Berechnung von Arbeitstagen in Zeitintervallen – Nutzer-definierte Funktionen mit PROC FCMP vs. MAKRO-Lösung

Thomas G. Grobe
AQUA-Institut
Maschmühlenweg 8-10
37073 Göttingen
Thomas.Grobe@Aqua-Institut.de

Zusammenfassung

In Datentabellen mit Angaben zu Zeitintervallen – jeweils definiert durch ein VON- und BIS-Datum – soll die Anzahl der regulären Arbeitstage (RA) ermittelt werden. Werden als RA zunächst alle Kalendertage von Montag bis Freitag gezählt, lässt sich eine Ermittlung der RA einfach mit der SAS-Funktion INTCK und 'WEEKDAY17W' als Intervall-Spezifikation realisieren. Nicht von der Zählung ausschließen lassen sich ohne Vorarbeiten so allerdings die typischerweise national unterschiedlich festgelegten Feiertage.
Zur Ermittlung von RA mit Berücksichtigung von Feiertagen findet sich gut dokumentierter SAS-Programmcode im Internet (1), mit dem unter Nutzung von PROC FCMP eine Nutzer-definierte Funktion generiert wird, die anschließend, wie eine übliche SAS-Funktion, in der SAS-Syntax verwendet werden kann. Der zitierte Programmcode wurde angepasst und um eine Möglichkeit zur Berücksichtigung regionalspezifischer Feiertage (auf Bundeslandebene [BL]) in Deutschland erweitert. Tests der Nutzer-definierten Funktionen zeigten allerdings Laufzeiten, die bei der Verarbeitung großer Datentabellen mit vielen Mio. Beobachtungen nur eingeschränkt akzeptabel wären (18,6 bzw. 55 Sekunden für 100.000 Intervalle, ohne bzw. mit Berücksichtigung BL-spezifischer Feiertage).
Vor diesem Hintergrund wurde ein SAS-Makro weiterentwickelt, welches als Alternative zur Nutzer-definierten Funktion bei identischen Daten identische Ergebnisse liefert. Die Laufzeiten der Makro-Lösung erwiesen sich als erheblich kürzer (0,06 bzw. 0,08 Sekunden für 100.000 Intervalle ohne bzw. mit Berücksichtigung BL-spezifischer Feiertage). Viele Wege führen zum Ziel – manche im Einzelfall erheblich schneller. Neben der Lösung für eine konkrete Aufgabe soll der Beitrag auch allgemeine Anregungen für die SAS-Programmierung liefern.

Schlüsselwörter: Zeitintervalle, Arbeitstage, PROC FCMP, SAS-Makro, PROC SQL

1 Hintergrund

Auf der Basis von Daten einer größeren Krankenkasse zu erkrankungsbedingten Arbeitsunfähigkeiten (AU) und Beschäftigungszeiten sollten Fehltage an regulären Arbeitstagen (RA) und damit zu jährlich mehr als 10 Mio. Zeitintervallen ermittelt werden. Arbeitsunfähigkeiten werden ärztlicherseits jeweils für bestimmte Zeiträume mit Angaben zum Datum des ersten sowie letzten Tages der AU ausgestellt. Typische Auswertungen von Krankenkassen zu Arbeitsunfähigkeiten berücksichtigen AU-Tage und Beschäftigungsverhältnisse an allen Kalendertagen. Arbeitgeberseitige Statistiken beruhen

demgegenüber oftmals ausschließlich auf Angaben zu RA und gemeldete Fehlzeiten an entsprechenden Tagen. Um annähernd vergleichbare Ergebnisse auf der Basis von Daten einer Krankenkasse zu ermitteln, müssen arbeitsfreie Tage an Wochenenden sowie Feiertage bei den Auswertungen unberücksichtigt bleiben. Nach entsprechend angepassten Zählungen dürften bei Berechnungen vergleichbare Ergebnisse wie bei vielen Arbeitgebern resultieren. Nicht berücksichtigt werden können spezifische oder individuell geregelte Arbeitszeiten, die in einigen Branchen eine wesentliche Bedeutung besitzen, zu denen jedoch einer Krankenkasse keine Informationen vorliegen.

2 Berechnung von Tagesintervallen

Zeiträume in Tagen lassen sich in SAS durch eine simple Subtraktion von Datumswerten berechnen. Sollen der jeweils angegebene Von- und Bis-Tage Bestandteile des Intervalls sein, muss zu der ermittelten Differenz der Wert 1 addiert werden.

```
Tage = BisDat - VonDat + 1;
```

Sollen Wochenenden (Samstag und Sonntag) bei der Berechnung unberücksichtigt bleiben, lässt sich dies in SAS gleichfalls relativ einfach realisieren. Zur Ermittlung von Zeitintervallen in sehr unterschiedlichen Maßeinheiten (z.B. auch in Quartalen [`qtr`] oder Jahren [`year`]) kann in SAS allgemein die Funktion `intck` genutzt werden, wobei die gewünschte Maßeinheit als erster Parameter der Funktion anzugeben ist. Ein entsprechender Parameter ist auch „`weekday`“, bei dessen Angabe ausschließlich Wochentage von Montag bis Freitag gezählt werden.

```
if weekday(VonDat) in(1,7) then
     TageRA = intck("weekday", VonDat, BisDat);
else TageRA = intck("weekday", VonDat-1, BisDat);
```

Dabei werden mit `intck`, wie auch bei einer einfachen Subtraktion von Datumswerten, Veränderungen einer Zählgröße, ermittelt, weshalb auch hier zur Bestimmung der Intervalldauer bei Einbeziehung des ersten und letzten Tages der Wert 1 addiert, oder, wie in der o.g. Syntax, das Von-Datum um einen Tag „vorverlegt“ werden muss. Diese Addition ist allerdings nicht erforderlich, sofern das Von-Datum auf ein Wochenende fällt (welches sinngemäß bei der Berechnung als Verlängerung des direkt vorausgehenden Arbeitstages angesehen wird), da in diesem Fall der erste Arbeitstag im Intervall ohnehin zu einer Veränderung der Zählgröße führt und von `intck` berücksichtigt wird.

Der Wochentag eines Datums lässt sich dabei mit der *Funktion* `weekday` ermitteln. Sie liefert für Sonntag den Wert 1, für Montag 2 usw. bis 7 für Samstag. Die oben genannte Bedingung `if weekday(VonDat) in(1,7)` überprüft also, ob der Von-Tag eines Intervalls ein Sonntag [`1`] oder Samstag [`7`] ist und berechnet die Intervalldauer in diesem Fall ohne eine „Vorverlegung“ des Von-Datums.

Der Funktionsparameter „`weekday`“ kann zum Ausschluss der Wochenendtage 1 und 7 wie in der zuvor beschriebenen Syntax verwendet werden, erlaubt jedoch optional auch den Ausschluss beliebiger anderer Wochentage. Diese müssen dem Parameter jeweils als Zahlenwerte mit einem abschließenden „w“ nachgestellt werden. Bei der Angabe des Parameters „`weekday234567w`“ würden von der Funktion `intck` in einem extremen Beispiel nur noch Sonntage gezählt. Möchte man die Auswahl der nicht berücksichtigten Wochentage in einem Programm flexibel halten, lässt sich dies durch eine Festlegung der Parameter in Makro-Variablen zu Beginn des Programms bewerkstelligen. Die nachfolgend dargestellte Syntax liefert dabei ein übereinstimmendes Ergebnis wie der vorausgehend dargestellte Code.

```
%LET CFWDL=17;  * Wochentage, Angabe ohne Trennzeichen ;
%LET SFWDL=1,7; * Wochentage, Angabe mit Trennung ;
if weekday(VonDat) in(&SFWDL.) then
     TageRA = intck("weekday&CFWDL.w", VonDat, BisDat);
else TageRA = intck("weekday&CFWDL.w", VonDat-1, BisDat);
```

3 Ermittlung von Feiertagen

Zur Berücksichtigung von Feiertagen auch außerhalb der Wochenenden bei der Berechnung der regulären Arbeitstage müssen national oder regional gültige Feiertage bekannt sein – standardmäßig gibt es hierfür in SAS, in Anbetracht regional unterschiedlicher Regelungen verständlicherweise, keine simpel aufrufbaren Funktionen.
Existieren ausschließlich Feiertage, die immer auf bestimmte Kalendertage fallen (wie in Deutschland z.B. der 1. Januar, 1. Mai, 3. Oktober und 25. Dezember), würde als Grundlage für nachfolgende Berechnungen eine einfache und unveränderliche Liste dieser Tage ausreichen.
Einige Feiertage in Deutschland fallen jedoch von Jahr zu Jahr auf unterschiedliche Kalendertage. Diese (christlichen) Feiertage weisen dabei i.d.R. einen konstanten zeitlichen Abstand zum Ostersonntag des jeweiligen Jahres auf, womit sich das Hauptproblem ihrer Berechnungen darauf beschränkt, das Datum des Ostersonntags innerhalb einzelner Jahre zu ermitteln. Für das Problem entwickelte bereits der Mathematiker Carl Friedrich Gauß einen Berechnungsalgorithmus. Ausführliche Informationen sind dem Wikipedia-Beitrag „Gaußsche Osterformel“ zu entnehmen. Die Umsetzung der Berechnung des Datums eines Ostersonntags für ein beliebiges Kalenderjahr in SAS zeigt die nachfolgend dargestellte Syntax (gemäß ergänzter Osterformel nach H. Lichtenberg).

Für die Berechnung des Ostersonntags muss lediglich das gewünschte Jahr in der Makro-Variablen `&YEAR.` spezifiziert sein, z.B. über das Statement `%LET YEAR = 2015;`. Nach Ablauf der Syntax ist der Datumswert für den Ostersonntag 2015 in der Makro-Variablen `&OSTSUN.` abgelegt und kann z.B. mit `%PUT &OSTSUN.;` ausgegeben werden, wobei für das Jahr 2015 der SAS-interne Datumswert 20183 ins Log geschrieben wird. Da im Data-Statement der Dateiname _null_ angegeben ist, wird mit dem Durchlauf der Syntax keine Datei erstellt. Mit

```
%PUT %SYSFUNC(putn(&OSTSUN.,ddmmyy10.));
```

erhält man bei Bedarf auch eine formatierte und lesbare Datumsangabe 05/04/2015, also die Angabe, dass im Jahr 2015 Ostersonntag auf den 5. April fällt.

```
data _null_; * ermittelt Ostersonntag im spezifizierten Jahr;
k = int( &YEAR./100 );
m = 15 + int( (3*k + 3)/4 ) - int( (8*k + 13)/25 );
s =  2 - int( (3*k + 3)/4 );
a = mod(&YEAR., 19);
d = mod(19*a + m, 30);
r = int( (d + int( a/11 ) )/29 );
og= 21 + d - r;
sz= 7 - mod( (&YEAR. + int( &YEAR./4 ) + s), 7);
oe= 7 - mod( (og-sz), 7);
os= og + oe;
OM= 3; OT=os; if os gt 31 then do;  OM=4; OT=os-31;  end;
OstSun= mdy(OM,OT,&YEAR.);  call symput("OSTSUN",OstSun);
run;
```

Erst spät wurde vom Autor des vorliegenden Beitrags bemerkt, dass sich die Lage des Ostersonntags in SAS-Software seit der Version 9 auch einfach mit der Funktion `Holiday` und dem Parameter `"EASTER"` ermitteln lässt, z.B. für 2015 mit

```
OstSunSAS = holiday("EASTER",2015);
```

Die Funktion berechnet identische Ergebnisse wie der dargestellte Programm-Code. Hingewiesen sei darauf, dass die Funktion Holiday erst ab dem Jahr 1582 (dem Jahr der Einführung des gregorianischen Kalenders) und bis zum Jahr 20000 Ergebnisse liefert. Gleiches gilt offensichtlich auch für andere SAS-Datumsfunktionen wie z.B. `mdy()`, weshalb auch mit dem dargestellten eigenen Programm ab dem Jahr 20001 keine SAS-Datumswerte mehr berechnet werden und b. B. von „weit vorausschauenden SAS-Nutzern“ nur auf die separaten Angaben des Programms zu Tag, Monat und Jahr eines Ostersonntags zurückgegriffen werden kann.

Weitere veränderliche Feiertage lassen sich überwiegend aus der Lage des Ostersonntags herleiten. So liegt Karfreitag immer zwei Tage vor Ostersonntag, Himmelfahrt 39 Tage nach Ostersonntag usw.. Aus den Informationen können – unter gleichzeitiger Berücksichtigung von Informationen zu den festgelegten regelmäßig arbeitsfreien Wochentagen sowie unveränderlichen Feiertagen – Listen mit arbeitsfreien (Feier-) Tagen außerhalb der regelmäßig freien Wochentage generiert werden, wobei mit entsprechend angepasster Syntax separat auch Informationen zu Tagen bereitgestellt werden können, an denen lediglich in bestimmten Regionen Deutschlands regulär nicht gearbeitet wird.

4 Funktion zur Berechnung von Arbeitstagen

Eine Google-Suche mit den Stichworten „working days“ und „sas“ lieferte als ersten Treffer einen Hinweis auf Programm-Code von Chris Hemedinger zur Ermittlung von regulären Arbeitstagen [1]. Vorgestellt wird eine Nutzung der Prozedur PROC FCMP, mit der eine Nutzer-definierte Funktion erstellt wird, die anschließend in einer SAS-Syntax wie eine reguläre SAS-Funktion zur Berechnung der RA verwendet werden kann. Die nachfolgende Syntax zeigt einen leicht modifizierten und auf wesentliche Elemente reduzierten Programmcode.

Zeile 1 ruft PROC FCMP auf und legt eine Bibliothek zur Ausgabe der Funktion fest. Zeile 2 legt den Namen „`workdayn`“ und die zwei Aufrufparameter der nachfolgend bis `endsub` definierten neuen Funktion fest. Grundsätzlich lassen sich mit nur einem Aufruf von PROC FCMP entsprechend auch mehrere neue Funktionen bereitstellen.
Zeile 3 benennt ein Array mit dem Namen „`holidays`“. Das Array „`holidays`“ wird nachfolgend mit Werten aus der Datei „`work.holidays_a`“ gefüllt, in diesem Fall mit Werten aus der Variablen „`dat`“, die Datumsangaben zu Feiertagen (außerhalb der wöchentlich regulär arbeitsfreien Tage) enthält.
Zeile 5 bis 7 ermitteln, analog wie bereits vorausgehend dargestellt und unter Nutzung der Funktion `intck`, die Tage zwischen dem Datum „`d1`“ und „`d2`“, die nicht auf wöchentlich regulär arbeitsfreie Tage entfallen.
In Zeile 8 bis 10 werden in einer Programmschleife alle Elemente des Arrays „`holidays`“ gelesen. Sofern sich ein Element des Arrays – hier ggf. ein zusätzliches Feiertagsdatum – innerhalb des Zeitintervalls von `d1` bis `d2` befindet, wird von der zuvor ermittelten Zahl der Arbeitstage in der Variablen „`diff`“ jeweils ein (weiterer) Tag abgezogen. Mit `return(diff);` wird schließlich festgelegt, dass der letztendlich in `diff` erfasste Wert als Ergebnis der Funktion ausgegeben werden soll. PROC FCMP wird schließlich mit `run` und `quit` abgeschlossen. Über die Option `cmplib` wird festgelegt, in welcher Bibliothek nachfolgend Nutzer-definierte Funktionen gesucht werden sollen.

```
01   proc fcmp  outlib=work.myfuncs.dates;
02   function workdayn(d1,d2);
03   array holidays[1] /nosymbols;
04   rc = read_array("work.holidays_a", holidays, "dat");
05   if weekday(d1) in(&SFWDL.) then
06     diff = intck("WEEKDAY&CFWDL.W",d1,d2);
07   else diff = intck("WEEKDAY&CFWDL.W",d1-1,d2);
08   do i = 1 to dim(holidays);
09     if (d1 <= holidays[i] <= d2) then diff = diff - 1;
10   end;                    return(diff);
11   endsub;
12   run; quit;              options cmplib=work.myfuncs;
```

Im Rahmen der Bearbeitung des Themas wurden erweiterte Varianten entsprechender Funktionen generiert, die auch eine regional differenzierte Berücksichtigung von Feier-

tagen erlauben. Dabei mussten Regionalkennungen vor dem Einlesen in Arrays obligat in numerische Werte umgesetzt werden, was als suboptimal anzusehen ist. Die Syntax kann auf Nachfrage vom Autor gern zur Verfügung gestellt werden. Weitere Informationen und Anwendungsbeispiele zu PROC FCMP aus der Praxis sind der im Literaturverzeichnis genannten Publikation zu entnehmen [2].

5 Makro zur Berechnung von Arbeitstagen

Aufbauend auf Listen bzw. Dateien zu Feiertagen in den betrachteten Zeiträumen können die Berechnungen der Arbeitstage ähnlich auch in einem Makro implementiert werden, welches, alternativ an Stelle der zuvor beschriebenen Funktion, innerhalb eines Data steps aufgerufen werden kann.
Sollen Merkmalswerte aus vielen Datenzeilen in eine oder in mehrere Makro-Variablen eingelesen werden, eignet sich hierzu i.d.R. am besten PROC SQL mit einem `into`-Statement (vgl. nachfolgende Syntax). Eine Anwendung von SQL wird dabei in SAS grundsätzlich mit PROC SQL eingeleitet, hier mit der Option `noprint`, um die nicht erwünschte Ausgabe von Ergebnissen in das Output-Fenster zu unterbinden. Bis zum `quit;` (vgl. Zeile 7) kann bzw. muss nachfolgend die spezifische SAS-SQL-Syntax verwendet werden.

```
01   proc sql noprint;          * Werte in Makro-Variablen einlesen ;
02   select left(put(count(dat),8.)) into :NA  from Holidays_a ;
03   select dat into :AD1 - :AD&NA.  from Holidays_a ;
04   select left(put(count(dat),8.)) into :NR  from Holidays_rs ;
05   select dat, FRegS  into :RD1 - :RD&NR., :RR1 - :RR&NR.
06                                  from Holidays_rs;
07   quit;
08   %MACRO WDAY(RESV,FROMV,TOV,REGV);
09   if &CFWDL.=9 then do;  * falls kein Wochentag ausgeschlossen ;
10     &RESV.=&TOV.-&FROMV.+1;
11   end;
12   else do;               * falls Wochentage ausgeschlossen ;
13     if weekday(&FROMV.) in(&SFWDL.) then
14       &RESV. = intck("WEEKDAY&CFWDL.W", &FROMV., &TOV.);
15     else &RESV. = intck("WEEKDAY&CFWDL.W", &FROMV.-1, &TOV.);
16   end;
17   %IF &RESV. ne   and &NA. gt 0 %THEN %DO;
18     %DO Z=1 %TO &NA.;
19        if &FROMV. le &&AD&Z. le &TOV. then &RESV.=&RESV.-1;
20     %END;
21   %END;
22   %IF &RESV. ne   and &REGV. ne   and &NR. gt 0 %THEN %DO;
23     %DO ZR=1 %TO &NR.;
24        if &FROMV. le &&RD&ZR. le &TOV.
25           and &REGV. in(&&RR&ZR.) then &RESV.=&RESV.-1;
26     %END;
27   %END;
28   %MEND WDAY;
```

Zunächst wird die Anzahl der Zeilen der Datei mit bundesweiten Feiertagsangaben (Zeile 2) sowie später auch der Datei mit regionalen Feiertagsangaben (Zeile 4) in die beiden Makro-Variablen `&NA.` und `&NR.` eingelesen. Die Angaben müssen linksbündig formatiert vorliegen, um sie anschließend als Namensbestandteile von (nummerierten) Makro-Variablen verwenden zu können.
Der Code in Zeile 3 liest alle Datumswerte zu den Feiertagen aus der Variablen „`dat`“, die in der Datei „`Holidays_a`“ zu bundesweiten Feiertagen gespeichert sind, in durchnummerierte Makro-Variablen `&AD1.` bis `&&AD&NA.`, wobei die Makro-Variable `&NA.` die Angabe zur Anzahl der Zeilen und damit auch zur Anzahl der Werte in der Tabelle „`Holidays_a`“ enthält. Je Wert bzw. Zeile wird so genau eine Makro-Variable gebildet und mit einer Datumsangabe gefüllt.
Analog füllt der Code in Zeile 5 und 6 zwei Gruppen von durchnummerierten Makro-Variablen mit Angaben zu regionalspezifischen Feiertagen (`&RD1.` bis `&&RD&NR.`) sowie mit Informationen zu jeweils betroffenen Regionen (`&RR1.` bis `&&RR&NR.`).

Der im vorausgehenden Text verwendete Code für Makro-Variablen, wie beispielsweise `&&RR&NR.`, entspricht der Form, in der der Inhalt einer Makro-Variable in der SAS-Syntax abgerufen werden kann, wenn der Name einer Makro-Variablen selbst erst durch den Inhalt einer weiteren Makro-Variable festgelegt ist. Beinhaltet `&NR.` z.B. den Wert 7, liefert der Code `&&RR&NR.` den *Wert* der Makro-Variablen `&RR7.` (der Code `&RR&NR.` würde demgegenüber eine *Verkettung der Inhalte der beiden Makro-Variablen* `&RR.` und `&NR.` liefern; wird der Name einer Makro-Variablen aus den Inhalten von *zwei* anderen Makro-Variablen zusammengesetzt, müssen zur Abfrage des Inhalts/Wertes der Variablen entsprechend *drei* &-Zeichen vorangestellt werden).

Der Code aus den Zeilen 8 bis 28 definiert das eigentliche Makro mit dem Namen „`WDAY`“. Beim Aufruf des Makros können maximal vier Variablen-Namen spezifiziert werden. „`RESV`“ benötigt obligat einen Variablen-Namen gemäß SAS-Konventionen, der für die Ausgabe der berechneten Arbeitstage verwendet wird. „`FROMV`“ benötigt die Angabe zum Namen einer Variablen mit dem Von-Datum und „`TOV`“ den Namen der Variablen mit dem Bis-Datum von Zeitintervallen in der Datei, für die Arbeitstage berechnet werden sollen. Im Gegensatz zu den vorausgehenden Angaben optional ist die zusätzliche Spezifikation einer Variablen in „`REGV`“, welche in den Daten Angaben zur regionalen Zuordnung enthält (hier alphanumerische Bundesland-Kennungen von 01 bis 16 für Schleswig-Holstein bis Thüringen, für die in der Datei „`Holidays_rs`“ regionalspezifische Feiertage spezifiziert sind).
Die Zeilen 9 bis 11 dienen zur ersten Berechnung der Intervalldauer, sofern in einem Sonderfall keine Wochentage als regelmäßig arbeitsfrei berücksichtigt werden sollen (wofür im übergeordneten Programmcode der Wert 9 an Stelle der typischen Nennungen der Wochentage von 1 und 7 angegeben werden muss). Zeilen 12 bis 16 beinhalten den Code zu Berechnung von Intervalllängen mit regelmäßigem Ausschluss von zuvor spezifizierten Wochentagen bei der Ermittlung von Arbeitstagen, der in ähnlicher Form

bereits im Zusammenhang mit PROC FCMP erläutert wurde. Zeilen 18 bis 20 bilden eine erste Makro-Schleife mit Durchgängen entsprechend der Anzahl der bundeseinheitlichen Feiertage. Sofern einer der in nummerierten Makro-Variablen abgelegten Feiertage in das jeweils betrachtete Zeitintervall fällt, wird die zuvor notierte Intervalldauer um einen Tag reduziert.
Eine zweite Makro-Schleife (Zeilen 23 bis 26) wir lediglich durchlaufen, sofern Angaben auch zu regionalspezifischen Feiertagen existieren. Neben der zeitlichen Relevanz von Feiertagen für das betrachtete Intervall wird durch eine Suche der Regionalkennung in der mit `&REGV.` spezifizierten Variablen in der entsprechenden Makro-Variablen `&&RR&ZR.` auch die regionale Relevanz der Feiertage überprüft und ggf. die Zahl der ermittelten Arbeitstage weiter reduziert.

6 Programmablauf und Performance

Die zuvor ausschnittsweise dargestellte Syntax zur Berechnung von Arbeitstagen zu dokumentierten Zeitintervallen verteilt sich auf drei SAS-Programme. Das SAS-Programm „**Holiday006.sas**“ enthält ausschließlich ein Makro „`HOLIDAY`“, welches nach dem einmaligen Durchlauf des Programmcodes (der Kompilierung) anschließend innerhalb derselben SAS-Session mit `%HOLIDAY(YMIN,YMAX,FWDL);` aufgerufen werden kann. Angegeben werden müssen für die Erstellung von mehreren angepassten Dateien mit Angaben zu Feiertagen das erste und letzte Kalenderjahr des Zeitraums, für den die Feiertage ermittelt werden sollen sowie als dritte Angabe eine Kennungen für die regelmäßig arbeitsfreien Wochentage. Sollen keine Wochentage als regelmäßig arbeitsfrei berücksichtigt werden, muss der Wert 9 angegeben werden. Der Aufruf `%HOLIDAY(2014,2015,1 7);` stellt beispielsweise Tabellen zu Feiertagen von Anfang 2014 bis Ende 2015 bereit, wobei Sonntage und Samstage als regelmäßig arbeitsfreie Tage berücksichtigt werden. Die aktuelle Implementierung berücksichtigt bundesweit gültige Feiertage in Deutschland sowie regionale Feiertage, sofern sie innerhalb einzelner Bundesländer für alle Regionen gelten (vgl. Eintrag „Feiertage in Deutschland“ in Wikipedia). Sollen abweichende Tage berücksichtigt werden, muss die Syntax des Makros „Holiday“ entsprechend angepasst werden.
Das SAS-Programm „**Holiday_function006.sas**“ stellt drei Nutzer-definierte Funktionen bereit. Die Funktion `workdayn(d1,d2)` berechnet Arbeitstage unter Berücksichtigung nationaler Feiertage (vgl. auch die vorausgehend erläuterte Syntax im Abschnitt 4), `workdayrs(d1,d2,regiovar)` berechnet die Zahl der ggf. zusätzlich anfallenden regionalen Feiertage und `workdayr(d1,d2,regiovar)` vereint die vorherigen Funktionen und berechnet Arbeitstage unter Berücksichtigung nationaler und regionaler Feiertage.
Das dritte Programm „**Holiday_macro006.sas**“ umfasst im Wesentlichen das zuvor dargestellte Makro `%WDAY` (vgl. Abschnitt 5). Vor einer Anwendung der Nutzer-definierten Funktionen sowie des Makros ist allgemein darauf zu achten, dass mit dem Makro `%HOLIDAY` passende Listen zu Feiertagen für die jeweils betrachteten Zeiträume bzw. Kalenderjahre bereitgestellt wurden.

Die Laufzeiten der Berechnungen wurden mit zufällig erzeugten Daten ermittelt. Ein einfaches Testprogramm ist nachfolgend dargestellt. Erzeugt wird hierdurch eine Datei mit 100.000 Beobachtungen, die eine konstante Regionalangabe (Variable `bl`), ein konstantes Von-Datum (`date1`) und ein zufälliges Bis-Datum (`date2`) enthält.

```
data test2; format date1 date2 ddmmyy10.;
do i=1 to 100000;
  bl="09"; date1=mdy(01,01,2015); date2=date1+int(ranuni(1)*364);
/* wtage1=date2-date1+1; */          * V1. einfache Zeitdifferenz ;
/* wtage2=workdayn(date1,date2);*/   * V2. eigene Funktion ;
   %WDAY(wtage3,date1,date2,bl);     * V3. Makro-Variante ;
   output test2;
end; run;
```

Im Abstract und nachfolgend genannte Laufzeiten unterscheiden sich und können grundsätzlich nur Anhaltspunkte liefern, da sie von Aufrufparametern sowie der verwendeten Soft- und Hardware-Umgebung abhängig sind. Wurde bei drei Durchläufen unter identischen Bedingungen in einer ersten Variante V1 lediglich eine einfache Zeitdifferenz berechnet und in der Variablen `wtage1` gespeichert, benötigte das Programm auf einem Testrechner (bei Auskommentierung der Varianten V2 und V3) lediglich 0,03 Sekunden. Bei der Berechnung von Arbeitstagen durch eine Nutzer-definierte Funktion `workdayn` in Variante V2 wurden demgegenüber gut 17 Sekunden benötigt, bei einer leicht verkürzter Syntax wie im vorliegenden Beitrag dargestellt waren es gut 14 Sekunden. In der dritten Variante V3 mit Verwendung des Makros `%WDAY` und Berechnung der Variablen `wtage3` benötigte ein Durchlauf zur Berechnung von Arbeitstagen unter Berücksichtigung von regionalspezifischen Feiertagen 0,07 Sekunden, ohne Bundeslandangabe und damit bei Ermittlung identischer Ergebnisse wie mit den Nutzer-definierten Funktionen nur 0,06 Sekunden. Die Nutzer-definierten Funktionen benötigen demnach eine vergleichsweise sehr lange Zeit. Die beobachteten Laufzeiten erhöhen sich in allen Varianten mit einer zunehmenden Zahl an Beobachtungen annähernd proportional. Bei 1.000.000 Beobachtungen wurden in Variante V1 0,23 Sekunden, in Variante V2 174 Sekunden (verkürzte Syntax: 145 Sekunden) und in Variante V3 ohne Regionalangabe 0,49 Sekunden gemessen. Die Berechnungen dauern mit den Nutzer-definierten Funktionen demnach mehr als 200 Mal länger als mit den Makro-Varianten. Ursachen für die relativ langen Laufzeiten bei der Verwendung der Nutzer-definierten Funktionen konnten zumindest ansatzweise eingegrenzt werden. Testläufe mit veränderter Syntax ergaben, dass lange Laufzeiten auftreten, sobald innerhalb einer mit PROC FCMP generierten Funktion ein Array definiert wird, selbst wenn nachfolgend keine Werte aus dem Array in Programmschleifen abgefragt werden. Dabei erscheinen die Laufzeiten der Nutzer-definierten Funktionen, zumindest im getesteten Bereich, wenig abhängig von der Anzahl der Elemente des Arrays, während in den Makro-Varianten die Laufzeiten – wie erwartet – mit jeder Erweiterung der Feiertagsliste um zusätzliche Einträge und den damit erforderlichen zusätzlichen Durchläufen von Programmschleifen steigen.

Nutzer-definierte Funktionen mit dynamischen Arrays scheinen nach den vorliegenden Erfahrungen für eine Verwendung von Berechnungen in sehr großen Dateien weniger gut geeignet zu sein. Vermutlich werden die Werte für ein dynamisches Array einer Nutzer-definierten Funktion bei der Verarbeitung jeder Zeile einer Datendatei erneut aus der entsprechenden externen Datei mit der (Feiertags-)Werteliste eingelesen. Hingewiesen werden muss allerdings zweifellos auch darauf, dass die hier beschriebenen „Probleme" nur bei der Verarbeitung von ausgesprochen umfangreichen Daten eine wesentliche Rolle spielen. Die Nutzung von dynamischen Arrays in PROC FCMP wird also keinesfalls grundsätzlich in Frage stellen.
Auf Anregung eines KSFE-Teilnehmers wurde nach Abschluss der KSFE-Tagung eine weitere Variante zur Ermittlung von Arbeitstagen implementiert. Diese beruht auf der Möglichkeit, für die Funktion `intck` auch weitgehend beliebige Zeitintervalle in Listen definieren zu können [3]. Dabei muss die Liste obligat eine Variable mit dem Namen „Begin" enthalten, in der alle Start-Tage der Zeitintervalle abgelegt sind. Umfasst die Liste alle für eine jeweilige Auswertung relevanten Arbeitstage, kann die Zahl der Arbeitstage direkt (und ähnlich wie mit dem Parameter „weekday") mit der Funktion `intck` ermittelt werden. Mit der Zeile

```
options intervalds=(Arbeitstage=Workday_a);
```

erhält SAS die Information, dass die Datei `work.workday_a` Intervallangaben erhält. Anschließend kann intck mit dem Intervall-Parameter `"Arbeitstage"` aufgerufen werden (eine geeignete Liste von Arbeitstagen wird vom Makro `HOLIDAY` erstellt). Die Laufzeit dieser Variante entspricht etwa der Laufzeit der Nutzer-definierten Funktion. Zur Ermittlung von regional unterschiedlichen Arbeitstagen müssten unterschiedliche Listen und Aufrufe verwendet werden.

Die vollständige Syntax in Form der erwähnten SAS-Programme kann vom Autor bei Bedarf gern auf Mail-Anfrage zur Verfügung gestellt werden.

Literatur

[1] Chris Hemedinger: Calculating the number of working days between two dates. 2011. Internet (Aufruf 12.03.2015): http://blogs.sas.com/content/sasdummy/2011/05/09/calculating-the-number-of-working-days-between-two-dates/

[2] Infos und Beispiele zu PROC FCMP (Aufruf 12.03.2015): http://support.sas.com/resources/papers/proceedings13/505-2013.pdf

[3] SAS-Dokumentation (SAS 9.4) zu Funktionen allgemein sowie auch zu intck (Aufruf: 28.03.2015): http://support.sas.com/documentation/cdl/en/lefunctionsref/67398/PDF/default/lefunctionsref.pdf