

Computing Ages in SAS®

Removal of the YRDIF Function from The Little SAS® Book: A Primer, Fourth Edition

by
Lora D. Delwiche and Susan J. Slaughter

Because of problems when calculating ages, the YRDIF function has been removed from *The Little SAS® Book: A Primer, Fourth Edition*, starting with the third printing. These problems are small, but nonetheless cause ages computed using YRDIF to be inaccurate in specific situations. This paper explains why the problems occur, and then discusses alternate methods for computing ages.

Computing ages with YRDIF The SAS Help and Documentation states that the YRDIF function

"Returns the difference in years between two dates."

Since age is the difference in years between two dates (a birth date and some other date), the YRDIF function has been used to compute ages in this way:

```
age = INT(YRDIF(birth-date, ending-date, 'ACTUAL'));
```

In this implementation, YRDIF first determines the actual number of days between two dates. It then divides the number of days from years with 365 days by 365, and divides the number of days from years with 366 days by 366. In practice, this method does not always return the difference between two dates *in the way that most people would expect*. When you calculate someone's age, you expect the value of age to increment by one year on each birthday. Dividing by 366 causes YRDIF to come up short in some situations. *As a result, ages computed using the YRDIF function may be incorrect if the starting or ending date falls in a leap year, and the ending date is the person's birthday.*

Please note that in this paper age is defined as the difference in years between a birth date and some later date. This paper does not address the question of differences between a birth date and an earlier date. The methods discussed here may produce unexpected results for negative differences in time.

Example This program illustrates the problem. Each line of data includes two dates. The first is the person's date of birth. The second is either the date of that person's first birthday, or the date before or after that person's first birthday. Only three birth dates (March 2, 2007, March 2, 2008, and March 2, 2009) are included.

```
/*A simple example to demonstrate behavior of the YRDIF function*/  
/*Compute age on day before first birthday, on birthday, and on day after*/  
DATA testyrdif;  
  INPUT (BirthDate TestDate) (DATE11. +1);  
  AgeDecimalYRDIF = YRDIF(BirthDate, TestDate, 'ACTUAL');  
  AgeIntegerYRDIF = INT(YRDIF(BirthDate, TestDate, 'ACTUAL'));  
  FORMAT BirthDate TestDate DATE11.;  
  DATALINES;  
02-MAR-2007 01-MAR-2008  
02-MAR-2007 02-MAR-2008  
02-MAR-2007 03-MAR-2008  
02-MAR-2008 01-MAR-2009  
02-MAR-2008 02-MAR-2009  
02-MAR-2008 03-MAR-2009  
02-MAR-2009 01-MAR-2010  
02-MAR-2009 02-MAR-2010  
02-MAR-2009 03-MAR-2010  
;
```

```
PROC PRINT DATA = testyrdif;
  TITLE 'Results for YRDIF Function';
RUN;
```

This program uses the YRDIF function to compute the difference in decimal years between the two dates (AgeDecimalYRDIF). Then the INT function is added to keep only the integer portion of the YRDIF result (AgeIntegerYRDIF).

Here is the output:

Results for YRDIF Function					1
Obs	BirthDate	TestDate	Age Decimal YRDIF	Age Integer YRDIF	
1	02-MAR-2007	01-MAR-2008	0.99955	0	
2	02-MAR-2007	02-MAR-2008	1.00228	1	
3	02-MAR-2007	03-MAR-2008	1.00502	1	
4	02-MAR-2008	01-MAR-2009	0.99498	0	
5	02-MAR-2008	02-MAR-2009	0.99772	0	
6	02-MAR-2008	03-MAR-2009	1.00046	1	
7	02-MAR-2009	01-MAR-2010	0.99726	0	
8	02-MAR-2009	02-MAR-2010	1.00000	1	
9	02-MAR-2009	03-MAR-2010	1.00274	1	

The results for the calculations on each person's birthday are highlighted in yellow. You can see that the integer value from YRDIF gives the child born in 2008, a leap year, an age of 0 on his first birthday while the children born in 2007 and 2009 have the correct ages. The YRDIF function works this way because it uses rules from the Securities Industry to compute the difference between two dates. The problem is that these rules don't always match the rules used to compute a person's age. The following two SAS Notes document this problem.

<http://support.sas.com/kb/3/036.html>

<http://support.sas.com/kb/36/977.html>

Alternate methods for computing ages People who want to compute ages have a choice of possible solutions. Which method you use depends on how precise your results need to be, and on whether you need age in decimal or integer values. Using YRDIF may be an acceptable solution if you do not need precise ages. Another solution that produces approximate results is simply to divide the difference between the two dates by 365.25:

```
age = INT((ending-date - birth-date) / 365.25);
```

To compute precise integer ages, this method is recommended:

```
age = INT(INTCK('MONTH', birth-date, ending-date)/12);
IF MONTH(birth-date) = MONTH(ending-date)
  THEN age = age - (DAY(birth-date) > DAY(ending-date));
```

This method uses the rather complex and versatile INTCK function. In this implementation, INTCK counts the number of times the first day of a month appears between the two dates. Dividing this number by 12 gives a rough value in years. Then the method checks to see if the starting month and ending month are the same. If they are the same, then it checks to see if the ending day of the month is before or after the starting day of the month. If the starting day of the month is greater than the ending day, then the age will be reduced by one.

The INTCK method computes accurate integer values of age. However, because it counts the number of months (thereby lumping together everyone born in a particular month), the decimal values produced by INTCK are not meaningful. If you need decimal values for ages, you cannot use INTCK.

To compute decimal values for age, use the YRDIF or 365.25 methods and remove the INT function so that age is no longer converted to an integer. The YRDIF and 365.25 methods give similar results for decimal ages, but YRDIF is more accurate. The small differences result from the fact that YRDIF will factor in leap years only if leap years fall between the starting and ending dates. Dividing by 365.25, on the other hand, always factors in leap years even when no leap years fall between the dates.

To compute current age (age at the time your SAS program runs), you simply find the difference between the current date and the birth date. The TODAY function returns the current date as a SAS date value. So to compute current age, you would insert the TODAY function in place of the ending date for any of these methods.

It is possible to use PROC FCMP to create a user-defined function for computing ages. The code for that can be found on SAS Institute's customer support web site.

<http://support.sas.com/kb/36/788.html>

Example This program computes age using each of these methods (YRDIF, dividing by 365.25, and INTCK) so that the results can be compared. For the YRDIF and 365.25 methods, age is computed both as a decimal and an integer value. For the INTCK method, age is computed only as an integer.

```

/*Comparing different ways of computing age*/
DATA testyrdif;
  INPUT (BirthDate TestDate) (DATE11. +1);
  *Decimal and integer ages using YRDIF;
  AgeDecimalYRDIF = YRDIF(BirthDate, TestDate, 'ACTUAL');
  AgeIntegerYRDIF = INT(YRDIF(BirthDate, TestDate, 'ACTUAL'));
  *Decimal and integer ages using 365.25;
  AgeDecimal_365_25 = (TestDate - BirthDate) / 365.25;
  AgeInteger_365_25 = INT((TestDate - BirthDate) / 365.25);
  *Integer ages using INTCK;
  AgeIntegerINTCK = INT(INTCK('MONTH', BirthDate, TestDate) / 12);
  IF MONTH(BirthDate) = MONTH(TestDate)
    THEN AgeIntegerINTCK = AgeIntegerINTCK - (DAY(BirthDate) > DAY(TestDate));
  FORMAT BirthDate TestDate DATE11.;
  DATALINES;
02-MAR-2007 01-MAR-2008
02-MAR-2007 02-MAR-2008
02-MAR-2007 03-MAR-2008
02-MAR-2008 01-MAR-2009
02-MAR-2008 02-MAR-2009
02-MAR-2008 03-MAR-2009
02-MAR-2009 01-MAR-2010
02-MAR-2009 02-MAR-2010
02-MAR-2009 03-MAR-2010
;

PROC PRINT DATA = testyrdif;
  VAR BirthDate TestDate AgeDecimal: AgeInteger;;
  TITLE 'Comparison of Methods for Computing Age';
RUN;

```

Here is the output:

Comparison of Methods for Computing Age							2
Obs	BirthDate	TestDate	Age Decimal YRDIF	Age Decimal_ 365_25	Age Integer YRDIF	Age Integer_ 365_25	Age Integer INTCK
1	02-MAR-2007	01-MAR-2008	0.99955	0.99932	0	0	0
2	02-MAR-2007	02-MAR-2008	1.00228	1.00205	1	1	1
3	02-MAR-2007	03-MAR-2008	1.00502	1.00479	1	1	1
4	02-MAR-2008	01-MAR-2009	0.99498	0.99658	0	0	0
5	02-MAR-2008	02-MAR-2009	0.99772	0.99932	0	0	1
6	02-MAR-2008	03-MAR-2009	1.00046	1.00205	1	1	1
7	02-MAR-2009	01-MAR-2010	0.99726	0.99658	0	0	0
8	02-MAR-2009	02-MAR-2010	1.00000	0.99932	1	0	1
9	02-MAR-2009	03-MAR-2010	1.00274	1.00205	1	1	1

For the dates in this example, only the INTCK method produces the correct integer value of age on the birthday for all birth dates. The YRDIF and dividing by 365.25 methods for computing decimal ages produce different answers, but the differences are very small.

Conclusions This paper presents a brief overview of the major methods for computing ages. For people who want more information, we recommend doing a search of conference papers on this topic. Some of those papers present other methods for computing ages, but they are generally variations on the methods shown here.

It is important to note that the dates used in these examples were chosen to illustrate the problems involved in computing ages. In practice, the frequency of such problems is small for any of these methods. In addition, the magnitude of the problem is small. (The people who had incorrect ages on their birthdays, had correct ages just one day later.)

At the time this paper was written (summer 2010), there was no single definitive method for computing ages in SAS. For applications in which approximate ages are acceptable, any of these methods may be sufficient. If you need to compute decimal ages, the YRDIF method is still best. For accurate integer values of ages, the INTCK method is best.