Intelligente Datenverarbeitung mit PROC IML

Felix Fritz, Stephan Meyer, Christof Weinhardt
Institut für Informationswirtschaft und
Marketing (IISM)
Karlsruher Institut für Technologie
Englerstr. 14
D-76131 Karlsruhe
Felix.Fritz2@kit.edu
Stephan.Meyer@kit.edu
Weinhardt@kit.edu

Zusammenfassung

SAS/IML bietet seit den 80er Jahren mit mehr als 300 Funktionen eine umfangreiche Bibliothek zur statistischen Programmierung in SAS. In diesem Beitrag zeigen wir die Vorteile von SAS/IML im Vergleich zu klassischen DATA Steps an einem einfachen finanzwirtschaftlichen Beispiel auf. Ziel dabei ist die Berechnung einer Liquiditätskennzahl für Aktien mit Hilfe von SAS/IML Features, wie beispielsweise Vektor-Operationen, Data Loop, oder Sortierungsroutinen. Wir veranschaulichen dabei, dass neben einer einfachen Programmierung auch die Performance bei SAS/IML überzeugt.

Schlüsselwörter: PROC IML, Datenverarbeitung, Data-Loop, Vektor-Operationen, Loc, Call Sort

1 Einführung

PROC IML („Interactive Matrix Language“) ist seit SAS V6. ein wesentlicher Bestandteil der statistischen Programmierung in der SAS Landschaft. Während klassische DATA Steps am besten zum Extrahieren, Mergen und Transformieren von Daten geeignet sind, überzeugt SAS/IML bei statistischen Anwendungen. Anders als im DATA Step ist die einfachste Datenstruktur in IML keine einzelne Beobachtung sondern eine Matrix. Ein grundlegender Unterschied ist die Art der Verarbeitung bei IML. Während bei einem DATA Step alle spezifizierten Beobachtungen sequentiell durchlaufen werden, ist SAS/IML in der Lage blockweise Daten ‚in-memory‘ zu verarbeiten und daraus resultierend, einfach Berechnungen über eine variable Anzahl an Beobachtungen zu ermöglichen. Eine Vielzahl von statistischen Analysen basiert theoretisch auf Matrixoperationen, sodass SAS/IML deren Umsetzung wesentlich erleichtert. Dabei müssen zu bearbeitende Elemente stets im Arbeitsspeicher gehalten werden, was einerseits Performancevorteile bei vielen Berechnungen mit sich bringt, jedoch die maximale Datengröße durch den verfügbaren Arbeitsspeicher beschränkt. Ein vollständiger Überblick über alle Funktionsweisen von SAS/IML wird in [1] gegeben. Im Folgenden veranschaulichen wir an einem finanzwirtschaftlichen Szenario einige der Features von

SAS/IML. Die hier vorgestellten Programme wurden in SAS 9.3 (bzw. SAS/IML 9.22) erstellt und getestet. Eine Kompatibilität mit früheren Versionen von SAS muss nicht gegeben sein.

2 Anwendungsszenario

Im Bereich der Finanzmarktforschung analysierte Zeitreihen und Datensätze überschreiten oftmals den dreistelligen Gigabytebereich und machen vollständige In-memory Berechnungen unmöglich. Statistikprogramme wie Stata, SPSS, oder R können mit solch großen Datensatz wenn überhaupt nur schwer umgehen. Im folgenden Anwendungsszenario liegen alle Rohdaten als Flatfiles (csv) vor, sodass sich SAS ideal eignet diese effizient zu verarbeiten. Das folgende Beispiel zeigt ein Rohdatensample aus sogenannten Trade-and-Quote Daten von Thomson Reuters Tick History für die Deutsche Telekom AG auf dem Börsenplatz Xetra[1]:

```
DTEGn.DE,09-OCT-2013,09:21:16.118,+2,Trade,,11.15,400,11.1150,,,"            [ACT_FLAG1];TRADE            [GV1_TEXT];
[PRC_QL2]",1820580,11:21:16.000,,,,,,,09-OCT-2013,,,,,,,,,,,11.07,11.175,11.07,3014117,33501.85,,,,,,,,,0.09,,,,
DTEGn.DE,09-OCT-2013,09:21:28.159,+2,Trade,,11.15,3165,11.1150,,,"            [ACT_FLAG1];TRADE            [GV1_TEXT];
[PRC_QL2]",1820590,11:21:28.000,,,,,,,09-OCT-2013,,,,,,,,,,,11.07,11.175,11.07,3017282,33537.14,,,,,,,,,0.09,,,,
DTEGn.DE,09-OCT-2013,09:21:28.159,+2,Trade,,11.15,2954,,,,"               [ACT_FLAG1];TRADE               [GV1_TEXT];
[PRC_QL2]",1820600,11:21:28.000,,,,,,,09-OCT-2013,,,,,,,,,,,11.07,11.175,11.07,3020236,33570.08,,,,,,,,,0.09,,,,
DTEGn.DE,09-OCT-2013,09:21:31.510,+2,Trade,,11.15,1815,11.1151,,,"            [ACT_FLAG1];TRADE            [GV1_TEXT];
[PRC_QL2]",1820610,11:21:31.000,,,,,,,09-OCT-2013,,,,,,,,,,,11.07,11.175,11.07,3022051,33590.31,,,,,,,,,0.09,,,,
```

Die ersten Spalten beinhalten einen Identifier, Datum, Uhrzeit, Zeitverschiebung und Beobachtungstyp. Außerdem beinhalten die Daten eine Reihe weiterer Variablen, wodurch der Datenumfang enorm steigt. Für unseren Fall sind zwei Beobachtungstypen von Interesse: *Trade* bezeichnet eine ausgeführte Transaktionen und *Quote* stellt Preise sowie Menge der besten Kaufs- bzw. Verkaufsangebote zu einem bestimmt Zeitpunkt dar.

Probleme im Bereich der statistischen Programmierung sind neben der Größe der Rohdaten, auch die Tatsache, dass die Daten kein festes Datengitter aufweisen (siehe Zeitstempel in den obigen Daten). Mit anderen Worten, die Abstände zwischen einzelnen Einträgen sind nicht nach einem festgelegten Muster angelegt, sondern können komplett unterschiedlich sein. Zudem erfordert jede Analyse eine intensive Datenbereinigung, da Datenfehler, fehlende Werte, oder andere Unreinheiten die Datenqualität beeinträchtigen.

2.1 Marktliquiditätskennzahlen berechnen

Die Marktmikrostukturforschung befasst sich mit den tiefliegenden Verhaltensweisen von Börsen und Handelsplätzen (vgl. [3]). Dabei sind der Preisentwicklungsprozess von Aktien und die Liquditätsbereitstellung von besonderem Interesse. Liquidität bezeichnet

[1] Xetra ist ein vollelektronisches Handelssystem, das von der Deutschen Börse AG in Frankfurt betrieben wird.

die Möglichkeit große Transaktionen, schnell und mit geringen Kosten abzuwickeln. Zur Messung der Liquidität an Handelsplätzen in Bezug auf individuelle Aktien wurden diverse Liquiditätsmaße entwickelt. Zum Beispiel bezeichnet der *Quoted Spread* den (relativen) Unterschied zwischen dem aktuell besten Kauf- bzw. Verkaufspreis[2]. Niedrige Spreads sind i.A. ein Hinweis auf einen liquiden Markt. Ein weiteres Maß ist der *Realized Spread*, welcher wie folgt definiert ist:

$$\boldsymbol{Realized\ Spread_{i,t}} = \boldsymbol{D_{i,t}} * \left(\frac{\boldsymbol{Price_{i,t}} - \boldsymbol{Mid_{i,t+x}}}{\boldsymbol{Mid_{i,t}}} \right)$$

wobei $D_{i,t}$ die Richtung der Transaktion anzeigt (Kauf = 1; Verkauf = -1), $Price_{i,t}$ der Ausführungspreis ist und $Mid_{i,t}$ der Durchschnitt von Geld- und Briefkurs für Aktie *i* zum Zeitpunkt *t* ist. Zur Berechnung des *Realized Spread* wird also der zukünftige Geld- und Briefkurs zum Zeitpunkt *t+x* benötigt. Aufgrund der Datenstruktur ist jedoch der Abstand zwischen der aktuellen Beobachtung und der „zukünftigen" Beobachtung mit der benötigten Information zum Zeitpunkt *t+x* ungewiss. In Kapitel 3.1 erklären wir unser Verfahren zur Berechnung dieser Liquiditätskennzahl mit SAS/IML und im Anschluss dessen Vor- und Nachteile in Vergleich zu Base SAS.

3 Umsetzung in PROC IML

Um das in Kapitel 2 aufgezeigte Anwendungsbeispiel mit PROC IML zu implementieren werden zunächst grundlegende Funktionalitäten von PROC IML eingeführt. Im Anschluss wird der eigentliche SAS Code dargestellt und mit der alternativen Umsetzung mit Base SAS (d.h. DATA Step, PROC SORT, PROC SQL) evaluiert.

3.1 IML Grundlagen

Im Gegensatz zu zeilenbasierten DATA Steps, ist SAS/IML eine vektor- bzw. matrixbasierte Sprache und alle Operationen sind als Verknüpfung von solchen zu verstehen. Grundlegende Kenntnisse der Vektor- bzw. Matrixalgebra (z.B. Addieren und Multiplizieren von Matrizen) werden vorausgesetzt. Im Folgenden verwenden wir die Begriffe Matrix und Vektor in der Regel gleichbedeutend, insbesondere wird der Fall einer (n×1)-Matrix nicht zwangsläufig als Vektor deklariert. Dies gilt auch für PROC IML selbst, da hier alle Datenelemente (n×n)-Matrizen (wobei $n \geq 0$) sind. Es ist zu beachten, dass eine Matrix immer nur ein Datentyp, d.h. double oder character, enthalten kann.
Für die Syntax der Basisoperationen verweisen wir auf das kompakte Tip Sheet[3] aus „Statistical Programming with SAS/IML Software" [1]. Eine komplette Dokumentation liefert der offizielle „SAS/IML 9.3 User's guide" [2]. Einige, für diesen Artikel wichtige Funktionalitäten werden nun kurz vorgestellt. Ein Vorteil von Matrizen im Vergleich zu DATA Steps bzw. SAS-Tabellen ist, dass Einträge gezielt angesprochen werden können. Zum Beispiel wird durch

[2] Sogenannte Brief- bzw. Geldkurse (im Englischen Ask bzw. Bid).
[3] http://blogs.sas.com/content/iml/files/2011/10/IMLTipSheet.pdf

```
y=x[2,3];
```

der Wert in der 2. Zeile und 3. Spalte der Matrix `x` einer Matrix `y` zugewiesen. Des Weiteren kann mit Hilfe der `loc()` Funktion gezielt nach der Position von gewissen Einträgen gesucht werden. Mit

```
a=loc(x>0);
```

enthält `a` die (zeilendominierten[4]) Positionen der Einträge der Matrix `x`, die größer 0 sind. Zu beachten ist, dass der Positionsmatrix `a` nur erstellt wird wenn die Bedingung mindestens einmal erfüllt ist. Das heißt, es muss (generell) vermieden werden, nicht initialisierte Matrizen zu verarbeiten, da dies bei den meisten Operationen und Funktionen zu Fehlern führt und damit zum Abbruch der abgeschickten Befehle[5] führt.
Eine oft genutzte Routine ist die Sortierungsfunktion von SAS/IML. Mit

```
CALL SORT (matrix, by, descend);
```

kann eine Matrix nach den Spalten in Vektor `by` sortiert werden, wobei standardmäßig aufsteigend sortiert wird, außer die entsprechenden Spalten werden in `descend` angegeben. Fehlende Werte in numerischen Matrizen werden als 0 interpretiert.
Da der Fokus dieses Artikels auf der effizienten Datenverarbeitung mit SAS/IML liegt, wird zunächst das Ein- bzw. Auslesen von Daten von einer SAS Tabelle in PROC IML dargestellt. Dabei muss als Erstes PROC IML gestartet und die zu verwendende Tabelle ausgewählt werden:

```
PROC IML;
     USE table1;
     EDIT table1;
     CLOSE table1;
```

`USE` und `EDIT` erklären jeweils, dass `table1` aktiv ist, wobei letzteres zusätzlich auch Veränderungen an der aktiven Tabelle erlaubt. Solange PROC IML nicht über den Befehl `QUIT;` beendet wird, bleibt das Programm und alle Elemente „interaktiv“ verfügbar. Wird PROC IML geschlossen, werden alle nicht gespeicherten Matrizen[6] gelöscht.
Aus einer aktiven Tabelle können nun Daten mit der `read` Anweisung in IML Matrizen geladen werden. Die Syntax von `read` erlaubt verschiedene Arten um Daten aus der Tabelle auszuwählen und in Matrizen zu laden. Die Grundform ist

```
READ range VAR operand WHERE(expression) INTO name;
```

wobei `operand` die zu lesenden Variablen enthält, `expression` Bedingungen für die zu lesenden Beobachtungen, ggf. auch aus nicht einzulesenden Variablen, bezeichnet und `name` der Name der Matrix ist. Wird kein Name angegeben werden Matrizen anhand der Variablennamen der Tabelle erstellt. `RANGE` gibt eine aus fünf Arten an Beobachtungen (Zeilen) der Tabelle eingelesen werden sollen. `ALL` liest alle Beobachtungen ein, sofern

[4] D.h. die Einträge einer Matrix werden zeilenweise durchnummeriert.
[5] Sollte es nicht gewährleistet sein, dass ein zu verwendendes Element `x` zu einem Zeitpunkt tatsächlich existiert, kann dies z.B. durch `if nrow(x)>0 then…` kontrolliert werden.
[6] Um Matrizen dauerhaft zu Speichern vergleiche [2], Kapitel 17.

die `WHERE` Bedingung erfüllt ist. In unserem Fall ist vor allem die `POINT` Anweisung von Interesse, da diese gezieltes Einlesen von Beobachtungen erlaubt. Zum Beispiel lädt

```
READ point {10,12,100} VAR _ALL_ into x;
```

die Beobachtungen 10, 12 und 100 aller Variablen der aktiven Tabelle in die Matrix `x`. Weitere Möglichkeiten Daten in Matrizen zu laden werden im SAS/IML Handbuch, Kapitel 7, beschrieben.
Der Befehl `FIND` kann benutzt werden, um die Position (Zeile) von Beobachtungen innerhalb einer SAS Tabelle zu finden. Die Syntax lautet

```
FIND range WHERE(expression) INTO matrixname ;
```

und ergibt sich analog zur `READ` Anweisung. Durch eine Kombination von `FIND` und `READ` können unter Anderem gezielt (variable) Zeilenbereiche sequentiell aus einer Tabelle geladen werden (Beispiel folgt in Abschnitt 3.2).
Die Rückgabe von IML Matrizen in SAS Tabellen kann über verschiedene Wege erfolgen und hängt zum Teil von der Problemstellung ab. Der erste Weg ist die Erstellung einer neuen SAS Tabelle aus (ggf. mehreren) Matrizen. Die Syntax lautet

```
CREATE table2 VAR operand;
APPEND;
CLOSE table2;
```

oder

```
CREATE table2 FROM matrix;
Append FROM x;
CLOSE table2;
```

Im ersten Fall bezeichnet `operand` die zu schreibenden Variablen, wobei Matrizen zeilendominiert behandelt werden. Sollen Variablen für jede Spalte einer Matrix erstellt werden muss der zweite Ansatz gewählt werden, wobei jeweils nur eine Matrix angegeben werden kann[7]. Alternativ zur Erstellung einer neuen Tabelle können bestehende Tabellen verändert werden, wenn diese mit `EDIT` aufgerufen wurden. Beobachtungen einer Tabelle können mit

```
DELETE range WHERE(expression);
```

gelöscht werden, wobei `range` und `expression` analog zu oben zu verwenden sind. Mit den gleichen Argumenten können mit

```
REPLACE range VAR operand WHERE (expression);
```

[7] Das heißt, mehrere Matrizen bzw. Vektoren müssen zuvor konkateniert werden. Die Variablennamen können über die Spaltenlabels der Matrix angepasst werden (vgl. [2], Kapitel 5).

Variablen verändert werden. Dazu müssen die Größe des zu ersetzenden Variablenbereichs (der SAS Tabelle) sowie der Name der Variable und der IML-Matrix übereinstimmen. Besonders praktisch ist die `POINT` Anweisung, womit zum Beispiel durch

```
EDIT tabelle3;
obs=12;
spalte5=0;
REPLACE POINT obs VAR {spalte5};
```

die 12. Beobachtung der Variable `spalte5` in `tabelle3` durch 0 ersetzt werden kann. PROC IML gibt für jede Ersetzung einen Hinweis bzgl. der Anzahl veränderter Beobachtungen aus. Diese Ausgabe kann bei vielen Ersetzungen, z.B. in einer Schleife, die Ausführung erheblich verlangsamen. Benachrichtigen können über die SAS Optionen (`options nonotes; …; options notes;`) deaktiviert bzw. aktiviert werden.
Als letzte SAS/IML Funktionalität wird die sogenannte „Data-loop“ eingeführt. Mit dieser können Daten in einer Schleife sequentiell in IML Matrizen geladen werden. Die Syntax lautet

```
DO DATA variable=start TO stop;
```

wobei die Schleifenparameter optional sind. Die Schleife wird generell verlassen, sobald eine sogenannte „end-of-file“ Bedingung eintritt, d.h. wenn das Tabellenende erreicht wurde. Ein einfaches Beispiel verdeutlicht die Funktionsweise:

```
USE tabelle4;
DO DATA i=1 TO 3;
     READ next 5 INTO x;
END;
```

Jeweils fünf Beobachtungen aller Variablen von `tabelle4` werden sequentiell in Matrix `x` geladen. Sollte `tabelle4` nur 8 Beobachtungen enthalten, wird die Schleife nach der zweiten Iteration automatisch verlassen. Alternativ hätte die Zählanweisung auch weggelassen werden können, wodurch die Schleife nachdem alle Beobachtungen gelesen wurden beendet wird.

Generell muss beim Arbeiten mit SAS/IML beachtet werden, dass alle Elemente (Matrizen, Vektoren, etc.) im Arbeitsspeicher gehalten werden bis PROC IML beendet wird oder ein Element mit `FREE matrixname` freigegeben wird. Matrizen können auch auf der Festplatte abgelegt und später wieder aufgerufen werden (vgl. Fußnote 7). Daher muss stets darauf geachtet werden, dass die einzulesenden Daten in den Arbeitsspeicher passen und ggf. Speicher für weitere Berechnungen zur Verfügung steht. Als Daumenregel haben sich 9 Byte pro Matrixeintrag als sinnvoll erwiesen. Das heißt bei einem Rechner mit 8 GByte RAM dürfen maximal ca. 888 Mio. Matrizeneinträge aktiv sein[8].

[8] Unter Windows (32/64bit) kann SAS/IML 9.22 maximal nur 2 GByte RAM nutzen. Unter Linux wird der gesamt Arbeitsspeicher, welcher der SAS Instanz beim Start zugewiesen wurde (vgl. Option `memsize`), verwendet.

3.2 Code mit Erklärung

In diesem Abschnitt soll das in Abschnitt 2 aufgezeigte Anwendungsproblem mit SAS/IML gelöst werden. Ziel ist es, aus hochfrequenten Finanzzeitreihen Liquiditätskennzahlen zu berechnen, die als Input zwei Beobachtungen (Zeitpunkte) benötigen. Allerdings ist der Abstand (in Beobachtungen) der beiden Inputs aufgrund der Datenstruktur variabel, wodurch ein einfacher Merge unbrauchbar wird.
Wegen der enormen Größe der Ausgangsdaten werden Daten pro Tag und Aktie geladen. Die Ausgangstabelle muss gewisse Vorbereitungen durchlaufen haben, die zum Beispiel im Zuge des Datenimports in einem DATA Step durchgeführt werden können. Insbesondere müssen die Variablen analog zum PROC IML Code benannt sein, ein Indikator (`newday`) für den Beginn eines neuen Tages eingeführt werden und die Variable `realizedspread15`, in welche die berechneten Werte geschrieben werden sollen, muss in der Tabelle initialisiert werden. Erläuterungen der einzelnen Schritte der PROC IML Umsetzung sind als Kommentar in den Programmcode eingefügt.

```
PROC IML;
     EDIT datentabelle NOBS n;

*Daten werden pro Tag eingelesen.
     FIND all WHERE( newday=1 ) INTO newd;
     newd=newd//(n+1);
     DO DATA i=1 TO nrow(newd)-1;
          readpos=newd[i,1]:(newd[i+1]-1);
          READ POINT readpos VAR {time msec type price bid_price
                              ask_price buysell newday rspread15};

*Finde Positionen von Trades bzw. Quotes;
          q=loc(type="Quote")`;
          tr=loc(type="Trade")`;

*Erstelle Vektor der um 15min (15*60sec) verschobenen Quote-
Zeitpunkte, Codiere  Quotes mit 0 (3.Spalte). Die 4.Spalte enthält
ursprüngliche Quote-Position;
          timeq15=(time[q,]-(15*60)) || msec[q,] ||
                   j(nrow(time[q,]),1,0) || q;
          dummy=j(nrow(tr),1,.);

*Analog für Trades;
          timetr=time[tr,] || msec[tr,] ||  j(nrow(tr),1,1) ||
                   j(nrow(tr),1,.);

*Falls möglich, werden nicht mehr benötigte Elemente freigegeben;
          FREE id dummy q;

*Die neuen Zeitstempel werden zusammengeführt und sortiert;
          timeall = timeq15 // timetr;
          FREE timeq15 timetr;
          CALL sort(timeall, {1,2});
```

```
*Finde ersten Trade;
          z=min(loc(timeall[,3]=1))-1;
          if z <=0 then z=1;

*Initialisiere (ursprüngliche) Position des ersten 15min Quotes;
          curid=timeall[min(loc(timeall[,3]=0)),4];

*Bestimme für jeden Trade die Position des zugehörigen Quotes 15min
später;
          DO l=z TO max(loc(timeall[,3]=1));
               IF timeall[l,3]=0 THEN curid=timeall[l,4];
                    ELSE timeall[l,4]=curid;
          END;

*Ergebnisvektor der Positionen;
          quoteid=timeall[loc(timeall[,3]=1),4];
          FREE timeall;

*Berechne aktuellen und zukünftigen Mid-Preis für alle Trades;
          mid15=0.5* (ask_price[quoteid,]+bid_price[quoteid,]);
          mid=0.5* (ask_price[tr,]+bid_price[tr,]);

*Berechne Realized Spread;
          rspread15=buysell[tr,]# (price[tr,]-mid15) / mid;

*Erstelle Vektor der Positionen der Trades in der SAS-Tabelle;
          towrite=readpos[,tr];
          REPLACE POINT towrite VAR {rspread15};
     END;
QUIT;
```

Eine Besonderheit dieser Vorgehensweise ist, dass zum einen die im Datensatz vorhandene Datenstruktur, d.h. Beobachtungen pro Tag pro Aktie, durch die Data-Loop ausgenutzt wird und zum anderen nur ein Teil der Variablen in der Tabelle bearbeitet bzw. neu geschrieben wird. Sollte solch eine Reduktion der Datengröße nicht möglich sein, muss stets darauf geachtet werden, dass der verfügbare Arbeitsspeicher ausreicht, um alle benötigten Daten einzulesen.

3.3 Alternative Umsetzung mit Base SAS und Vergleich der Performance

Die im letzten Abschnitt vorgeführte Berechnung des *Realized Spreads* kann natürlich auch mit Hilfe üblicher Base SAS Operationen erreicht werden. Eine Umsetzung könnte zum Beispiel wie folgt aussehen, wobei wir hier nur die verwendeten Schritte angeben, statt der kompletten Codes:

1. Erstelle mit einem DATA Step eine Kopie der ursprünglichen Daten mit einem um 15 Minuten verschobenen Zeitstempel, sowie einem Indikator.

2. Füge die Kopie an die ursprüngliche Tabelle mit PROC Append an.

3. Sortiere die zusammengeführten Daten mit PROC Sort nach Zeitpunkt und Aktie.

4. Ziehe in einem DATA Step die benötigten Werte der Variable ($\boldsymbol{Mid_{i,t+x}}$) der markierten Zeilen in die entsprechenden unmarkierten Zeilen der ursprünglichen Daten. Lösche außerdem die markierten Zeilen und berechne den *Realized Spread*.

Tabelle 1: Eigenschaften der Datensätze und Performance-Ergebnisse.

		Sample 1	Sample 2	Sample 3	Sample 4	Sample 5
	Größe [MByte]	1456	4209	7625	13892	29220
	Beobachtungen [Mio.]	7,549	21,817	39,528	72,014	151,474
PROC IML	realtime [Sek]	25	51	183	366	783
	user cpu time [Sek]	14	37	79	143	309
	system cpu time [Sek]	7	14	41	77	168
	memory [Mbyte]	20	20	22	42	42
Base SAS	realtime [Sek]	69	389	720	1287	2766
	user cpu time [Sek]	18	80	156	279	597
	system cpu time [Sek]	29	119	220	375	780
	memory [Mbyte]	4271	6743	6741	6743	6741
	Zeitersparnis IML	64%	87%	75%	72%	72%

Diese Vorgehensweise stellt vermutlich die „naivste“ Möglichkeit dar, das vorliegende Problem zu lösen. Es ist ebenfalls offensichtlich, dass auch mit Base SAS effizientere Wege existieren, die natürlich mehr Programmieraufwand benötigen. Trotzdem möchten wir den beschriebenen Ansatz als Benchmark für die vorgestellte Implementierung mit SAS/IML verwenden. Der beispielhafte Performancevergleich zwischen den beiden Implementierungen erfolgt anhand von fünf Datensätzen mit einer Größe von ca. 1,5 GByte bis 30 GByte (siehe Tabelle 1). Es wird die 64bit SAS 9.3 Version auf einem virtualisierten Linux Server[9] verwendet. Die SAS Systemoptionen bei Programmaufruf bzgl. Arbeitsspeichernutzung sind wie folgt: `-maxmemsize 8G -maxmemquery 8G -sortsize MAX`. Details zu den Datensätzen sind in Tabelle 1 aufgelistet.
Abbildung 1 zeigt die Rechendauer der beiden alternativen Implementierungen, jeweils angewendet auf die fünf Beispiel-Datensätze. Es stellt sich heraus, dass die Implementierung mit SAS/IML deutlich schneller ist und die Rechendauer auch bei großen Datenmengen nicht explodiert. Tabelle 1 beinhaltet einige Angaben des System-Logs aus jedem Durchlauf. Da es sich um nur je einen Durchlauf pro Implementierung handelt, können natürlich gewisse Schwankungen enthalten sein – die Grundaussage ist jedoch

[9] Es handelt sich um einen Dell Power Edge Server mit zwei Intel Xeon E5640 Prozessoren und 24 GByte RAM, wobei die CentOS 6.5 Virtualisierung auf 12 Kernen und 14GByte RAM läuft. Die Daten befinden sich auf einer 2Tbyte Festplatte von Western Digital (SAS 6Gb/s, 32Mbyte Cache, 7200Rpm).

eindeutig: Die SAS/IML Implementierung ist in allen Belangen überlegen. Insbesondere der RAM-Bedarf scheint überraschend zu sein, da IML eigentlich alle Datenelemente im Speicher halten muss. Der Grund hierfür liegt in der geschickten Nutzung der vorliegenden Datenstruktur (Einlesen pro Tag und Aktie) sowie an der Tatsache, dass nur die benötigten Variablen eingelesen werden, statt die gesamte Tabelle zu verarbeiten. Der hohe Speicherbedarf der Base SAS Implementierung ist durch den PROC Sort Schritt zu erklären, was insbesondere bedeutet, dass sich der Vorsprung der IML Implementierung noch weiter ausbauen sollte, falls weniger Arbeitsspeicher zur Verfügung steht. Der hier gezeigte Vorteil von bis zu 87% weniger Rechenzeit ist jedoch bereits bei unserer Hardware-Konfiguration enorm. Durch den hohen I/O Bedarf von Base SAS bei großen Datensätzen, profitiert die entsprechende Implementierung stärker von einem höheren I/O Durchsatz z.B. durch RAIDs oder SSD-Festplatten.

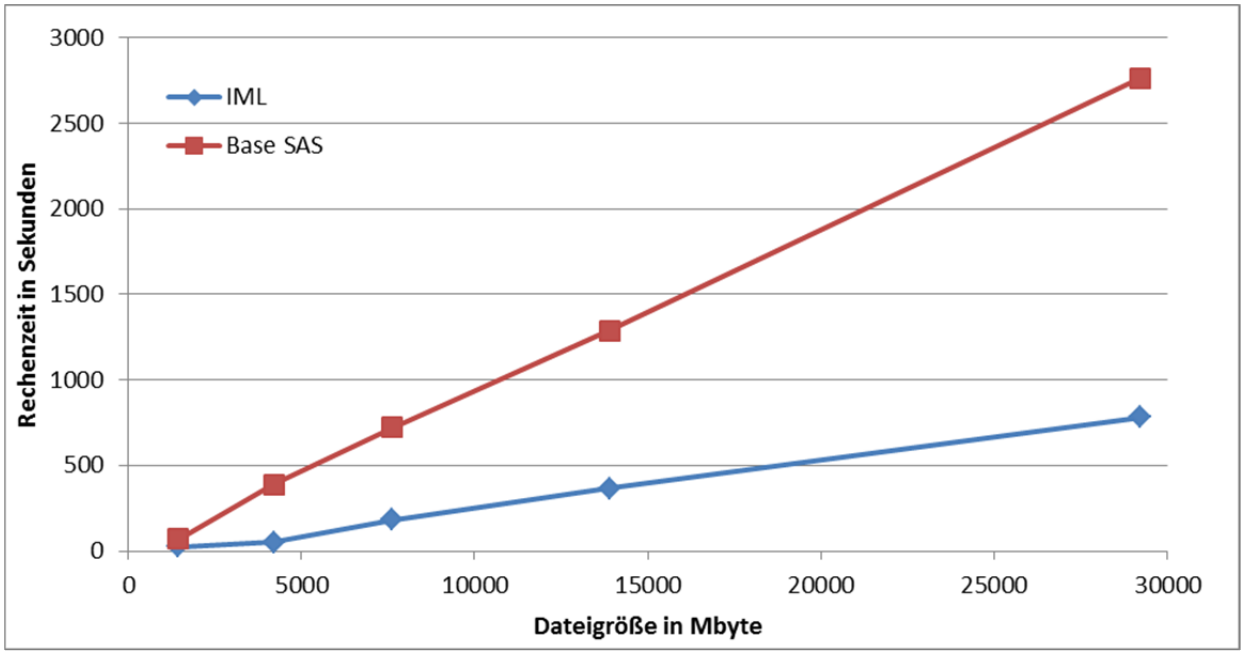

Abbildung 1: Vergleich der Rechendauer

3.4 Vor- und Nachteile von SAS/IML und weitere Funktionalitäten

Abschließend möchten wir die vorgestellten SAS/IML Funktionalitäten bzgl. ihrer Vor- und Nachteile diskutieren, sowie gewisse Limitierungen aufzeigen.

Wie bereits mehrfach erwähnt, ist der größte Vorteil von SAS/IML – neben dem eigentlichen Zweck der statistischen Programmierung – dass die verwendeten Daten relativ einfach gezielt ausgewählt werden können und anschließend komplett im Arbeitsspeicher gehalten werden. Gerade bei „Random Access“ oder Sortierungen ist dieser Vorteil enorm. Auch die zugrundeliegende Matrixstruktur von SAS/IML kann diverse Probleme programmiertechnisch Vereinfachen. Das gezielte und trotzdem dynamische Zusammenführen von Werten aus vielen Tabellen kann so zum Beispiel stark vereinfacht werden. Wie hier gezeigt, lassen sich einzelne Variablen und Bereiche unkompli-

ziert auslesen, verknüpfen und zurückschreiben. Ähnliches lässt sich auch mit Base SAS erreichen – zum Beispiel durch geschicktes Teilen, Bearbeiten bzw. Zusammenführen von Tabellen – jedoch ist aus unserer Sicht der Aufwand und die Komplexität der entsprechenden Programmierung größer. Eine von uns getestete Implementierung, die im Gegensatz zum „naiven" Ansatz stark optimiert wurde, ergaben „nur" ca. 40 Prozent Rechenzeitersparnis, im Gegensatz zu über 80 Prozent mit SAS/IML.

Andererseits zeigen sich beim Einsatz von SAS/IML zur Datenverarbeitung auch Nachteile. Beispielsweise ist das klassische Zusammenverführen („Mergen") von Datensätzen mit SAS/IML eher schwierig und bringt keinen Performance-Vorteil. Generell lässt sich sagen, das bei „Standard-Routinen" wie Zusammenführen von Tabellen, Löschen von Beobachtungen oder (zeilenweise) Berechnung neuer Variablen kein Vorteil erzielt werden kann, da auch SAS/IML in diesen Fällen eine Tabelle komplett lesen und schreiben muss.

Der Vorteil der In-memory Bearbeitung von Daten durch SAS/IML, kann natürlich gleichzeitig ein großer Nachteil werden. Typische Anwendungsszenarien von SAS beinhalten oft riesige Datenmengen, die den verfügbaren Arbeitsspeicher der meisten Systeme um ein Vielfaches übersteigen. Die verwendete „Data-Loop" kann dieses Problem vermeiden, falls eine innere Datenstruktur vorhanden ist, die es erlaubt, die Daten sequentiell zu verarbeiten (hier Beobachtungen pro Tag und Aktie). Sollte der Anwendungsfall es erfordern die komplett Daten gleichzeitig zu benötigen, ist der verfügbare Arbeitsspeicher eine starke Begrenzung und der Einsatz von Base SAS Methoden ist zwingend erforderlich.

4 Fazit

In diesem Artikel haben wir dargestellt, wie die SAS/IML Programmiersprache zur Verarbeitung von großen Datenmengen verwendet werden kann. Anhand des Anwendungsszenarios, in dem Kennzahlen aus sehr großen Finanzdaten berechnet werden sollen, wurde gezeigt, wie große Datensätze mit Hilfe von In-memory Berechnungen in SAS/IML performant durchgeführt werden können. Im Gegensatz zu Funktionalitäten von Base SAS (DATA Step, PROC SQL, PROC SORT, etc.) wird durch gezieltes Einlesen, Bearbeiten und Zurückschreiben von Variablen bzw. Beobachtungen ein Performancevorteil von bis zu 87 Prozent erzielt. Wir vermuten, dass diese Vorteile auf schwächeren Systemen noch stärker zum Tragen kommen. Gerade bei Standard-PCs oder Laptops die keine hohen I/O-Durchsatzraten durch Festplatten-RAIDs haben, kann eine geschickte Nutzung von In-memory Berechnungen größere Vorteile bringen – bei gleichzeitig relativ einfacher und intuitiver Programmierung. Auf Hochleistungsservern mit SPDE-Libraries oder SAS High-Performance Analytics Systemen wird vermutlich kein Vorteil erzielt werden, da der I/O-Durchsatz bereits sehr hoch ist bzw. Daten sowieso In-memory verarbeitet werden.

Abschließend ist zu erwähnen, dass wir die aufgezeigten Möglichkeiten in SAS/IML nicht als Ersatz für Base SAS verstehen, sondern als nützliche Ergänzung. Speziell für Probleme, die mit Base SAS nur umständlich (z.B. über Workarounds mittels Makro-Programmierung) oder ineffizient gelöst werden können, bietet SAS/IML zum Teil passende Lösungswege. Wir empfehlen daher auch SAS-Anwendern, die sich nicht mit komplexen statistischen Methoden und Berechnungen beschäftigen, die grundlegende Funktionsweise von SAS/IML kennenzulernen, um diese als Ergänzung zur intelligenten Datenverarbeitung zu nutzen.

Danksagung:
Wir danken der Börse Stuttgart für eine finanzielle Unterstützung.

Literatur

[1] Rick Wicklin, Statistical Programming with SAS/IML Software, 2010, SAS Institute Inc., Cary, NC, USA

[2] SAS Institute Inc., SAS/IML® 9.3 User's Guide, 2011, Institute Inc., Cary, NC, USA

[3] Harris, L., Trading and Exchanges, 2003, Oxford University Press, New York.