Bewertung von SMR-Produkten

Martin Vonwald

November 1999

Bewertung von SMR-Produkten

eingereicht von

Martin Vonwald

Diplomarbeit

zur Erlangung des akademischen Grades
Magister der Sozial- und Wirtschaftswissenschaften
(Mag. rer. soc. oec.)

Sozial- und Wirtschaftswissenschaftliche Fakultät
Universität Wien
Technisch - Naturwissenschaftliche Fakultät
Technische Universität Wien

Studienrichtung: Wirtschaftsinformatik

Betreuer/Begutachter:
Ao. Univ.-Prof. Dr. Stefan Pichler

Wien, im November 1999

Vorwort

An dieser Stelle möchte ich mich bei meinem Diplomarbeitsbetreuer Prof. Stefan Pichler bedanken, der mit seinen wertvollen Anregungen entscheidend zum guten Gelingen dieser Arbeit beigetragen hat. Ein weiteres Dankeschön geht an meine Korrekturleser Michaela Escuyer und Karl Selitsch, welche dieser Arbeit den letzten Schliff gegeben haben. Nicht zuletzt gebührt den Mitgliedern der LyX-Usergroup und natürlich den LyX-Entwickler mein Dank, die es mir ermöglichten diese Arbeit in der vorliegenden Form umzusetzen.

Martin Vonwald

Inhaltsverzeichnis

1. Einführung **1**

1.1. Motivation 1

1.2. SMR und davon abhängige Finanzinstrumente 3

1.2.1. Sekundärmarktrendite 3

1.2.2. SMR-Floater 5

1.3. Literaturüberblick 6

2. Bewertungsmodell **7**

3. Programmbedienung **11**

3.1. Assistent 11

3.1.1. Kalibrierung 12

3.1.2. Bewertung 13

3.2. Makros 14

4. Programmbeschreibung **17**

4.1. Klassen 17

4.1.1. Klasse „Spotrates“ 17

4.1.2. Klasse „CMT“ 19

4.1.3. Klasse „SMR“ 21

4.1.4. Klasse „SMRRegression“ 22

4.2. Makros 24

4.2.1. SMRKalibrierung 24

4.2.2. SMRBewertung 25

4.2.3. SMRDetailBewertung 26

4.3. Beispiel 27

4.3.1. Kalibrierung 27

4.3.2. Bewertung 31

5. Dateiformate **33**

5.1. Zinssätze 33

5.2. SMR 35

5.3. Modellparameter 36

A. Quellcode **37**

A.1. Module 37

A.1.1. Makros 37

A.1.2. Zeichenketten 43

A.2. Klassenmodule 44

A.2.1. SpotRates 44

A.2.2. CMT 55

A.2.3. SMR 58

A.2.4. SMRRegression 62

1. Einführung

1.1. Motivation

Eine der bedeutsamsten Gruppen von Zahlungsströmen im Kredit- und Anleihenbereich sowie auf den stark wachsenden Swapmärkten ist die Gruppe der Floater. Dies sind Zahlungsströme, deren Kupon im Gegensatz zu Anleihen an einen bestimmten *variablen* Zinssatz gebunden sind. Trotz der großen Bedeutung von Floatern existiert jedoch relativ wenig Literatur zu diesem Thema (vgl. [3], [13]).

Eine speziell für den österreichischen Kapitalmarkt wichtige Gruppe von Finanztitel sind sog. „SMR-Floater“. Dies sind Floater mit Bindung an einen Kapitalmarktzinssatz mit variabler Laufzeit, einer sog. Sekundärmarktrendite (kurz: SMR). Eine SMR kann im wesentlichen als gewichteter Durchschnitt von Endfälligkeitsrenditen von Anleihen eines bestimmten Marktsegments betrachtet werden (vgl. Abschnitt 1.2.1), wobei jedoch zu beachten ist, dass sich die Zusammensetzung des Marktsegments häufig ändern kann, da immer wieder Anleihen auslaufen oder neue Anleihen emmitiert werden. Vor allem diese Eigenschaft macht eine Bewertung von SMR-abhängigen Produkten problematisch.

Um SMR-Produkte bewerten zu können, muss die SMR entsprechend modelliert werden. Grundsätzlich gibt es hierzu zwei einfache Ansätze, die jedoch beide mehr oder weniger gravierende Nachteile aufweisen:

- Eine endogene Modellierung der SMR basierend auf einem Zinsstrukturmodell.

  Dabei trifft man die Annahme, dass bereits zum Bewertungszeitpunkt alle zukünftigen Zahlungsströme und Umlaufvolumina aller Anleihen bekannt sind. Dies ist jedoch ausgesprochen problematisch, wenn man bedenkt, dass jederzeit neue Anleihen emittiert werden können und diese dann in die Berechnung der SMR eingehen.

  Weiters nimmt man bei diesem Ansatz ebenfalls an, dass die Preise aller Anleihen ihren arbitragefreien Werten entsprechen. Abgesehen von Bid-Ask-Spreads, die dazu führen, dass die Marktpreise nicht exakt den theoretischen Preisen entsprechen, ist der österreichische Kapitalmarkt unvollständig. Dies führt dazu, dass abweichende Bewertungen nicht arbitriert werden können.

- Modellierung der SMR als exogenen stochastischen Prozess.

  Der gravierendste Nachteil dieses Ansatzes liegt darin, dass man entweder einen sehr einfachen Prozess verwendet und dabei ausgesprochen unrealistische Annahmen treffen muss (z.B. Unkorreliertheit zwischen SMR und Zinsstruktur), oder aber man bedient sich eines aufwendigeren Prozesses und hat dann Probleme bei der Prozessspezifikation und der Parameterschätzung zu lösen. Ein weiterer Nachteil liegt darin, dass der überwiegende Teil der Bewertung von Parametern abhängt, die an Preise von SMR-abhängigen Instrumenten kalibriert werden müssen, wodurch sich eine relativ hohe Ungenauigkeit ergeben kann.

Ein weiterer Ansatz, der erstmals in Pichler [13] vorgestellt wurde, teilt in der Modellierung die SMR in einen endogenen Teil und einen exogenen Fehlerprozess auf. Im endogenen Teil werden Änderungen der SMR durch Änderungen von mehreren Constant Maturity Treasury Rates (CMT) beschrieben. Eine CMT ist eine Par-Coupon Yield, welche aus den Preisen von Staatsanleihen mit standardisierten Laufzeiten ermitteln wird (vgl. [2] und Abschnitt 4.1.2). Der exogene Fehlerprozess hingegen beschreibt die Abweichungen der modellmäßigen SMR von der realen SMR. Im Gegensatz zur Modellierung der SMR als exogenen stochastischen Prozess ist bei der Bewertung nur ein sehr kleiner Teil des Barwertes von zu kalibrierenden Parametern abhängig.

Der Vorteil dieses Ansatzes ist es, die einzelnen Kuponzahlungen in einen auf herkömmliche Weise hedgebaren Teil und einen Fehlerteil zu trennen. Somit wäre es möglich, große Teile von SMR-Produkten durch Zerobonds nachzubilden. Die verbleibenden relativ kleinen Abweichungen werden durch den Fehlerprozess beschrieben und müssten durch andere SMR-Produkte gehedgt werden.

Genau diesem Ansatz widmet sich auch diese Arbeit. Zu Beginn wird eine zusammenfassende Darstellung des Bewertungsmodells gegeben und anschließend wird eine Softwareapplikation vorgestellt, welche auf Basis dieses Modells entwickelt wurde. Da es sich hierbei um die erstmalige Umsetzung dieses Modells in Form eines konkreten Bewertungstool handelt, wurde auch Bedacht auf eine sehr genaue Dokumentation gelegt. Gerade auch die vielfältigen Anwendungsmöglichkeiten eines solchen Tools, wie z.B.

- Bewertung von SMR-abhängigen Anleihen und Krediten,
- Bewertung von SMR-abhängigen Leasing- und Mietverträgen oder
- Risikobeurteilung von SMR-abhängigen Kapitalmarktfloatern zwecks Gesamtbanksteuerung

waren ein ausgesprochen großer Anreiz für diese Arbeit. Bei der Entwicklung mit Hilfe von Microsoft Excel 97 wurde vor allem auf eine einfache Bedienung Wert gelegt ohne jedoch die Flexibilität einzuschränken. Um den universellen Einsatz zu gewährleisten, wurden die eigentlichen Berechnungsroutinen in Form eines Addin realisiert. Die Benutzerschnittstelle wurde in zwei Bereiche aufgeteilt:

- Ein dialoggesteuerter Assistent führt den Anwender durch den Modellkalibrierungsvorgang und den darauffolgenden Bewertungsvorgang.
- Mehrere Excel-Makros erlauben die simultane Kalibrierung verschiedener Modelle und die Bewertung mehrerer SMR-Floater innerhalb einer Excel-Arbeitsmappe.

Die Kalibrierung und Bewertung erfolgt in beiden Fällen in zwei getrennten Abläufen, wobei die Modellparameter für spätere Bewertungsvorgänge gespeichert werden können. Die zur Kalibrierung des Modells notwendigen Spotrate- und SMR-Zeitreihen können einerseits in einem programmeigenen Format andererseits im Format der ProfitLine-Software der Österreichischen Kontrollbank vorliegen.

Im ersten Kapitel werden die Begriffe SMR und SMR-Floater vorgestellt und erläutert. Außerdem wird noch ein kurzer Literaturüberblick gegeben. In Kapitel 2 wird das verwendete Bewertungsmodell im Detail vorgestellt. Kapitel 3 widmet sich dann dem erstellten Bewertungstool. Es werden die zwei unabhängigen Programmmodule vorgestellt und deren Bedienung erläutert. Der Aufbau der einzelnen Programmteile wird in Kapitel 4 beschrieben. Außerdem wird anhand eines konkreten Zahlenbeispiels der genaue Ablauf des Kalibrierungs- und Bewertungsvorganges gezeigt. Abschließend wird dann in Kapitel 5 auf die möglichen Dateiformate eingegangen, in welchen die notwendigen Daten (z.B. SMR- oder Spotrates-Zeitreihen) vorliegen können.

1.2. SMR und davon abhängige Finanzinstrumente

1.2.1. Sekundärmarktrendite

Eine Sekundärmarktrendite (SMR) ist ein Kapitalmarktzinssatz mit variabler Laufzeit, wobei dieser als gewichteter Durchschnitt von Endfälligkeitsrenditen von bestimmten Gruppen von Anleihen betrachtet werden kann. Die Berechnung der SMR ist von folgenden Punkten abhängig (vgl. [13]):

- Welche Gruppe von Anleihen wird verwendet?

  Neben der Währung und der Verzinsungsform (variabel-/festverzinst) spielt hierbei auch die Bonitätsklasse der Schuldner eine Rolle. Außerdem kann eine Unterscheidung auch nach Restlaufzeit und eventuell vorhandenen Kündigungsrechten erfolgen.

- Wie wird die Endfälligkeitsrendite berechnet?

  Eine weit verbreitete Methode zur Bestimmung der Endfälligkeitsrendite ist die Verwendung der Effektivverzinsung. Diese beruht auf der Annahme, dass alle innerhalb der Laufzeit anfallenden Zahlungen zu einem bestimmten fixen Zinssatz wiederveranlagt werden können.

  Die österreichische Kontrollbank hingegen verwendet zur Berechnung der SMR die sog. „reale“ Rendite, die einem fix vorgegebenen Zinssatz entspricht (7,5% oder 8%).

- Wie werden die Endfälligkeitsrenditen gewichtet?

  Neben der Gewichtung nach Börsenumsatz (Umsatzgewichtung) kommt auch eine Gewichtung nach dem ausstehenden Nominale (Umlaufgewichtung) zur Anwendung.

Im folgenden wird auf die Berechnungsmethode der österreichischen Kontrollbank eingegangen. Diese nimmt folgende Gruppeneinteilung vor:

- Anleihengruppen

  Die Anleihen werden in 15 Gruppen aus festverzinsten Emmisionen, 10 Gruppen aus Kapitalmarktfloatern und 7 Gruppen aus Geldmarktfloatern eingeteilt.

- Restlaufzeit

  Neben einer SMR für alle Restlaufzeiten wird auch eine SMR für Restlaufzeiten größer als ein Jahr und eine SMR für Restlaufzeiten größer als drei Jahre berechnet.

Für die Berechnung der SMR werden ausschließlich Tagesschlusskurse der Wiener Börse herangezogen. Die Österreichische Kontrollbank verwendet für die Berechnung der Endfälligkeitsrendite folgenden Algorithmus (vgl. [12]):

- Berechnung der „realen“ Renditen $Y_{0,075}$ und $Y_{0,08}$ für jede Anleihe in der betrachteten Gruppe:

$$Y_{k^*i} = \left(\frac{\sum_{j=1}^{m_i} Z_{ij} \cdot (1+k^*)^{T_{im_i}-T_{ij}}}{PV_i} \right)^{\frac{1}{T_{im_i}}} - 1 \tag{1}$$

  Y_{k^*i} Reale Rendite für Anleihe i und Wiederveranlagungssatz k^* mit $k^* = 0,075$ bzw. $k^* = 0,08$
  m_i Anzahl der Zahlungen der Anleihe i
  Z_{ij} Zahlungen der Anleihe i; $j = 1, \ldots, m_i$
  T_{ij} Zahlungszeitpunkte der Anleihe i; $j = 1, \ldots, m_i$
  PV_i Marktwert der Anleihe i (entspricht dem Börsenkurs inklusive Stückzinsen). Ist der Börsenkurs nicht beobachtbar, wird der letzte beobachtete Börsenkurs herangezogen.

- Umlaufgewichtung über die gesamte Gruppe:

$$Y_{k^*} = \frac{\sum_{i=1}^{I} Y_{k^*i} \cdot U_i}{\sum_{i=1}^{I} U_i} \tag{2}$$

  Y_{k^*} Umlaufgewichteter Durchschnitt der „realen“ Renditen mit Wiederveranlagungssatz k^* innerhalb der Gruppe.
  I Anzahl der Anleihen innerhalb der betrachteten Gruppe.
  U_i Noch nicht getilgtes Nominale der Anleihe i.

- Berechnung der SMR:

$$SMR = \frac{Y_{0,08} \cdot 0,08 - Y_{0,075} \cdot 0,075}{0,08 - 0,075 + Y_{0,08} - Y_{0,075}} \tag{3}$$

Bei der letzten Formel erkennt man, dass die SMR ein nichtlinear gewichteter Durchschnitt der „realen“ Renditen $Y_{0,075}$ und $Y_{0,08}$ ist.

1.2.2. SMR-Floater

Kapitalmarktfloater mit Bindung an eine SMR bezeichnet man als SMR-Floater. Die wichtigsten Unterscheidungsmerkmale werden im folgenden angeführt:

- Welche SMR dient als Referenzzinssatz?
- Wann sind die Anpassungszeitpunkte und welche Regeln werden verwendet?

  Falls beides bei Vetragsabschluss genau festgelegt wird, spricht man von einer fixen Zinsbindung, anderenfalls von einer nicht fixen Zinsbindung.
- Wann wird die Höhe der Kuponzahlung festgelegt?

  Erfolgt die Anpassung am Anfang der Kuponperiode spricht man von einer regulären Zinsanpassung. Bei einer Anpassung am Ende der Kuponperiode nennt man dies in-arrears Zinsanpassung. Eine weitere Möglichkeit ist die Durchschnittsanpassung. Diese liegt dann vor, wenn die Kuponzahlung von der durchschnittlichen Höhe des Referenzzinssatzes während einer Referenzperiode abhängig ist.
- Wie wird die Höhe der Kuponzahlung ermittelt?

  Wenn man unterstellt, dass die Höhe der Kuponzahlung als Funktion des Referenzzinssatzes darzustellen ist, kann man u.a. zwischen folgenden Anpassungsfunktionen unterscheiden:
- Perfekt indiziert

  Die Verzinsung entspricht dem Referenzzinssatz.
- Margin

  Entspricht einem Auf- bzw. Abschlag auf den Referenzzinssatz.
- Rundung

  Hierbei wird der Referenzzinssatz auf einen bestimmten Wert gerundet.
- Zinsober-/untergrenze (Cap/Floor)

  Es wird festgelegt, dass die Verzinsung einen bestimmten Wert nicht über- bzw. unterschreiten darf.

Natürlich sind auch Kombinationen der einzelnen Anpassungsfunktionen erlaubt, z.B. ein Margin von 0,5% zusammen mit einer Rundung auf 0,125%. Auch eine Kombination von Zinsober- und Untergrenze ist möglich.

1.3. Literaturüberblick

Eine der ersten Arbeiten die sich mit der Bewertung von Geldmarktfloater befasst hat, wurde von Ramaswamy und Sundaresan [14] erstellt. Ausgehend von einem Einfaktor-Modell wurde ein allgemeiner Bewertungsansatz für Geldmarktfloater (Durchschnitts-Zinsanpassung) entworfen. Unter Anwendung des Forward Measure wurde von Li und Raghavan [9] ein Modell für Floater mit in-arrears Zinsanpassung vorgestellt. Die Grundlagen für die Anwendung des Forward Measure finden sich in Musiela und Rutkowski [11] und Miltersen, Sandmann und Sondermann [10], sowie sehr ausführlich in Jamshidian [7].

Zur Bewertung von Kapitalmarktfloatern mit Bindung an einen Zinssatz mit konstanter Laufzeit (Constant Maturity Produkten) findet sich in Jamshidian [7] eine ausführliche Darstellung, welche auf das Modell aus Brotherton-Ratcliffe und Iben [1] zurückzuführen ist. Eine leichter zugängliche Beschreibung des Ansatzes findet sich in Hull [6].

Eine der ersten umfassenden Darstellungen des Bewertungsproblems für Kapitalmarktfloater mit Bindung an einen Zinssatz mit variabler Laufzeit wurde in Pichler [13] vorgestellt. Im wesentlichen der gleiche Ansatz wurde auch in einem Arbeitspapier von Frühwirth [4] verwendet.

2. Bewertungsmodell

Dieser Abschnitt gibt eine Überblick über das in dieser Arbeit verwendete Bewertungsmodell (vgl. [13]). Der exakte Rechenvorgang wird anhand eines konkreten Beispieles in Abschnitt 4.3 beschrieben.

Der verwendete Bewertungsansatz basiert auf einer endogenen Modellierung der SMR und einem zusätzlichen exogenen Fehlerterm, d.h. die SMR wird in zwei Teile aufgeteilt:

- Einen prognostizierbaren SMR-Teil, basierend auf den zum Bewertungszeitpunkt verfügbaren Informationen.
- Einen Fehlerprozess, der mit den dem Zinsstrukturmodell zugrundeliegenden Faktoren unkorreliert ist und die Abweichungen von der modellmäßigen SMR angibt.

Der Zusammenhang lässt sich also wie folgt darstellen:

$$SMR_T = SMR_T|I_T + u_T \tag{4}$$

SMR_T	Tatsächliche SMR zum Zeitpunkt $T \geq t$
$SMR_T\|I_T$	Prognostizierbare SMR unter der verfügbaren Informationsmenge I_T
u_T	Realisation des Fehlerprozesses zum Zeitpunkt T

Da sich gezeigt hat, dass eine hohe Korrelation zwischen der SMR und einigen CMTs vorliegt, wird die SMR als Linearkombination aus mehreren CMTs dargestellt. Der lineare Zusammenhang kann durch folgende OLS Regression bestimmt werden:

$$\Delta SMR_t = \sum_{k=1}^{K} \beta_k \cdot \Delta CMT_{kt} + \varepsilon_t \tag{5}$$

CMT_{kt}	CMT mit Laufzeit k zum Zeitpunkt t
β_k	Lineare Gewichtung der CMT mit Laufzeit k
ε_t	Residualterm mit $\mathsf{E}\varepsilon_t = 0$; es gilt: $\varepsilon_t = \Delta u_t$ sowie $u_t = u_0 + \sum_{j=1}^{t} \varepsilon_j$

2. Bewertungsmodell

Unter der Annahme, dass der Fehlerprozess und die Diskontierungsfaktoren unkorreliert sind, kann der CMT-abhängige Teil getrennt vom SMR-spezifischen Teil bewertet werden. Für den CMT-abhängigen Teil einer SMR-gebundenen Kuponzahlung gilt

$$V_{CMT}(t,T) = \sum_{k=1}^{K} \beta_k \cdot V_k(t,T), \tag{6}$$

wobei $V_k(t,T)$ den Barwert einer zu T fälligen Kuponzahlung mit perfekter regulärer Zinsanpassung an die CMT mit Laufzeit k bezeichnet. Dieser wird auf folgende Art ermittelt (vgl. [6]):

$$\begin{aligned}
V(t,T) &= P(t,T) \cdot \left[f - \frac{1}{2} \cdot f^2 \cdot \sigma_{FSR}^2 \cdot (T-h) \cdot \frac{PV_N''(f)}{PV_N'(f)} \right] \qquad (7)\\
PV_N'(f) &= -f \cdot \sum_{i=1}^{N} i \cdot (1+f)^{-i-1} - N \cdot (1+f)^{-N-1}\\
PV_N''(f) &= f \cdot \sum_{i=1}^{N} \left[i \cdot (i+1) \cdot (1+f)^{-i-2} \right] + N \cdot (N+1) \cdot (1+f)^{-N-2}\\
FSR_{\tau,N}(t) &= \frac{P(t,t+\tau) - P(t,t+\tau+N)}{H_\tau(t,N)}\\
H_\tau(t,N) &= h \cdot \sum_{i=1}^{N/h} P(t,t+\tau+i \cdot h)
\end{aligned}$$

f	Forward-Swap-Rate $FSR_{T-h-t,N}(t)$
σ_{FSR}^2	Volatilität der Forward-Swap-Rate
h	Kuponperiode
$H_\tau(t,N)$	Annuitätenfaktor: Der zu t beobachtbare Preis einer zu $t+\tau$ beginnenden Annuität über N Jahre und Frequenz h.

Zur Modellierung des Fehlerprozesses wurde ein Ornstein-Uhlenbeck-Prozess mit folgendem Aussehen verwendet:

$$du_t = a \cdot (\gamma - u_t) \cdot dt + b \cdot dW_t \qquad a,b > 0 \tag{8}$$

a	Mean Reversion Parameter
b	Volatilitätsparameter
γ	Langfristiger Mittlerwert

Für den diskreten Fall greift man auf die Euler-Approximation des Ornstein-Uhlenbeck-Prozesses, den sog. AR(1)-Prozess (vgl. [8]), zurück. Es gilt:

$$du_t \approx \Delta u_t = \varepsilon_t = \rho \cdot \varepsilon_{t-1} + e_t \qquad e_t \sim \text{i.i.d.} \tag{9}$$

Für die Residualvarianz gilt weiters:

$$\sigma_\varepsilon^2 = \frac{b^2}{2a} \cdot \left(1 - e^{-2 \cdot a \cdot \Delta t}\right) \qquad \text{mit } a = -\rho \tag{10}$$

Dabei bezeichnet Δt das Zeitintervall zwischen zwei Beobachtungen. Der Erwartungswert unter dem risikoneutralen Maß Q für den Fehlerprozess zu einem beliebigen zukünftigen Zeitpunkt T, ist folgendermaßen gegeben:

$$\mathsf{E}_Q u_T = \gamma^* + (u_t - \gamma^*) \cdot e^{-a(T-t)} \tag{11}$$

$$\gamma^* = \gamma - \frac{\lambda \cdot b}{a} \tag{12}$$

Der Parameter γ^* entspricht dem risikoadjustierten langfristigen Mittelwert des Fehlerprozesses. Die momentane Ausprägung u_t des Fehlerprozesses ergibt sich zu:

$$u_t = SMR_t - \sum_{k=1}^{K} \beta_k \cdot CMT_{kt} \tag{13}$$

Den Mean Reversion Paramater a erhält man ebenso wie die Realisation von u_t aus der Regression. Lediglich der Parameter γ^* muss durch Kalibrierung an Marktdaten ermittelt werden (vgl. Abschnitt 4.3.1).

Kombiniert man die beiden Bewertungen ergibt sich als Barwert für eine zum Zeitpunkt T fällige Kuponzahlung:

$$\begin{aligned} V(t,T) &= V_{CMT}(t,T) + P(t,T) \cdot \mathsf{E}_Q u_{T-h} \\ &= \underbrace{\sum_{k=1}^{K} [\beta_k \cdot V_k(t,T)]}_{\text{CMT}-\text{abhngiger Teil}} + \underbrace{P(t,T) \cdot \left[\gamma^* + (u_t - \gamma^*) \cdot e^{-a(T-h-t)}\right]}_{\text{Barwert des Erwartungswertes des Fehlerprozesses}} \end{aligned} \tag{14}$$

β_k	Lineare Gewichtung der CMT mit Laufzeit k (vgl. Formel 5)
$V_k(t,T)$	Barwert einer zu T fälligen Kuponzahlung mit perfekter regulärer Zinsanpassung an die CMT mit Laufzeit k (vgl. Formel 7)
γ^*	Risikoadjustierter langfristiger Mittelwert des Fehlerprozesses (vgl. Formel 12)
u_t	Momentane Ausprägung des Fehlerprozesses (vgl. Formel 13)
a	Mean Reversion Parameter (vgl. Formel 8)
h	Länge der Kuponperiode in Jahren

3. Programmbedienung

Das Programm wurde in zwei Bereiche unterteilt:

- Einen Assistenten, der dialoggesteuert durch den Kalibrierungs- und Bewertungsprozess führt (Datei `smr.xls`).
- Eine Arbeitsmappe, welche bespielhaft die Makros zur Kalibrierung und Bewertung verwendet (Datei `smrmakro.xls`).

Auf jeden Fall wird die Datei `smr.xla` benötigt, welche die eigentlichen Kalibrierungs- und Bewertungsfunktionen enthält.

3.1. Assistent

Nach dem Öffnen der Arbeitsmappe `smr.xls` erscheint sofort der Einstiegsdialog des Assistenten (vgl. Abbildung 3.1). Falls man eine neue Kalibrierung durchführen will, wählt man „Neue Kalibrierung“. Will man eine Bewertung basierend auf einer bestehenden Kalibrierung durchführen, muss man den Pfadnamen der Datei mit den Kalibrierungswerten angeben und wählt dann „Weiter“.

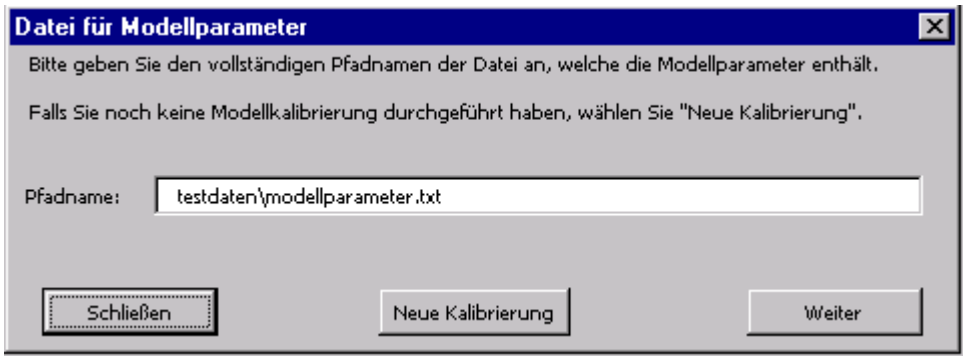


Abbildung 3.1.: Einstiegsdialog des Assistenten

Der nachfolgende Dialog ist abhängig davon, ob man sich für eine neue Kalibrierung oder eine Bewertung entschieden hat.

3.1.1. Kalibrierung

Nachdem man sich im Einstiegsdialog für eine neue Kalibrierung entschieden hat wird man in den folgenden Dialogen aufgefordert, die Dateinamen für die Zeitreihen der Zinssätze sowie der Sekundärmarktrendite anzugeben, woraufhin diese eingelesen werden (Für die unterstützten Dateiformate vgl. Abschnitt 5). Nach dem Einlesen der Zeitreihe der Sekundärmarktrendite wird überprüft, ob sich der Datumsbereich der beiden Zeitreihen ausreichend überschneidet. Nur in diesem Fall kann mit der Kalibrierung fortgefahren werden und man gelangt zum Hauptdialog der Kalibrierung (vgl. Abbildung 3.2).

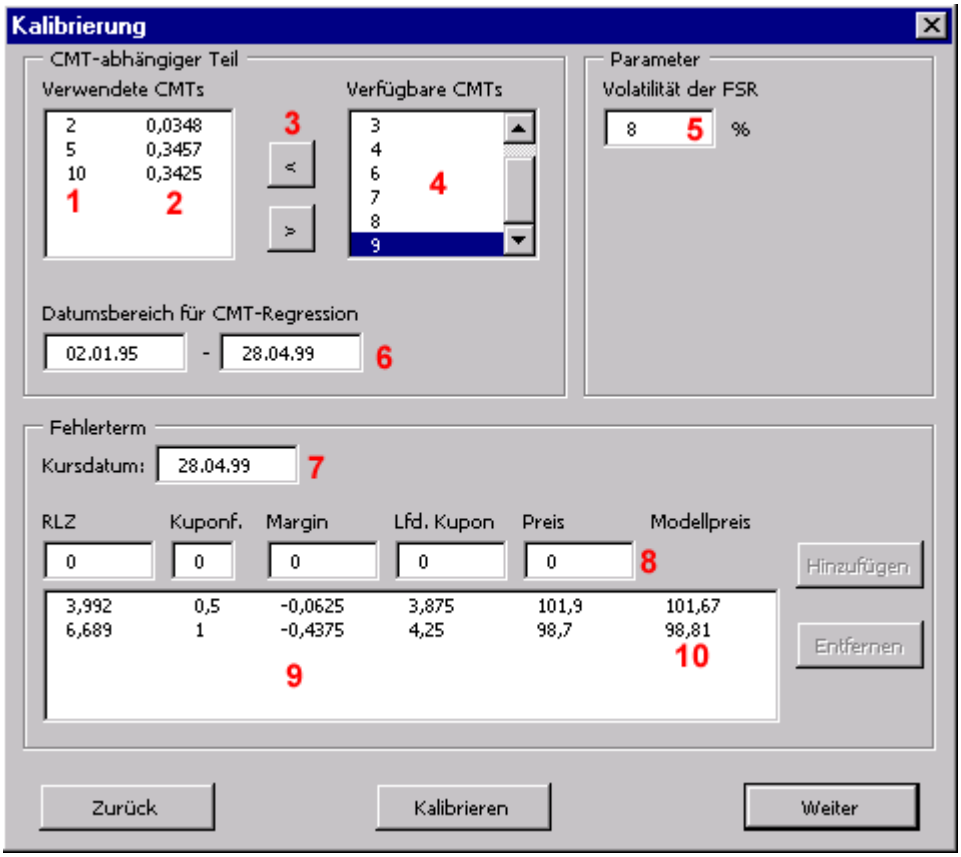


Abbildung 3.2.: Hauptdialog der Kalibrierung: 1: Laufzeiten der zu verwendenden CMTs; 2: Ermittelte β-Werte (nach Kalibrierung); 3: Hinzufügen/Entfernen von CMTs; 4: Verfügbare CMT-Zeitreihen; 5: Volatilität der Forward Swap Rate (wird vereinfachend als von der Laufzeit unabhängig angenommen); 6: Zu verwendender Zeitbereich für die Regression; 7: Kursdatum der für die Kalibrierung zu verwendenden SMR-abhängigen Anleihen; 8: Stammdaten und Kurse einer Anleihe; wird mit „Hinzufügen“ in die Anleihen-Liste 9 aufgenommen; 9: Daten der eingegebenen Anleihen; 10: Modellpreis (nach Kalibrierung)

Der Hauptdialog gliedert sich in drei Teile:

1. Auswahl der zu verwendenden CMTs sowie des Zeitbereichs für die Regression.
2. Angabe der Volatilität der Forward Swap Rate zur Bewertung der CMT-angepassten Zahlungen.
3. Stammdaten und Kurse jener Anleihen, die zur Kalibrierung des Fehlerterms (konkret: γ^*) verwendet werden sollen.

Die Bedeutung der einzelnen Werte wurde bereits in Abschnitt 2 erklärt. Erwähnenswert ist, dass in der Anleihenliste (9) einzelne Werte markiert werden können. Diese werden bei der darauffolgenden Kalibrierung *nicht* berücksichtigt.

Nachdem eine Kalibrierung durchgeführt wurde (Schaltfläche „Kalibrieren“), werden in der Anleihenliste zusätzlich die ermittelten Modellpreise angezeigt. Sollte es hier zu größeren Abweichungen kommen, ist es ratsam die eingegebenen Werte nochmals zu überprüfen bzw. die Auswahl der Anleihen zu überdenken.

Sobald die Kalibrierung abgeschlossen ist gelangt man mittels „Weiter“ in den letzten Dialog, innerhalb dessen man angeben kann, in welcher Datei die ermittelten Werte für eine spätere Bewertung gespeichert werden sollen. Außerdem können in diesem Dialog die Kalibrierungswerte mittels „Drucken“ ausgedruckt werden.

3.1.2. Bewertung

Nach dem man im Einstiegsdialog die Datei mit den Kalibrierungswerten und im nächsten Dialog die Datei für die aktuellen Zinssätze angegeben hat, gelangt man in den Dialog für die aktuelle Sekundärmarktrendite (vgl. Abbildung 3.3). Da zur Bewertung nur die aktuelle SMR benötigt wird, kann hier neben einer Datei, welche mehrere SMR-Daten enthalten kann, auch eine SMR mit dem zugehörigen Datum direkt angeben werden. Nach dem SMR-Dialog gelangt man in den Hauptdialog der

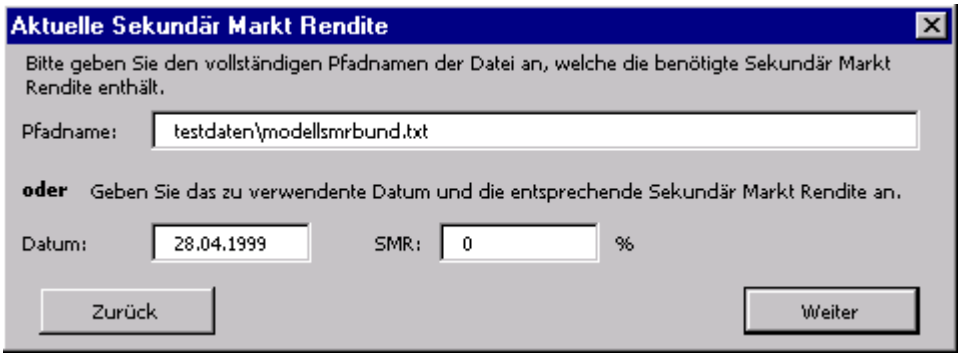


Abbildung 3.3.: Angabe der aktuellen Sekundärmarktrendite

Bewertung (vgl. Abbildung 3.4). Neben der Volatilität der Forward Swap Rate müssen im Bereich „Grunddaten“ die Daten der zu bewertenden Anleihe sowie im Bereich „Bewertung“ das Kursdatum eingegeben werden. Danach kann diese Anleihe bewertet werden (Schaltfläche „Bewerten“), woraufhin der Modellpreis unterhalb des Kursdatums angezeigt wird.

Der Bereich „Ausstattung“ wurde *nicht* implementiert, da eine Bewertung von Optionsrechten den Rahmen dieser Arbeit sprengen würde.

Abbildung 3.4.: Hauptdialog der Bewertung

3.2. Makros

Das Excel-Addin `smr.xla` stellt unter anderem auch Makrofunktionen zur Verfügung, die es erlauben ohne den Assistenten innerhalb einer Arbeitsmappe mehrere Kalibrierungen und Bewertungen gleichzeitig durchzuführen. Die Arbeitsmappe `smrmakro.xls` enthält einige Beispiele zur Anwendung dieser Makros. Eine genaue Beschreibung der Parameter ist in Abschnitt 4.2 zu finden.

In den Abbildungen 3.5 und 3.6 sind Beispiele zur Kalibrierung und Bewertung dargestellt.

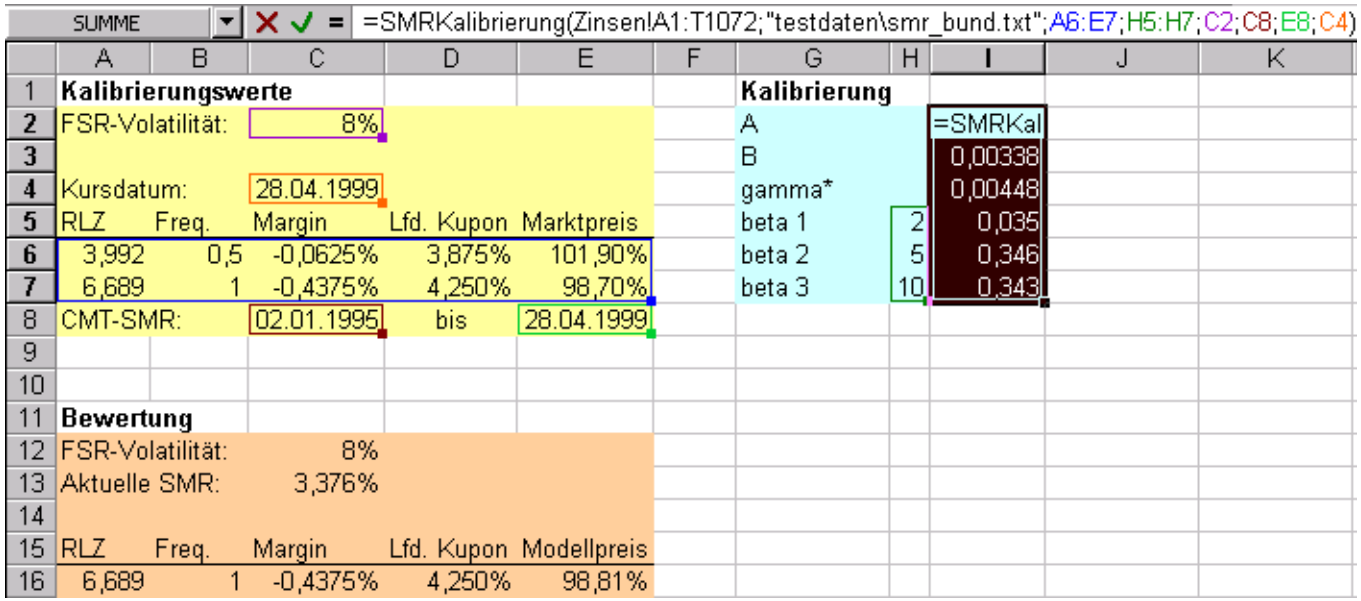


Abbildung 3.5.: Makrofunktion zur Kalibrierung: Der markierte Bereich [I2:I7] enthält die Matrix-Formel zur Kalibrierung. Die Zinssätze werden aus dem Bereich [A1:T1072] der Tabelle „Zinsen“ ermittelt. Die SMR-Zeitreihe wird aus der Datei `testdaten\smr_bund.txt` eingelesen. Die Daten der Kalibrierungs-Anleihen kommen aus dem Bereich [A6:E7] der aktuellen Tabelle; die Laufzeiten der zu verwendenten CMTs aus dem Bereich [H5:H7]. Weiters werden die FSR-Volatilität aus der Zelle C2, das Start- sowie Enddatum der Regression aus den Zellen C8 und E8 und das Kursdatum aus der Zelle C4 entnommen.

SUMME | =smrbewertung($C$4;$C$12;Zinsen!$A$1:$T$1072;$C$13;A16;B16;C16;D16;$I$2;$I$3;$I$4;$H$5:$H$7;$I$5:$I$7)

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Kalibrierungswerte**						**Kalibrierung**						
2	FSR-Volatilität:		8%				A		0,26653				
3							B		0,00338				
4	Kursdatum:		28.04.1999				gamma*		0,00448				
5	RLZ	Freq.	Margin	Lfd. Kupon	Marktpreis		beta 1	2	0,035				
6	3,992	0,5	-0,0625%	3,875%	101,90%		beta 2	5	0,346				
7	6,689	1	-0,4375%	4,250%	98,70%		beta 3	10	0,343				
8	CMT-SMR:		02.01.1995	bis	28.04.1999								
9													
10													
11	**Bewertung**												
12	FSR-Volatilität:		8%										
13	Aktuelle SMR:		3,376%										
14													
15	RLZ	Freq.	Margin	Lfd. Kupon	Modellpreis								
16	6,689	1	-0,4375%	4,250%	=smrbewert								

Abbildung 3.6.: Makrofunktion zur Bewertung: Die markierte Zelle E16 enthält die Formel zur Bewertung. Kursdatum sowie FSR-Volatilität werden aus den Zellen C4 bzw. C12 entnommen. Die Zinssätze werden aus dem Bereich [A1:T1072] der Tabelle „Zinsen“ ermittelt. Der Wert der aktuellen SMR ist in Zelle C13 enthalten. Die Daten der zu bewertenden Anleihe stehen in den Zellen A16 bis D16. Die Modellparameter A, B und γ^* stehen in den Zellen I2 bis I4. Die CMT-Laufzeiten sind in der Matrix [H5:H7], die β-Werte in der Matrix [I5:I7] enthalten.

4. Programmbeschreibung

Dieser Abschnitt gibt einen Überblick über die Funktionsweise des Programmes. Dazu werden in 4.1 und 4.2 die Klassen und Makros beschrieben, die im AddIn `smr.xla` enthalten sind. In 4.3 wird abschließend anhand eines konkreten Beispiels der Ablauf des Kalibrierungs- und Bewertungsvorganges erläutert.

4.1. Klassen

4.1.1. Klasse „Spotrates“

Diese Klasse repräsentiert eine Zinsstruktur für verschiedene Datumswerte. Die Werte der Zinsstruktur können aus einer Datei (vgl. Abschnitt 5) bzw. direkt aus einem Tabellenblatt eingelesen und danach beliebig abgefragt werden. Zwischen den vorhandenen Laufzeiten werden die Zinssätze linear interpoliert.

Methode	Funktion
`getAF`	Berechnet den Annuitätenfaktor basierend auf der aktuellen Zinsstruktur.
`getAvailableDates`	Liefert ein einspaltiges Datenfeld, welches alle verfügbaren Datumswerte in absteigender Reihenfolge enthält.
`getAvailableMaturities`	Liefert ein einspaltiges Datenfeld, welches alle verfügbaren Laufzeiten in aufsteigender Reihenfolge enthält.
`getDiscountFactor`	Berechnet den Diskontierungsfaktor basierend auf der aktuellen Zinsstruktur.
`getFirstDate`	Ermittelt das älteste Datum für welches eine Zinsstruktur vorliegt.
`getForwardRate`	Berechnet die Forward Rate basierend auf der aktuellen Zinsstruktur.
`getForwardSwapRate`	Berechnet die Forward Swap Rate basierend auf der aktuellen Zinsstruktur.
`getHistoryAF`	Berechnet den Annuitätenfaktor basierend auf der Zinsstruktur zum angegebenen Datum.
`getHistoryDiscountFactor`	Berechnet den Diskontierungsfaktor basierend auf der Zinsstruktur zum angegebenen Datum.
`getHistoryForwardRate`	Berechnet die Forward Rate basierend auf der Zinsstruktur zum angegebenen Datum.

`getHistoryForwardSwapRate`	Berechnet die Forward Swap Rate basierend auf der Zinsstruktur zum angegebenen Datum.
`getHistoryRateFor`	Liefert den Zinssatz für die angegebene Laufzeit zum angegebenen Datum. Liegt diese Information nicht vor, wird der Wert `0` zurückgegeben.
`getHistorySwapRate`	Berechnet die Swap Rate basierend auf der Zinsstruktur zum angegebenen Datum.
`getLastDate`	Ermittelt das jüngste Datum für welches eine Zinsstruktur vorliegt.
`getRateFor`	Liefert den Zinssatz für die angegebene Laufzeit für das aktuelle Datum.
`getSwapRate`	Berechnet die Swap Rate basierend auf der aktuellen Zinsstruktur.
`isOK`	Liefert `true`, falls das Objekt korrekt initialisiert worden ist und eine Zinsstruktur eingelesen wurde.
`read`	Liest die Zinsstruktur aus einer Datei ein. Konnte die angegebene Datei gefunden werden, wird `readStandard` aufgerufen, anderenfalls `readPL`.
`readFromMatrix`	Liest die Zinsstruktur aus einem Tabellenblatt ein. Falls der markierte Bereich zweispaltig ist, müssen in der ersten Spalte die Laufzeiten und in der zweiten Spalte die aktuellen Zinssätze stehen. Enthält der Bereich mehr als zwei Spalten, müssen in der ersten Zeile die Laufzeiten stehen, in der ersten Spalte die Datumswerte und in den restlichen Feldern die entsprechenden Zinssätze.
`readPL`	Liest die Zinsstruktur aus mehreren Dateien im ProfitLine-Format ein. Die Namen der Dateien müssen dem Format XXXYY.ZZZ entsprechen, wobei XXX und ZZZ beliebige Buchstaben sein dürfen und YY der Laufzeit entspricht.
`readStandard`	Liest die Zinsstruktur aus einer Datei im Standard-Format ein. Die Datei muss einer Zinsmatrix entsprechend aufgebaut sein (vgl. `readFromMatrix`), wobei die Spalten durch einen Tabulator (`chr(9)`) getrennt sein müssen.
`setRates`	Liest die aktuelle Zinsstruktur aus einem Tabellenblatt ein. Der markierte Bereich muss zweispaltig sein, wobei in der ersten Spalte die Laufzeiten und in der zweiten Spalte die aktuellen Zinssätze stehen müssen.

Verwendete Berechnungsmethoden

- Annuitätenfaktor

$$H_\tau(t,N) = h \cdot \sum_{i=1}^{N/h} P(t, t+\tau+i\cdot h) \tag{15}$$

Der zu t beobachtbare Preis einer zu $t+\tau$ beginnenden Annuität über N Jahre mit Kuponfrequenz h.

- Forward Rate

$$ {}_tF_T = \frac{(1+r_T)^T}{(1+r_t)^t} - 1 \tag{16} $$

- Forward Swap Rate

$$ FSR_{\tau,N}(t) = \frac{P(t,t+\tau) - P(t,t+\tau+N)}{H_\tau(t,N)} \tag{17} $$

- Swap Rate

$$ SR_N(t) = \frac{1 - P(t,t+N)}{H_0(t,N)} \tag{18} $$

4.1.2. Klasse „CMT“

Diese Klasse liefert basierend auf einer Zinsstruktur (Objekt der Spotrates-Klasse) die Constant Maturity Treasury-Werte.

Methode	Funktion
`getAvailableDates`	Liefert ein einspaltiges Datenfeld, welches alle verfügbaren Datumswerte in absteigender Reihenfolge enthält.
`getAvailableMaturities`	Liefert ein einspaltiges Datenfeld, welches alle verfügbaren Laufzeiten in aufsteigender Reihenfolge enthält.
`getConvexityAdjustment`	Berechnet das sog. Convexity Adjustment basierend auf der aktuellen Zinsstruktur.
`getFirstDate`	Ermittelt das älteste Datum für welches eine Zinsstruktur vorliegt.
`getHistoryConvexityAdjustment`	Berechnet das sog. Convexity Adjustment basierend auf der Zinsstruktur für das angegebene Datum.
`getHistoryRateFor`	Liefert den CMT-Wert für die angegebene Laufzeit zum angegebenen Datum. Liegt diese Information nicht vor, wird der Wert `0` zurückgegeben.
`getLastDate`	Ermittelt das jüngste Datum für welches eine Zinsstruktur vorliegt.
`getPaymentValue`	Berechnet den Barwert einer Kuponzahlung mit perfekter regulärer Zinsanpassung an eine CMT.
`getRateFor`	Liefert den aktuellen CMT-Wert für die angegebene Laufzeit. Liegt diese Information nicht vor, wird der Wert `0` zurückgegeben.
`getValue`	Berechnet den Barwert einer Anleihe mit perfekter regulärer Zinsanpassung an eine CMT.
`setSpotRates`	Weist das zu verwendende SpotRates-Objekt zu. Alle Berechnungen basieren auf der Zinsstruktur diese Objektes.

Verwendete Berechnungsmethoden

- CMT-Wert

$$CMT_T(t) = \frac{1 - P(t,T)}{\sum_{i=1}^{T} P(t,i)} \tag{19}$$

- Convexity Adjustment

$$\begin{aligned} CA &= -\frac{1}{2} \cdot f^2 \cdot \sigma_{FSR}^2 \cdot T \cdot \frac{PV_N''(f)}{PV_N'(f)} && (20)\\ f &= FSR_{T-h-t,N}(t) \\ PV_N'(f) &= -f \cdot \sum_{i=1}^{N} \left[i \cdot (1+f)^{-i-1} \right] - N \cdot (1+f)^{-N-1} && (21)\\ PV_N''(f) &= f \cdot \sum_{i=1}^{N} \left[i \cdot (i+1) \cdot (1+f)^{-i-2} \right] + N \cdot (N+1) \cdot (1+f)^{-N-2} && (22) \end{aligned}$$

- Barwert einer Kuponzahlung mit perfekter regulärer Zinsanpassung an eine CMT

$$V(t,T) = P(t,T) \cdot [FSR_{T-h-t,N}(t) - CA] \tag{23}$$

4.1.3. Klasse „SMR“

Diese Klasse abstrahiert SMR-Werte für verschiedene Datumswerte. Die SMR-Werte können aus einer Datei (vgl. Abschnitt 5) bzw. direkt aus einem Tabellenblatt eingelesen und danach beliebig abgefragt werden.

Methode	Funktion
`getAvailableDates`	Liefert ein einspaltiges Datenfeld, welches alle verfügbaren Datumswerte in absteigender Reihenfolge enthält.
`getFirstDate`	Ermittelt das älteste Datum für welches SMR-Werte vorliegen.
`getHistoryRate`	Liefert den SMR-Wert für das angegebene Datum. Liegt diese Information nicht vor, wird der Wert `0` zurückgegeben.
`getLastDate`	Ermittelt das jüngste Datum für welches SMR-Werte vorliegen.
`getRate`	Liefert den SMR-Wert für das aktuelle Datum. Liegt diese Information nicht vor, wird der Wert `0` zurückgegeben.
`isOK`	Liefert `true`, falls das Objekt korrekt initialisiert worden ist und SMR-Werte eingelesen wurden.
`read`	Liest die Zinsstruktur aus einer Datei ein. Konnte die angegebene Datei gefunden werden, wird `readStandard` aufgerufen, anderenfalls `readPL`.
`readFromMatrix`	Liest die SMR-Werte aus einem Tabellenblatt ein. Der markierte Bereich muss zweispaltig sein und in der ersten Spalte die Datumswerte und in der zweiten Spalte die entsprechenden SMR-Werte enthalten.
`readPL`	Liest die SMR-Werte aus einer Datei im ProfitLine-Format ein.
`readStandard`	Liest die SMR-Werte aus einer Datei im Standard-Format ein. Die Datei muss genau so aufgebaut sein wie eine SMR-Matrix (vgl. `readFromMatrix`), wobei die Spalten durch einen Tabulator (`chr(9)`) getrennt sein müssen.
`setRate`	Setzt den SMR-Werte für das aktuelle Datum.

4.1.4. Klasse „SMRRegression“

In dieser Klasse ist der Algorithmus zur Bewertung SMR-abhängiger Kuponzahlungen implementiert.

Methode	Funktion
`addCalibrateValue`	Fügt die Stammdaten einer Anleihe zur internen Liste hinzu, welche für die Fehlerterm-Kalibrierung verwendet wird.
`berechneParameter`	Berechnet die Modellparameter *a*, *b*, γ sowie γ^*.
`berechneUT`	Berechnet das aktuelle u_T (Differenz zwischen aktueller SMR und der CMT-Linearkombination). Die β-Werte müssen vorher geschätzt werden.
`calibrate`	Bereitet die Daten zur Kalibrierung vor (`createArrays`) und ruft danach die Methoden `schätzeRoh`, `schätzeBetas` und `berechnePa-rameter` auf.
`checkSecondDimension`	Hilfsfunktion um sicherzustellen, dass ein Array zweidimensional ist.
`createArrays`	Erstellt die Arrays für die CMT- und SMR-Werte sowie die ΔCMT- und ΔSMR-Werte, falls eine Kalibrierung durchgeführt wird (keine Bewertung).
`eDach`	Hilfsfunktion für `schätzeRoh`.
`getA`	Liefert den Modellparameter *a*.
`getActualDate`	Liefert das „aktuelle“ Datum.
`getB`	Liefert den Modellparameter *b*.
`getBetas`	Liefert ein einspaltiges Array mit den β-Werten.
`getCalibrationCMTMaturities`	Liefert die Laufzeiten der CMTs, welche für den CMT-abhängigen Teil verwendet werden.
`getCalibrationEndDate`	Liefert das Enddatum, bis zu welchem die SMR-Regression durchgeführt werden soll.
`getCalibrationStartDate`	Liefert das Anfangsdatum, ab welchem die SMR-Regression durchgeführt werden soll.
`getCalibrationValues`	Liefert ein fünfspaltiges Array, welches die Stammdaten jener Anleihen enthält, die zur Kalibrierung des Fehlerterms verwendet werden sollen.
`getDetailValue`	Liefert ein vierspaltiges Array. Ausgehend von den übergebenen Stammdaten wird eine Bewertung vorgenommen. Das Array enthält (in genau dieser Reihenfolge) folgende Barwerte: CMT-Teil, SMR-Teil, Stückzinsen, Margin.
`getFSRVolatility`	Liefert die Volatilität der Forward Swap Rate.
`getGamma`	Liefert das ermittelte γ.
`getGammaStern`	Liefert das ermittelte γ^*.
`getLambda`	Liefert das ermittelte λ.
`getPresentValue`	Bewertung des SMR-abhängigen Fehlerterms.

`getValue`	Führt eine Bewertung für die übergebenen Stammdaten durch. Entspricht der Summe von `getDetailValue`.
`initValuation`	Initialisiert das Objekt für eine Bewertung. Es werden die notwendigen Arrays aufgebaut (`createArrays`) und das aktuelle u_T berechnet.
`isCalibrated`	Liefert `true`, falls das Objekt gültige Modellparameter enthält.
`readFromFile`	Liest die Modellparameter aus einer Datei, welche mit `writeToFile` erstellt wurde.
`resetCalibration`	Setzt die Kalibrierung zurück.
`schätzeBetas`	Schätzt aus den ΔSMR- und ΔCMT-Arrays die β-Werte.
`schätzeRoh`	Ermittelt den Wert ρ der zur Schätzung der β-Werte notwendig ist.
`setActualDate`	Setzt das „aktuelle“ Datum.
`setCMT`	Übergibt das zu verwendende CMT-Objekt.
`setFSRVolatility`	Setzt die Volatilität der Forward Swap Rate.
`setParameter`	Setzt die Modellparameter a, b und γ^* sowie die Laufzeiten der CMTs und die β-Werte.
`setSMR`	Übergibt das zu verwendende SMR-Objekt.
`setSpotRates`	Übergibt das zu verwendende SpotRates-Objekt.
`writeToFile`	Schreibt die Modellparameter in eine Datei.

Auf die Berechnungsmethoden wird detailiert im konkreten Beispiel (Abschnitt 4.3) eingegangen.

4.2. Makros

4.2.1. SMRKalibrierung

Definition:
```
Public Function SMRKalibrierung(
    vspots as Variant,
    vsmr as Variant,
    vkali as Variant,
    vcmt as Variant,
    fsrvol as Variant,
    smrvon as Variant,
    smrbis as Variant,
    kursdatum as Date) As Variant
```

Parameter	Beschreibung
`vspots`	Entweder der Name der Datei mit den Zeitreihen der Zinssätze oder ein einzelner Zellenbereich, der die Zeitreihen enthält.
`vsmr`	Entweder der Name der Datei mit der Zeitreihe der SMR oder ein einzelner Zellenbereich, der die Zeitreihe enthält.
`vkali`	Ein Zellenbereich, der die Stammdaten jener Anleihen enthält, welche zur Kalibrierung verwendet werden sollen. Der Zellenbereich muss exakt fünf Spalten breit sein, in denen die Werte Restlaufzeit, Kuponfrequenz, Margin, laufender Kupon und Marktpreis in genau dieser Reihenfolge enthalten sind. Jede Zeile beschreibt eine einzelne Anleihe.
`vcmt`	Ein Zellenbereich mit exakt einer Spalte, welcher die zu verwendenden CMTs angibt.
`fsrvol`	Volatilität der Forward Swap Rate.
`smrvon`	Anfangsdatum, ab dem die SMR-Regression durchgeführt wird.
`smrbis`	Enddatum, bis zu dem die SMR-Regression durchgeführt wird.
`kursdatum`	Kursdatum der Anleihen (siehe `vkali`)

Rückgabe: Man erhält eine einspaltige Matrix, welche die Parameter a, b, und γ^* in den ersten drei Zeilen enthält und in den folgenden Zeilen die β-Werte.

4.2.2. SMRBewertung

Definition:
```
Public Function SMRBewertung(
    kursdatum As Date,
    vol As Double,
    vspots As Variant,
    vsmr As Variant,
    rlz As Double,
    freq As Double,
    margin As Double,
    kupon As Double,
    par As Variant,
    Optional parB As Double = 0,
    Optional parGammaStern As Double = 0,
    Optional parCMTs As Variant,
    Optional parBetas As Variant) As Double
```

Parameter	Beschreibung
`kursdatum`	Datum für die Bewertung. Notwendig, falls Zinssätze bzw. SMR-Werte für mehrere Datumswerte angegeben wurden.
`vol`	Volatilität der Forward Swap Rate.
`vspots`	Entweder der Name der Datei, welche die Zinssätze enthält oder ein Zellbereich, welcher in der ersten Zeile die Laufzeiten enthält, in der ersten Spalte die Datumswerte und in den restlichen Feldern die entsprechenden Zinssätze.
`vsmr`	Entweder der Name der Datei, welche die SMR-Zeitreihe enthält oder ein Zellbereich, welcher in der ersten Spalte die Datumswerte und in der zweiten Spalte die entsprechenden SMR-Werte enthält.
`rlz`	Restlaufzeit der zu bewertenden Anleihe
`freq`	Kuponfrequenz
`margin`	Margin
`kupon`	Laufender Kupon
`par`	Entweder der Name der Datei, welche die Modellparameter enthält oder der Modellparameter *a*. Alle folgenden Parameter müssen nur dann angegeben werden, falls an dieser Stelle der Parameter *a* angegeben wurde.
`parB`	Modellparameter *b*
`parGammaStern`	Modellparameter γ^*
`parCMTs`	Ein Zellenbereich mit exakt einer Spalte, welcher die Laufzeiten der verwendeten CMTs enthält.
`parBetas`	Ein Zellenbereich mit exakt einer Spalte, welcher die β-Werte enthält.

Rückgabe: Modellpreis der Anleihe

4.2.3. SMRDetailBewertung

Definition:
```
Public Function SMRDetailBewertung(
    kursdatum As Date,
    vol As Double,
    vspots As Variant,
    vsmr As Variant,
    rlz As Double,
    freq As Double,
    margin As Double,
    kupon As Double,
    par As Variant,
    Optional parV As Double = 0,
    Optional parGammaStern As Double = 0,
    Optional parCMTs As Variant,
    Optional parBetas As Variant) As Variant
```

Die Parameter entsprechen denen der Funktion `SMRBewertung`.

Rückgabe: Ein einzeilige Matrix welche die Barwerte des CMT-Teils, des SMR-Teils, der Stückzinsen und der Margin in genau dieser Reihenfolge enthält. Die Summe dieser vier Werte entspricht dem Rückgabewert der Funktion `SMRBewertung`.

4.3. Beispiel

In diesem Abschnitt wird anhand eines konkreten Beispiels der Kalibrierungs- und Bewertungsvorgang dargestellt. Es werden jeweils die relevanten Formeln sowie die Rechenergebnisse angeführt. Die Methoden, welche die Berechnungen durchführen, werden im Format `{Klasse:Methode}` bzw. `{Prozedur}` angeführt (z.B. `{SMR:createArrays}`), wobei der Programmablauf bei Einsatz der Makros beschrieben wird.

Dieses Beispiel ist in der Arbeitsmappe `smrmakro.xls` auf der Seite „Beispiel“ zu finden (vgl. Abbildung 4.1).

	A	B	C	D	E	F	G	H	I
1	**Kalibrierungswerte**						**Kalibrierung**		
2	FSR-Volatilität:		8%				A		0,26650
3							B		0,11031
4	Kursdatum:		28.04.1999				gamma*		0,00448
5	RLZ	Freq.	Margin	Lfd. Kupon	Marktpreis		beta 1	2	0,035
6	3,992	0,5	-0,0625%	3,875%	101,90%		beta 2	5	0,346
7	6,689	1	-0,4375%	4,250%	98,70%		beta 3	10	0,343
8	CMT-SMR:		02.01.1995	bis	28.04.1999				
9									
10									
11	**Bewertung**								
12	FSR-Volatilität:		8%						
13	Aktuelle SMR:		3,376%						
14									
15	RLZ	Freq.	Margin	Lfd. Kupon	Modellpreis				
16	6,689	1	-0,4375%	4,250%	98,81%				

Abbildung 4.1.: Beispiel

4.3.1. Kalibrierung

In diesem Abschnitt wird die Kalibrierung des Bewertungsmodells beschrieben. Es werden mittels einer AR(1)-Regression die linearen Gewichtungen der CMTs (vgl. Formel 5) sowie der Mean Reversion Parameter a (vgl. Formel 8) ermittelt. Der risikoadjustierte langfristige Mittelwert γ^* des Fehlerprozesses wird anschließend mittels eines einfachen iterativen Verfahrens an den Daten ausgewählter Anleihen kalibriert.

Die Regression wird im Zeitraum 2.1.1995 bis 28.4.1999 durchgeführt. Zur Kalibrierung des Fehlerterms werden die Anleihen 13772 und 42804 (vgl. Tabelle 4.5) zusammen mit ihren Marktdaten vom 28.4.1999 verwendet (vgl. Tabelle 4.6).

WP-Knr.	Emittent	Laufzeit Ende	KF.	Margin	Rundung
13772	Erste Bank der ö. SK AG	30.4.2003	6M	0%	-0,125%
42804	Österreichische PSK AG	11.1.2006	1J	-0,375%	-0,125%

Tabelle 4.5.: Stammdaten der verwendeten Anleihen

WP-Knr.	Restlaufzeit	Lfd. Kupon	Marktpreis
13772	3,992	3,875%	101,90%
42804	6,689	4,250%	98,70%

Tabelle 4.6.: Marktdaten vom 28.4.1999 der verwendeten Anleihen

In einem ersten Schritt werden die Objekte für die Zinssätze, SMR- und CMT-Werte erzeugt `{SMR-Kalibrierung}` und mit den entsprechenden Daten gefüllt. Diese werden dann zusammen mit den notwendigen Stamm- und Marktdaten der Anleihen dem SMRRegression-Objekt übergeben und danach die Kalibrierung durch Aufruf der Methode `{SMRRegression:calibrate}` angestoßen. Es werden nun die folgenden Vorgänge ausgeführt:

1. Erstellung der notwendigen Datenfelder (ΔCMT, ΔSMR) für die Regression (Seite 28).
2. Schätzung des ρ-Wertes für die AR1-Regression (Seite 28).
3. Schätzung der β-Werte (Seite 29).
4. Berechnung der Modellparameter (Seite 30).

Erstellung der Datenfelder

Da im verwendeten Modell die Änderungen der SMR durch Änderungen einer Linearkombination der CMT-Werte beschrieben werden (vgl. Formel 6), ist es notwendig die ΔSMR- und ΔCMT-Werte zu berechnen und für eine weitere Verwendung temporär ab zu speichern `{SMRRegression:createArrays}`.

Schätzung von ρ

Da für die (diskrete) Modellierung des Fehlerterms ein AR(1)-Prozess verwendet wird, muss zuerst der Parameter ρ berechnet werden `{SMRRegression:schätzeRoh}` (vgl. [5]), bevor die β-Werte geschätzt werden können. Man beginnt indem man *erste* Schätzwerte für die β-Werte berechnet:

$$\hat{\beta} = \left(X^T \cdot X\right)^{-1} \cdot X^T \cdot y = \begin{pmatrix} -0,0051 \\ 0,3328 \\ 0,3297 \end{pmatrix} \tag{24}$$

Dabei bezeichnen X und y folgende Matrizen:

$$X = \begin{pmatrix} \Delta CMT_{11} & \cdots & \Delta CMT_{1K} \\ \Delta CMT_{21} & \cdots & \Delta CMT_{2K} \\ \vdots & & \vdots \\ \Delta CMT_{T1} & \cdots & \Delta CMT_{TK} \end{pmatrix}$$

$$y = \begin{pmatrix} \Delta SMR_1 \\ \Delta SMR_2 \\ \vdots \\ \Delta SMR_T \end{pmatrix}$$

Ausgehend von diesen Werten kann man nun zuerst einen Schätzwert für e_t ermitteln

$$\hat{e}_t = y_t - \sum_{i=1}^{j} \hat{\beta}_i \cdot x_{t_i i} \tag{25}$$

und anschließend ρ schätzen:

$$\hat{\rho} = \frac{\sum_{t=2}^{T} \hat{e}_t \hat{e}_{t-1}}{\sum_{t=2}^{T} \hat{e}_{t-1}^2} = -0,2665 \tag{26}$$

Schätzung der β-Werte

Nachdem $\hat{\rho}$ berechnet wurde, können nun die β-Werte auf folgende Art geschätzt werden `{SMRRegression:schätzeBetas}`:

$$\hat{\hat{\beta}} = \left(X^{*^T} X^*\right)^{-1} X^{*^T} y^* = \begin{pmatrix} 0,0350 \\ 0,3456 \\ 0,3425 \end{pmatrix} \tag{27}$$

Dabei gilt:

$$X^* = \begin{pmatrix} \sqrt{1-\hat{\rho}^2} \cdot \Delta CMT_{11} & \cdots & \sqrt{1-\hat{\rho}^2} \cdot \Delta CMT_{1K} \\ \Delta CMT_{21} - \hat{\rho} \cdot \Delta CMT_{11} & \cdots & \Delta CMT_{2j} - \hat{\rho} \cdot \Delta CMT_{1K} \\ \Delta CMT_{31} - \hat{\rho} \cdot \Delta CMT_{21} & \cdots & \Delta CMT_{3j} - \hat{\rho} \cdot \Delta CMT_{2K} \\ \vdots & & \vdots \\ \Delta CMT_{T1} - \hat{\rho} \cdot \Delta CMT_{T-1,1} & \cdots & \Delta CMT_{Tj} - \hat{\rho} \cdot \Delta CMT_{TK} \end{pmatrix}$$

$$y^* = \begin{pmatrix} \sqrt{1-\hat{\rho}^2} \cdot \Delta SMR_1 \\ \Delta SMR_2 - \hat{\rho} \cdot \Delta SMR_1 \\ \Delta SMR_3 - \hat{\rho} \cdot \Delta SMR_2 \\ \vdots \\ \Delta SMR_T - \hat{\rho} \cdot \Delta SMR_{T-1} \end{pmatrix}$$

Weiters berechnet man einen Schätzwert für die Residualvarianz[1]:

$$\hat{\sigma}_\varepsilon^2 = \frac{\left(y^* - X^* \hat{\beta}\right)^T \left(y^* - X^* \hat{\beta}\right)}{T-K} = 0,0000000463 \tag{28}$$

Auf Basis der geschätzten β-Werte kann man den momentanen Wert des Fehlerterms mittels Formel 13 berechnen zu $u_t = 0,6758\%$ `{SMRRegression:berechneUT}`.

[1] Hierbei ist anzumerken, dass es für die spätere Bewertung nicht notwendig ist, die Residualvarianz σ_ε^2 und im weiteren Verlauf den Parameter *b* zu berechnen.

Berechnung der Modellparameter

Abschließend müssen noch die Modellparameter basierend auf den Ergebissen der Regression berechnet werden `{SMRRegression:berechneParameter}`:

$$a = -\rho = 0,2665 \tag{29}$$

$$b = \sqrt{\frac{2 \cdot a \cdot \sigma_\varepsilon^2}{1 - e^{-2 \cdot a \cdot \Delta t}}} = 0,11031 \tag{30}$$

$$\gamma = \frac{\sum_{t=1}^{T} u_t}{T} = 1,11\% \tag{31}$$

$$\gamma^* = 0,448\%$$

Der Wert γ^* wird mittels eines einfachen iterativen Verfahrens basierend auf den Marktdaten der angegebenen Anleihen ermittelt (vgl. Abschnitt A.2.4 ab Zeile 600).

4.3.2. Bewertung

Basierend auf der Kalibrierung aus vorherigem Abschnitt wird nun die Anleihe 42804 bewertet {SMRBewertung}. Dazu wird für jede einzelne Kuponzahlungen eine Bewertung durchgeführt {SMR-Regression:getDetailValue} (vgl. Tabelle 4.7). Zu beachten ist dabei, dass für die erste Zahlungen nur der laufende Kupon heranzuziehen ist, aber weder Fehlerterm noch Margin zu berücksichtigen sind. Es gilt die aus Abschnitt 2 bekannte Formel:

$$V(t,T) = \underbrace{\sum_{k=1}^{K} [\beta_k \cdot V_k(t,T)]}_{\text{CMT-abhngiger Teil}} + \underbrace{P(t,T) \cdot \left[\gamma^* + (u_t - \gamma^*) \cdot e^{-a(T-h-t)}\right]}_{\text{Barwert des Erwartungswertes des Fehlerprozesses}}$$

Dabei bezeichnet $V_k(t,T)$ den Barwert einer CMT-Komponente (Bewertung vgl. Formel 23).

Zeitpunkt	CMT-Teil	Fehlerterm	Margin	Summe
0,689	4,175	-	-	4,175
1,689	2,722	0,609	-0,418	2,913
2,689	2,831	0,549	-0,405	2,975
3,689	2,914	0,500	-0,390	3,024
4,689	2,947	0,457	-0,375	3,029
5,689	2,930	0,421	-0,359	2,992
6,689	2,820	0,389	-0,342	2,867
	21,339	**2,923**	**-2,288**	**21,974**

Tabelle 4.7.: Bewertung der Kuponzahlungen

Nun müssen von der Summe der Barwerte der einzelnen Kuponzahlung noch die Stückzinsen abgezogen und der Barwert der Rückzahlung addiert werden. Somit ergibt sich folgender Wert:

Kuponzahlungen	21,974
Rückzahlung	71,156
Stückzinsen	-1,322
Gesamt	**98,808**

5. Dateiformate

Die Klassen `SMR`, `SpotRates` und `SMRRegression` können ihre Daten aus speziell formatierten Dateien einlesen. Dieser Abschnitt beschreibt das Format dieser Dateien.

5.1. Zinssätze

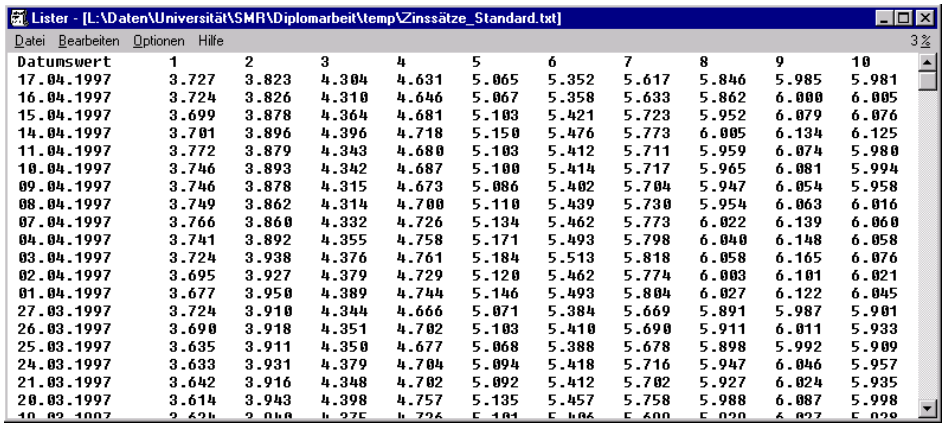
Die Klasse `SpotRates` kann zwei verschiedene Datenformate verarbeiten:

- Standardformat

  Die Datei ist eine tabellenartig aufgebaute Textdatei (vgl. Abbildung 5.1), wobei die Spalten durch Tabulator-Zeichen getrennt sind. Die erste Spalte enthält ab der zweiten Zeile die Datumswerte in absteigender Reihenfolge. Die erste Zeile enthält ab der zweiten Spalte die Laufzeiten. Der Wert in der ersten Zeile und Spalte wird ignoriert. Ab der zweiten Zeile und Spalte müssen die entsprechenden Zinssätze gespeichert sein.

- ProfitLine-Format

  Die Daten können auch im Export-Format der Software ProfitLine (vgl. Abbildung 5.2) vorliegen. Da hierbei jede Datei nur eine Zeitreihe für eine Laufzeit enthalten kann, müssen mehrere Dateien exportiert werden. Die Dateinamen der einzelnen Dateien müssen dem Muster `XXXYY.ZZZ` folgen, wobei `XXX` beliebige Buchstaben (keine Zahlen) sein dürfen, die für alle unterschiedlichen Laufzeiten identisch sein müssen und `YY` die Laufzeit mit „ . “ (Punkt) als Kommazeichen bezeichnet. Gültige Dateinamen wären z.B. `zins1.0.txt` oder `Spotrates_9.5.txt`.

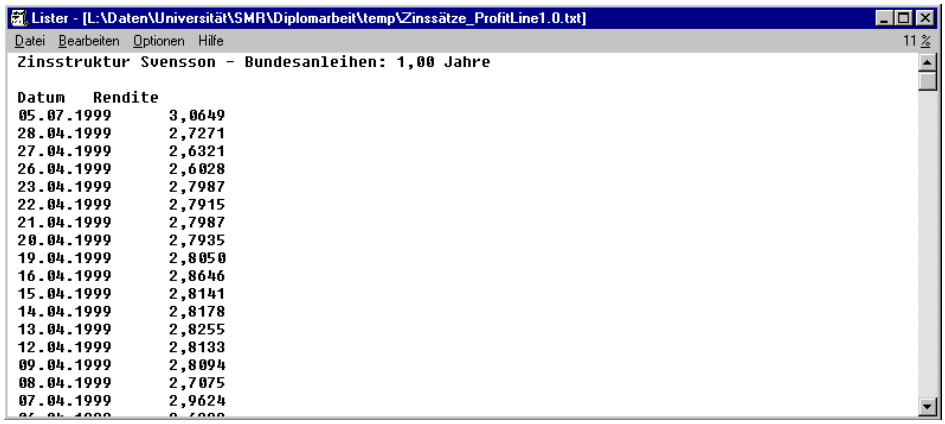


Lister - [L:\Daten\Universität\SMR\Diplomarbeit\temp\Zinssätze_Standard.txt]

Datei Bearbeiten Optionen Hilfe 3 %

Datumswert	1	2	3	4	5	6	7	8	9	10
17.04.1997	3.727	3.823	4.304	4.631	5.065	5.352	5.617	5.846	5.985	5.981
16.04.1997	3.724	3.826	4.310	4.646	5.067	5.358	5.633	5.862	6.000	6.005
15.04.1997	3.699	3.878	4.364	4.681	5.103	5.421	5.723	5.952	6.079	6.076
14.04.1997	3.701	3.896	4.396	4.718	5.150	5.476	5.773	6.005	6.134	6.125
11.04.1997	3.772	3.879	4.343	4.680	5.103	5.412	5.711	5.959	6.074	5.980
10.04.1997	3.746	3.893	4.342	4.687	5.100	5.414	5.717	5.965	6.081	5.994
09.04.1997	3.746	3.878	4.315	4.673	5.086	5.402	5.704	5.947	6.054	5.958
08.04.1997	3.749	3.862	4.314	4.700	5.110	5.439	5.730	5.954	6.063	6.016
07.04.1997	3.766	3.860	4.332	4.726	5.134	5.462	5.773	6.022	6.139	6.060
04.04.1997	3.741	3.892	4.355	4.758	5.171	5.493	5.798	6.040	6.148	6.058
03.04.1997	3.724	3.938	4.376	4.761	5.184	5.513	5.818	6.058	6.165	6.076
02.04.1997	3.695	3.927	4.379	4.729	5.120	5.462	5.774	6.003	6.101	6.021
01.04.1997	3.677	3.950	4.389	4.744	5.146	5.493	5.804	6.027	6.122	6.045
27.03.1997	3.724	3.910	4.344	4.666	5.071	5.384	5.669	5.891	5.987	5.901
26.03.1997	3.690	3.918	4.351	4.702	5.103	5.410	5.690	5.911	6.011	5.933
25.03.1997	3.635	3.911	4.350	4.677	5.068	5.388	5.678	5.898	5.992	5.909
24.03.1997	3.633	3.931	4.379	4.704	5.094	5.418	5.716	5.947	6.046	5.957
21.03.1997	3.642	3.916	4.348	4.702	5.092	5.412	5.702	5.927	6.024	5.935
20.03.1997	3.614	3.943	4.398	4.757	5.135	5.457	5.758	5.988	6.087	5.998

Abbildung 5.1.: Standardformat für Zinssätze



Lister - [L:\Daten\Universität\SMR\Diplomarbeit\temp\Zinssätze_ProfitLine1.0.txt]

Datei Bearbeiten Optionen Hilfe 11 %

Zinsstruktur Svensson - Bundesanleihen: 1,00 Jahre

Datum	Rendite
05.07.1999	3,0649
28.04.1999	2,7271
27.04.1999	2,6321
26.04.1999	2,6028
23.04.1999	2,7987
22.04.1999	2,7915
21.04.1999	2,7987
20.04.1999	2,7935
19.04.1999	2,8050
16.04.1999	2,8646
15.04.1999	2,8141
14.04.1999	2,8178
13.04.1999	2,8255
12.04.1999	2,8133
09.04.1999	2,8094
08.04.1999	2,7075
07.04.1999	2,9624

Abbildung 5.2.: ProfitLine-Format für Zinssätze

5.2. SMR

Ebenso wie die Klasse `SpotRates` kann auch die Klasse `SMR` zwei verschiedene Datenformate verarbeiten:

- Standardformat

  In diesem Format muss die Datei zwei durch Tabulator-Zeichen getrennte Spalten enthalten. Die erste Spalte enthält dabei die Datumswerte in absteigender Reihenfolge; die zweite Spalte die entsprechenden SMR-Werte (vgl. Abbildung 5.3).

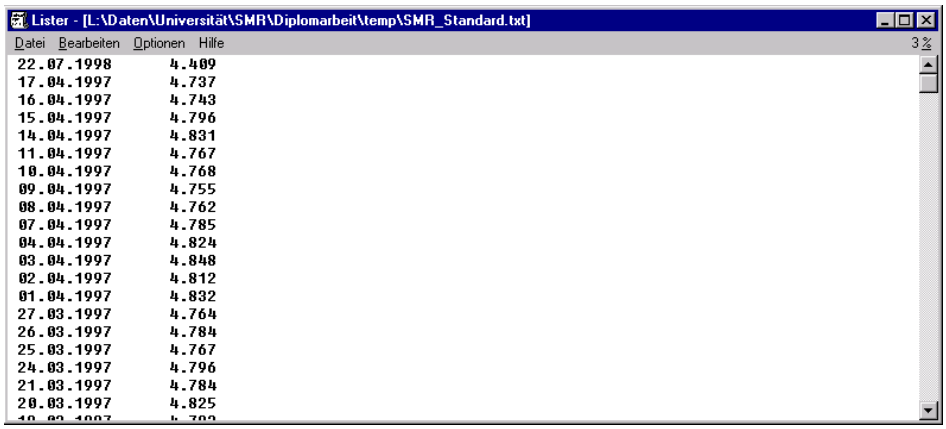


Abbildung 5.3.: Standardformat für SMR-Zeitreihen

- ProfitLine-Format

  Aus ProfitLine exportierte SMR-Zeitreihen (vgl. Abbildung 5.4) können ohne Einschränkung eingelesen werden.

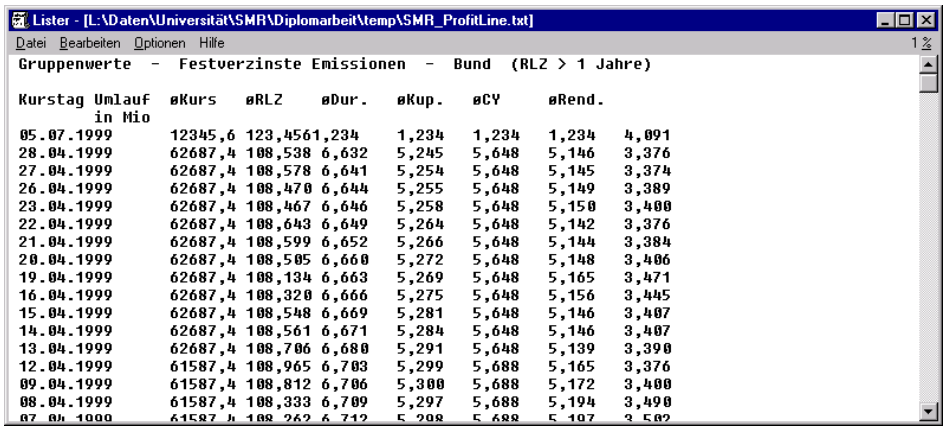


Abbildung 5.4.: ProfitLine-Format für SMR-Zeitreihen

5.3. Modellparameter

Die Klasse SMRRegression kann die ermittelten Modellparameter in eine Textdatei (vgl. Abbildung 5.5) schreiben `{SMRRegression:writeToFile}` und die Parameter zur Bewertung aus der Textdatei wieder einlesen `{SMRRegression:readFromFile}`. Die erste Zeile muss den Text

```
SMR-Regression Modellparameter V1
```

enthalten. Die weiteren Zeilen sind im Format

```
Parameter = Wert
```

gehalten. Als Parameter werden erkannt:

- `A`
- `B`
- `Gamma*`
- `Betas`
- `CMT`

Für die Parameter `Betas` und `CMT` müssen die Einzelwerte durch „ ; “ getrennt werden.

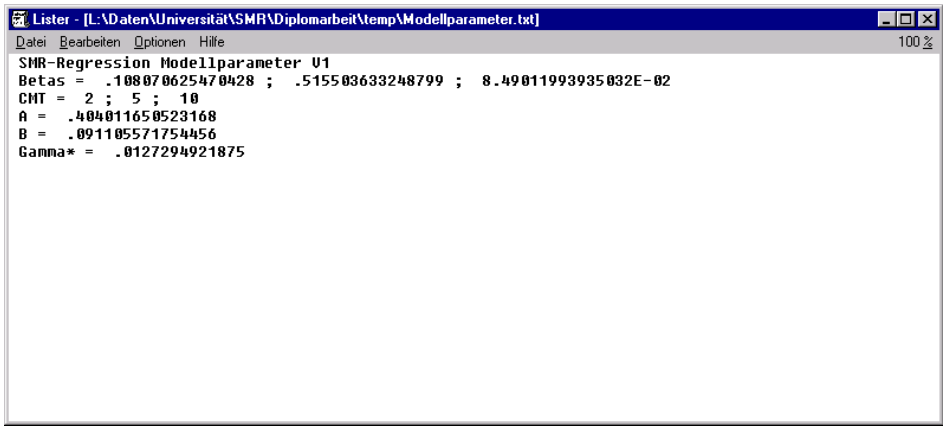


Abbildung 5.5.: Dateiformat für Modellparameter

A. Quellcode

A.1. Module

A.1.1. Makros

```
Attribute VB_Name = "Makros"
Option Explicit
Option Base 1
Dim regression() As Variant, spotrates() As Variant, smrs() As Variant
Dim regrcount As Integer, spotcount As Integer, smrcount As Integer

Private Function getSMRRegression(kursdatum As Date, vspots As Variant, _
                    vsmr As Variant, _
                    par As Variant, _
                    Optional parB As Double = 0, _
                    Optional parGammaStern As Double = 0, _
                    Optional parCMTs As Variant, _
                    Optional parBetas As Variant) As Object
    Dim i As Integer, typ As Integer
    '*****************************************************
    ' Spotrates eingelesen ?
    '*****************************************************
    Dim mySpots As Object, myCMT As Object
    ' Typ abfragen
    Dim vgl1 As Variant, vgl2 As Variant
    typ = VarType(vspots)
    If typ = vbString Then
        vgl1 = vspots
    Else
        vgl1 = "Range"
    End If
    ' Objekt schon vorhanden
    For i = 1 To spotcount
        If spotrates(1, i) = vgl1 Then
            Exit For
        End If
    Next i
    If i > 0 And i <= spotcount Then
        ' Vorhandenes Objekt verwenden
        Set mySpots = spotrates(2, i)
        If typ = vbString Then
            mySpots.read vspots
        Else
```

```
            mySpots.readFromMatrix vspots, kursdatum
        End If
        Set myCMT = spotrates(3, i)
    Else
        Set mySpots = New spotrates
        If typ = vbString Then
            mySpots.read (vspots)
        Else
            mySpots.readFromMatrix vspots, kursdatum
            'mySpots.setRates sr
        End If
        If mySpots.isOK Then
            ' CMT erstellen
            Set myCMT = New cmt
            myCMT.setSpotRates mySpots
            ' Merken
            spotcount = spotcount + 1
            ReDim Preserve spotrates(3, spotcount)
            spotrates(1, spotcount) = vgl1
            Set spotrates(2, spotcount) = mySpots
            Set spotrates(3, spotcount) = myCMT
        Else
            ' Spotrates nicht gefunden
            getSMRRegression = Empty
            Exit Function
        End If
    End If
    '*****************************************************
    ' SMR eingelesen ?
    '*****************************************************
    ' Typ abfragen
    Dim smr As Double
    typ = VarType(vsmr)
    If typ = vbString Then
        vgl1 = vsmr
        vgl2 = "Datei"
    ElseIf typ = vbDecimal Or typ = vbDouble Or typ = vbSingle Or _
            typ = vbInteger Or typ = vbLong Then
        smr = vsmr
        vgl1 = smr
        'vgl2 = mySpots.getLastDate
        vgl2 = kursdatum
    Else
        getSMRRegression = Empty
        Exit Function
    End If
    ' Objekt schon vorhanden
    Dim mySMR As Object
    For i = 1 To smrcount
        If smrs(1, i) = vgl1 And smrs(2, i) = vgl2 Then
            Exit For
        End If
    Next i
```

```
If i > 0 And i <= smrcount Then
    ' Gefunden
    Set mySMR = smrs(3, i)
Else
    ' neu erstellen
    Set mySMR = New smr
    If typ = vbString Then
        Dim filename As String
        filename = vgl1                                          100
        mySMR.read filename
    Else
        'mySMR.setRate mySpots.getLastDate, smr
        mySMR.setRate kursdatum, smr
    End If
    If mySMR.isOK Then
        ' Merken
        smrcount = smrcount + 1
        ReDim Preserve smrs(3, smrcount)
        smrs(1, smrcount) = vgl1                                 110
        smrs(2, smrcount) = vgl2
        Set smrs(3, smrcount) = mySMR
    Else
        ' SMR nicht ok
        getSMRRegression = Empty
        Exit Function
    End If
End If
'*****************************************************
' Regression eingelesen ?                                     120
'*****************************************************
Dim myRegr As Object
Dim pt As Integer, parvgl As String
pt = VarType(par.value)
If pt = vbString Then
    parvgl = par.value
Else
    parvgl = " !!! Modellparameter direkt angegeben !!! "
End If
' Objekt vorhanden ?                                          130
For i = 1 To regrcount
    If regression(1, i) = parvgl Then
        Exit For
    End If
Next i
If i > 0 And i <= regrcount Then
    ' Objekt ist vorhanden
    Set myRegr = regression(2, i)
Else
    ' Objekt erstellen                                        140
    Set myRegr = New SMRRegression
    If pt = vbString Then
        If myRegr.readFromFile(par.value) Then
            regrcount = regrcount + 1
```

```
                ReDim Preserve regression(2, regrcount)
                regression(1, regrcount) = par
                Set regression(2, regrcount) = myRegr
            Else
                ' Parameter nicht eingelesen
                getSMRRegression = Empty
                Exit Function
            End If
        End If
    End If
    If Not pt = vbString Then
        ' Parameterwerte setzen
        Dim sos As Double
        sos = par.value
        If Not myRegr.setParameter(sos, parB, parGammaStern, parCMTs, _
                                   parBetas) Then
            getSMRRegression = Empty
            Exit Function
        End If
    End If
    ' Spotrates/CMT/SMR/Kursdatum in Regression setzen
    myRegr.setSpotRates mySpots
    myRegr.setCMT myCMT
    myRegr.setSMR mySMR
    myRegr.setActualDate kursdatum
    Set getSMRRegression = myRegr
End Function

' Bewertungsfunktion
Public Function SMRBewertung(kursdatum As Date, vol As Double, _
                vspots As Variant, vsmr As Variant, _
                rlz As Double, freq As Double, _
                margin As Double, kupon As Double, _
                par As Variant, _
                Optional parB As Double = 0, _
                Optional parGammaStern As Double = 0, _
                Optional parCMTs As Variant, _
                Optional parBetas As Variant) As Double
    Dim myRegr As Variant
    Set myRegr = getSMRRegression(kursdatum, vspots, vsmr, _
                par, parB, parGammaStern, parCMTs, parBetas)
    ' Bewerten
    If myRegr.initValuation() Then
        SMRBewertung = myRegr.getValue(0, rlz, vol, freq, margin, kupon)
    Else
        SMRBewertung = 0
    End If

End Function

' Detailierte Bewertungsfunktion
Public Function SMRDetailBewertung(kursdatum As Date, vol As Double, _
                vspots As Variant, vsmr As Variant, _
```

```
                rlz As Double, freq As Double, _
                margin As Double, kupon As Double, _
                par As Variant, _
                Optional parB As Double = 0, _
                Optional parGammaStern As Double = 0, _
                Optional parCMTs As Variant, _
                Optional parBetas As Variant) As Variant
    Dim myRegr As Variant
    Set myRegr = getSMRRegression(kursdatum, vspots, vsmr, _
                par, parB, parGammaStern, parCMTs, parBetas)
    ' Bewerten
    If myRegr.initValuation() Then
        SMRDetailBewertung = myRegr.getDetailValue(0, rlz, vol, freq, _
                                                   margin, kupon)
    Else
        SMRDetailBewertung = 0
    End If
End Function

' Rücksetzen
Public Sub reset()
    smrcount = 0
    spotcount = 0
    regrcount = 0
End Sub


Public Function SMRKalibrierung(vspot As Variant, vsmr As Variant, _
        vkali As Variant, vcmt As Variant, fsrvol As Double, _
        smrvon As Date, smrbis As Date, kursdatum As Date) As Variant
    Dim i As Integer
    ' Spotrates erstellen
    Dim r As Object
   Set r = New spotrates
    If VarType(vspot) = vbString Then
        r.read vspot
    Else
        r.readFromMatrix vspot, kursdatum
    End If
    ' CMT erstellen
    Dim c As Object
    Set c = New cmt
    c.setSpotRates r
    ' SMR erstellen
    Dim s As Object
    Set s = New smr
    If VarType(vsmr) = vbString Then
        Dim ichweissesnicht As String
        ichweissesnicht = vsmr
        s.read ichweissesnicht
    Else
        s.readFromMatrix vsmr
    End If
```

```
    ' SMRRegression erstellen
    Dim sr As Object
    Set sr = New SMRRegression
    sr.setSMR s
    sr.setCMT c
    sr.setSpotRates r
    ' CMT-Laufzeiten auslesen
    Dim cmt As Variant, cmtcount As Integer
    cmt = vcmt
    cmtcount = UBound(cmt, 1)                                          260
    Dim cmts As Variant
    ReDim cmts(cmtcount)
    For i = 1 To cmtcount
        cmts(i) = cmt(i, 1)
    Next i
    ' FSR-Volatilität setzen
    sr.setFSRVolatility fsrvol
    ' Kalibrierungswerte setzen
    Dim kali As Variant
    kali = vkali                                                       270
    i = 1
    For i = 1 To UBound(kali)
        sr.addCalibrateValue kali(i, 5), kali(i, 1), kali(i, 2), _
                             kali(i, 3), kali(i, 4)

    Next i
    sr.setActualDate kursdatum
    'Kalibrieren
    sr.calibrate cmts, smrvon, smrbis
    ' Werte auslesen                                                   280
    Dim betas As Variant
    betas = sr.getBetas
    Dim ret As Variant
    ReDim ret(3 + UBound(betas) - LBound(betas) + 1, 1)
    ret(1, 1) = sr.getA
    ret(2, 1) = sr.getB
    ret(3, 1) = sr.getGammaStern
    For i = LBound(betas) To UBound(betas)
        ret(i - LBound(betas) + 4, 1) = betas(i)
    Next i                                                             290
    SMRKalibrierung = ret
End Function
```

A.1.2. Zeichenketten

```
Attribute VB_Name = "Zeichenketten"
Option Explicit

Public Function doubleValue(txt As String) As Double
    ' Alle "," finden
    Dim c As Integer
    Dim kommas(20) As Integer, k As Integer
    c = 1
    k = 0
    While InStr(c, txt, ",") > 0
        c = InStr(c, txt, ",")
        k = k + 1
        kommas(k) = c
        c = c + 1
    Wend
    ' Alle "." finden
    Dim dots(20) As Integer, d As Integer
    c = 1
    d = 0
    While InStr(c, txt, ".") > 0
        c = InStr(c, txt, ".")
        d = d + 1
        dots(d) = c
        c = c + 1
    Wend
    ' Alle "," durch "." ersetzen
    Dim i As Integer
    For i = 1 To k
        Mid(txt, kommas(i), 1) = "."
    Next i
    ' Alle "." entfernen
    For i = 1 To d
        txt = Left(txt, dots(i) - i) + Mid(txt, dots(i) - i + 2)
    Next i
    doubleValue = Val(txt)
End Function
```

A.2. Klassenmodule

A.2.1. SpotRates

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "SpotRates"
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit
Option Base 1
Private myRates, myMaturity, myDates As Variant
Private initialized As Boolean

Private Sub Class_Initialize()
    initialized = False
End Sub

'*********************************************************************
' Spotrates einlesen
'*********************************************************************
Public Function read(ByVal filename As String) As Boolean
    ' Existiert Datei ?
    Dim h As String
    h = Dir(filename)
    If h <> "" Then
        ' Ja -> Standardformat
        readStandard (filename)
    Else
        ' Nein -> ProfitLine-Format
        readPL (filename)
    End If
    read = initialized
End Function

'*********************************************************************
' ProfitLine-Spotrates einlesen
'*********************************************************************
Private Sub readPL(ByVal filename As String)
    Dim aktname, pfad, zeile As String
    Dim laufzeit, wert As Double
    Dim spotcount, i, pos, ha, hb As Integer
    Dim datum As Date
    ' Pfad aus filename ermitteln
    pos = 0
    While InStr(pos + 1, filename, "\") > 0
        pos = InStr(pos + 1, filename, "\")
    Wend
    If pos > 1 Then
```

```
    pfad = Left(filename, pos)
Else
    pfad = ""
End If
' Verfügbaren Datumsbereich ermitteln
Dim validDateCount As Integer, datePointer As Integer
Dim firstFile As Boolean
validDateCount = 0
firstFile = True
ReDim myDates(1) As Date
aktname = Dir(filename + "*.txt")
While aktname <> ""
    datePointer = 1
    ' Aktuelle Datei einlesen
    On Error GoTo datumsfehler
    Open pfad + aktname For Input As #1
    ' Die ersten Zeilen überspringen
    While Not EOF(1) And (InStr(zeile, "Datum") = 0 Or _
                          InStr(zeile, "Rendite") = 0)
        Line Input #1, zeile
    Wend
    On Error GoTo 0
    ' Einzelne Zeilen einlesen
    Do While Not EOF(1)
        Line Input #1, zeile
        i = InStr(zeile, Chr(9))
        If i > 2 Then
            datum = CDate(Left(zeile, i - 1))
            If firstFile And datePointer > validDateCount Then
                ' Datum am Ende hinzufügen
                ReDim Preserve myDates(datePointer) As Date
                myDates(datePointer) = datum
                validDateCount = validDateCount + 1
                datePointer = datePointer + 1
            ElseIf datePointer > validDateCount Then
                Exit Do
            ElseIf myDates(datePointer) = datum Then
                ' Datum schon vorhanden -> überspringen
                datePointer = datePointer + 1
            ElseIf myDates(datePointer) > datum Then
                ' Keine Spotrate für diese(s) Datum/Laufzeit vorhanden
                While datePointer <= validDateCount And _
                                     myDates(datePointer) > datum
                    For ha = datePointer To validDateCount - 1
                        myDates(ha) = myDates(ha + 1)
                    Next ha
                    validDateCount = validDateCount - 1
                Wend
                datePointer = datePointer + 1
                If validDateCount > 0 Then
                    ReDim Preserve myDates(validDateCount) As Date
                End If
            ElseIf myDates(datePointer) < datum Then
```

```
                    ' Noch kein Datumseintrag vorhanden -> ignorieren
                End If
            End If
        Loop
        Close #1
datumsfehler:
        ' Nächsten Dateinamen ermitteln
        aktname = Dir
        firstFile = False
    Wend
    ' Gültige Daten gefunden ?
    If validDateCount = 0 Then
        Exit Sub
    End If
    ' Array initialisieren
    ReDim myRates(validDateCount, 1) As Double
    ReDim myMaturity(1) As Double
    '********************
    ' Spotrates einlesen
    '********************
    Dim spotPointer
    spotcount = 0
    aktname = Dir(filename + "*.txt")
    While aktname <> ""
        ' Array dimensionieren
        spotcount = spotcount + 1
        datePointer = 1
        ReDim Preserve myRates(validDateCount, spotcount)
        ReDim Preserve myMaturity(spotcount)
        ' Laufzeit ermitteln
        i = 2
        While i < Len(aktname) And Val(Mid(aktname, i)) = 0
            i = i + 1
        Wend
        If i < Len(aktname) Then
            laufzeit = Val(Mid(aktname, i))
        Else
            GoTo dateifehler
        End If
        spotPointer = spotcount - 1
        Do While (spotPointer >= 1)
            If myMaturity(spotPointer) < laufzeit Then
                Exit Do
            End If
            For ha = 1 To validDateCount
                myRates(ha, spotPointer + 1) = myRates(ha, spotPointer)
            Next ha
            myMaturity(spotPointer + 1) = myMaturity(spotPointer)
            spotPointer = spotPointer - 1
        Loop
        spotPointer = spotPointer + 1
        myMaturity(spotPointer) = laufzeit
        ' Aktuelle Datei einlesen
```

```
        On Error GoTo dateifehler
        Open pfad + aktname For Input As #1
        ' Die ersten Zeilen überspringen
        While Not EOF(1) And (InStr(zeile, "Datum") = 0 Or _
                              InStr(zeile, "Rendite") = 0)
            Line Input #1, zeile                                    160
        Wend
        On Error GoTo 0
        ' Einzelne Zeilen einlesen
        Do While Not EOF(1)
            Line Input #1, zeile
            i = InStr(zeile, Chr(9))
            If i > 2 Then
                datum = CDate(Left(zeile, i - 1))
                wert = doubleValue(Mid(zeile, i + 1)) / 100
                If myDates(datePointer) = datum Then                170
                    myRates(datePointer, spotPointer) = wert
                    datePointer = datePointer + 1
                    If datePointer > UBound(myDates) Then
                        Exit Do
                    End If
                End If
            End If
        Loop
        Close #1
        GoTo dateiok                                                180
dateifehler:
        spotcount = spotcount - 1
        If spotcount > 0 Then
            ReDim Preserve myRates(validDateCount, spotcount)
            ReDim Preserve myMaturity(spotcount)
        End If
dateiok:
        ' Nächsten Dateinamen ermitteln
        aktname = Dir
    Wend                                                            190
    initialized = True
End Sub

'*********************************************************************
' Standard-Spotrates einlesen
'*********************************************************************
Private Sub readStandard(ByVal filename As String)
    Dim zeile As String, htxt As String, anfang As Integer
    Dim i As Integer, z As Integer, count As Integer, matcount As Integer
    initialized = False                                             200
    ' Datei öffnen
    On Error GoTo dateifehler
    Open filename For Input As #1
    On Error GoTo 0
    ' Erste Zeile mit Laufzeiten einlesen
    ReDim myMaturity(1) As Double
    Line Input #1, zeile
```

```
matcount = 0
Do While InStr(zeile, Chr(9)) > 0
    matcount = matcount + 1                                          210
    ReDim Preserve myMaturity(matcount)
    zeile = Trim(Mid(zeile, InStr(zeile, Chr(9)) + 1))
    If InStr(zeile, Chr(9)) > 0 Then
        htxt = Left(zeile, InStr(zeile, Chr(9)) - 1)
    Else
        htxt = zeile
    End If
    myMaturity(matcount) = Val(htxt)
Loop
If matcount > 0 Then                                                 220
    ' Datumsbereich einlesen
    anfang = Seek(1)
    ReDim myDates(1) As Date
    count = 0
    Do While Not EOF(1)
        Line Input #1, zeile
        i = InStr(zeile, Chr(9))
        If i < 9 Then
            Exit Do
        End If                                                       230
        ' Datum
        count = count + 1
        ReDim Preserve myDates(count)
        myDates(count) = CDate(Left(zeile, i - 1))
    Loop
    If count > 0 Then
        Seek #1, anfang
        ' Spotrates
        ReDim myRates(count, matcount) As Double
        z = 0                                                        240
        Do While Not EOF(1) And z < count
            z = z + 1
            Line Input #1, zeile
            i = 0
            Do While InStr(zeile, Chr(9)) > 0
                i = i + 1
                zeile = Trim(Mid(zeile, InStr(zeile, Chr(9)) + 1))
                If InStr(zeile, Chr(9)) > 0 Then
                    htxt = Left(zeile, InStr(zeile, Chr(9)) - 1)
                Else                                                 250
                    htxt = zeile
                End If
                myRates(z, i) = Val(htxt) / 100
            Loop
        Loop
        initialized = (z = count)
    End If
End If
' Datei schließen
Close #1                                                             260
```

```
dateifehler:
End Sub

'*********************************************************************
' Spotrates aus Tabelle/Array auslesen
'*********************************************************************
Public Sub readFromMatrix(vmatrix As Variant, datum As Date)
    initialized = False
    Dim c As Integer, r As Integer
    ' Zuweisen, um aus Range-Objekt ein Array zu machen                    270
    Dim matrix As Variant, row As Integer, col As Integer
    matrix = vmatrix
    row = UBound(matrix, 1)
    col = UBound(matrix, 2)
    If col > 2 Then
        ' FORMAT: History
        '          X     LZ1 LZ2 LZ3 ...
        '          D1    R11 R12 R13 ...
        '          D2    R21 R22 R23 ...
        ' Laufzeiten auslesen                                              280
        ReDim myMaturity(col - 1)
        For c = 2 To col
            myMaturity(c - 1) = matrix(1, c)
        Next c
        ' Datum+Rates auslesen
        ReDim myDates(row - 1)
        ReDim myRates(row - 1, col - 1)
        For r = 2 To row
            myDates(r - 1) = matrix(r, 1)
            For c = 2 To col                                             290
                myRates(r - 1, c - 1) = matrix(r, c)
            Next c
        Next r
    Else
        ' FORMAT: Nur Aktuell
        ' LZ1 R1
        ' LZ2 R2
        ' LZ3 R3
        ' Laufzeiten auslesen
        ReDim myMaturity(row)                                            300
        For r = 1 To row
            myMaturity(r) = matrix(r, 1)
        Next r
        ' Datum+Rates auslesen
        ReDim myDates(1)
        ReDim myRates(1, row)
        myDates(1) = datum
        For r = 1 To row
            myRates(1, r) = matrix(r, 2)
        Next r                                                           310
    End If
    initialized = True
End Sub
```

```
'*****************************************************************
' Aktuelle Spotrates setzen
'*****************************************************************
Public Sub setRates(rates As Variant, Optional datum As Date = Empty)
    initialized = False
    If IsEmpty(datum) Then                                          320
        datum = Date
    End If
    '*********************************************
    ' Typ prüfen
    '*********************************************
    Dim typ As Integer, i As Integer
    If VarType(rates) < vbArray Then
        Exit Sub
    End If
    If UBound(rates, 2) - LBound(rates, 2) <> 1 Then                330
        Exit Sub
    End If
    ' Laufzeiten prüfen
    For i = LBound(rates) To UBound(rates)
        typ = VarType(rates(i, 1))
        If Not (typ = vbDouble Or typ = vbSingle Or typ = vbDecimal Or _
                                    typ = vbInteger Or typ = vbLong) Then
            Exit Sub
        End If
        If rates(i, 1) <= 0 Then Exit Sub                           340
    Next i
    ' Zinssätze prüfen
    For i = LBound(rates) To UBound(rates)
        typ = VarType(rates(i, 2))
        If Not (typ = vbDouble Or typ = vbSingle Or typ = vbDecimal Or _
                                    typ = vbInteger Or typ = vbLong) Then
            Exit Sub
        End If
        If rates(i, 2) <= 0 Then Exit Sub
    Next i                                                          350
    '*********************************************
    ' Laufzeiten kopieren
    '*********************************************
    Dim count As Integer
    count = UBound(rates) - LBound(rates) + 1
    ReDim myMaturity(count) As Double
    For i = 1 To count
        myMaturity(i) = rates(i + LBound(rates) - 1, 1)
    Next i
    '*********************************************                  360
    ' Datumsfeld ausfüllen
    '*********************************************
    ReDim myDates(1) As Date
    myDates(1) = datum
    '*********************************************
    ' Zinssätze kopieren
```

```
    '*********************************************
    ReDim myRates(1, count) As Double
    For i = 1 To count
        myRates(1, i) = rates(i + LBound(rates) - 1, 2)
    Next i
    initialized = True
End Sub

'*****************************************************************
' Daten abfragen
'*****************************************************************
Public Function isOK() As Boolean
    isOK = initialized
End Function

Public Function getFirstDate() As Date
    getFirstDate = myDates(UBound(myDates))
End Function

Public Function getLastDate() As Date
    getLastDate = myDates(1)
End Function

'*****************************************************************
' Aktuellen Zinssatz für angegebene Laufzeit abfragen
'*****************************************************************
Public Function getRateFor(ByVal maturity As Double)
    getRateFor = getHistoryRateFor(myDates(1), maturity)
End Function

'*****************************************************************
' Historischen Zinssatz für angegebene Laufzeit abfragen
'*****************************************************************
Public Function getHistoryRateFor(ByVal datum As Date, _
                                  ByVal maturity As Double)
    ' Passendes Datum ermitteln
    Dim dateindex As Integer
    dateindex = 1
    While dateindex < UBound(myDates) And myDates(dateindex) > datum
        dateindex = dateindex + 1
    Wend
    If myDates(dateindex) <> datum Then
        'MsgBox ("Zinssatz für ungültiges Datum abgefragt")
        getHistoryRateFor = 0
        Exit Function
    End If
    ' Passende Laufzeit finden
    Dim index As Integer
    index = 1
    While (myMaturity(index) < maturity) And index < UBound(myMaturity)
        index = index + 1
    Wend
    ' Zinssatz berechnen
```

```
    If (index = 1) Then                                                    420
        getHistoryRateFor = myRates(dateindex, index)
    ElseIf (myMaturity(index) > maturity) Then
        getHistoryRateFor = (myMaturity(index) - maturity) / _
                    (myMaturity(index) - myMaturity(index - 1)) * _
                    myRates(dateindex, index - 1) + _
                    (maturity - myMaturity(index - 1)) / _
                    (myMaturity(index) - myMaturity(index - 1)) * _
                    myRates(dateindex, index)
    Else
        getHistoryRateFor = myRates(dateindex, index)                      430
    End If
End Function

'*********************************************************************
' Diskontierungsfaktor abfragen
'*********************************************************************
' Der zum Zeitpunkt valuetime gültige Diskontierungsfaktor für
' eine Laufzeit bis endtime bzw. Restlaufzeit=endtime-valuetime
Public Function getDiscountFactor(ByVal valuetime As Double, _
                                  ByVal endtime As Double)                 440
    getDiscountFactor = getHistoryDiscountFactor(myDates(1), _
                                                 valuetime, endtime)
End Function

Public Function getHistoryDiscountFactor(ByVal datum As Date, _
            ByVal valuetime As Double, ByVal endtime As Double)
    ' Vasicek: Basis 61
    If valuetime <> 0 Then
        MsgBox "Sorry, nicht implementiert", vbCritical, _
                                        "Diskontierungsfaktor"             450
        Exit Function
    End If
    getHistoryDiscountFactor = (1 + getHistoryRateFor(datum, endtime)) ^ _
                                                                  -endtime
End Function

'*********************************************************************
' Annuitätenfaktor
'*********************************************************************
' Pichler 47(62)                                                        460
Public Function getHistoryAF(ByVal datum As Date, _
            ByVal time As Double, ByVal offset As Double, _
            ByVal maturity As Double, ByVal freq As Double)
    Dim i As Integer
    Dim wert As Double
    wert = 0
    For i = 1 To maturity / freq
        wert = wert + getHistoryDiscountFactor(datum, time, _
                                               time + offset + i * freq)
    Next i                                                                 470
    getHistoryAF = freq * wert
End Function
```

```
Public Function getAF(ByVal time As Double, ByVal offset As Double, _
            ByVal maturity As Double, ByVal freq As Double)
    getAF = getHistoryAF(myDates(1), time, offset, maturity, freq)
End Function

'*********************************************************************
' Swap Rate                                                          480
'*********************************************************************
' Pichler 47(61), 24(14,15)
Public Function getHistorySwapRate(ByVal datum As Date, _
                ByVal valuetime As Double, ByVal maturity As Double)
    getHistorySwapRate = (1 - getHistoryDiscountFactor(datum, valuetime, _
                                                valuetime + maturity)) / _
                getHistoryAF(datum, valuetime, 0, maturity, 1)
End Function

Public Function getSwapRate(ByVal valuetime As Double, _                490
                            ByVal maturity As Double)
    getSwapRate = getHistorySwapRate(myDates(1), valuetime, maturity)
End Function

'*********************************************************************
' Forward Rate
'*********************************************************************
' Aussenegg 18
Public Function getHistoryForwardRate(ByVal datum As Date, _
                    ByVal valuetime As Double, ByVal endtime As Double)  500
    getHistoryForwardRate = ((1 + getHistoryRateFor(datum, endtime)) ^ _
                                                        endtime) / _
                    ((1 + getHistoryRateFor(datum, valuetime)) ^ _
                                                        valuetime) - 1
End Function

Public Function getForwardRate(ByVal valuetime As Double, _
                                ByVal endtime As Double)
    getForwardRate = getHistoryForwardRate(myDates(1), valuetime, endtime)
End Function                                                             510

'*********************************************************************
' Forward Swap Rate
'*********************************************************************
' Pichler 47(63)
Public Function getHistoryForwardSwapRate(ByVal datum As Date, _
            ByVal valuetime As Double, _
            ByVal maturity As Double, ByVal offset As Double)
    getHistoryForwardSwapRate = _
        (getHistoryDiscountFactor(datum, valuetime, valuetime + offset) - _  520
        getHistoryDiscountFactor(datum, valuetime, _
                                    valuetime + offset + maturity)) / _
        getHistoryAF(datum, valuetime, offset, maturity, 1)
End Function
```

```
Public Function getForwardSwapRate(ByVal valuetime As Double, _
                ByVal maturity As Double, ByVal offset As Double)
    getForwardSwapRate = getHistoryForwardSwapRate(myDates(1), _
                                                   valuetime, maturity, offset)
End Function                                                                    530

'*********************************************************************
' Datumsarray abfragen
'*********************************************************************
Public Function getAvailableDates()
    Dim tmpdate As Variant
    ReDim tmpdate(UBound(myDates)) As Double
    Dim i As Integer
    For i = 1 To UBound(myDates)
        tmpdate(i) = myDates(i)                                                 540
    Next
    getAvailableDates = tmpdate
End Function

'*********************************************************************
' Laufzeiten abfragen
'*********************************************************************
Public Function getAvailableMaturities()
    Dim tmpmat As Variant
    ReDim tmpmat(UBound(myMaturity)) As Double                                  550
    Dim i As Integer
    For i = 1 To UBound(myMaturity)
        tmpmat(i) = myMaturity(i)
    Next
    getAvailableMaturities = tmpmat
End Function
```

A.2.2. CMT

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "CMT"
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit
Private mySpotRates As Object
Private myLastDate As Date

'*********************************************************************
' SpotMySpotRates setzen
'*********************************************************************
Public Sub setSpotRates(sr As Object)
    Set mySpotRates = sr
    myLastDate = mySpotRates.getAvailableDates()(1)
End Sub

'*********************************************************************
' Daten abfragen
'*********************************************************************
Public Function getFirstDate() As Date
    getFirstDate = mySpotRates.getFirstDate()
End Function

Public Function getLastDate() As Date
    getLastDate = mySpotRates.getLastDate()
End Function

'*********************************************************************
' CMT-Wert für angegebene Laufzeit abfragen
'*********************************************************************
Public Function getRateFor(ByVal maturity As Double)
    getRateFor = getHistoryRateFor(myLastDate, maturity)
End Function

Public Function getHistoryRateFor(ByVal datum As Date, _
                                  ByVal maturity As Double)
    If maturity <> Int(maturity) Then
        MsgBox ("CMT nur für ganzjährige Laufzeiten implementiert.")
        getHistoryRateFor = 0
        Exit Function
    End If
    Dim i As Integer
    Dim wert As Double
    wert = 0
    For i = maturity To 1 Step -1
        wert = wert + mySpotRates.getHistoryDiscountFactor(datum, 0, i)
    Next i
```

```
    getHistoryRateFor = (1 - _
            mySpotRates.getHistoryDiscountFactor(datum, 0, maturity)) / _
            wert
End Function

'*********************************************************************
' Convexity Adjustment
'*********************************************************************
' Pichler 71(88-90)
Public Function getConvexityAdjustment(ByVal valuetime As Double, _
            ByVal maturity As Double, ByVal endtime As Double, _
            ByVal vol As Double, ByVal freq As Double) As Double
    getConvexityAdjustment = getHistoryConvexityAdjustment( _
            myLastDate, valuetime, maturity, endtime, vol, freq)
End Function

Public Function getHistoryConvexityAdjustment(ByVal datum As Date, _
            ByVal valuetime As Double, _
            ByVal maturity As Double, ByVal endtime As Double, _
            ByVal vol As Double, ByVal freq As Double) As Double
    Dim i As Integer
    Dim f, pvs, pvss, N As Double
    f = mySpotRates.getHistoryForwardSwapRate(datum, valuetime, maturity, _
                                              endtime - freq - valuetime)
    pvs = 0
    For i = 1 To maturity
        pvs = pvs + i * (1 + f) ^ (-i - 1)
    Next i
    pvs = -f * pvs - maturity * (1 + f) ^ (-maturity - 1)

    pvss = 0
    For i = 1 To maturity
        pvss = pvss + i * (i + 1) * (1 + f) ^ (-i - 2)
    Next i
    pvss = f * pvss + maturity * (maturity + 1) * (1 + f) ^ (-maturity - 2)
    getHistoryConvexityAdjustment = -0.5 * (f ^ 2) * (vol ^ 2) * _
                                    endtime * pvss / pvs
End Function

'*********************************************************************
' Bewertung für eine Zahlung
'*********************************************************************
' Pichler 71(88)
Public Function getPaymentValue(ByVal valuetime As Double, _
            ByVal maturity As Double, ByVal endtime As Double, _
            ByVal vol As Double, ByVal freq As Double)
    getPaymentValue = mySpotRates.getDiscountFactor(valuetime, endtime) * _
        (mySpotRates.getForwardSwapRate(valuetime, maturity, _
                                        endtime - freq - valuetime) - _
        getConvexityAdjustment(valuetime, maturity, endtime, vol, freq))
End Function

'*********************************************************************
```

```
' Bewertung
'*********************************************************************
Public Function getValue(ByVal valuetime As Double, _
                ByVal maturity As Double, ByVal endtime As Double, _
                ByVal vol As Double, ByVal freq As Double)
    Dim i As Integer
    Dim v As Double
    v = mySpotRates.getDiscountFactor(valuetime, endtime)
    For i = 1 To maturity / freq
        v = v + getPaymentValue(valuetime, maturity, _
            valuetime + i * freq, vol, freq)
    Next
    getValue = v
End Function

'*********************************************************************
' Datumsarray abfragen
'*********************************************************************
Public Function getAvailableDates()
    getAvailableDates = mySpotRates.getAvailableDates()
End Function

'*********************************************************************
' Laufzeiten abfragen
'*********************************************************************
Public Function getAvailableMaturities()
    getAvailableMaturities = mySpotRates.getAvailableMaturities()
End Function
```

A.2.3. SMR

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "SMR"
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit
Option Base 1
Private myRates, myDates As Variant
Private initialized As Boolean

Private Sub Class_Initialize()
    initialized = False
End Sub

'*********************************************************************
' SMR einlesen
'*********************************************************************
Public Sub read(filename As String)
    On Error GoTo dateifehler
    Open filename For Input As #1
    On Error GoTo 0
    ' Erste Zeile einlesen und auf Tab prüfen
    Dim zeile As String
    Line Input #1, zeile
    Close #1
    ' Kein Tab -> PL
    If InStr(zeile, Chr(9)) = 0 Then
        readPL filename
    Else
        readStandard filename
    End If
dateifehler:
End Sub

'*********************************************************************
' ProfitLine-SMR einlesen
'*********************************************************************
Private Sub readPL(filename As String)
    Dim zeile As String
    Dim wert As Double
    Dim i, pos, h As Integer
    Dim datum As Date
    Dim datePointer As Integer
    ReDim myDates(1) As Date
    ReDim myRates(1) As Double
    datePointer = 1
    ' Datei einlesen
    On Error GoTo dateifehler
```

```
    Open filename For Input As #1
    ' Die ersten Zeilen überspringen
    Line Input #1, zeile
    Line Input #1, zeile
    Line Input #1, zeile
    Line Input #1, zeile
    On Error GoTo 0
    ' Einzelne Zeilen einlesen
    While Not EOF(1)                                                    60
        Line Input #1, zeile
        ' Position der Rendite ermitteln
        i = 1
        h = 1
        While i < 8 And InStr(h, zeile, Chr(9))
            h = InStr(h, zeile, Chr(9)) + 1
            i = i + 1
        Wend
        ' End-Position des Datums
        i = InStr(zeile, Chr(9))                                        70
        If i > 2 And h > 2 Then
            datum = CDate(Left(zeile, i - 1))
            wert = doubleValue(Mid(zeile, h)) / 100
            ReDim Preserve myDates(datePointer) As Date
            myDates(datePointer) = datum
            ReDim Preserve myRates(datePointer) As Double
            myRates(datePointer) = wert
            datePointer = datePointer + 1
        End If
    Wend                                                                80
    Close #1
    initialized = True
dateifehler:
End Sub

'*********************************************************************
' Standard-SMR einlesen
'*********************************************************************
Private Sub readStandard(filename As String)
    Dim zeile As String                                                 90
    Dim wert As Double
    Dim i, pos, h As Integer
    Dim datum As Date
    Dim datePointer As Integer
    ReDim myDates(1) As Date
    ReDim myRates(1) As Double
    datePointer = 1
    ' Datei einlesen
    On Error GoTo dateifehler
    Open filename For Input As #1                                       100
    On Error GoTo 0
    ' Einzelne Zeilen einlesen
    While Not EOF(1)
        Line Input #1, zeile
```

```
        ' End-Position des Datums
        i = InStr(zeile, Chr(9))
        If i > 2 Then
            datum = CDate(Left(zeile, i - 1))
            wert = Val(Mid(zeile, i + 1)) / 100
            ReDim Preserve myDates(datePointer) As Date
            myDates(datePointer) = datum
            ReDim Preserve myRates(datePointer) As Double
            myRates(datePointer) = wert
            datePointer = datePointer + 1
        End If
    Wend
    Close #1
    initialized = True
dateifehler:
End Sub

'*********************************************************************
' SMR aus Tabelle/Array auslesen
'*********************************************************************
Public Sub readFromMatrix(vmatrix As Variant)
    initialized = False
    Dim r As Integer
    ' Zuweisen, um as Range ein Array zu erstellen
    Dim matrix As Variant, row As Integer
    matrix = vmatrix
    row = UBound(matrix, 1)
    ' Datum+Rates auslesen
    ReDim myDates(row)
    ReDim myRates(row)
    For r = 1 To row
        myDates(r) = matrix(r, 1)
        myRates(r) = matrix(r, 2)
    Next r
    initialized = True
End Sub

'*********************************************************************
' Aktuelle SMR setzen
'*********************************************************************
Public Sub setRate(datum As Date, wert As Double)
    ReDim myDates(1) As Date
    myDates(1) = datum
    ReDim myRates(1) As Double
    myRates(1) = wert
    initialized = True
End Sub

'*********************************************************************
' Daten abfragen
'*********************************************************************
Public Function isOK() As Boolean
    isOK = initialized
```

```
End Function

Public Function getFirstDate() As Date                                    160
    getFirstDate = myDates(UBound(myDates))
End Function

Public Function getLastDate() As Date
    getLastDate = myDates(1)
End Function

'*********************************************************************
' SMR abfragen
'*********************************************************************  170
Public Function getHistoryRate(datum As Date)
    ' Passendes Datum ermitteln
    Dim dateindex As Integer
    dateindex = 1
    While dateindex < UBound(myDates) And myDates(dateindex) > datum
        dateindex = dateindex + 1
    Wend
    If myDates(dateindex) <> datum Then
        getHistoryRate = 0
        'MsgBox ("SMR für ungültiges Datum abgefragt")                    180
        Exit Function
    End If
    getHistoryRate = myRates(dateindex)
End Function

Public Function getRate() As Double
    getRate = myRates(1)
End Function

'*********************************************************************  190
' Datumsarray abfragen
'*********************************************************************
Public Function getAvailableDates()
    Dim tmpdate As Variant
    ReDim tmpdate(UBound(myDates)) As Double
    Dim i As Integer
    For i = 1 To UBound(myDates)
        tmpdate(i) = myDates(i)
    Next
    getAvailableDates = tmpdate                                           200
End Function
```

A.2.4. SMRRegression

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "SMRRegression"
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit
Option Base 1
' Fertig kalibriert ?
Private myCalibrated As Boolean
' Anzahl der Börsentage -> Verwendet für delta t
Private myDeltat As Double
' Parameter
Private myCMT As Object, mySMR As Object
Private myRates As Object
' Aktuelles Datum
Private actualDate As Date ' Gesetzt: readFromFile,calibrate
' SMR/CMT-Werte
Private mySMRArray As Variant, myCMTArray As Variant, _
        myCMTMaturity As Variant
Private myDeltaSMR As Variant, myDeltaCMT As Variant
Private myCount As Integer, myCMTCount As Integer
Private myUTAktuell As Double
' Ergebnisse der Delta-Regression
Private myRho As Double
Private myBetas As Variant
Private myVariance As Double
' Ergebnisse der Regression
Private myA As Double, myB As Double
Private myGamma As Double, myGammaStern As Double
' Kalibrierungswerte
Private myStartDate As Date, myEndDate As Date
Private myCalibrationCMTMaturities As Variant
Private myCalCount As Integer
Private myFSRVolatility As Double
Private myCalValue(10), myCalEndtime(10), myCalFreq(10), _
        myCalMargin(10), myCalKupon(10) As Double

' Objekt initialisieren
Private Sub Class_Initialize()
    myCalCount = 0
    myDeltat = 1 / 246
    myCalibrated = False
End Sub

'*************************************************************************
' Parameter aus Datei einlesen
'*************************************************************************
Public Function readFromFile(filename As String) As Boolean
```

```
myCalibrated = False
readFromFile = False
' Datei öffnen
On Error GoTo dateifehler
Open filename For Input As #1
' Kennung einlesen
Dim txt As String, i As Integer, count As Integer
Line Input #1, txt
If txt <> "SMR-Regression Modellparameter V1" Then
    MsgBox ("Ungültige Parameterdatei.")
    Exit Function
End If
' Werte einlesen
Do While Not EOF(1)
    Line Input #1, txt
    If Left(txt, 5) = "Betas" Then
        If InStr(txt, "=") > 0 Then
            ReDim myBetas(1)
            txt = Trim(Mid(txt, InStr(txt, "=") + 1)) + " ;"
            count = 1
            Do While InStr(txt, ";") > 0
                ReDim Preserve myBetas(count)
                myBetas(count) = Val(txt)
                txt = Trim(Mid(txt, InStr(txt, ";") + 1))
                count = count + 1
            Loop
        End If
    ElseIf Left(txt, 3) = "CMT" Then
        If InStr(txt, "=") > 0 Then
            ReDim myCMTMaturity(1)
            txt = Trim(Mid(txt, InStr(txt, "=") + 1)) + " ;"
            count = 1
            Do While InStr(txt, ";") > 0
                ReDim Preserve myCMTMaturity(count)
                myCMTMaturity(count) = Val(txt)
                txt = Trim(Mid(txt, InStr(txt, ";") + 1))
                count = count + 1
            Loop
            myCMTCount = count - 1
        End If
    ElseIf Left(txt, 2) = "A " Then
        myA = Val(Trim(Mid(txt, InStr(txt, "=") + 1)))
    ElseIf Left(txt, 2) = "B " Then
        myB = Val(Trim(Mid(txt, InStr(txt, "=") + 1)))
    ElseIf Left(txt, 7) = "Gamma* " Then
        myGammaStern = Val(Trim(Mid(txt, InStr(txt, "=") + 1)))
    End If
Loop
' Datei schließen
Close #1
On Error GoTo 0
' Fertig
myCalibrated = True
```

```
    readFromFile = True
    Exit Function
dateifehler:
    On Error GoTo 0
End Function

'*************************************************************************
' Parameter setzen
'*************************************************************************
Public Function setParameter(A As Double, B As Double, _
                gammaStern As Double, _
                vcmts As Variant, vbetas As Variant) As Boolean
    On Error GoTo parameterfehler
    setParameter = False
    myA = A
    myB = B
    myGammaStern = gammaStern
    myCMTCount = 0
    ' CMTs einlesen
    Dim cmts As Variant
    cmts = vcmts
    myCMTCount = UBound(cmts, 1)
    ReDim myCMTMaturity(myCMTCount)
    Dim i As Integer
    For i = 1 To myCMTCount
        myCMTMaturity(i) = cmts(i, 1)
    Next i
    ' Betas einlesen
    ReDim myBetas(myCMTCount)
    Dim betas As Variant
    betas = vbetas
    For i = 1 To myCMTCount
        myBetas(i) = betas(i, 1)
    Next i
    On Error GoTo 0
    ' Fertig
    myCalibrated = True
    setParameter = True
    Exit Function
parameterfehler:
    On Error GoTo 0
End Function

'*************************************************************************
' Bewertung initialisieren nach dem Einlesen der Parameter
' und setzen der Spots,CMT,SMR
'*************************************************************************
Public Function initValuation() As Boolean
    ' Bewertungsdatum
    If actualDate = Empty Then
        actualDate = Application.min(myRates.getLastDate, _
                                     myCMT.getLastDate, mySMR.getLastDate)
    End If
```

```
    ' Array erstellen
    Dim tmp As Variant, i As Integer
    ReDim tmp(UBound(myCMTMaturity))
    For i = 1 To UBound(myCMTMaturity)
        tmp(i) = myCMTMaturity(i)
    Next i
    If Not createArrays(tmp, actualDate, actualDate) Then
        ' Fertig
        initValuation = False
        Exit Function
    End If
    ' UT berechnen
    berechneUT
    initValuation = True
End Function

'*************************************************************************
' Parameter in Datei schreiben
'*************************************************************************
Public Function writeToFile(filename As String) As Boolean
    ' Datei öffnen
    On Error GoTo dateifehler
    Open filename For Output As #1
    '  Kennung
    Print #1, "SMR-Regression Modellparameter V1"
    ' Parameter schreiben
    ' Betas
    Dim txt As String, i As Integer
    txt = ""
    For i = 1 To UBound(myBetas)
        txt = txt + Str(myBetas(i)) + " ; "
    Next i
    txt = Left(txt, Len(txt) - 3)
    Print #1, "Betas = " + txt
    ' CMT-Laufzeiten
    txt = ""
    For i = 1 To UBound(myCMTMaturity)
        txt = txt + Str(myCMTMaturity(i)) + " ; "
    Next i
    txt = Left(txt, Len(txt) - 3)
    Print #1, "CMT = " + txt
    ' Restliche Parameter
    Print #1, "A = " + Str(myA)
    Print #1, "B = " + Str(myB)
    Print #1, "Gamma* = " + Str(myGammaStern)
    ' Datei schließen
    Close #1
    On Error GoTo 0
    writeToFile = True
    Exit Function
dateifehler:
    writeToFile = False
End Function
```

```
'*********************************************************************
' FSR-Volatilität setzen/abfragen
'*********************************************************************
Public Sub setFSRVolatility(vol As Double)
    myFSRVolatility = vol
End Sub

Public Function getFSRVolatility() As Double
    getFSRVolatility = myFSRVolatility                                  220
End Function

'*********************************************************************
' Kalibrierungs-Daten löschen
'*********************************************************************
Public Sub resetCalibration()
    myCalCount = 0
    myFSRVolatility = 0
    myStartDate = Empty
    myEndDate = Empty                                                   230
End Sub

'*********************************************************************
' Kalibrierungs-Daten hinzufügen
'*********************************************************************
Public Function addCalibrateValue(ByVal value As Double, _
            ByVal endtime As Double, _
            ByVal freq As Double, _
            ByVal margin As Double, ByVal kupon As Double)
    myCalCount = myCalCount + 1                                         240
    myCalValue(myCalCount) = value
    myCalEndtime(myCalCount) = endtime
    myCalFreq(myCalCount) = freq
    myCalMargin(myCalCount) = margin
    myCalKupon(myCalCount) = kupon
End Function

Public Function getCalibrationValues() As Variant
    Dim tmp As Variant, i As Integer
    ReDim tmp(myCalCount, 5)                                            250
    For i = 1 To myCalCount
        tmp(i, 1) = myCalEndtime(i)
        tmp(i, 2) = myCalFreq(i)
        tmp(i, 3) = myCalMargin(i)
        tmp(i, 4) = myCalKupon(i)
        tmp(i, 5) = myCalValue(i)
    Next i
    getCalibrationValues = tmp
End Function
                                                                        260
'*********************************************************************
' Kalibrierung
'*********************************************************************
```

```
Public Sub calibrate(ByVal cmtmat As Variant, ByVal startDate As Date, _
                     ByVal endDate As Date)
    If myCalCount = 0 Then
        MsgBox "Keine Kalibierungswerte angegeben."
        Exit Sub
    End If
    On Error GoTo fehlendeDaten                                          270
    If actualDate = Empty Then
        actualDate = Application.min(myRates.getLastDate, _
                                     myCMT.getLastDate, mySMR.getLastDate)
    End If
    On Error GoTo 0
    GoTo datenok
fehlendeDaten:
    MsgBox "Fehlende Daten."
    Exit Sub
datenok:                                                                 280
    ' Werte für spätere Abfrage merken
    myStartDate = startDate
    myEndDate = endDate
    myCalibrationCMTMaturities = cmtmat
    ' SMR/CMT-Werte einlesen
    If createArrays(cmtmat, startDate, endDate) = False Then
        Exit Sub
    End If
    If myCount < 10 Then
        MsgBox ("Zuwenige CMT/SMR-Kombinationen zum Kalibrieren vorhanden")  290
        Exit Sub
    End If
    ' Parameter schätzen
    schätzeRoh
    schätzeBetas
    berechneParameter
    ' Fertig
    myCalibrated = True
End Sub
                                                                         300
'************************************************************************
' Fertig kalibriert ?
'************************************************************************
Public Function isCalibrated() As Boolean
    isCalibrated = myCalibrated
End Function

'************************************************************************
' Datumsbereich der Kalibrierung abfragen
'************************************************************************  310
Public Function getCalibrationStartDate() As Date
    getCalibrationStartDate = myStartDate
End Function

Public Function getCalibrationEndDate() As Date
    getCalibrationEndDate = myEndDate
```

```
End Function

'*************************************************************************
' CMT-Laufzeiten der Kalibrierung abfragen                                  320
'*************************************************************************
Public Function getCalibrationCMTMaturities() As Variant
    getCalibrationCMTMaturities = myCalibrationCMTMaturities
End Function

'*************************************************************************
' Parameter abfragen
'*************************************************************************
Public Function getBetas() As Variant
    getBetas = myBetas                                                     330
End Function

'*************************************************************************
' GammaStern abfragen
'*************************************************************************
Public Function getGammaStern() As Double
    getGammaStern = myGammaStern
End Function

'*************************************************************************  340
' Gamma abfragen
'*************************************************************************
Public Function getGamma() As Double
    getGamma = myGamma
End Function

'*************************************************************************
' A abfragen
'*************************************************************************
Public Function getA() As Double                                           350
    getA = myA
End Function

'*************************************************************************
' B abfragen
'*************************************************************************
Public Function getB() As Double
    getB = myB
End Function
                                                                           360
'*************************************************************************
' Lambda abfragen
'*************************************************************************
Public Function getLambda() As Double
    getLambda = (myGamma – myGammaStern) * myA / myB
End Function

'*************************************************************************
' Abfragen: Aktuelles Datum = Datum der verwendeten Zinsstruktur
```

```
'*************************************************************************
Public Function getActualDate() As Date
    If actualDate = Empty Then
        actualDate = Application.min(myRates.getLastDate, _
                                     myCMT.getLastDate, mySMR.getLastDate)
    End If
    getActualDate = actualDate
End Function

'*************************************************************************
' Setzen: Aktuelles Datum = Datum der verwendeten Zinsstruktur
'*************************************************************************
Public Function setActualDate(datum As Date) As Boolean
    setActualDate = False
    On Error GoTo fehler
    If Not (myRates.getHistoryRateFor(datum, 5) = 0 Or _
            myCMT.getHistoryRateFor(datum, 5) = 0 Or _
            mySMR.getHistoryRate(datum) = 0) Then
        actualDate = datum
        setActualDate = True
    End If
fehler:
    On Error GoTo 0
End Function

'*************************************************************************
' Aktuelles UT berechnen
'*************************************************************************
Private Sub berechneUT()
    Dim c As Integer
    Dim u As Double
    u = 0
    For c = 1 To myCMTCount
        u = u + myBetas(c) * myCMT.getHistoryRateFor(actualDate, _
                                                     myCMTMaturity(c))
    Next c
    myUTAktuell = mySMR.getHistoryRate(actualDate) - u
End Sub

'*************************************************************************
' Arrays füllen
'*************************************************************************
Private Function createArrays(cmtmat As Variant, start As Date, _
                              ende As Date) As Boolean
    Dim smrDates As Variant, smr As Variant
    Dim mat As Integer, d As Integer, cmtDates As Variant, cmt As Variant
    ' Datumsbereich
    cmtDates = myCMT.getAvailableDates
    ' Laufzeiten
    ReDim myCMTMaturity(UBound(cmtmat) - LBound(cmtmat) + 1)
    ' Werte-Array
    myCMTCount = UBound(cmtmat) - LBound(cmtmat) + 1
    ReDim cmt(UBound(cmtDates) - LBound(cmtDates) + 1, myCMTCount)
```

```
ReDim smr(UBound(cmtDates) - LBound(cmtDates) + 1)
' Werte kopieren
Dim wert As Double
Dim smrcounter As Integer, cmtcounter As Integer
d = UBound(cmtDates)
smrcounter = 1
cmtcounter = 1
While d >= LBound(cmtDates)                                          430
    ' Nur Daten innerhalb des gegebenen Datumsbereichs verwenden
    If cmtDates(d) >= start And cmtDates(d) <= ende Then
        ' SMR-Wert kopieren
        wert = mySMR.getHistoryRate(cmtDates(d))
        If wert > 0 Then
            smr(smrcounter) = wert
            smrcounter = smrcounter + 1
            ' CMT-Werte kopieren
            For mat = LBound(cmtmat) To UBound(cmtmat)
                cmt(cmtcounter, mat - LBound(cmtmat) + 1) = _        440
                        myCMT.getHistoryRateFor(cmtDates(d), cmtmat(mat))
            Next mat
            cmtcounter = cmtcounter + 1
        End If
    End If
    d = d - 1
Wend
myCount = cmtcounter - 1
If myCount < 1 Then
    MsgBox ("Keine CMT/SMR-Kombinationen vorhanden")                 450
    createArrays = False
    Exit Function
End If
' SMR zuweisen
ReDim Preserve smr(myCount)
mySMRArray = smr
' CMT muß kopiert werden
ReDim myCMTArray(myCount, myCMTCount)
For d = 1 To myCount
    For mat = 1 To myCMTCount                                        460
        myCMTArray(d, mat) = cmt(d, mat)
    Next mat
Next d
' CMT-Laufzeiten kopieren
For mat = LBound(cmtmat) To UBound(cmtmat)
    myCMTMaturity(mat - LBound(cmtmat) + 1) = cmtmat(mat)
Next mat
If myCount > 1 Then ' myCount==1 falls Bewertung
    ' CMT-Delta-Werte
    ReDim myDeltaCMT(UBound(myCMTArray) - 1, myCMTCount)             470
    Dim c As Integer, y As Integer
    For c = 1 To myCMTCount
        For y = 1 To UBound(myCMTArray) - 1
            myDeltaCMT(y, c) = myCMTArray(y + 1, c) - myCMTArray(y, c)
        Next y
```

```
        Next c
        ' SMR-Delta-Werte (zweite Dimension notwendig für Matrixoperationen)
        ReDim myDeltaSMR(UBound(mySMRArray) - 1, 1)
        For d = 1 To UBound(mySMRArray) - 1
            myDeltaSMR(d, 1) = mySMRArray(d + 1) - mySMRArray(d)
        Next d
    End If
    ' Fertig
    createArrays = True
End Function

'*************************************************************************
' Hilfsfunktion um sicherzustellen, dass Array zweidimensional ist
'*************************************************************************
Private Function checkSecondDimension(B As Variant) As Variant
    Dim t As Integer
    ' Falls zweite Dimension fehlt -> hinzufügen
    On Error GoTo addSecond
    t = UBound(B, 2)
    On Error GoTo 0
    GoTo secondok
addSecond:
    Dim neuesb As Variant
    ReDim neuesb(LBound(B) To UBound(B), 1) As Double
    For t = LBound(B) To UBound(B)
        neuesb(t, 1) = B(t)
    Next t
    B = neuesb
secondok:
    checkSecondDimension = B
End Function

'*************************************************************************
' Roh schätzen
'*************************************************************************
Private Sub schätzeRoh()
    Dim xt As Variant, B As Variant
    xt = Application.Transpose(myDeltaCMT)
    B = Application.MMult(xt, myDeltaCMT)
    B = Application.MInverse(B)
    B = Application.MMult(B, xt)
    B = Application.MMult(B, myDeltaSMR)
    ' Zweite Dimension prüfen
    B = checkSecondDimension(B)
    ' rho schätzen
    Dim t As Integer
    Dim summeo, summeu As Double
    ' Obere Summe berechnen
    summeo = 0
    For t = 2 To myCount - 1
        summeo = summeo + eDach(B, t) * eDach(B, t - 1)
    Next t
    ' Untere Summe berechnen
```

```
    summeu = 0
    For t = 2 To myCount – 1
        summeu = summeu + (eDach(B, t – 1) ^ 2)
    Next t
    myRho = summeo / summeu
End Sub

Private Function eDach(B, t As Integer) As Double
    Dim i As Integer
    Dim summe As Double
    summe = 0
    For i = LBound(B) To UBound(B)
        summe = summe + B(i, 1) * myDeltaCMT(t, i)
    Next
    eDach = myDeltaSMR(t, 1) – summe
End Function

'************************************************************************
' Betas schätzen
'************************************************************************
Private Sub schätzeBetas()
    Dim xs, ys As Variant
    ReDim ys(myCount – 1, 1) As Double
    ReDim xs(myCount – 1, myCMTCount) As Double
    Dim summe As Double
    Dim z, B As Integer
    ' Array füllen: 1 Zeile
    ys(1, 1) = Math.Sqr(1 – myRho ^ 2) * myDeltaSMR(1, 1)
    For B = 1 To myCMTCount
        xs(1, B) = Math.Sqr(1 – myRho ^ 2) * myDeltaCMT(1, B)
    Next
    ' Array füllen: Weitere Zeilen
    For z = 2 To myCount – 1
        ys(z, 1) = myDeltaSMR(z, 1) – myRho * myDeltaSMR(z – 1, 1)
        For B = 1 To myCMTCount
            xs(z, B) = myDeltaCMT(z, B) – myRho * myDeltaCMT(z – 1, B)
        Next B
    Next z
    ' Betas schätzen
    Dim xst, bDach As Variant
    xst = Application.Transpose(xs)
    bDach = Application.MMult(xst, xs)
    bDach = Application.MInverse(bDach)
    bDach = Application.MMult(bDach, xst)
    bDach = Application.MMult(bDach, ys)
    ' Zweite Dimension prüfen
    bDach = checkSecondDimension(bDach)
    ' Betas kopieren
    myBetas = bDach
    ReDim myBetas(1 To myCMTCount) As Double
    For z = 1 To myCMTCount
        myBetas(z) = bDach(z, 1)
    Next z
```

```
    ' Varianz
    Dim h, ht As Variant
    h = Application.MMult(xs, bDach)
    For z = LBound(h, 1) To UBound(h, 1)
        h(z, 1) = ys(z, 1) - h(z, 1)
    Next z
    ht = Application.Transpose(h)
    h = Application.MMult(ht, h)
    myVariance = h(1) / (myCount - myCMTCount)                          590
    ' Aktuelles ut berechnen
    berechneUT
End Sub

'*************************************************************************
' Parameter für Regression berechnen
'*************************************************************************
Private Sub berechneParameter()
    myA = -myRho
    myB = Sqr((myVariance * 2 * myA) / (1 - Exp(-2 * myA * myDeltat)))  600
    ' Gamma berechnen
    ' Durchschnittliche Differenz zwischen SMR und Linearkombination
    Dim z As Integer, c As Integer, lk As Double
    myGamma = 0
    For z = 1 To myCount
        lk = 0
        For c = 1 To myCMTCount
            lk = lk + myBetas(c) * myCMTArray(z, c)
        Next c
        myGamma = myGamma + mySMRArray(z) - lk                          610
    Next z
    myGamma = myGamma / myCount
    myGammaStern = 0.013
    Dim iter, f, i, k1, k2 As Integer, min, gs1, gs2 As Double
    Dim fehler(5) As Double
    iter = 0
    gs1 = -0.2
    gs2 = 0.2
    min = 1
    While iter < 50 And (gs2 - gs1) > 0.0001 'min > 0.000001            620
        ' Fehler berechnen
        For f = 1 To 5
            myGammaStern = gs1 + (gs2 - gs1) * (f - 1) / 4
            fehler(f) = 0
            For i = 1 To myCalCount

                Dim dummy As Double
                dummy = getValue(0, myCalEndtime(i), myFSRVolatility, _
                                 myCalFreq(i), myCalMargin(i), myCalKupon(i))
                fehler(f) = fehler(f) + (myCalValue(i) - _              630
                    dummy) ^ 2
            Next i
        Next f
        ' Zwei kleinste Fehler suchen
```

```
        min = 10000
        For f = 1 To 5
            If fehler(f) < min Then
                min = fehler(f)
                k1 = f
            End If
        Next f
        myGammaStern = gs1 + (gs2 – gs1) * (k1 – 1) / 4
        min = 10000
        For f = 1 To 5
            If f <> k1 And fehler(f) < min Then
                min = fehler(f)
                k2 = f
            End If
        Next f
        If k1 > 1 Then
            k1 = k1 – 1
        End If
        If k2 < 5 Then
            k2 = k2 + 1
        End If
        ' Neue Grenzen
        If k1 = 1 And k2 = 5 Then
            f = gs1
            gs1 = gs1 + (gs2 – gs1) * (k1 – 0.9) / 4
            gs2 = f + (gs2 – f) * (k2 – 1.1) / 4
        Else
            f = gs1
            gs1 = gs1 + (gs2 – gs1) * (k1 – 1) / 4
            gs2 = f + (gs2 – f) * (k2 – 1) / 4
        End If

        iter = iter + 1
    Wend
End Sub

'*************************************************************************
' CMT-Bewertung setzen
'*************************************************************************
Public Sub setCMT(cmt As Object)
    Set myCMT = cmt
End Sub

'*************************************************************************
' SMR setzen
'*************************************************************************
Public Sub setSMR(smr As Object)
    Set mySMR = smr
End Sub

'*************************************************************************
' Spotrates setzen
'*************************************************************************
```

```
Public Sub setSpotRates(r As Object)
    Set myRates = r
End Sub

'*********************************************************************
' Bewertung des SMR-abhängigen Fehlerterms
'*********************************************************************
Public Function getPresentValue(ByVal valuetime As Double, _
                ByVal endtime As Double, _
                ByVal freq As Double) As Double
    getPresentValue = myRates.getHistoryDiscountFactor(actualDate, _
            valuetime, endtime) * _
            (myGammaStern + (myUTAktuell - myGammaStern) * _
            Exp(-myA * (endtime - freq - valuetime)))
End Function

'*********************************************************************
' Bewertung
'*********************************************************************
Public Function getDetailValue(ByVal valuetime As Double, _
            ByVal endtime As Double, _
            ByVal vol As Double, ByVal freq As Double, _
            ByVal margin As Double, ByVal kupon As Double) As Variant
    Dim i, c, anz As Integer
    Dim start, cmt, smr, time As Double
    Dim cmtteil As Double, smrteil As Double, marginteil As Double
    Dim stückzinsen As Double
    cmtteil = 0
    smrteil = 0
    marginteil = 0
    stückzinsen = 0
    ' Ersten Zahlungszeitpunkt ermitteln
    start = endtime
    anz = 1
    While ((start - freq) > 0)
        start = start - freq
        anz = anz + 1
    Wend
    ' Zahlungen bewerten
    For i = 1 To anz
        time = start + (i - 1) * freq
        ' CMT-Teil
        If i = 1 Then
            ' Bei i=1 -> laufenden Kupon heranziehen !
            cmt = kupon * freq * myRates.getHistoryDiscountFactor( _
                                    actualDate, valuetime, time)
        Else
            ' Linearkombination
            cmt = 0
            For c = 1 To myCMTCount
                Dim dummy As Double
                dummy = myRates.getHistoryForwardSwapRate(actualDate, _
                        valuetime, myCMTMaturity(c), time - freq - valuetime)
```

```
                dummy = dummy + myCMT.getHistoryConvexityAdjustment( _
                    actualDate, valuetime, myCMTMaturity(c), time, vol, freq)
                cmt = cmt + myBetas(c) * dummy
            Next c
            cmt = cmt * myRates.getHistoryDiscountFactor( _
                        actualDate, valuetime, time) * freq
        End If
        cmtteil = cmtteil + cmt
        ' SMR-Teil
        If i = 1 Then                                                    750
            smr = 0
        Else
            smr = getPresentValue(valuetime, time, freq) * freq
        End If
        smrteil = smrteil + smr
        ' Margin
        If i > 1 Then
            marginteil = marginteil + margin * _
                    myRates.getHistoryDiscountFactor(actualDate, _
                                                valuetime, time)         760
        End If
    Next i
    ' Stückzinsen abziehen
    stückzinsen = -(freq - start) * kupon
    ' Rückzahlung dazu
    cmtteil = cmtteil + myRates.getHistoryDiscountFactor( _
                        actualDate, valuetime, endtime)
    ' Rückgabe
    Dim ret As Variant
    ReDim ret(4)                                                         770
    ret(1) = cmtteil
    ret(2) = smrteil
    ret(3) = stückzinsen
    ret(4) = marginteil
    getDetailValue = ret
End Function

Public Function getValue(ByVal valuetime As Double, _
            ByVal endtime As Double, _
            ByVal vol As Double, ByVal freq As Double, _                 780
            ByVal margin As Double, ByVal kupon As Double) As Double
    Dim v As Variant
    v = getDetailValue(valuetime, endtime, vol, freq, margin, kupon)
    Dim i As Integer, ret As Double
    ret = 0
    For i = LBound(v) To UBound(v)
        ret = ret + v(i)
    Next i
    getValue = ret
End Function                                                             790
```

Literaturverzeichnis

[1] R. Brotherton-Ratcliffe, B. Iben: Yield Curve Applications of Swap Products, in: R. Schwartz, C. Smith: Advanced Strategies in Financial Risk Management, New York Institute of Finance, New York, 1993.

[2] Derivatives Week: CMT-Based Derivatives, Institutional Investors, 1994.

[3] F.J. Fabozzi: Floating Rate Instruments: Characteristics, Valuation, and Portfolio Strategy, Prentice Hall, Englewood Cliffs, 1986.

[4] M. Frühwirth: Suggestions for the Pricing of Secondary Market Yield based Floating Rate Notes subject to default Risk, Working Paper, Wirtschaftsuniversität Wien, Wien, 1998.

[5] W. Griffiths, R. Hill, G. Judge: Learning and Practicing Econometrics, John Wiley & Sons Inc., 1993.

[6] J.C. Hull: Options, Futures, and other Derivatives, Prentice Hall, Upper Saddle River, 1997.

[7] F. Jamshidian: LIBOR and Swap Market Models and Measures, Finance and Stochastics 1, p. 293-330, 1997.

[8] P. E. Kloeden, E. Platen und H. Schurz: Numerical Solutions of SDE Through Computer Experiments, Springer, New York, 1994.

[9] A. Li und V.R. Raghavan: LIBOR In-Arrears Swaps, The Journal of Derivatives 3, p. 44-48, 1996.

[10] K.R. Miltersen, K. Sandmann, D.Sondermann: Closed Form Solutions for Term Structure Derivatices with Log-Normal Interest Rates, The Journal of Finance 52, p. 409-430, March 1997.

[11] M. Musiela, M. Rutkowski: Continuos-Time Term Structure Models: Forward Measure Approach, Finance and Stochastics 1, p.261-291, 1997.

[12] Österreichische Kontrollbank: Dokumentation zur Berechnung der SMR, Wien, 1997.

[13] S. Pichler: Bewertung von Zahlungsströmen mit variabler Verzinsung, Deutscher Universitäts Verlag, Gabler Edition Wissenschaft, 1999.

[14] K. Ramaswamy, S. M. Sundaresan: The Valuation of Floating-Rate Instruments: Theory and Evidence, Journal of Financial Economics 17, p. 251-272, 1986.