Chrono for Octave

version 0.1.4, January 2019

Andrew Janke

This manual is for Chrono, version 0.1.4.

Table of Contents

1 Introduction

> Time is an illusion. Lunchtime doubly so.
>
> —*Douglas Adams*

This is the manual for the Chrono package version 0.1.4 for GNU Octave.

This document is a work in progress. You are invited to help improve it and submit patches.

Chrono provides date/time functionality for Octave by supplying Matlab-compatible implementations for the `datetime`, `duration`, and `calendarDuration` classes, along with related functions.

Chrono's classes are designed to be convenient to use while still being efficient. The data representations used by Chrono are designed to be efficient and suitable for working with large-ish data sets. A "large-ish" data set is one that can have millions of elements or rows, but still fits in main computer memory. Chrono's main relational and arithmetic operations are all implemented using vectorized operations on primitive Octave data types.

Chrono was written by Andrew Janke `<floss@apjanke.net>`. Support can be found on the Chrono project GitHub page (`https://github.com/apjanke/octave-chrono`).

2 Getting Started

The easiest way to obtain Chrono is by using Octave's `pkg` package manager. To install the latest development (unstable) head of Chrono, run this in Octave:

```
pkg install https://github.com/apjanke/octave-chrono/archive/master.zip
```

Soon, once the code gets a bit more stable, numbered releases for Chrono will be available, and you should use those instead.

For development, you can obtain the source code for Chrono from the project repo on GitHub at `https://github.com/apjanke/octave-chrono`. Upon first installation, run the `octave_chrono_make_local` script to build the octfiles so Chrono will work. Then add the `inst` directory in the repo to your Octave path.

3 Date Representation

Chrono provides the ‘datetime‘ class for representing points in time.

3.1 `datetime` Class

A `datetime` is an array object that represents points in time in the familiar Gregorian calendar.

This is an attempt to reproduce the functionality of Matlab's `datetime`. It also contains some Octave-specific extensions.

The underlying representation is that of a datenum (a `double` containing the number of days since the Matlab epoch), but encapsulating it in an object provides several benefits: friendly human-readable display, type safety, automatic type conversion, and time zone support. In addition to the underlying datenum array, a `datetime` inclues an optional `TimeZone` property indicating what time zone the datetimes are in.

3.1.1 `datenum` Compatibility

While the underlying data representation of `datetime` is compatible with (in fact, identical to) that of datenums, you cannot directly combine them via assignment, concatenation, or most arithmetic operations.

This is because of the signature of the `datetime` constructor. When combining objects and primitive types like `double`, the primitive type is promoted to an object by calling the other object's one-argument constructor on it. However, the one-argument numeric-input consstructor for `datetime` does not accept datenums: it interprets its input as datevecs instead. This is due to a design decision on Matlab's part; for compatibility, Octave does not alter that interface.

To combine `datetime`s with datenums, you can convert the datenums to `datetime`s by calling `datetime.ofDatenum` or `datetime(x, 'ConvertFrom', 'datenum')`, or you can convert the `datetime`s to datenums by accessing its `dnums` field with `x.dnums`.

Examples:

```
dt = datetime('2011-03-04')
dn = datenum('2017-01-01')
[dt dn]
    ⇒ error: datenum: expected date vector containing [YEAR, MONTH, DAY, HOUR, MINUTE
[dt datetime.ofDatenum(dn)]
    ⇒ 04-Mar-2011   01-Jan-2017
```

4 Time Zones

Chrono has support for representing dates in time zones and for converting between time zones.

A `datetime` may be "zoned" or "zoneless". A zoneless `datetime` does not have a time zone associated with it. This is represented by an empty `TimeZone` property on the `datetime` object. A zoneless `datetime` represents the local time in some unknown time zone, and assumes a continuous time scale (no DST shifts).

A zoned `datetime` is associated with a time zone. It is represented by having the time zone's IANA zone identifier (e.g. `'UTC'` or `'America/New_York'`) in its `TimeZone` property. A zoned `datetime` represents the local time in that time zone.

By default, the `datetime` constructor creates unzoned `datetime`s. To make a zoned `datetime`, either pass the `'TimeZone'` option to the constructor, or set the `TimeZone` property after object creation. Setting the `TimeZone` property on a zoneless `datetime` declares that it's a local time in that time zone. Setting the `TimeZone` property on a zoned `datetime` turns it back into a zoneless `datetime` without changing the local time it represents.

You can tell a zoned from a zoneless time zone in the object display because the time zone is included for zoned `datetime`s.

```
% Create an unzoned datetime
d = datetime('2011-03-04 06:00:00')
    ⇒  04-Mar-2011 06:00:00

% Create a zoned datetime
d_ny = datetime('2011-03-04 06:00:00', 'TimeZone', 'America/New_York')
    ⇒  04-Mar-2011 06:00:00 America/New_York
% This is equivalent
d_ny = datetime('2011-03-04 06:00:00');
d_ny.TimeZone = 'America/New_York'
    ⇒  04-Mar-2011 06:00:00 America/New_York

% Convert it to Chicago time
d_chi.TimeZone = 'America/Chicago'
    ⇒  04-Mar-2011 05:00:00 America/Chicago
```

When you combine two zoned `datetime`s via concatenation, assignment, or arithmetic, if their time zones differ, they are converted to the time zone of the left-hand input.

```
d_ny = datetime('2011-03-04 06:00:00', 'TimeZone', 'America/New_York')
d_la = datetime('2011-03-04 06:00:00', 'TimeZone', 'America/Los_Angeles')
d_la - d_ny
    ⇒ 03:00:00
```

You cannot combine a zoned and an unzoned `datetime`. This results in an error being raised.

> **Warning:** Normalization of "nonexistent" times (like between 02:00 and 03:00 on a "spring forward" DST change day) is not implemented yet. The results of converting a zoneless local time into a time zone where that local time did not exist are currently undefined.

4.1 Defined Time Zones

Chrono's time zone data is drawn from the IANA Time Zone Database (`https://www.iana.org/time-zones`), also known as the "Olson Database". Chrono includes a copy of this database in its distribution so it can work on Windows, which does not supply it like Unix systems do.

You can use the `timezones` function to list the time zones known to Chrono. These will be all the time zones in the IANA database on your system (for Linux and macOS) or in the IANA time zone database redistributed with Chrono (for Windows).

> **Note:** The IANA Time Zone Database only covers dates from about the year 1880 to 2038. Converting time zones for `datetime`s outside that range is currently unimplemented. (Chrono needs to add support for proleptic POSIX time zone rules, which are used to govern behavior outside that date range.)

5 Durations

5.1 `duration` Class

A `duration` represents a period of time in fixed-length seconds (or minutes, hours, or whatever you want to measure it in.)

A `duration` has a resolution of about a nanosecond for typical dates. The underlying representation is a `double` representing the number of days elapsed, similar to a datenum, except it's interpreted as relative to some other reference point you provide, instead of being relative to the Matlab/Octave epoch.

You can add or subtract a `duration` to a `datetime` to get another `datetime`. You can also add or subtract `durations` to each other.

5.2 `calendarDuration` Class

A `calendarDuration` represents a period of time in variable-length calendar components. For example, years and months can have varying numbers of days, and days in time zones with Daylight Saving Time have varying numbers of hours. A `calendarDuration` does arithmetic with "whole" calendar periods.

`calendarDuration`s and `duration`s cannot be directly combined, because they are not semantically equivalent. (This may be relaxed in the future to allow `duration`s to be interpreted as numbers of days when combined with `calendarDuration`s.)

```
d = datetime('2011-03-04 00:00:00')
    ⇒ 04-Mar-2011
cdur = calendarDuration(1, 3, 0)
    ⇒ 1y 3mo
d2 = d + cdur
    ⇒ 04-Jun-2012
```

6 Missing Functionality

Chrono is based on Matlab's date/time API and supports most of its major functionality. But not all of it is implemented yet. The missing parts are currently:

- POSIX time zone support for years outside the IANA time zone database coverage
- Week-of-year (ISO calendar) calculations
- Various `'ConvertFrom'` forms for `datetime` and `duration`
- Support for LDML formatting for `datetime`
- Various functions: `between`, `caldiff`, `dateshift`, `week`
- `isdst`, `isweekend`
- `calendarDuration.split`
- `duration.Format` support
- `UTCOffset` and `DSTOffset` fields in the output of `timezones()`
- Plotting support

It is the author's hope that all these will be implemented some day.

7 Function Reference

7.1 Functions by Category

7.1.1 Date Representation

7.1.2 Durations

7.2 Functions Alphabetically

7.2.1 calendarDuration

`calendarDuration` [Class]

Durations in time using variable-length calendar periods, such as days, months, and years, which may vary in length over time. (For example, a calendar month may have 28, 30, or 31 days.)

obj = `calendarDuration ()` [Constructor]

Constructs a new scalar `calendarDuration` of zero elapsed time.

obj = `calendarDuration (`*Y*, *M*, *D*`)` [Constructor]
obj = `calendarDuration (`*Y*, *M*, *D*, *H*, *MI*, *S*`)` [Constructor]

Constructs new `calendarDuration` arrays based on input values.

7.2.2 calmonths

out = `calmonths (`*x*`)` [Function File]

Create a `calendarDuration` that is a given number of calendar months long.

Input *x* is a numeric array specifying the number of calendar months.

This is a shorthand alternative to calling the `calendarDuration` constructor with `calendarDuration(0, x, 0)`.

Returns a new `calendarDuration` object of the same size as *x*.

See Section 7.2.1 [calendarDuration], page 9.

7.2.3 calyears

out = `calyears (`*x*`)` [Function]

Construct a `calendarDuration` a given number of years long.

This is a shorthand for calling `calendarDuration(`*x*`, 0, 0)`.

See Section 7.2.1 [calendarDuration], page 9.

7.2.4 datetime

`datetime` [Class]

Represents points in time using the Gregorian calendar.

The underlying values are doubles representing the number of days since the Matlab epoch of "January 0, year 0". This has a precision of around nanoseconds for typical times.

`double dnums` [Instance Variable of `datetime`]

The underlying datenums that represent the points in time.

`char TimeZone` [Instance Variable of `datetime`]

The time zone this `datetime` array is in. Empty if this does not have a time zone associated with it ("unzoned"). The name of an IANA time zone if this does.

`char Format` [Instance Variable of `datetime`]
The format to display this `datetime` in. Currently unsupported.

obj = `datetime ()` [Constructor]
Constructs a new scalar `datetime` containing the current local time, with no time zone attached.

obj = `datetime (`*datevec*`)` [Constructor]
obj = `datetime (`*datestrs*`)` [Constructor]
obj = `datetime (`*in*`, 'ConvertFrom', `*inType*`)` [Constructor]
obj = `datetime (`*Y*`, `*M*`, `*D*`, `*H*`, `*MI*`, `*S*`)` [Constructor]
obj = `datetime (`*Y*`, `*M*`, `*D*`, `*H*`, `*MI*`, `*MS*`)` [Constructor]
obj = `datetime (...,` *'Format'*, *Format*, *'InputFormat'*, *InputFormat*, *'Locale'*, *InputLocale*, *'PivotYear'*, *PivotYear*, *'TimeZone'*, *TimeZone*`)` [Constructor]
Constructs a new `datetime` array based on input values.

7.2.5 days

out = `days (`*x*`)` [Function]
Duration in days.

If *x* is numeric, then *out* is a `duration` array in units of fixed-length 24-hour days, with the same size as *x*.

If *x* is a `duration`, then returns a `double` array the same size as *x* indicating the number of fixed-length days that each duration is.

7.2.6 duration

`duration` [Class]
Represents durations or periods of time as an amount of fixed-length time (i.e. fixed-length seconds). It does not care about calendar things like months and days that vary in length over time.

This is an attempt to reproduce the functionality of Matlab's `duration`. It also contains some Octave-specific extensions.

obj = `duration ()` [Constructor]
Constructs a new scalar `duration` of zero elapsed time.

obj = `duration (`*durationstrs*`)` [Constructor]
obj = `duration (`*durationstrs*`, 'InputFormat', `*InputFormat*`)` [Constructor]
obj = `duration (`*H*`, `*MI*`, `*S*`)` [Constructor]
obj = `duration (`*H*`, `*MI*`, `*S*`, `*MS*`)` [Constructor]
Constructs a new `duration` array based on input values.

7.2.7 hours

out = `hours (`*x*`)` [Function File]
Create a `duration` *x* hours long, or get the hours in a `duration` *x*.

If input is numeric, returns a `duration` array that is that many hours in time.

If input is a `duration`, converts the `duration` to a number of hours.

Returns an array the same size as *x*.

7.2.8 isdatetime

tf = `isdatetime` (*x*) [Function]

True if input is a `datetime` array, false otherwise.

Returns a logical array the same size as *x*.

7.2.9 isduration

tf = `isduration` (*x*) [Function]

True if input is a `duration` array, false otherwise.

Returns a logical array the same size as *x*.

7.2.10 milliseconds

out = `milliseconds` (*x*) [Function File]

Create a `duration` *x* milliseconds long, or get the milliseconds in a `duration` *x*.

If input is numeric, returns a `duration` array that is that many milliseconds in time.

If input is a `duration`, converts the `duration` to a number of milliseconds.

Returns an array the same size as *x*.

7.2.11 minutes

out = `hours` (*x*) [Function File]

Create a `duration` *x* hours long, or get the hours in a `duration` *x*.

7.2.12 NaT

out = `NaT` () [Function]

out = `NaT` (*sz*) [Function]

"Not-a-Time". Constructs a new `datetime` array of all `NaT` values of the given size. If no input *sz* is given, the result is a scalar `NaT`.

`NaT` is the `datetime` equivalent of `NaN`. It represents a missing or invalid value. `NaT` values never compare equal to, greater than, or less than any value, including other `NaT`s. Doing arithmetic with a `NaT` and any other value results in a `NaT`.

7.2.13 seconds

out = `seconds` (*x*) [Function File]

Create a `duration` *x* seconds long, or get the seconds in a `duration` *x*.

If input is numeric, returns a `duration` array that is that many seconds in time.

If input is a `duration`, converts the `duration` to a number of seconds.

Returns an array the same size as *x*.

7.2.14 timezones

out = `timezones` () [Function]
out = `timezones` (*area*) [Function]

List all the time zones defined on this system.

This lists all the time zones that are defined in the IANA time zone database used by this Octave. (On Linux and macOS, that will generally be the system time zone database from `/usr/share/zoneinfo`. On Windows, it will be the database redistributed with the Chrono package.

If the return is captured, the output is returned as a table if your Octave has table support, or a struct if it does not. It will have fields/variables containing column vectors:

`Name` The IANA zone name, as cellstr.

`Area` The geographical area the zone is in, as cellstr.

Compatibility note: Matlab also includes UTCOffset and DSTOffset fields in the output; these are currently unimplemented.

7.2.15 years

out = `years` (*x*) [Function File]

Create a `duration` *x* years long, or get the years in a `duration` *x*.

If input is numeric, returns a `duration` array in units of fixed-length years of 365.2425 days each.

If input is a `duration`, converts the `duration` to a number of fixed-length years as double.

Note: `years` creates fixed-length years, which may not be what you want. To create a duration of calendar years (which account for actual leap days), use `calyears`.

See Section 7.2.3 [calyears], page 9.

8 Copying

8.1 Package Copyright

Chrono for Octave is covered by the GNU GPLv3, the Unicode License, and Public Domain.

All the code in the package is GNU GPLv3.

The IANA Time Zone Database redistributed with the package is Public Domain.

The Windows Zones file redistributed with the package is covered by the Unicode License (`http://www.unicode.org/copyright.html`).

8.2 Manual Copyright

This manual is for Chrono, version 0.1.4.