

# Ein Vorschlag zur Formelfüllung

Hans Peter Wolf

email: `pwolf@wiwi.uni-bielefeld.de`  
Bereich für Statistik und Datenverarbeitung  
Fakultät für Wirtschaftswissenschaften  
Universität Bielefeld

06.06.2000, R/formfill/ff2.rev, mod: 27.11.02

## Überblick

<b>1</b>	<b>Problemstellung</b>	<b>2</b>
<b>2</b>	<b>Ein Beispiel, was gehen sollte</b>	<b>2</b>
<b>3</b>	<b>Ein konzeptioneller Vorschlag</b>	<b>3</b>
<b>4</b>	<b>Was der Anwender zu tun hat</b>	<b>4</b>
<b>5</b>	<b>Die Lösung für einzelne Füllabschnitte</b> – die innere Funktion	<b>6</b>
<b>6</b>	<b>Eine allgemeine Lösung für viele Füllabschnitte</b> – die äußere Funktion	<b>11</b>
<b>7</b>	<b>Anhang A: Die beiden Hilfsdateien</b>	<b>13</b>
<b>8</b>	<b>Anhang B: Ein Ausschnitt aus dem Quellfile</b>	<b>14</b>

# 1 Problemstellung

Statistische Formeln erzeugen bei Lernenden bisweilen Unverständnis. Besonders auch deshalb, weil sie – die Leidenden – bei eigenen Auswertungen zu anderen Ergebnissen kommen. Deshalb kann eine Vorführung der Auswertung hilfreich sein. Zur Erstellung einer solchen Demonstration ist ein Tool wünschenswert, das für spezifische Datenkonstellationen Schritte auf dem Weg zum Ziel in Form von Zwischenergebnissen zeigt. Der Mangel eines solchen Werkzeugs hat meinen Freund *Dietrich Trenkler* so sehr gestört, daß er vorgeschlug, ein solches zu entwickeln. Mit seinen langjährigen APL-Erfahrungen hat er daraufhin einen Entwurf erstellt, der leider nicht mein Wohlwollen fand. In diesem Papier wird eine Alternative beschrieben, wie die Füllung von Formeln aus meiner Sicht viel geeigneter beschrieben werden kann.

Mit dem hier beschriebenen Werkzeug lassen sich außerdem vorher definierte Ergebnis-Formulare füllen, die dem spröden R-Output ein angemessenes Outfit verleihen.

## 2 Ein Beispiel, was gehen sollte

Betrachten wir zunächst ein sehr kleines Beispiel. Es soll zu einem vorgegebenen  $\alpha$  der Prozentpunkt  $z_{1-\alpha/2}$  der Standardnormalverteilung berechnet werden. Je nach  $\alpha$  ergibt sich natürlich ein anderer Wert. Nehmen wir zum Beispiel an, es sei  $\alpha = 0.01$ . Wie lautet dann die gesuchte Wahrscheinlichkeit? Hier ist die Antwort:

$$z = z_{1-\alpha/2} = z_{0.995} = 2.576$$

Man möchte natürlich nur an einer einzigen Stelle den Wert für  $\alpha$  festlegen, dann soll simulabim auf Knopfdruck der Text mit den richtigen Zahlen *gefüllt* vorliegen.

Dieses Beispiel ist vielleicht etwas zu niedlich, um den gewünschten Effekt zu verdeutlichen. Deshalb betrachten wir einer Anregung von DT folgend ein Konfidenzintervall für den Parameter  $p$  einer Bernoulli-Verteilung  $\mathcal{B}(1, p)$ , das sich gemäß

$$\text{KI} = \frac{1}{n + z^2} \left( n\hat{p} + \frac{1}{2} \pm z\sqrt{n\hat{p}(1 - \hat{p}) + \frac{z^2}{4}} \right)$$

berechnen läßt.

Nach einem Beispiel von Schlittgen errechnet sich für  $n = 69$  und  $\hat{p} = 0.768$  ein 0.99%-Konfidenzintervall wie folgt:

$$\begin{aligned}
 KI_{0.99}(p) &= \frac{1}{69+2.576^2} \left( 69 \times 0.768 + \frac{2.576^2}{2} \pm 2.576 \sqrt{69 \times 0.768(1 - 0.768) + \frac{2.576^2}{4}} \right) \\
 &= \frac{1}{75.63} \left( 52.99 + 3.317 \pm 2.576 \sqrt{12.29 + 1.659} \right) \\
 &= 0.01322 \left( 56.31 \pm 2.576 \sqrt{13.95} \right) \\
 &= 0.01322 \left( 56.31 \pm 2.576 \times 3.735 \right) \\
 &= 0.01322 \left( 56.31 \pm 9.622 \right) \\
 &= [0.6173, 0.8717]
 \end{aligned}$$

Diese kleinen Umrechnungsschritte kann jeder schnell nachrechnen. Mit Hilfe des gedachten Werkzeugs ließen sich für beliebige andere Datenkonstellationen sofort entsprechende Füllergebnisse erstellen. Soweit die Wunschwelt.

### 3 Ein konzeptioneller Vorschlag

Der beschriebene Wunsch soll noch vor Weihnachten in Erfüllung gehen. Deshalb folgt sogleich ein Umsetzungsvorschlag.

Nach diesem Vorschlag läßt sich in einer Rohdatei (dem  $\text{\LaTeX}$ -Dokument) eine neu entwickelte Umgebung verwenden, in der Texte mit auszuwertenden R-Anweisungen gespickt werden können. Eine R-Funktion liest diese Umgebungen, wertet die gefundenen Anweisungen aus und schreibt das Ergebnis in eine Hilfsdatei. Der  $\text{\LaTeX}$ -Formatierer bettet die erstellte Hilfsdatei an Stelle der Umgebung in den Gesamttext ein.

## 4 Was der Anwender zu tun hat

Was der Anwender zu tun hat, wird zunächst allgemein und dann noch einmal anhand der vorliegenden Datei `ff2.rev` beschrieben.

1. Zunächst ist die neue Umgebung mit dem Namen `formfill` zu definieren:

```
\newenvironment{formfill}[2]{\input{#1}}{}
```

2. Mit dieser neuen Umgebung läßt sich ein Abschnitt beschreiben, deren Auswertungsergebnis in einer Hilfsdatei abgelegt wird. Dazu muß der Name der Hilfs- $\text{\TeX}$ -Datei der Umgebung mitgegeben werden. Wie an der Definition der Umgebung zu erkennen ist, wird der Inhalt der Hilfsdatei bei der Formatierung (siehe: `\input`) integriert.

Als zweiter Parameter ist dieser Umgebung die Rohform des zu füllenden Abschnittes mitzugeben. Dieser Parameter wird bei der Formatierung nicht benutzt, so daß er durchaus Sonderzeichen enthalten kann. Der Body der Umgebung, sonst üblicherweise das Fleisch des Gerüsts, bleibt leer.

3. Die Rohform eines zu füllenden Abschnitts kann drei verschiedene Qualitäten enthalten:
  - Klartext, der später eins zu eins im formatierten Papier enthalten ist,
  - R-Zuweisungen, die nicht ausgedruckt werden sollen, aber für die zu ersetzenden Anweisungen Hilfsvariablen bereitstellen,
  - und R-Anweisungen, die später durch ihre Werte ersetzt sein sollen.

Blöcke von R-Zuweisungen beginnen mit Zeilen, die die Zeichenfolge `#:` enthalten und werden durch Zeilen beendet, in denen das Zeichenpaar `:#` zu finden ist. Durch ihre Werte zu ersetzende R-Anweisungen werden durch das Umschaltzeichen `#` von normalen Textzeichen abgegrenzt, wie bei  $\text{\TeX}$  Formeln im Text durch Klammerung in  $\text{\$}$ -Zeichen getrennt werden.

4. Zur Umsetzung der im  $\text{\LaTeX}$ -Dokument beschriebenen Ersetzungswünsche wird in diesem Papier die R-Funktion `ff` beschrieben:

```
1 <* 1>≡  
  <Definition der Funktion ff 21>
```

Diese Funktion muß in den R-Interpreter geladen und mit dem Quelldateinamen als Argument gestartet werden. Für jede `formfill`-Umgebung erstellt `ff` die vereinbarte(n) Hilfsdatei(en) mit den aufbereiteten Texten.

5. Nun kann die Quelldatei ge $\text{\LaTeX}$ t werden.
6. Probieren geht vor studieren.

Um eine noch bessere Vorstellung von den zu erledigenden Tätigkeiten zu bekommen, wollen wir eine Auflistung von konkreten Handlungsanweisungen anfügen.

1. Die Definition der Umgebung muß irgendwo vorn in der Quelldatei stehen. Hier ist der Code:

```
\newenvironment{formfill}[2]{\input{#1}}{}
```

2. Einen zu füllenden Abschnitt, dessen Bearbeitungsergebnis in der Datei `ffbsp1.tex` abgelegt werden soll, leiten wir ein mit:

```
\begin{formfill}{ffbsp1.tex}{
```

3. Im Text wollen wir beispielsweise den Wert der Formel  $z_{1-\alpha/2}$  eingesetzt bekommen. Dazu lassen wir zum Beispiel die Variable `z` auswerten, die im nächsten Schritt erklärt wird:

```
... Wahrscheinlichkeit? Hier ist die Antwort:  
\[  
z=z_{1-\alpha/2}=z_{\#1-\alpha/2\#}=#z#  
\]
```

4. Die Hilfsvariablen `z` und `alpha` müssen wir definieren:

```
#:  
  alpha<-0.01  
  z    <-qnorm(1-alpha/2)  
:#
```

5. Am Ende des zu füllenden Absatzes müssen wir diesen beenden und schreiben:

```
}\end{formfill}
```

6. Wir lassen `ff(ff2.rev)` von R durch einen Batchaufruf:

```
echo "source(\"ff2.sch\");ff(\"ff2.rev\")" | R BATCH
```

oder per Hand veranlaßt auswerten. Ist `ff` in der Umgebung bereits erreichbar, muß sie natürlich nicht geladen werden.

7. `latex ff2`

8. Weiterverarbeitung wie gewohnt.

Vielleicht klingt das Prozedere nach dem ersten Hinhören kompliziert, jedoch sind nur drei Dinge zu tun: Schreibe den Quellfile, starte `ff` und `TEXe` wie üblich.

## 5 Die Lösung für einzelne Füllabschnitte – die innere Funktion

Wir wollen zur Umsetzung der Wünsche bzw. von `ff` zunächst ein leichteres Problem lösen: Gegeben sei eine Variable, auf der als Elemente die Zeilen eines auszuwertenden und zu füllenden Rohabschnittes stehen. Dann soll die Funktion `ff.local` die beschriebenen Definitionen umsetzen, die zu ersetzenden Ausdrücke auswerten und ersetzen und das Ergebnis in den angegebenen File schreiben.

Die Funktion `ff.local` ersetzt, wie oben schon beschrieben, in einem Text jede in zwei *Fluchtsymbole* oder besser *Trennzeichen* eingeschlossene Zeichenkette durch ihren Wert, wie R ihn berechnet. Das Default-Fluchtzeichen ist der Lattenzaun.

Beispiel: `#2+2#` wird überführt in 4. Um auch mit verschiedenen Variablen operieren zu können, werden vor der Evaluation der zu ersetzenden Ausdrücke alle Terme ausgewertet, die mit durch Fluchtsymbol Doppelpunkt beginnen und mit Doppelpunkt Fluchsymbol enden. So kann mit

```
#:  
x <- 1:10  
:#
```

die Variable `x` belegt und in `#sum(x)#` verwendet werden. Für weitere Verwendungen läßt sich zum Beispiel die Variable `y` als Quadrat von `x` einführen:

```
#:  
y <- x*x  
:#
```

Als Nebenbedingung müssen zu ersetzende Ausdrücke innerhalb einer Zeile stehen. Erlaubt sind auch mehrere Ausdrücke pro Zeile. Mehrere Definitionen dürfen dagegen nicht in einer Zeile auftauchen.

Der Funktion `ff.local` werden als erstes Argument die Textzeilen `tz` übergeben. Als zweiter Parameter ist der Name der Outputdatei zu übergeben. Weiter folgen die Genauigkeit des Ergebnisses und das ausgewählte Fluchtsymbol. Die genaue Syntax kann der Kopfzeile entnommen werden.

```
2 <initialisiere lokale Funktion 2>≡  
  ff.local<-  
  function(tz,out.file="tmpout.tex",path="",digits=4,escape.symbol="#"){  
    <Rumpf von ff.local 3>  
  }
```

Innerhalb der Funktion werden zunächst die Definitionen und dann die Ersetzungswünsche umgesetzt. Zum Schluß wird das Ergebnis in der Hilfsdatei abgelegt und eine kurze Meldung ausgegeben.

```
3  <Rumpf von ff.local 3>≡
    cat("processing for ",out.file,": starts  \n")
    <suche Definitionen und evaluiere diese 4>
    cat("definitions for ",out.file,": evaluated\n")
    <suche Ersetzungsaufträge und führe sie aus 10>
    cat("replacements for ",out.file,": finished\n")
    if(0<nchar(path)) out.file<-paste(path,"/",out.file,sep="")
    cat(paste(tz,"\n"),file=out.file,sep="")
    cat("output file      ",out.file,": generated\n")
    return(tz)
```

Wenden wir uns zunächst der Extraktion von auszuwertenden Definitionen zu. Im wesentlichen muß nach und nach die jeweils nächste Definition gesucht und ausgewertet werden. Ist dieses geschehen, werden die Definitionen selbst nicht mehr benötigt und können entfernt werden.

```
4  <suche Definitionen und evaluiere diese 4>≡
    <initialisiere einige Dinge zur Evaluation von Definitionen 5>
    repeat{
        <stoppe, wenn keine Definition mehr zu finden ist 6>
        <suche nächste Definition zur Aktivierung 7>
        <evaluiere die gefundenen Ausdrücke 8>
    }
    <entferne Definitionen aus dem Text 9>
```

Auf `tz` stehen Zeilenweise die Zeilen. Durch eine schnelle Inspektion lassen sich alle Anfänge und Enden von Definitionsblöcken finden. Für die spätere Entfernung der Definitionen von `tz` werden die Grenzen auf `ff.defs.pos` festgehalten.

```
5  <initialisiere einige Dinge zur Evaluation von Definitionen 5>≡
    ff.begin<-grep(paste(escape.symbol,":",sep=""),tz)
    ff.end  <-grep(paste(":",escape.symbol,sep=""),tz)
    ff.defs.pos<-cbind(ff.begin,ff.end)
    cat(" definition(s) to be evaluated:\n")
    for(ff.i in 1:length(ff.begin)){
        cat(" from ",ff.begin,"to",ff.end,"\n")
        cat(paste(" ",tz[ff.begin[ff.i]:ff.end[ff.i]],"\n"))
    }
```

Wenn keine weiteren Definitionen mehr gefunden werden, kann die Definitionsauswertung abgebrochen werden.

```
6  <stoppe, wenn keine Definition mehr zu finden ist 6>≡
    if(length(ff.begin)==0 ) break
```

In diesem Abschnitt werden alle Definitionen des nächsten Definitionsblocks extrahiert. Zuerst wird der Block auf `ff.defs` abgelegt. `ff.begin` und `ff.end`, die die Blockgrenzen halten, werden verkürzt. Die Zeichenketten `#:` bzw. `:#` stören die Evaluation und werden entfernt, so daß am Schluß auf `ff.defs` die zu aktivierenden Definitionen stehen.

```
7 (suche nächste Definition zur Aktivierung 7)≡
  ff.defs<-tz[ff.begin[1]:ff.end[1]]
  ff.begin<-ff.begin[-1]; ff.end<-ff.end[-1]

  ff.chars<-substring(ff.defs[1],1:nchar(ff.defs[1]),1:nchar(ff.defs[1]))
  ff.von <-seq(ff.chars)[ff.chars==escape.symbol&c(ff.chars[-1]," ")==" :"]
  ff.defs[1]<-substring(ff.defs[1],ff.von+2)

  ff.h<-1:nchar(ff.defs[ff.length<-length(ff.defs)])
  ff.chars<-substring(ff.defs[ff.length],ff.h,ff.h)
  ff.bis <-seq(ff.chars)[c(ff.chars[-1]," ")==escape.symbol&ff.chars==" :"]
  ff.defs[ff.length]<-substring(ff.defs[ff.length],1,ff.bis-1)
```

Alle Definitionen des extrahierten Blockes müssen evaluiert werden.

```
8 (evaluiere die gefundenen Ausdrücke 8)≡
  for(ff.d in ff.defs){
    cat(" definition(s) to be evaluated: ",ff.d,"\n")
    eval(parse(text=ff.d))
    cat(" evaluation(s) done\n")
  }
```

Von `tz` werden die Zeilen mit den Definitionen entfernt.

```
9 (entferne Definitionen aus dem Text 9)≡
  for(ff.h in 1:length(ff.defs.pos[,1]))
    tz<-tz[-(ff.defs.pos[ff.h,1]:ff.defs.pos[ff.h,2])]
```

Als zweites sind die Ersetzungsaufträge umzusetzen. Dazu ist die nächste Zeile mit einem Ersetzungsauftrag zu suchen. Der nächste Auftrag ist zu evaluieren, und das Ergebnis ist zurückzuschreiben.

```
10 <suche Ersetzungsaufträge und führe sie aus 10>≡
    <initialisiere einige Dinge für die Ersetzungen 11>
    repeat{
        <stoppe, wenn keine Ersetzung mehr zu finden ist 12>
        <zerlege nächste potentielle Zeile 13>
        ff.found<-T
        <suche nach Ersetzungskandidaten 14>
        if(ff.found){
            <extrahiere Ersetzungen 15>
            <evaluiere gefundenen Ausdruck 16>
            <plaziere die ermittelten Werte in den Text 17>
        }
        <beende Abarbeitung der betrachteten Zeile 20>
    }
}
```

Als Vorbereitung lassen sich leicht alle Zeilen mit Fluchtsymbolen finden, die potentielle Kandidaten für Ersetzungen darstellen.

```
11 <initialisiere einige Dinge für die Ersetzungen 11>≡
    ff.begin.end<-grep(escape.symbol,tz)
    cat(" lines with replacements: \n")
    cat(paste(" ",tz[ff.begin.end],"\n"))
```

Wenn keine Ersetzungen (mehr) gefunden werden, kann die Ersetzungsarbeit beendet werden.

```
12 <stoppe, wenn keine Ersetzung mehr zu finden ist 12>≡
    if(length(ff.begin.end)==0) break
```

Die nächste potentielle Zeile wird zuerst in ihre Zeichen zerlegt.

```
13 <zerlege nächste potentielle Zeile 13>≡
    ff.repl <-tz[ff.begin.end[1]]
    ff.chars<-substring(ff.repl,1:nchar(ff.repl),1:nchar(ff.repl))
```

Auf `ff.positions` werden die Stellen der Fluchtsymbole der potentiellen Zeile abgelegt. Dann folgen einige weitere Überprüfungen, um auch sicher zu sein, daß eine Ersetzung notwendig ist. Ist dies nicht der Fall, wird `ff.found` auf F gesetzt.

```
14 <suche nach Ersetzungskandidaten 14>≡
    ff.positions<-seq(ff.chars)[ff.chars==escape.symbol]
    if(length(ff.positions)<2) ff.found<-F
    if(ff.chars[ff.positions[1]+1]==":") ff.found<-F
    if(0!=length(ff.positions)%2) ff.found<-F
```

Wenn `ff.found` wahr ist, werden die Ersetzungen der Zeile extrahiert. Dabei dürfen die Trennzeichen selbst nicht übernommen werden.

```
15 <extrahiere Ersetzungen 15>≡  
    ff.positions<-matrix(ff.positions,2)  
    ff.eval <-substring(ff.repl,ff.positions[1,]+1,ff.positions[2,]-1)
```

Da pro Zeile mehrere Ersetzungen gefordert sein können, geschieht die Abarbeitung in einer Schleife.

```
16 <evaluiere gefundenen Ausdruck 16>≡  
    ff.out<-NULL  
    for(ff.e in ff.eval){  
        cat(" expression to be evaluated / replaced:", ff.e, "\n")  
        ff.out<-c(ff.out,eval(parse(text=ff.e)))  
        cat(" evaluation done, result: ",ff.out[length(ff.out)]," \n")  
    }
```

Die berechneten Ersetzungen werden auf die Variable `tz` zurückgeschrieben. Dabei wird die vorgegebene Zahlengenauigkeit berücksichtigt. Es erschien praktisch, zur Vermeidung von Sonderfällen die aktuelle Zeile `ff.chars` vorn und hinten um jeweils einen Buchstaben zu verlängern. Diese Zugaben werden später wieder entfernt. Modifikation 27.11.02: `ff.out` auch für Zeichenketten gangbar gemacht.

```
17 <plaziere die ermittelten Werte in den Text 17>≡  
    if(is.numeric(ff.out)) ff.out<-as.character(signif(ff.out,digits))  
    ff.chars<-c("a",ff.chars[0==cumsum(ff.chars==escape.symbol)%%2],"z")  
    ff.chars<-paste(ff.chars,collapse="")
```

Die zusammengefügte Zeile wird an den Stellen der Fluchtsymbole aufgebrochen und als Zeile unter die Ersetzungen geschrieben. Die spaltenweise Betrachtung der so entstandenen Matrix liefert nach Entfernung des ersten Elementes, nach erneutem Zusammenfügen und nach Entfernung des ersten und letzten Zeichens das gewünschte Ergebnis.

```
18 <plaziere die ermittelten Werte in den Text 17>+≡  
    ff.chars<-strsplit(ff.chars,escape.symbol)[[1]]  
    ff.chars<-rbind(c(" ",ff.out),ff.chars)[-1]  
    ff.chars<-paste(ff.chars,collapse="")  
    ff.repl <-substring(ff.chars,2,nchar(ff.chars)-1)
```

Das Zurückschreiben erfordert eine letzte Anweisung.

```
19 <plaziere die ermittelten Werte in den Text 17>+≡  
    tz[ff.begin.end[1]]<-ff.repl  
    cat(" REPLACED line:",ff.repl,"\n")
```

Haben wir eine Zeile erledigt, kann sie aus der Menge der zu bearbeitenden Zeilen entfernt werden, also von `ff.begin.end`.

```
20 <beende Abarbeitung der betrachteten Zeile 20>≡  
    ff.begin.end<-ff.begin.end[-1]
```

## 6 Eine allgemeine Lösung für viele Füllabschnitte – die äußere Funktion

Nachdem im Prinzip das Problem gelöst ist, muß noch eine passende Einbettung, die Funktion `ff` selbst, entworfen werden.

In dieser ist die oben beschriebene Funktion zu verdrahten, die zu bearbeitende Datei (ggf. auch der Pfad) ist einzulesen, die Füllteile (`formfill`-Umgebungen) sind zu finden und die Füllaufträge müssen abgehandelt werden. Neben dem Namen der Quelldatei können die Zahlengenauigkeit und das Fluchtsymbol (Defaults: 4 bzw. `#`) gesetzt werden.

```
21 <Definition der Funktion ff 21>≡  
    ff<-function(in.file,path="",digits=4,escape.symbol="#"){  
        cat("processing of in.file ",in.file," : starts\n")  
        <initialisiere ggf. Pfad 22>  
        <initialisiere lokale Funktion 2>  
        <lese Datei ein 23>  
        <extrahiere Füllteile 24>  
        cat("environments extracted\n")  
        <handele Füllaufträge ab 25>  
    }
```

Wenn kein Pfad explizit angegeben ist, sollten die Hilfdaten später in dem Verzeichnis zu finden sein, aus dem der Quellfile stammt.

```
22 <initialisiere ggf. Pfad 22>≡  
    if(0==nchar(path)){  
        ff.h<-unlist(strsplit(in.file,"/"))  
        path<-paste(ff.h[-length(ff.h)],collapse="/")  
    }else{  
        in.file<-paste(path,in.file,sep="/")  
    }
```

Die Datei `in.file` wird zeilenweise als Text eingelesen.

```
23 <lese Datei ein 23>≡  
    ff.tz<-scan(in.file,"",sep="\n")
```

Füllteile beginnen mit `\begin{formfill}` und werden mit `\end{formfill}` abgeschlossen. Wenn keine solche Umgebung gefunden wird, soll der gesamte File abgearbeitet werden. Für das weitere muß der erste Umgebungsparameter mit den Hilfsdateinamen extrahiert werden. Außerdem sind Umgebungskopf und -ende zu entfernen. Es wird eine kurze Meldung über die gefundenen Dateinamen ausgegeben.

```
24 <extrahiere Füllteile 24>≡
  ff.pos<-grep("\\\\begin\\{formfill\\}",ff.tz)
  if(length(ff.pos)==0){
    ff.pos<-matrix(c(1,length(ff.tz)),1,2);ff.names<-"tmpout.tex"
  }else{
    ff.names<-sub("\\\\begin\\{formfill\\}","",ff.tz[ff.pos])
    ff.names<-gsub("\\{","",ff.names)
    ff.names<-unlist(lapply(strsplit(ff.names,"\\}"),"[" ,1))
    ff.pos<-cbind(ff.pos,grep("\\\\end\\{formfill\\}",ff.tz))
  }
  cat("new file(s):",paste(ff.names,collapse=" "),"\n")
```

Mit Hilfe der Funktion `ff.local` ist die Bearbeitung der gefundenen Füllteile kein Problem mehr.

```
25 <handele Füllaufträge ab 25>≡
  for(ff.n in seq(along=ff.names)){
    ff.local(
      ff.tz[(ff.pos[ff.n,1]+1):(ff.pos[ff.n,2]-1)],
      out.file=ff.names[ff.n],
      path=path,
      digits=digits,
      escape.symbol=escape.symbol
    )
  }
```

Damit ist die Umsetzung des Vorschlag zur Erfüllung des Formelfüllungswunsches abgeschlossen.

## 7 Anhang A: Die beiden Hilfsdateien

ffbsp1.tex:

Betrachten wir zun\u00e4chst ein sehr kleines Beispiel.

Es soll zu einem vorgegebenen  $\alpha$  der Prozentpunkt  $z_{1-\alpha/2}$  der Standardnormalverteilung berechnet werden.

Je nach  $\alpha$  ergibt sich nat\u00fcrlich ein anderer Wert. Nehmen wir zum Beispiel an, es sei  $\alpha=0.01$ . Wie lautet dann die gesuchte Wahrscheinlichkeit? Hier ist die Antwort:

```
\[
z=z_{1-\alpha/2}=z_{0.995}= 2.576
\]
```

ffbsp2.tex:

Nach einem Beispiel von Schlittgen errechnet sich f\u00fcr  $n=69$  und  $\hat{p}=0.768$  ein 0.99%-Konfidenzintervall wie folgt:

```
\[
\begin{array}{r@{=}l}
KI_{0.99}(p) & \frac{1}{69+2.576^2} \\
& \left( \begin{array}{l} 69 \times 0.768 + \frac{2.576^2}{2} \pm \\ 2.576 \sqrt{69 \times 0.768(1-0.768) + \frac{2.576^2}{4}} \end{array} \right) \\
& \frac{1}{75.63} \\
& \left( \begin{array}{l} 52.99 + \quad 3.317 \pm \\ 2.576 \sqrt{12.29 + \quad 1.659} \end{array} \right) \\
& 0.01322 \\
& \left( \begin{array}{l} 56.31 \pm \\ 2.576 \sqrt{13.95} \end{array} \right) \\
& 0.01322 \\
& \left( \begin{array}{l} 56.31 \pm \\ 2.576 \times 3.735 \end{array} \right) \\
& 0.01322 \\
& \left( \begin{array}{l} 56.31 \pm \\ 9.622 \end{array} \right) \\
& [0.6173, \\
& 0.8717] \\
\end{array}
\]
```

## 8 Anhang B: Ein Ausschnitt aus dem Quellfile

```
\section{Ein Beispiel, was gehen sollte}

\newenvironment{formfill}[2]{\input{#1}}{}

\begin{formfill}{ffbsp1.tex}{
Betrachten wir zun\u"achst ein sehr kleines Beispiel.
Es soll zu einem vorgegebenen  $\alpha$  der Prozentpunkt  $z_{1-\alpha/2}$  der
Standardnormalverteilung berechnet werden.
Je nach  $\alpha$  ergibt sich nat\u"urlich ein anderer Wert. Nehmen wir
zum Beispiel an, es sei  $\alpha = \#alpha\#$ . Wie lautet dann die gesuchte
Wahrscheinlichkeit? Hier ist die Antwort:
\[
z = z_{1-\alpha/2} = z_{\#1-alpha/2\#} = \#z\#
\]
#:
  alpha <- 0.01
  z     <- qnorm(1-alpha/2)
:#
}\end{formfill}
```

Man m\u00f6chte nat\u00fcrlich nur an einer einzigen Stelle den Wert f\u00fcr  $\alpha$  festlegen, dann soll `simsalabim` auf Knopfdruck der Text mit den richtigen Zahlen `{\em gef\u00fcllt}` vorliegen.

Dieses Beispiel ist vielleicht etwas zu niedlich, um den gew\u00fcnschten Effekt zu verdeutlichen. Deshalb betrachten wir einer Anregung von DT folgend ein Konfidenzintervall f\u00fcr den Parameter  $p$  einer Bernoulli-Verteilung  $\mathcal{B}(1,p)$ , das sich gem\u00e4\u00df

```
\[
\mbox{KI} =
  \frac{1}{n + z^2}
  \left(
    n \hat{p} + \frac{1}{2} \pm
    z \sqrt{n \hat{p} (1 - \hat{p}) + \frac{z^2}{4}}
  \right)
\]
```

berechnen l\u00e4\u00dft.

```
@
\begin{formfill}{ffbsp2.tex}{
#:
  n     <- 69
  p     <- 0.768
  conf  <- 0.99
  alpha <- (1-conf)
```

