Title

stata.com

Datetime durations — Obtaining and working with durations

Description

This entry describes functions that calculate durations, such as the number of years between two dates (for example, a person's age). These functions account for leap years and leap days and produce results that are more consistent than simply taking arithmetic differences of numerical dates and converting to another unit.

This entry also describes functions that convert durations from one unit (for example, milliseconds) to another (for example, hours).

Quick start

Calculate age of a subject in integer years on the date of a survey based on a numerically encoded Stata date `dob` that gives the subject's date of birth and a numerically encoded Stata date `date_of_survey`

```
generate subject_age = age(dob, date_of_survey)
```

As above, but calculate the age as a noninteger; that is, include the fractional part

```
generate subject_fage = age_frac(dob, date_of_survey)
```

Calculate age on date `d` for persons born on 29feb as having their birthday on 28feb in nonleap years (rather than the default of 01mar)

```
generate celebrate = age(dob, d, "28feb")
```

Calculate the difference in number of months, rounded down to an integer, between two Stata dates, `d1` and `d2`

```
generate diff_months = datediff(d1, d2, "month")
```

As above, but include the fractional part of the difference

```
generate diff_fmonths = datediff_frac(d1, d2, "month")
```

Calculate the difference in number of hours, rounded down to an integer, between two Stata datetime/c variables, `t1` and `t2`

```
generate diff_hours = clockdiff(t1, t2, "hour")
```

As above, but include the fractional part of the difference

```
generate diff_fhours = clockdiff_frac(t1, t2, "hour")
```

As above, but use a conversion function to calculate hours with a fractional part

```
generate diff_fhours2 = hours(t2 - t1)
```

Calculate the difference in number of minutes, rounded down to an integer, between two Stata datetime/C variables, `tvar1` and `tvar2`

```
generate diff_minutes = Clockdiff(tvar1, tvar2, "minute")
```

Syntax

Syntax is presented under the following headings:

Functions for calculating durations

Description	Function	Value returned
age	`age(`$e_{d\,\mathrm{DOB}}$`,`e_d`[ ,`s_{nl}`])`	years rounded down to an integer
age with fraction	`age_frac(`$e_{d\,\mathrm{DOB}}$`,`e_d`[ ,`s_{nl}`])`	years with fractional part
datetime/C difference	`Clockdiff(`e_{tC1}`,`e_{tC2}`,`s_{tu}`)`	integer (rounded down)
datetime/c difference	`clockdiff(`e_{tc1}`,`e_{tc2}`,`s_{tu}`)`	integer (rounded down)
datetime/C difference with fraction	`Clockdiff_frac(`e_{tC1}`,`e_{tC2}`,`s_{tu}`)`	floating point
datetime/c difference with fraction	`clockdiff_frac(`e_{tc1}`,`e_{tc2}`,`s_{tu}`)`	floating point
date difference	`datediff(`e_{d1}`,`e_{d2}`,`s_{du}`[ ,`s_{nl}`])`	integer (rounded down)
date difference with fraction	`datediff_frac(`e_{d1}`,`e_{d2}`,`s_{du}`[ ,`s_{nl}`])`	floating point

e_d, $e_{d\,\mathrm{DOB}}$, e_{d1}, and e_{d2} are Stata dates.
e_{tC1} and e_{tC2} are Stata datetime/C values.
e_{tc1} and e_{tc2} are Stata datetime/c values.
s_{nl} is a string specifying nonleap-year birthdays or anniversaries of 29feb and may be
`"01mar"`, `"1mar"`, `"mar01"`, or `"mar1"` (the default); or
`"28feb"` or `"feb28"` (case insensitive).
s_{tu} is a string specifying time units:
`"day"` or `"d"` for day;
`"hour"` or `"h"` for hour;
`"minute"`, `"min"`, or `"m"` for minute;
`"second"`, `"sec"`, or `"s"` for second; or
`"millisecond"` or `"ms"` for millisecond (case insensitive).
s_{du} is a string specifying date units:
`"day"` or `"d"` for day;
`"month"`, `"mon"`, or `"m"` for month; or
`"year"` or `"y"` for year (case insensitive).

Notes:

1. The string s_{nl} specifying nonleap-year birthdays or anniversaries is an optional argument. It rarely needs to be specified. See example 3 below.
2. When $e_d < e_{d\,\mathrm{DOB}}$, `age(`$e_{d\,\mathrm{DOB}}$`,`$e_d$`[ ,`$s_{nl}$`])` and `age_frac(`$e_{d\,\mathrm{DOB}}$`,`$e_d$`[ ,`$s_{nl}$`])` return *missing* (`.`).
3. `Clockdiff(`e_{tC1}`,`e_{tC2}`,`s_{tu}`)` $= -$`Clockdiff(`e_{tC2}`,`e_{tC1}`,`s_{tu}`)`. `clockdiff()`, `Clockdiff_frac()`, `clockdiff_frac()`, `datediff()`, and `datediff_frac()` have the same anticommutative property.

Functions for converting units of a duration

Desired conversion	Function	Value returned
milliseconds to hours	`hours(`*ms*`)`	$ms/(60 \times 60 \times 1000)$
milliseconds to minutes	`minutes(`*ms*`)`	$ms/(60 \times 1000)$
milliseconds to seconds	`seconds(`*ms*`)`	$ms/1000$
hours to milliseconds	`msofhours(`*h*`)`*	$h \times 60 \times 60 \times 1000$
minutes to milliseconds	`msofminutes(`*m*`)`*	$m \times 60 \times 1000$
seconds to milliseconds	`msofseconds(`*s*`)`*	$s \times 1000$

* Stata datetime values are in milliseconds and must be stored as `double`s. When using millisecond results to add to or subtract from a Stata datetime, store the results as `double`s.

Remarks and examples

stata.com

Remarks are presented under the following headings:

We assume you have read [D] **Datetime** and are familiar with how Stata stores dates and datetimes. String dates and times must be converted into numeric values to become Stata dates and datetimes. Stata date and time values are durations (positive or negative) from 01jan1960. Stata date values record the number of days from 01jan1960. Stata datetime/c values record the number of milliseconds from 01jan1960 00:00:00. Stata datetime/C is the same as datetime/c, except that it accounts for leap seconds and encodes Coordinated Universal Time (UTC).

There are other types of Stata date and time values, ones for weeks, months, quarters, half years, and years, but the functions described here are intended for use with daily dates or datetimes.

Calculating ages and differences of dates

The `age()` function calculates age just as one would expect. Typing

```
. generate subject_age = age(date_of_birth, current_date)
```

produces integers that are a person's age in years on `current_date` given birthdate `date_of_birth`. The variables `date_of_birth` and `current_date` must be Stata dates.

The arguments of `age()` need not be variables, but they must be Stata date values, which are numeric. To get Stata date values for literal dates, we can use the date pseudofunction `td()` and use its results as arguments to `age()`. For example,

```
. display age(td(05feb1927), td(24may2006))
79
```

shows that an individual born on 05feb1927 was 79 years old on 24may2006.

`age_frac()` returns age including the fractional part. For example, let's use `age_frac()` with the dates we specified above:

```
. display age_frac(td(05feb1927), td(24may2006))
79.29589
```

The `datediff()` and `datediff_frac()` functions produce results in units of years, months, or days. For example, to determine the number of months between 05feb1927 and 24may2006, first as an integer (rounded down) and as a number including the fractional part, we type

```
. display datediff(td(05feb1927), td(24may2006), "month")
951
. display datediff_frac(td(05feb1927), td(24may2006), "month")
951.6129
```

The optional last argument, s_{nl}, for `age()`, `age_frac()`, `datediff()`, and `datediff_frac()` was not specified in any of the above examples. It applies only to a date of birth (or starting date) on 29feb when the ending date is not in a leap year. The argument controls whether to use 01mar (the default) or 28feb as the birthday (or anniversary) in nonleap years. Setting this argument is important only when the data you are using have a set rule for determining the age of persons born on 29feb. For example, you might have data on the dates when people first get their driver's licenses. You would want the argument to match the legal rule for the data. See example 3.

The functions `age()` and `age_frac()` are based on `datediff()` and `datediff_frac()`, respectively,

$$\texttt{age}(e_{d\,\mathrm{DOB}},e_d,s_{nl}) = \texttt{datediff}(e_{d\,\mathrm{DOB}},e_d,\texttt{"year"},s_{nl})$$

and

$$\texttt{age\_frac}(e_{d\,\mathrm{DOB}},e_d,s_{nl}) = \texttt{datediff\_frac}(e_{d\,\mathrm{DOB}},e_d,\texttt{"year"},s_{nl})$$

when $e_d \geq e_{d\,\mathrm{DOB}}$. When $e_d < e_{d\,\mathrm{DOB}}$, `age()` and `age_frac()` return *missing* (`.`).

`datediff(...,"year",...)` and `datediff_frac(...,"year",...)` calculate the number of years between two dates just as one would expect. The only wrinkles are leap days and leap years. See *Methods and formulas* in [FN] **Date and time functions** for details.

The usefulness of these functions is solely in the way they handle leap days and leap years. Suppose, for example, you are doing an analysis of age of onset of some disorder. If you use values from `age_frac()` as time in a survival model, these times will match up perfectly with recorded ages (or ages from `age()` of course). If instead you used

```
. generate time_years = (onset_date - date_of_birth)/365.25
```

as your time variable, there would be minor discrepancies between this time and ages at birthdays. See examples below.

`datediff(...,"month",...)` and `datediff_frac(...,"month",...)` calculate the number of months between two dates as one would expect for starting days 1–28. For example, a starting date on the 28th of the month will have month anniversaries on the 28th of all other months. When the day of the starting date is 29, 30, or 31, other months may not have this day of the month. The last day of February will be 28 or 29. When the starting date is on the 31st, the months ending on the 30th obviously do not have a 31st. In these cases, the first day of the next month is considered the month anniversary. (This is consistent with the default handling of 29feb start dates when calculating year anniversaries in nonleap years; the nonleap year anniversaries are on 01mar.)

Fractional months are also a bit tricky because lengths of months vary. There is an example below, and see *Methods and formulas* in [FN] **Date and time functions** for how they are calculated.

Note that `datediff(...,"year",...)`, `datediff_frac(...,"year",...)`, `datediff(...,"month",...)`, and `datediff_frac(...,"month",...)` all match up. That is, on an ending date on which `datediff(...,"year",...)` increases by one from the previous day, the value of `datediff_frac(...,"year",...)` is exactly an integer and equal to `datediff(...,"year",...)`. On this ending date, `datediff_frac(...,"month",...)` is also an integer and equal to 12 times the year difference.

`datediff(`e_{d1}`,`e_{d2}`,"day",`s_{nl}`)` and `datediff_frac(`e_{d1}`,`e_{d2}`,"day",`s_{nl}`)` have no complications in how they are calculated. Both are equal to $e_{d2} - e_{d1}$ and are always integers. The optional argument s_{nl} has no bearing on the calculation and is ignored if specified.

▷ Example 1: Ages

Calculating ages is straightforward, but we do need to show how `age_frac()` calculates the fractional part of age. Here is an example.

We have a dataset with string dates. Date of birth is recorded in the variable `str_dob`, and the end date for calculating age is in `str_end_date`.

```
. use https://www.stata-press.com/data/r17/ages
(Fictional data for calculating ages)
. describe
Contains data from https://www.stata-press.com/data/r17/ages.dta
 Observations:             5                  Fictional data for calculating
                                                ages
    Variables:             2                  30 Oct 2020 17:35
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
str_dob         str9    %9s                   Date of birth
str_end_date    str9    %9s                   End date
-------------------------------------------------------------------------------
Sorted by:
. list, abbreviate(12)
```

	str_dob	str_end_date
1.	28/8/1967	27/8/2019
2.	28/8/1967	28/8/2019
3.	28/8/1967	29/8/2019
4.	28/8/1967	28/8/2020
5.	28/8/1967	29/8/2020

We must convert the strings to numeric Stata dates, which we do using the `date()` function with a mask of `"DMY"` because the date components are in the order day, month, year. We format the new encoded date variables using format `%td`, the simplest format specification for daily dates.

```
. generate dob = date(str_dob, "DMY")
. generate end_date = date(str_end_date, "DMY")
. format dob end_date %td
. list str_dob dob str_end_date end_date, abbreviate(12)
```

	str_dob	dob	str_end_date	end_date
1.	28/8/1967	28aug1967	27/8/2019	27aug2019
2.	28/8/1967	28aug1967	28/8/2019	28aug2019
3.	28/8/1967	28aug1967	29/8/2019	29aug2019
4.	28/8/1967	28aug1967	28/8/2020	28aug2020
5.	28/8/1967	28aug1967	29/8/2020	29aug2020

This person was born on 28aug1967, and we compute his or her age and age with the fractional part on the dates in `end_date`.

```
. generate age = age(dob, end_date)
. generate double fage = age_frac(dob, end_date)
. format fage %12.0g
. list dob end_date age fage
```

	dob	end_date	age	fage
1.	28aug1967	27aug2019	51	51.99726027
2.	28aug1967	28aug2019	52	52
3.	28aug1967	29aug2019	52	52.00273224
4.	28aug1967	28aug2020	53	53
5.	28aug1967	29aug2020	53	53.00273973

Note that the fractional parts on end dates of 29aug2019 and 29aug2020 differ. There are 366 days between 28aug2019 and 28aug2020 because 2020 is a leap year. So the fractional part for 29aug2019 is $1/366 = 0.00273224$. There are 365 days between 28aug2020 and 28aug2021, so the fractional part for 29aug2020 is $1/365 = 0.00273973$.

◁

▷ Example 2: Differences in months

Here we show an example of how `datediff()` and `datediff_frac()` calculate date differences in units of months.

We load a dataset with Stata date variables `start` and `end`. First, we generate `months` using `datediff(start, end, "month")` to get the integer difference (rounded down) in months. Then, we generate `fmonths` using `datediff_frac(start, end, "month")` to get the difference including the fractional part. We also put `datediff(start, end, "day")` into a variable to get differences in days to help us see how the fractional parts are calculated.

```
. use https://www.stata-press.com/data/r17/month_differences, clear
(Fictional data for calculating date differences)
. generate months = datediff(start, end, "month")
. generate double fmonths = datediff_frac(start, end, "month")
. generate days = datediff(start, end, "day")
. format fmonths %12.0g
```

```
. list start end months fmonths days, sepby(start)
```

	start	end	months	fmonths	days
1.	15jan2019	15jan2019	0	0	0
2.	15jan2019	16jan2019	0	.0322580645	1
3.	15jan2019	15feb2019	1	1	31
4.	15jan2019	16feb2019	1	1.035714286	32
5.	15jan2019	15mar2019	2	2	59
6.	15jan2019	16mar2019	2	2.032258065	60
7.	15jan2019	15apr2019	3	3	90
8.	15jan2019	16apr2019	3	3.033333333	91
9.	31jan2019	01feb2019	0	.0344827586	1
10.	31jan2019	28feb2019	0	.9655172414	28
11.	31jan2019	01mar2019	1	1	29
12.	31jan2019	02mar2019	1	1.033333333	30
13.	31jan2019	31mar2019	2	2	59
14.	31jan2019	01apr2019	2	2.032258065	60
15.	31jan2019	30apr2019	2	2.967741935	89
16.	31jan2019	01may2019	3	3	90

Let's first look at the start date 15jan2019. `months` increases by one on 15feb2019 and then again on 15mar2019 and 15apr2019. On these days, `datediff_frac(..., "month")` is an integer.

The fractional month difference between 15jan2019 and 16jan2019 is $1/31 = 0.032258$. The denominator is 31 because the next month anniversary is 15feb2019, which is 31 days from 15jan2019. The fractional part of the difference between 15jan2019 and 16feb2019 is $1/28 = 0.035714$ because there are 28 days between the month anniversaries 15feb2019 and 15mar2019. The fractional part of the difference between 15jan2019 and 16apr2019 is $1/30 = 0.033333$ because there are 30 days between the month anniversaries 15apr2019 and 15may2019.

For the start date 31jan2019, monthly anniversaries are 01mar2019, 31mar2019, and 01may2019. Fractional differences are calculated based on the number of days between the monthly anniversaries. For example, there are 29 days between 31jan2019 and 01mar2019, so the fractional difference between 31jan2019 and 01feb2019 is $1/29 = 0.034483$.

The optional fourth argument, s_{nl}, of `datediff(`e_{d1}`,`e_{d2}`,"month",`s_{nl}`)` applies only when the start date, e_{d1}, falls on 29feb. See the next example for what this option does with ages in years. It works similarly when units are months.

◁

▷ Example 3: Born on a leap day

If you are a "leapling"—born on 29feb—when do you have a birthday in nonleap years? On 28feb or 01mar? Or do you not have a birthday at all in nonleap years (Sullivan 1923)?

In the United Kingdom, a leapling legally becomes 18 on 01mar. In Taiwan, it is 28feb. In the United States, there is no legal statute concerning leap-day birthdates.

The functions `age()`, `age_frac()`, `datediff()`, and `datediff_frac()` all have an optional last argument that sets the day of the birthday (or anniversary) in nonleap years. Here is an example using `age()` and `age_frac()`.

We load a dataset with Stata date variables `dob` (containing date of birth) and `end_date`. We generate `age1` using `age()` with the `"01mar"` argument (which is the default if it is not specified). The `age2` variable is generated using `"28feb"`. We also generate the variables `fage1` and `fage2` using `age_frac()` with different last arguments.

```
. use https://www.stata-press.com/data/r17/leap_day, clear
(Fictional leapling data)
. generate age1 = age(dob, end_date, "01mar")
. generate double fage1 = age_frac(dob, end_date, "01mar")
. generate age2 = age(dob, end_date, "28feb")
. generate double fage2 = age_frac(dob, end_date, "28feb")
. generate year = year(end_date)
. format fage1 fage2 %12.0g
. list dob end_date age1 age2 fage1 fage2, sepby(year)
```

	dob	end_date	age1	age2	fage1	fage2
1.	29feb2004	27feb2019	14	14	14.99452055	14.99726027
2.	29feb2004	28feb2019	14	15	14.99726027	15
3.	29feb2004	01mar2019	15	15	15	15.00273224
4.	29feb2004	28feb2020	15	15	15.99726027	15.99726776
5.	29feb2004	29feb2020	16	16	16	16
6.	29feb2004	01mar2020	16	16	16.00273224	16.00273973

Changes in `age1` and `age2` (that is, birthdays) in nonleap years occur on the day specified by the last argument to `age()`. Note that birthdays in leap years are, of course, on 29feb regardless of the last argument. Fractional parts from `age_frac()` differ because they are based on the number of days between birthdays on either side of `end_date`, which will be 365 or 366. So fractional parts are multiples of $1/365$ or $1/366$.

It is worth mentioning again that `age()`, `age_frac()`, `datediff()`, and `datediff_frac()` all match up sensibly, but if there are leaplings, the last argument must be the same (or not be specified) for them to match up. See *Methods and formulas* in [FN] **Date and time functions**.

◁

Calculating differences of datetimes

The `clockdiff()` function calculates differences of datetime/c values in units of days, hours, minutes, seconds, or milliseconds, with the result rounded down to an integer. The `Clockdiff()` function does the same, except it calculates differences for datetime/C values (UTC times with leap seconds).

The `clockdiff_frac()` and `Clockdiff_frac()` functions calculate the corresponding differences for datetime/c and datetime/C values, respectively, but the fractional part of the difference is also included.

▷ Example 4: Differences of datetime/c values

We have a dataset with string datetimes. A start datetime is recorded in the variable `str_start`, and an end datetime is in `str_end`.

```
. use https://www.stata-press.com/data/r17/time_differences, clear
(Fictional data for calculating time differences)
. list, abbreviate(9)
```

	str_start	str_end
1.	2015-06-30 00:00:00	2015-06-30 23:59:59
2.	2015-06-30 00:00:00	2015-06-30 23:59:60
3.	2015-06-30 00:00:00	2015-07-01 00:00:00
4.	2015-06-30 00:00:00	2015-07-01 23:59:59
5.	2015-06-30 00:00:00	2015-07-02 00:00:00

We must convert the strings to numeric Stata datetimes, which we do using the `clock()` function with a mask of `"YMDhms"`. We format the new encoded datetime variables using format `%tc`, the simplest format specification for datetime/c.

```
. generate double cstart = clock(str_start, "YMDhms")
. generate double cend   = clock(str_end,   "YMDhms")
(1 missing value generated)
. format cstart cend %tc
. list str_end cend
```

	str_end	cend
1.	2015-06-30 23:59:59	30jun2015 23:59:59
2.	2015-06-30 23:59:60	.
3.	2015-07-01 00:00:00	01jul2015 00:00:00
4.	2015-07-01 23:59:59	01jul2015 23:59:59
5.	2015-07-02 00:00:00	02jul2015 00:00:00

One of the string values became missing when it was encoded. It was the value `"2015-06-30 23:59:60"`. This is a leap second, which was added to the end of the day on 30jun2015. There is no encoding for leap seconds in datetime/c. That is why it is missing. We snuck in this leap second to illustrate a point later about datetime/C.

We now use `clockdiff()` to calculate differences in seconds and hours between the datetime/c variables `cstart` and `cend`.

```
. generate csecs  = clockdiff(cstart, cend, "second")
(1 missing value generated)
. generate chours = clockdiff(cstart, cend, "hour")
(1 missing value generated)
. list cstart cend csecs chours
```

	cstart	cend	csecs	chours
1.	30jun2015 00:00:00	30jun2015 23:59:59	86399	23
2.	30jun2015 00:00:00	.	.	.
3.	30jun2015 00:00:00	01jul2015 00:00:00	86400	24
4.	30jun2015 00:00:00	01jul2015 23:59:59	172799	47
5.	30jun2015 00:00:00	02jul2015 00:00:00	172800	48

`clockdiff()` calculates values rounded down to integers, and the results are what we expect. Integer hours starting at 30jun2015 00:00:00 are 23 hours at 30jun2015 23:59:59. Integer hours become 24 hours one second later at 01jul2015 00:00:00.

Rather than use `clockdiff()`, we could take the difference between the datetime/c variables `cstart` and `cend` and use the conversion functions `seconds()` and `hours()`.

```
. generate double csecs2  = seconds(cend - cstart)
(1 missing value generated)
. generate double chours2 = hours(cend - cstart)
(1 missing value generated)
. format %12.0g chours2
. list csecs csecs2 chours chours2
```

	csecs	csecs2	chours	chours2
1.	86399	86399	23	23.99972222
2.	.	.	.	.
3.	86400	86400	24	24
4.	172799	172799	47	47.99972222
5.	172800	172800	48	48

The results are consistent with our earlier results. The number of seconds are exactly the same in `csecs` and `csecs2` because they are integers. Hours in `chours2` are not integers, but rounded down to integers, they agree with hours produced by `clockdiff()`.

If we want to calculate the difference between `cstart` and `cend` in hours with the fractional part, we can use `clockdiff_frac()` as follows:

```
. generate double fchours = clockdiff_frac(cstart, cend, "hour")
(1 missing value generated)
. format %12.0g fchours
. list chours chours2 fchours
```

	chours	chours2	fchours
1.	23	23.99972222	23.99972222
2.	.	.	.
3.	24	24	24
4.	47	47.99972222	47.99972222
5.	48	48	48

As expected, `fchours` is the same as `chours2`.

◁

▷ Example 5: Differences of datetime/C values

What if we are using datetime/C values, that is, datetimes with leap seconds? Let's redo the previous example encoding the strings using `Clock()` to produce `Cstart` and `Cend` as datetime/C. Then, we generate a variable `Csecs` using `Clockdiff(Cstart, Cend, "second")`, `Chours` using `clockdiff(Cstart, Cend, "hour")`, and `fChours` using `Clockdiff_frac(Cstart, Cend, "hour")`.

```
. generate double Cstart = Clock(str_start, "YMDhms")
. generate double Cend   = Clock(str_end,   "YMDhms")
```

```
. format Cstart Cend %tC
. generate Csecs  = Clockdiff(Cstart, Cend, "second")
. generate Chours = Clockdiff(Cstart, Cend, "hour")
. generate double fChours = Clockdiff_frac(Cstart, Cend, "hour")
. format %12.0g fChours
. list Cstart Cend Csecs Chours fChours
```

1.

Cstart	Cend	Csecs	Chours
30jun2015 00:00:00	30jun2015 23:59:59	86399	23

fChours
23.9994446

2.

Cstart	Cend	Csecs	Chours
30jun2015 00:00:00	30jun2015 23:59:60	86400	23

fChours
23.9997223

3.

Cstart	Cend	Csecs	Chours
30jun2015 00:00:00	01jul2015 00:00:00	86401	24

fChours
24

4.

Cstart	Cend	Csecs	Chours
30jun2015 00:00:00	01jul2015 23:59:59	172800	47

fChours
47.99972222

5.

Cstart	Cend	Csecs	Chours
30jun2015 00:00:00	02jul2015 00:00:00	172801	48

fChours
48

In the previous example, the difference between the times of the first observation was 23.99972222 hours; now it is 23.99944460 hours. The difference for the first observation in this example is further from 24 hours because there are now two seconds between `Cend` and 24 hours from `Cstart`, whereas before there was only one second because the leap second was treated as if it did not exist.

The other difference is the denominator of the fractional part. From the earlier example using datetime/c values and `clockdiff_frac()`, we note that $1 - 0.99972222 = 0.00027778 = 1/3600$, where 3,600 is the number of seconds in an hour. In this example using datetime/C values and `Clockdiff_frac()`, we see that $1 - 0.99944460 = 0.00055540 = 2/3601$, where 3,601 is the number of seconds in the hour containing the leap second.

For the second-to-last observation, the fractional part of the difference is 0.99972222, the same as the fractional part in the previous example. So in this example, the hour differences with the fractional part are not evenly spaced, and this would be true even without the second observation with the leap

second in the data. If the lack of uniform spacing is a problem and there are no leap seconds in your data, you may want to consider converting your datetime/C data to datetime/c.

◁

Reference

Sullivan, A. 1923. *The Pirates of Penzance or the Slave of Duty*, libretto by W. S. Gilbert. New York: G. Schirmer.

Also see

[D] **Datetime** — Date and time values and variables

[D] **Datetime business calendars** — Business calendars

[D] **Datetime conversion** — Converting strings to Stata dates

[D] **Datetime display formats** — Display formats for dates and times

[D] **Datetime relative dates** — Obtaining dates and date information from other dates

[D] **Datetime values from other software** — Date and time conversion from other software