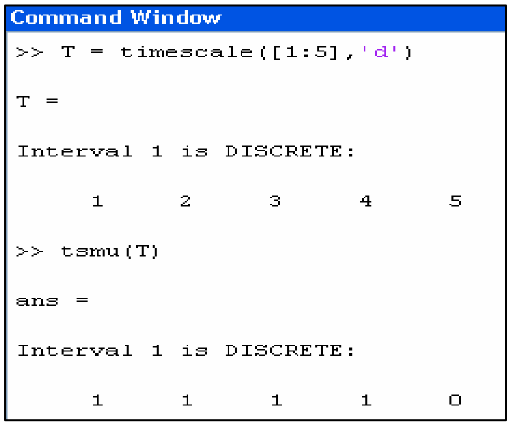
T: time scale object

Output:

mu: time scale object containing the graininess

This function calculates the graininess, μ(t), which is the distance from the current point to the next point. The graininess is defined by μ(t) = σ(t) – t. The solution is returned as a time scale of the same size and type, however by convention the last value will be zero. For continuous time scales, μ(t) = 0.

Example: Entering a discrete time scale at the right and performing the tsmu operation generates the following solution.



Command Window

```
>> T = timescale([1:5],'d')

T =

Interval 1 is DISCRETE:

     1     2     3     4     5

>> tsmu(T)

ans =

Interval 1 is DISCRETE:

     1     1     1     1     0
```

tsnu **Calling Syntax:** nu = tsnu(T)

Inputs:

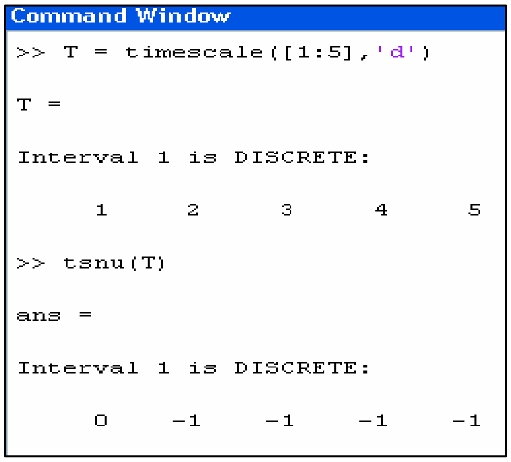
T: time scale object

Output:

nu: time scale object containing the backward graininess

This function calculates the backward graininess, which is the distance from the current point to the previous point. The solution is returned as a time scale of the same size and type, however by convention the last value will be zero. For continuous time scales, the backward graininess equals 0.

Example: Entering a discrete time scale at the right and performing the tsnu operation generates the following solution.


Command Window

```
>> T = timescale([1:5],'d')

T =

Interval 1 is DISCRETE:

     1     2     3     4     5

>> tsnu(T)

ans =

Interval 1 is DISCRETE:

     0    -1    -1    -1    -1
```

tsplot **Calling Syntax:** hndl = tsplot (X1, Y1, X2, Y2,…)
or hndl = tsplot (varargin)

Inputs:

varargin: multiple time scale objects

Output:

hndl: the plot handle

TSPLOT plots time scale objects against each other, Y1 vs. X1, Y2 vs. X2, etc. Each X_n and Y_n must be compatible (same interval sizes and types).

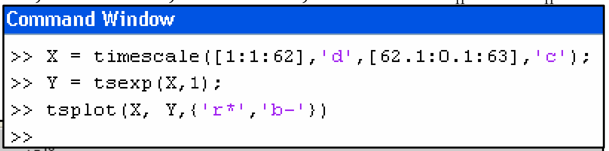
Special syntax options:

TSPLOT is the same as TSPLOT (X1, tszeros (size (X1))) and plots an even number of time scales against one another.

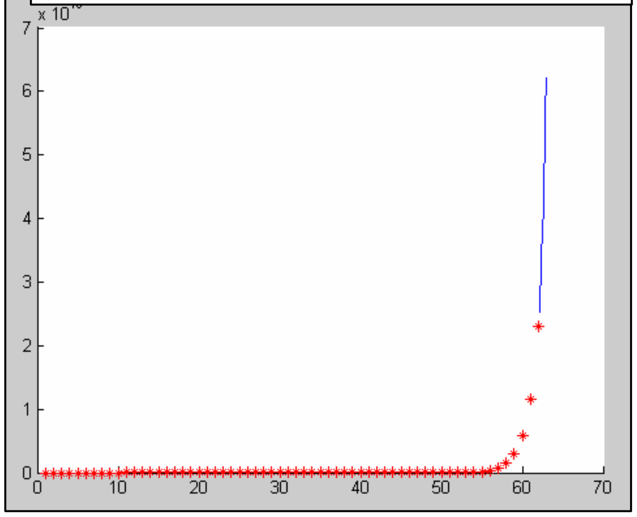
TSPLOT (X1, Y1, S1 X2, Y2, S2…) will use cell array S to determine the line-type for the plot. For example,

tsplot (X, Y, {‘r*’, ‘b-’})

will plot Y vs. X with discrete intervals showing as red stars and continuous intervals showing as a blue line (cf. HELP PLOT for a comprehensive list of plot options).


Command Window

```
>> X = timescale([1:1:62],'d',[62.1:0.1:63],'c');
>> Y = tsexp(X,1);
>> tsplot(X, Y,{'r*','b-'})
>>
```



The default behavior (no S input) is

S = {‘ . ’, ‘ - ’},

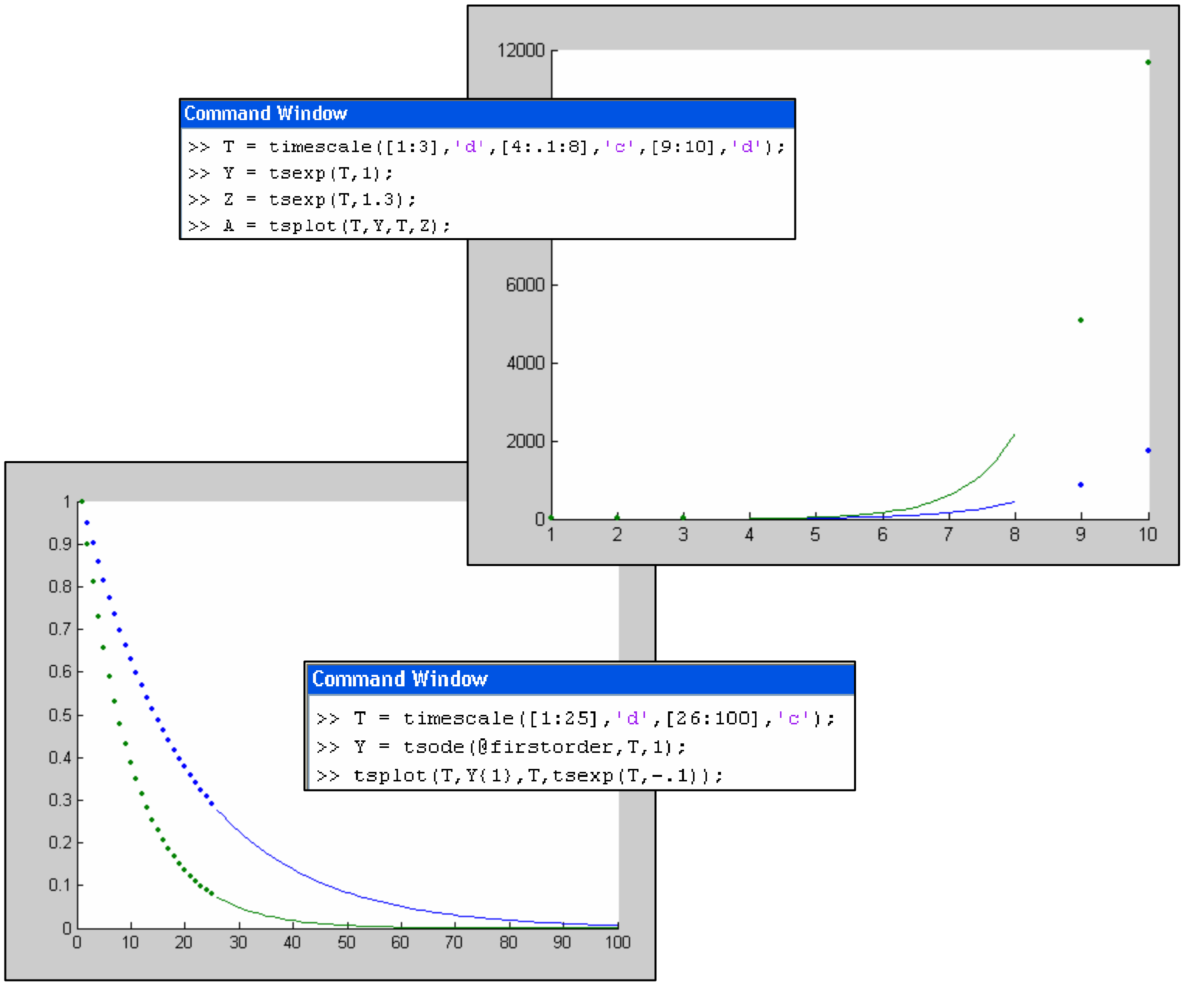
i.e. dots for discrete intervals and connecting lines for continuous intervals, both of the same color. If no line/color styles are specified, TSPLOT will cycle through the current axis color order for each plot.

HNDL = TSPLOT (X1, Y1,…) will return a cell array HNDL containing handles to all individual interval plots. For example, if X1 and Y1 contain a discrete interval, then a continuous interval, then another discrete interval, then HNDL will have one element, a vector with three handles in it.

HNDL{1} = [h1, h2, h3]

where h1 is the handle to the first (discrete) interval, h2 is the handle to the second, etc. The handles for X2 and Y2 are in cell HNDL {2} and so forth.

Additional examples of command window entries with the resulting plot are given:



```
>> T = timescale([1:3],'d',[4:.1:8],'c',[9:10],'d');
>> Y = tsexp(T,1);
>> Z = tsexp(T,1.3);
>> A = tsplot(T,Y,T,Z);
```

```
>> T = timescale([1:25],'d',[26:100],'c');
>> Y = tsode(@firstorder,T,1);
>> tsplot(T,Y{1},T,tsexp(T,-.1));
```

Time Scale Calculus Operations

Time scale mathematical operations are significantly different from real number operations. This is emphasized when we talk about time scale calculus. In most cases, calculations for the continuous part of a time scale are done as the usual real calculus. For example, the derivative of a continuous interval is the usual derivate such that the Hilger derivative f^{Δ} is

$$f^{\Delta} = f'$$

Examined below are two important time scale calculus functions – tsexp for the time scale exponential and tsode which is the time scale O.D.E solver.

tsexp **Calling Syntax:** Y = tsexp(T, alpha, [t_0])

Inputs:

T: time scale object

α: scalar exponent

t_0: initial value (optional)

Output:

Y: time scale object containing the solution of the time scale exponential.

TSEXP (T, α, t_0) returns a time scale exponential of the input time scale. The commands are illustrated below for time scale

The initial condition (t_0) must be contained in T. If there is no t_0 input, the first value of the timescale T is used as the initial value. If t_0 is not the first term of the timescale, the exponential (Y) of all values before t_0 is set to zero.

This function calculates the time scale exponential slightly differently from the formulae proposed by Bohner – Peterson which includes the rather complex cylinder transformation. The implemented calculation is split into three parts:

i. Uniform discrete interval – when the graininess of an interval is constant and non zero, the exponential is calculated using the formula described by Bohner – Peterson [74].

ii. $$Y(t) = (1+\alpha\mu(t))^{(t-t_0)/\mu(t)}$$

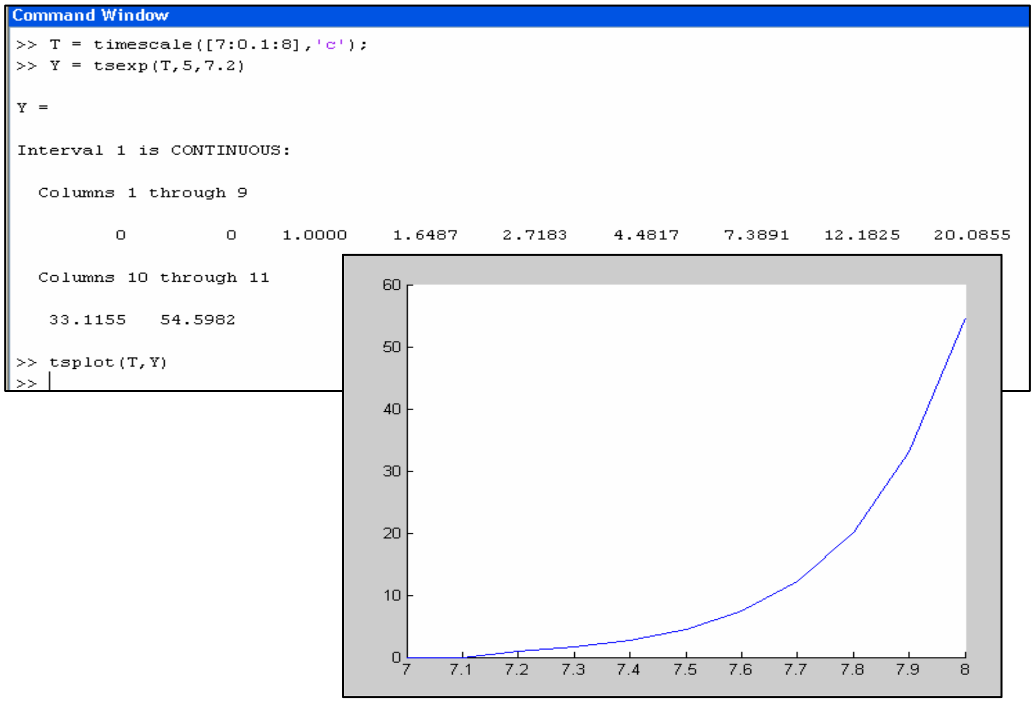
iii. Non-uniform discrete interval – when the graininess of an interval is not constant, the exponential is derived by recursion.

iv. Continuous interval – when the graininess is zero, the exponential is obtained using the usual real exponential function (exp) provided by MATLAB.

v. $$Y(t) = e^{\alpha(t-t_0)}$$

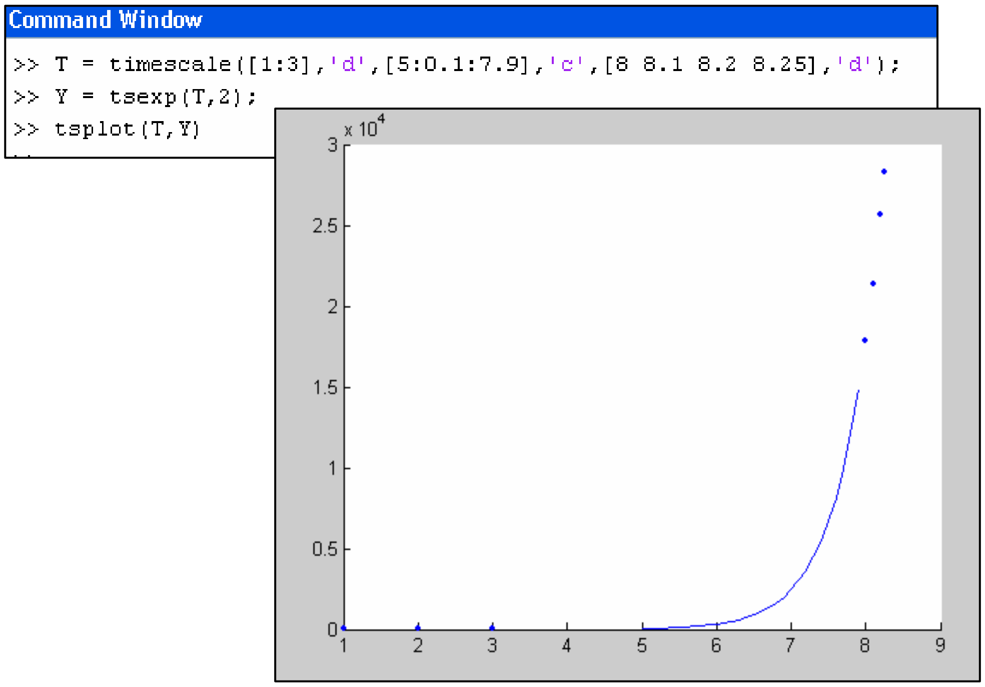
Here are some examples of using the ***tsexp*** function:

A single continuous interval time scale with alpha = 5 and an initial value of 7.2:



```
>> T = timescale([7:0.1:8],'c');
>> Y = tsexp(T,5,7.2)

Y =

Interval 1 is CONTINUOUS:

  Columns 1 through 9

         0         0    1.0000    1.6487    2.7183    4.4817    7.3891   12.1825   20.0855

  Columns 10 through 11

   33.1155   54.5982

>> tsplot(T,Y)
>>
```

A discrete, continuous, discrete type time scale with alpha = 2, and an initial value of 1:



```
>> T = timescale([1:3],'d',[5:0.1:7.9],'c',[8 8.1 8.2 8.25],'d');
>> Y = tsexp(T,2);
>> tsplot(T,Y)
```

tsode **Calling Syntax:** Z= tsode (@f, T, Y_0)

Inputs:

ODEFUN: file containing the differential equation

T: time scale object

Y_0: initial condition (number or vector)

Output:

Z: cell array containing as many output time scales as initial conditions.

ODEFUN is the file that contains the ODE function. It returns a number or vector (according to the order of the D.E.), that contains dy values. The initial condition (Y_0) is a number for a first order D.E. When the order is greater than one, the initial condition input is a vector with N members. The output Z is a cell array that contains N output time scales (Y1, Y2, Y3…) corresponding to each initial condition.

This function integrates the system of differential equations y' = f (t, y) from time t_0 to t_f in the timescale T with initial condition(s) Y_0. The solution is obtained differently for continuous and discrete intervals:

i. Discrete Interval: when the interval is discrete, the solution is obtained by a recursive difference method:

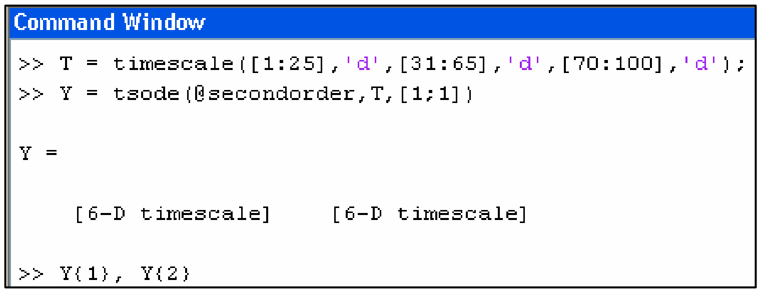
$$Y(t+1) = f(y(t)) \times \mu(t) + y(t)$$

where $Y(t+1)$ is the next value in the solution time scale, $f(y(t))$ is the result of the current solution passed through the function contained in the ODE file (f) and $\mu(t)$ is the graininess of the time scale T.

ii. Continuous Interval: in this case, the differential equation is solved using the ODE23 solver provided by MATLAB which solves non-stiff differential equations using the low order method.

Accessing the Output Data (Cell Array):

The function stores the output into a cell array, requiring the user to access the data output by calling the corresponding element of the cell array. In a first order ODE there is only one element in the cell array, and in a second order ODE there are two elements, likewise for each type of differential equation. To access the elements of the cell array for a second order ODE, view the figure below:

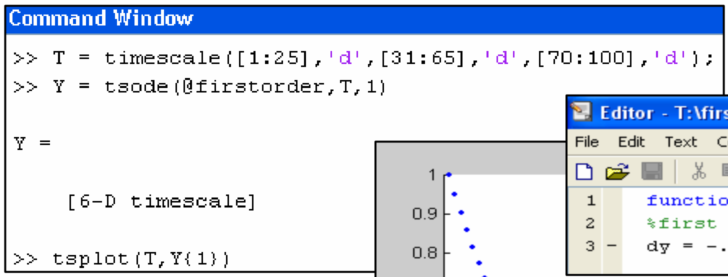


```
>> T = timescale([1:25],'d',[31:65],'d',[70:100],'d');
>> Y = tsode(@secondorder,T,[1;1])

Y =

    [6-D timescale]    [6-D timescale]

>> Y{1}, Y{2}
```

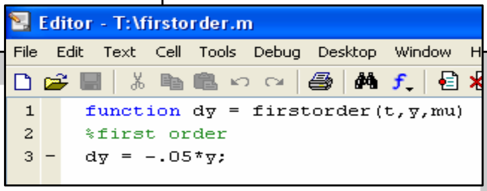
Y{1}: accesses the data for the dy1 equation

Y{2}: accesses the data for the dy2 equation

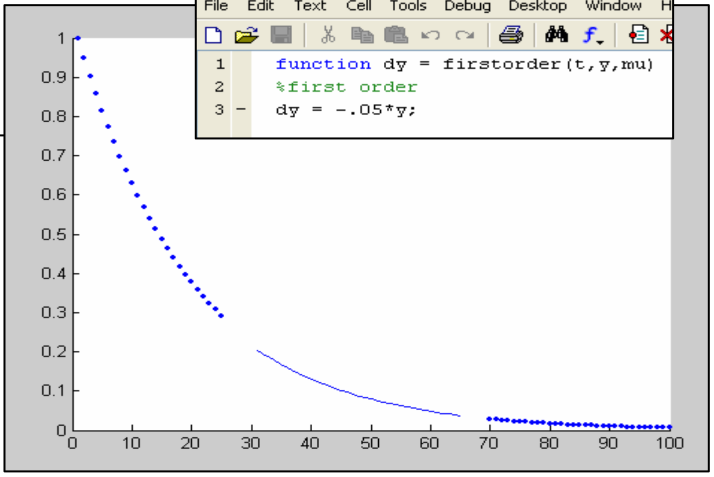
Examples of using the ***tsode*** function are given below:



```
>> T = timescale([1:25],'d',[31:65],'d',[70:100],'d');
>> Y = tsode(@firstorder,T,1)

Y =

    [6-D timescale]

>> tsplot(T,Y{1})
```



```
function dy = firstorder(t,y,mu)
%first order
dy = -.05*y;
```

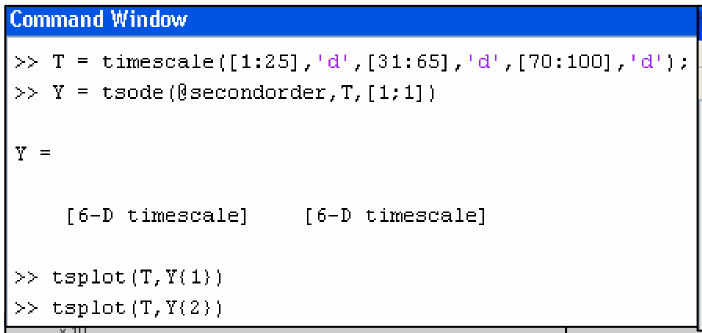


Here is the solution of a first order differential equation using a discrete, continuous, discrete type time scale and the equation

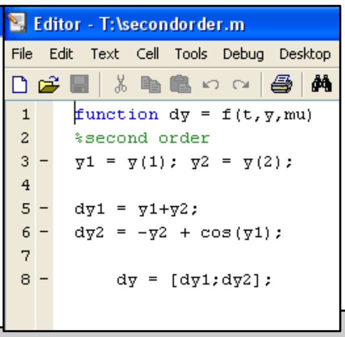
dy = -0.05 * y

Here is a solution of a second order differential equation using a discrete, continuous, discrete time scale and the second order equation

$$dy1 = y1 + y2$$
$$dy2 = -y2 + \cos(y1)$$

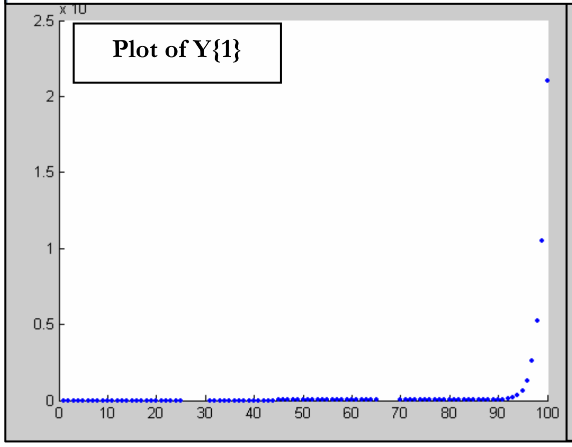


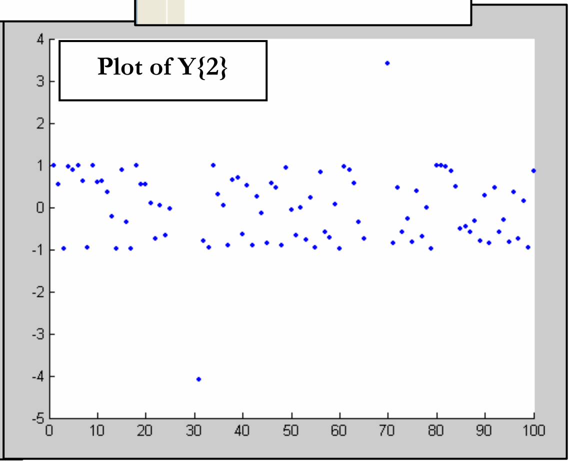
```
>> T = timescale([1:25],'d',[31:65],'d',[70:100],'d');
>> Y = tsode(@secondorder,T,[1;1])

Y =

    [6-D timescale]    [6-D timescale]

>> tsplot(T,Y{1})
>> tsplot(T,Y{2})
```



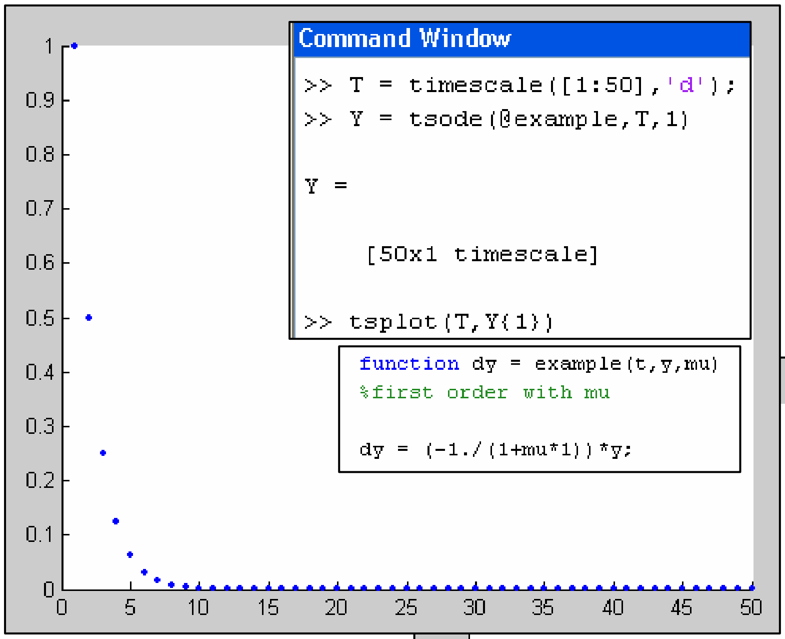
Editor - T:\secondorder.m
File Edit Text Cell Tools Debug Desktop

```
function dy = f(t,y,mu)
%second order
y1 = y(1); y2 = y(2);

dy1 = y1+y2;
dy2 = -y2 + cos(y1);

    dy = [dy1;dy2];
```





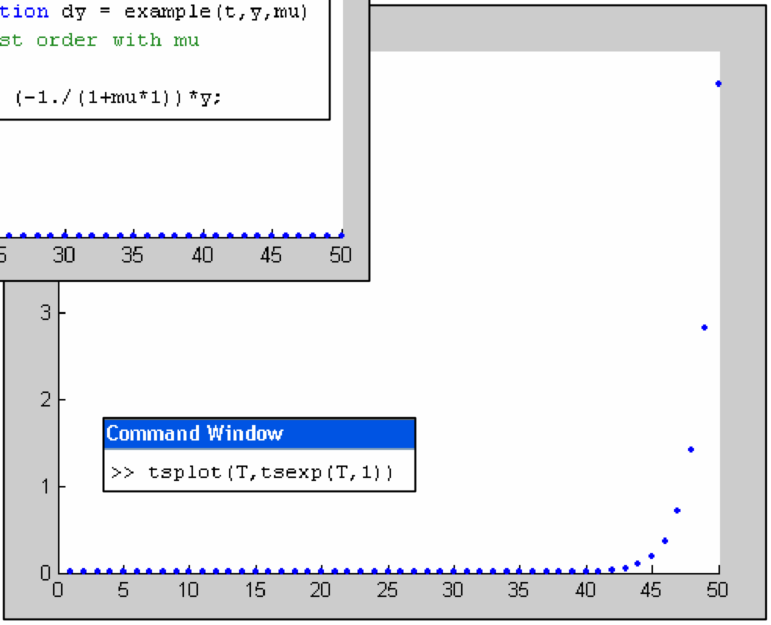
The ***tsode*** function automatically passes the graininess to the user-defined function. This allows the user to create differential equations that involve the graininess. One example of using the graininess in a single order differential equation is given below.



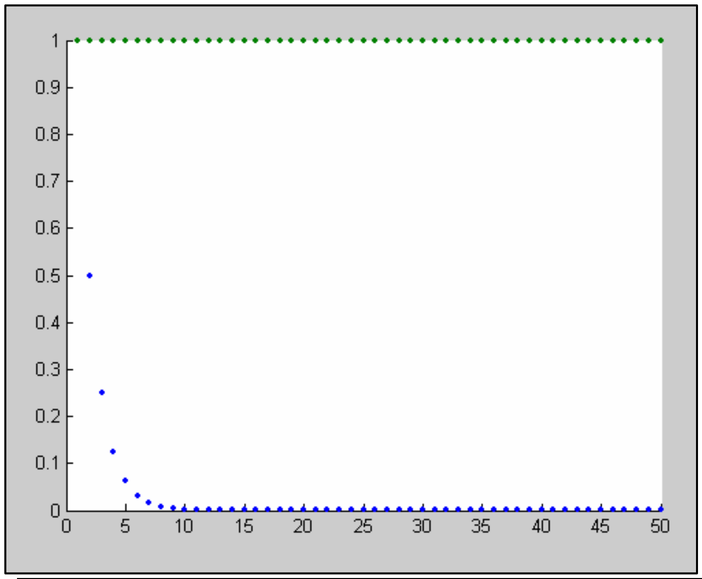
```
Command Window
>> T = timescale([1:50],'d');
>> Y = tsode(@example,T,1)

Y =

     [50x1 timescale]

>> tsplot(T,Y{1})
```

```
function dy = example(t,y,mu)
%first order with mu

dy = (-1./(1+mu*1))*y;
```

Here is a plot of the solution to the ordinary differential equation using the graininess.



```
Command Window
>> tsplot(T,tsexp(T,1))
```



After viewing these two plots one can reason that if the two are multiplied they should produce a solution of 1's, which is verified at left.

Trigonometric & Hyperbolic Functions

The trigonometric functions sine, cosine, hyperbolic sine, and hyperbolic cosine exist on time scales but are calculated slightly different than their counterparts on real numbers.

tscos **Calling Syntax:** Y = tscos(T, alpha, [t_0])

Inputs:

T: time scale object

α: scalar exponent

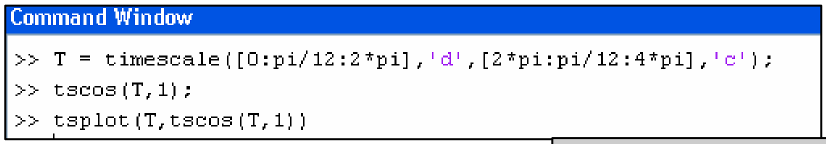
t_0: initial value (optional)

Output:

Y: time scale object containing the solution to the time scale cosine

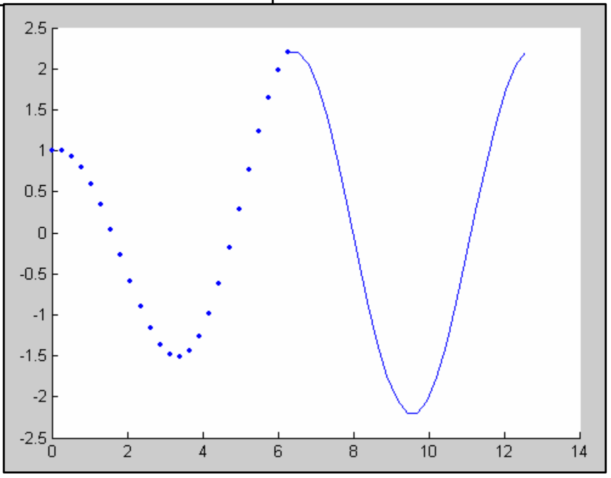
TSCOS (T, α, t0) returns a time scale cosine of T.

This function calculates the time scale cosine using the formula: $\cos_p = \dfrac{e_{ip} + e_{-ip}}{2}$.



Command Window

```
>> T = timescale([0:pi/12:2*pi],'d',[2*pi:pi/12:4*pi],'c');
>> tscos(T,1);
>> tsplot(T,tscos(T,1))
```



Example: Entering the mixed type time scale interval above, generates the plot to the right. Notice how the amplitude grows for a discrete interval and produces a typical cosine wave for a continuous interval.

Entering a continuous time scale interval into the tscos function generates a typical cosine wave on real numbers.

The function discards imaginary values and returns only real valued results.

tscosh **Calling Syntax:** Y = tscosh(T, alpha, [t_0])

Inputs:

T: time scale object

α: scalar exponent

t_0: initial value (optional)

Output:

Y: time scale object containing the solution to the time scale hyperbolic cosine

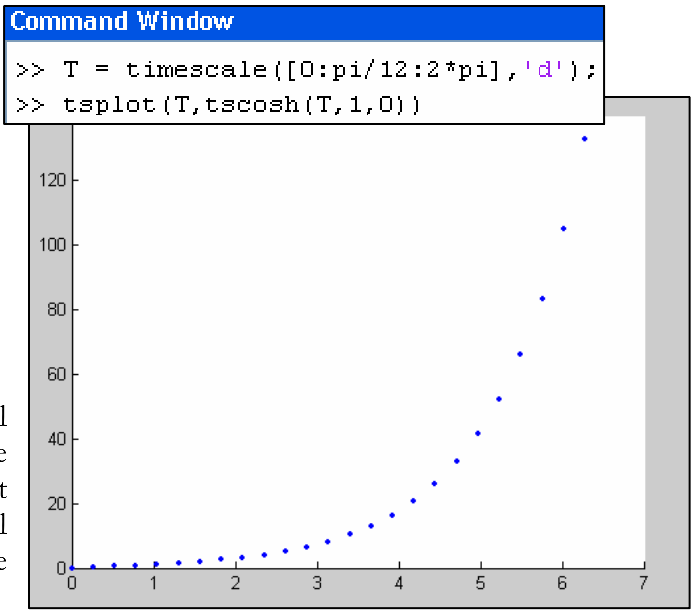
TSCOSH (T, α, t0) returns a time scale hyperbolic sine of T.

This function calculates the time scale sine using the formula, $\cosh_p = \dfrac{e_p + e_{-p}}{2}$.

Example: Entering the discrete timescale at right, and plotting the timescale hyperbolic cosine against the original timescale yields the exponential curve to the right.

When considering the formula used to calculate the time scale hyperbolic cosine, the plotted graph is easy to understand.

The initial value input is an optional input that must be contained in the time scale. If the initial value is not the first point of the time scale, the function will fill every point preceding the initial value with zeros.


Command Window

```
>> T = timescale([0:pi/12:2*pi],'d');
>> tsplot(T,tscosh(T,1,0))
```

tssin **Calling Syntax:** Y = tssin(T, alpha, [t_0])

Inputs:

T: time scale object

α: scalar exponent

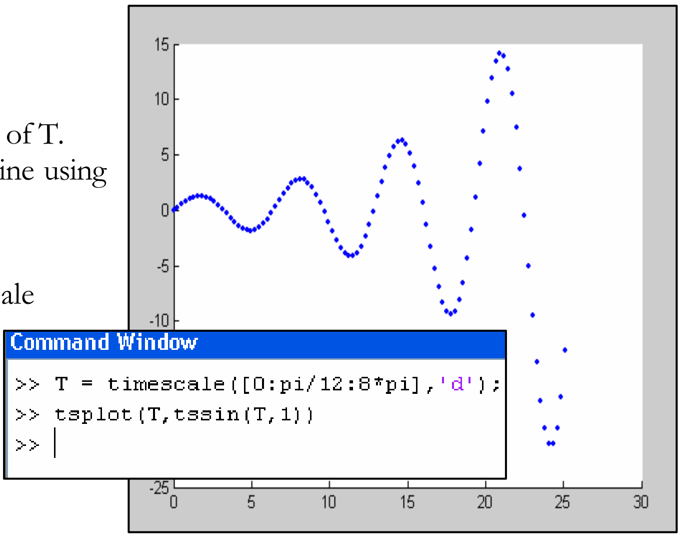
t_0: initial value (optional)

Output:

Y: time scale object containing the solution to the time scale sine

TSSIN (T, α, t0) returns a time scale sine of T. This function calculates the time scale sine using the formula: $\sin_p = \dfrac{e_{ip} - e_{-ip}}{2i}$.

Example: Entering the discrete time scale interval to the right, performing the sine function, and plotting the results produces the following graph at right. The amplitude is dependent on μp^2, so in discrete calculations the amplitude increases with μ.

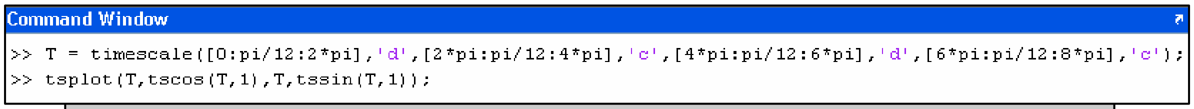

Command Window

```
>> T = timescale([0:pi/12:8*pi],'d');
>> tsplot(T,tssin(T,1))
>> |
```

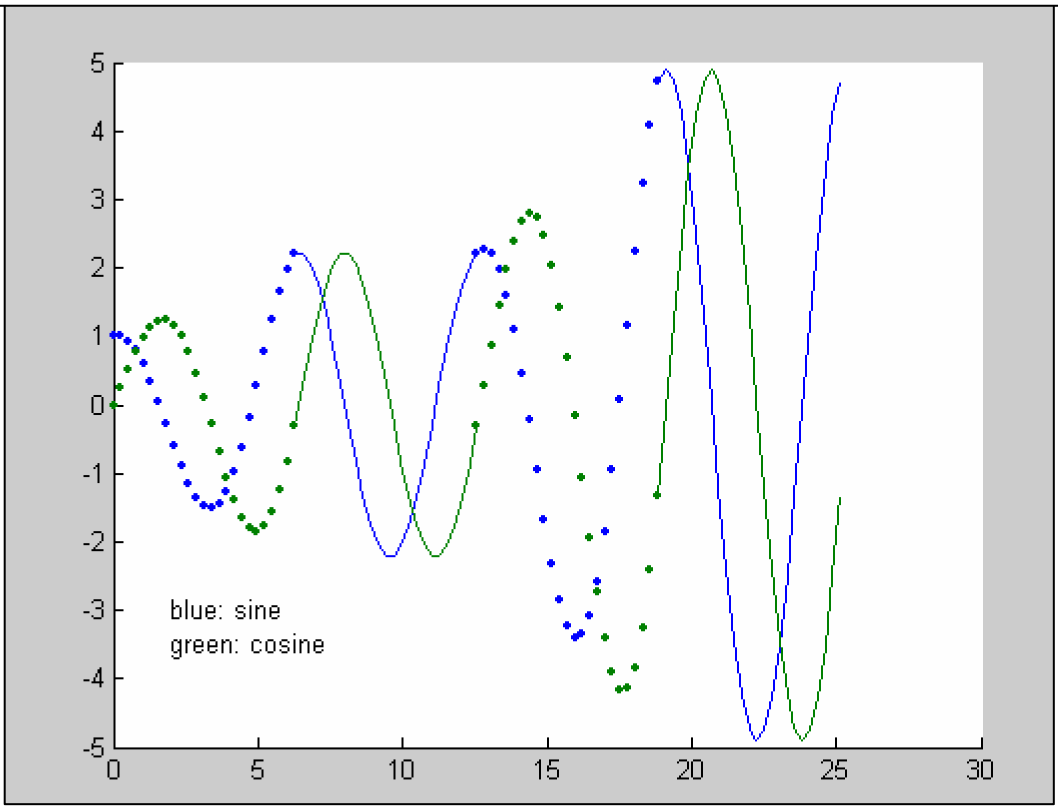
The function discards imaginary values and returns only real valued results.

Entering a continuous time scale interval into the tssin function generates a typical sine wave on real numbers.

The following multiple interval, multiple type time scale and plotting the solutions against one another will yield the graph below.



```
>> T = timescale([0:pi/12:2*pi],'d',[2*pi:pi/12:4*pi],'c',[4*pi:pi/12:6*pi],'d',[6*pi:pi/12:8*pi],'c');
>> tsplot(T,tscos(T,1),T,tssin(T,1));
```



tssinh **Calling Syntax:** Y = tssinh(T, alpha, [t_0])

Inputs:

T: time scale object

α: scalar exponent

t_0: initial value (optional)

Output:

Y: time scale object containing the solution to the time scale hyperbolic sine

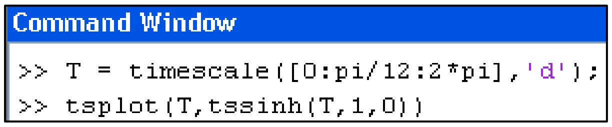
TSSINH (T, α, t0) returns a time scale hyperbolic sine of T.

This function calculates the time scale sine using the formula, $\sinh_p = \frac{e_p - e_{-p}}{2}$.

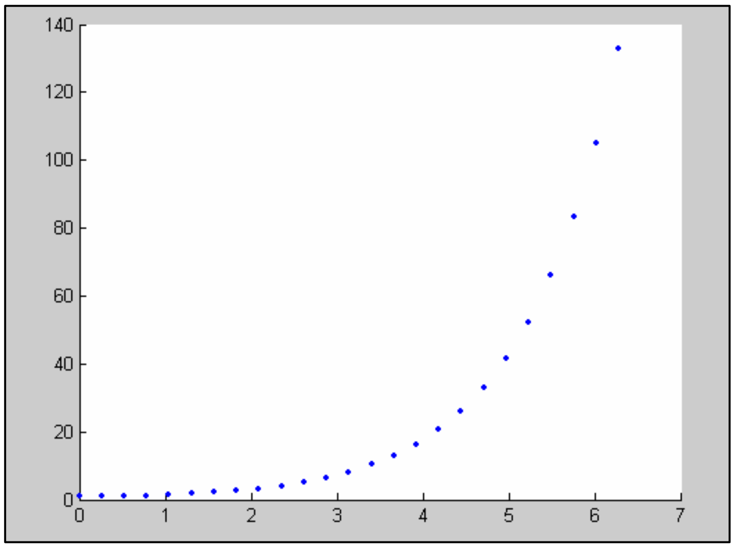
Example: Entering the discrete timescale at below, and plotting the timescale hyperbolic sine against the original timescale yields the exponential curve below.

When considering the formula used to calculate the time scale hyperbolic sine, the plotted graph is easy to understand.

The initial value input is an optional input that must be contained in the time scale. If the initial value is not the first point of the time scale, the function will fill every point preceding the initial value with zeros.



```
>> T = timescale([0:pi/12:2*pi],'d');
>> tsplot(T,tssinh(T,1,0))
```



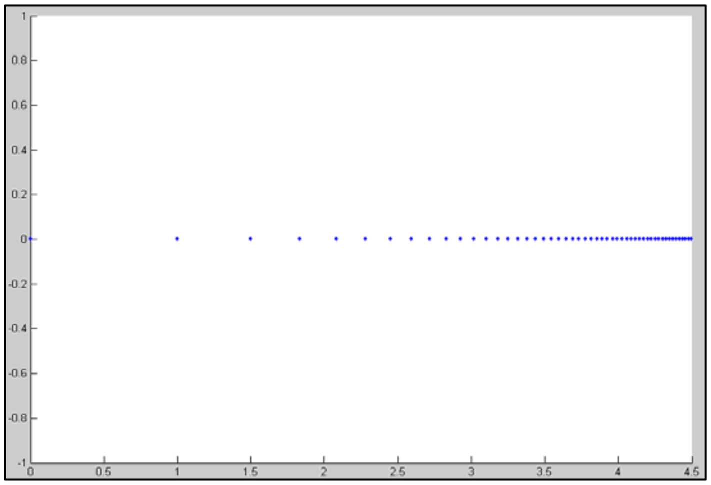
Example Time Scale Functions

Example functions were written to create example time scales which are useful. These are time scales commonly found in timescale calculus, hence, frequently used in testing and calculations.

tsharmonic **Calling Syntax:** Y = tsharmonic (n)

TSHARMONIC (N) is a harmonic time scale with N points. The commands and plot are illustrated below for time scale

T = tsharmonic (50); tsplot (T)



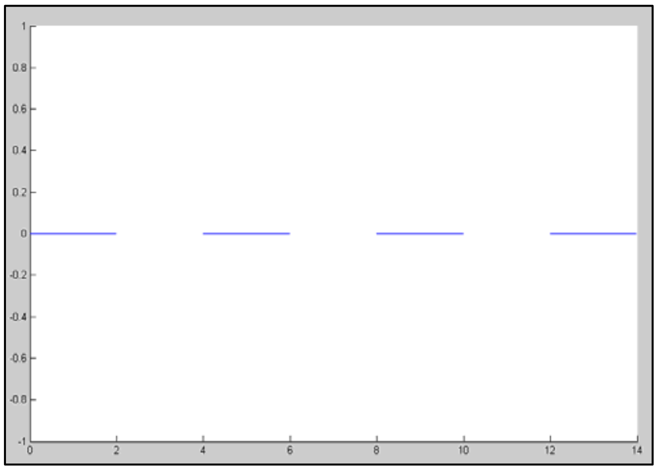
tspab **Calling Syntax:** Y = tspab(a, x, b, n)

Inputs:

- a: length of intervals
- x: graininess of the cont. interval
- b: length of jump
- n: number of a-b pairs

TSPAB (A, X, B, N) returns a time scale of the form P (a, b). P (a, b) contains N continuous intervals of length A separated by a jump of length B. X is the graininess of the continuous interval.

For example, consider:
T = tspab (2, 1, 2, 4)



tsqz **Calling syntax:** Y = tsqz(q, a, b, h)

Inputs:

q: scalar

a: initial value in Z

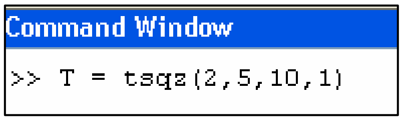
b: final value in Z

h: (optional) graininess

TSQZ (Q, A, B, H) returns a time scale object of the form q^{hZ} where Z is the set of all integers and h is the graininess. Z goes from A to B. H is an optional input; when H is not entered, its default value is 1. The timescale

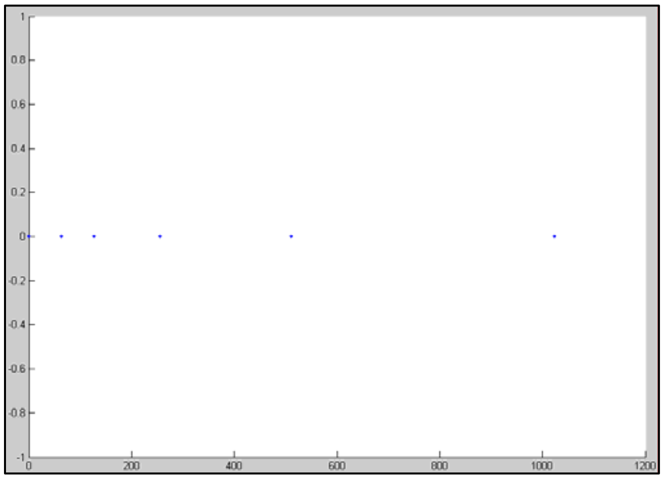
T = tsqz (2, 5, 10, 1)

is purely discrete, yielding the plot to the right.



Command Window

```
>> T = tsqz(2,5,10,1)
```



Miscellaneous Time Scale Functions

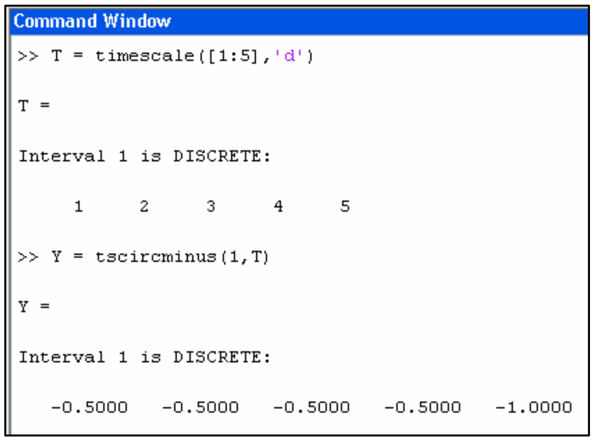
These functions prove to be useful to use inside other functions.

tscircminus **Calling Syntax:** Y = tscircminus(z , T)

TSCIRCMINUS (z , T) returns a timescale object containing the solution to the circle minus operation of a scalar and a given time scale. The function uses the equation from Bohner-Peterson:

$$\Theta z = \frac{-z}{1+\mu(t)z}.$$

The function finds the graininess of the time scale and uses it to perform calculations. The last value of the timescale is based on a graininess of 0 and simply returns a $-z$.


Command Window

```
>> T = timescale([1:5],'d')

T =

Interval 1 is DISCRETE:

     1     2     3     4     5

>> Y = tscircminus(1,T)

Y =

Interval 1 is DISCRETE:

   -0.5000   -0.5000   -0.5000   -0.5000   -1.0000
```

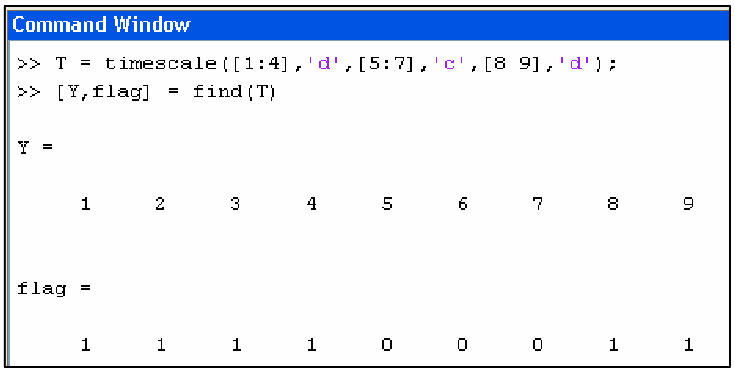
tsfind **Calling Syntax:** Y = tsfind(T)

TSFIND (T) returns instances of nonzero entries in the time scale object T, such that T(I) returns the nonzero values. The optional flag output indicates whether the corresponding indices point to a continuous (FLAG = 0) or discrete (FLAG = 1) interval. TSFIND returns empty if there are no nonzero entries in T.

For example, consider

T = timescale([1:4],'c',[4.1:0.1:4.5],'d', [10:1:13],'c')

[I, flag] = find(T)


Command Window

```
>> T = timescale([1:4],'d',[5:7],'c',[8 9],'d');
>> [Y,flag] = find(T)

Y =

     1     2     3     4     5     6     7     8     9


flag =

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

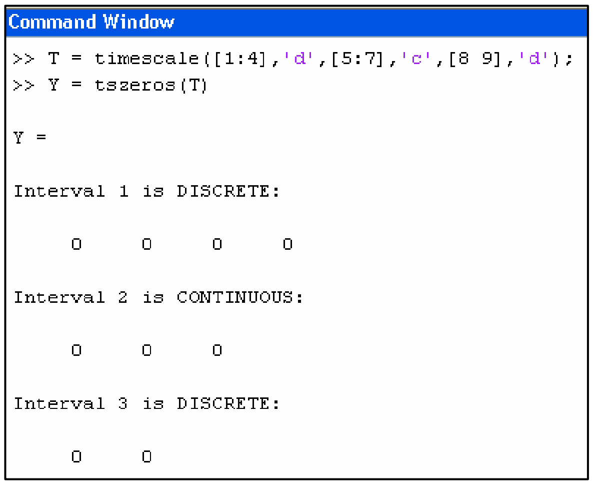
tszeros **Calling syntax:** Y = tszeros(T)

TSZEROS (T) requires an input time scale T and returns a time scale exactly like T, but filled with zeros instead of the t values of T.

For example:

T = timescale([1:4], 'd',[5:7], 'c', [8 9], 'd')

Y = tszeros(T)



Command Window

```
>> T = timescale([1:4],'d',[5:7],'c',[8 9],'d');
>> Y = tszeros(T)

Y =

Interval 1 is DISCRETE:

     0     0     0     0

Interval 2 is CONTINUOUS:

     0     0     0

Interval 3 is DISCRETE:

     0     0
```

Index