Updating Carter-Guthrie

Ken Butler

August 1, 2016

Introduction

Carter and Guthrie (2004) proposed a method of modelling cricket matches. Their aim was to provide an alternative method of deciding interrupted matches, in the manner of Duckworth and Lewis (1998). What was interesting to me is that Carter and Guthrie (2004) estimate a *probability* of winning (which is then held fixed over interruptions), and it seemed to me that one could estimate and update the probability of winning as the game progresses, which would be a useful adjunct for spectators.

Data for the update come from `yorkrdata` (Ganesh 2016b), by the author of the R package `yorkr` (Ganesh 2016a). (I do not actually use the latter package, although it inspired this investigation.)

Gathering and arranging data

Getting a match file

We need to start by loading packages `tidyr` (Wickham (2016)) and `dplyr` (Wickham and Francois (2016)):

```
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

The data of Ganesh (2016b) are in the form of `.RData` files (saved dataframes), which were downloaded as a `.zip` file and extracted to the working directory. (For more recent matches I may need to grab `.yaml` files from `cricsheet.org` and process them using functions in `yorkr`.) One such file is `Afghanistan-Bangladesh-2015-02-18.RData`. It will be convenient to write a function for reading in files of this kind:

```
readMatch=function(fname) {
  load(fname)
  overs
}
```

The data frame for each match was originally called `overs`, so when it is `load`ed, it acquires the name it had when it was `save`d. This is inconvenient, so we abstract it into a function.

What does one of these data frames contain?

```r
fname="Afghanistan-Bangladesh-2015-02-18.RData"
d=readMatch(fname)
glimpse(d)
```

```
## Observations: 565
## Variables: 25
## $ ball            <chr> "1st.0.1", "1st.0.2", "1st.0.3", "1st.0.4", "1...
## $ team            <fctr> Bangladesh, Bangladesh, Bangladesh, Banglades...
## $ batsman         <fctr> Anamul Haque, Anamul Haque, Anamul Haque, Ana...
## $ bowler          <fctr> Hamid Hassan, Hamid Hassan, Hamid Hassan, Ham...
## $ nonStriker      <fctr> Tamim Iqbal, Tamim Iqbal, Tamim Iqbal, Tamim ...
## $ byes            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ legbyes         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0...
## $ noballs         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ wides           <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ nonBoundary     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ penalty         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ runs            <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0...
## $ extras          <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0...
## $ totalRuns       <dbl> 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0...
## $ wicketFielder   <chr> "nobody", "nobody", "nobody", "nobody", "nobod...
## $ wicketKind      <chr> "not-out", "not-out", "not-out", "not-out", "n...
## $ wicketPlayerOut <chr> "nobody", "nobody", "nobody", "nobody", "nobod...
## $ date            <date> 2015-02-18, 2015-02-18, 2015-02-18, 2015-02-1...
## $ matchType       <fctr> ODI, ODI, ODI, ODI, ODI, ODI, ODI, ODI, ODI, ...
## $ overs           <dbl> 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50...
## $ venue           <fctr> Manuka Oval, Manuka Oval, Manuka Oval, Manuka...
## $ team1           <fctr> Afghanistan, Afghanistan, Afghanistan, Afghan...
## $ team2           <fctr> Bangladesh, Bangladesh, Bangladesh, Banglades...
## $ winner          <fctr> Bangladesh, Bangladesh, Bangladesh, Banglades...
## $ result          <chr> "NA", "NA", "NA", "NA", "NA", "NA", "NA", "NA"...
```

The data frame contains one row for each ball. The columns include a coded representation of innings (team batting first or second), over and ball within over, the name of the batsman and bowler, the number of runs from that ball (regular `runs` and the various types of extras), the kind of wicket from that ball (if any), and at the end, some information about the match: the date, the type of match, the number of overs per side, the venue, the names of the teams, the winner and the result. These last are the same thing repeated on every line.

It will be convenient to have a function that returns some information about the match from its filename:

```r
matchInfo=function(fname) {
  d=readMatch(fname)
  data.frame(fname=fname,date=d$date[1],type=d$matchType[1])
}
```

and to test it:

```r
matchInfo(fname)
```

```
##                                    fname       date type
## 1 Afghanistan-Bangladesh-2015-02-18.RData 2015-02-18  ODI
```

Data frame of what we want

We don't need all the information in the match data frame, and we do need to re-process some of what there is. Specifically:

- grab the ball code and the scoring info
- separate out the ball code into the inns, over and ball within that innings
- create new variables, making sure that numeric things are numeric and that we know whether each ball contains a wicket or is a extra, defined here as a no ball or a wide that will result in another ball. (I need to be more careful about counting these, since the number of balls left as calculated later will be wrong.) Other extras, such as leg byes, are treated as regular runs.
- the total number of balls so far in the innings (which is wrong in the presence of extras as above)
- treat each innings separately. The Carter-Guthrie analysis uses only the first innings.
- grab only the variables we want.

```
processMatch=function(fname) {
  readMatch(fname) %>% select(ball,noballs,wides,totalRuns,wicketKind) %>%
    separate(ball,into=c("inns","over","ball")) %>%
    group_by(inns) %>%
    mutate(over=as.numeric(over),
           ball=as.numeric(ball),
           isWkt=(wicketKind!="not-out"),
           wktDown=cumsum(isWkt),
           isExtra=(noballs+wides>0),
           totalBalls=6*over+ball) %>%
    select(inns,totalBalls,isExtra,isWkt,wktDown,totalRuns)
}
```

Testing:

```
d=processMatch(fname)
d
```

```
## Source: local data frame [565 x 6]
## Groups: inns [2]
##
##     inns totalBalls isExtra isWkt wktDown totalRuns
##    <chr>      <dbl>   <lgl> <lgl>   <int>     <dbl>
## 1    1st          1    TRUE FALSE       0         1
## 2    1st          2   FALSE FALSE       0         0
## 3    1st          3   FALSE FALSE       0         0
## 4    1st          4   FALSE FALSE       0         0
## 5    1st          5   FALSE FALSE       0         1
## 6    1st          6   FALSE FALSE       0         0
## 7    1st          7   FALSE FALSE       0         0
## 8    1st          7   FALSE FALSE       0         0
## 9    1st          8   FALSE FALSE       0         1
## 10   1st          9   FALSE FALSE       0         0
## # ... with 555 more rows
```

It is also useful to summarize a match, as done by the function below

```
summarizeMatch=function(fname) {
  processMatch(fname) %>% summarize(
                                 runTotal=max(cumsum(totalRuns)),
                                 wktTotal=max(wktDown),
                                 ballTotal=max(totalBalls)
```

```
                                   )
}
```

and to test:

```
summarizeMatch(fname)
```

```
## # A tibble: 2 x 4
##    inns runTotal wktTotal ballTotal
##   <chr>    <dbl>    <int>     <dbl>
## 1   1st      267       10       300
## 2   2nd      162       10       257
```

The team batting first scored 267 and were all out on the last ball of their 50 overs (300 balls); the team batting second were all out for 162 in the 43rd over, and thus lost by 105 runs.

It will also be necessary to determine whether the innings of the team batting first was "complete" (that is, either the team was all out or batted out the full 50 overs). This could fail to happen because the match is interrupted, and for the estimation, we want to ignore these matches:

```
firstComplete=function(fname) {
  summarizeMatch(fname) %>%
    mutate(complete=ifelse(wktTotal[1]==10,T,ifelse(ballTotal[1]>=300,T,F))) %>%
    mutate(fname=fname) %>%
    select(c(fname,complete)) %>%
    slice(1)
}
firstComplete(fname)
```

```
## # A tibble: 1 x 2
##                                       fname complete
##                                       <chr>    <lgl>
## 1 Afghanistan-Bangladesh-2015-02-18.RData     TRUE
```

Getting all the files

The files we want to consider are `.RData` files *with a number before the dot* (the tail end of the match date). There are some other `.RData` files that were in the `.zip`, but we don't want to consider those:

```
files=list.files(pattern = "[0-9].RData$")
head(files)
```

```
## [1] "Afghanistan-Australia-2012-08-25.RData"
## [2] "Afghanistan-Bangladesh-2015-02-18.RData"
## [3] "Afghanistan-Canada-2012-03-18.RData"
## [4] "Afghanistan-England-2012-09-21.RData"
## [5] "Afghanistan-England-2015-03-13.RData"
## [6] "Afghanistan-Hong Kong-2014-03-18.RData"
```

We are going to apply our functions to a list of filenames (since they all take filenames as input: that was a design decision). The normal way to do this would be to use `sapply`: if the functions each returned a scalar, the results would be glued together into a vector. Here, though, each function returns a data frame, so we use `lapply` to produce a **list** of data frames, which we then run through `bind_rows` to produce a big data frame with these rows stacked atop each other. This is something we'll be doing a lot, so let's make a function to do it all. `v` is a vector (of eg. file names) and `f` is a function that returns a data frame, so that my `xx` is a list of data frames:

```
atop=function(v,f) {
  xx=lapply(v,f)
  bind_rows(xx)
}
```

Since it doesn't have a data frame as a first argument (I couldn't make that work), it won't work in a pipe.

We need to test this, so let's check whether each of our match files has a complete first innings (which we'll need to check anyway):

```
isComplete=atop(files,firstComplete)
isComplete
```

```
## # A tibble: 2,099 x 2
##                                      fname complete
##                                      <chr>    <lgl>
## 1   Afghanistan-Australia-2012-08-25.RData     TRUE
## 2  Afghanistan-Bangladesh-2015-02-18.RData     TRUE
## 3      Afghanistan-Canada-2012-03-18.RData    FALSE
## 4     Afghanistan-England-2012-09-21.RData    FALSE
## 5     Afghanistan-England-2015-03-13.RData    FALSE
## 6   Afghanistan-Hong Kong-2014-03-18.RData    FALSE
## 7   Afghanistan-Hong Kong-2015-07-21.RData    FALSE
## 8       Afghanistan-India-2010-05-01.RData    FALSE
## 9       Afghanistan-India-2012-09-19.RData    FALSE
## 10      Afghanistan-India-2014-03-05.RData     TRUE
## # ... with 2,089 more rows
```

The second line is the match we've been using to test. What is up with the lines here that are `FALSE`?

```
isComplete %>% slice(3:9) -> tmp
atop(tmp$fname,summarizeMatch)
```

```
## # A tibble: 14 x 4
##     inns runTotal wktTotal ballTotal
##    <chr>    <dbl>    <int>     <dbl>
## 1    1st      174        8       123
## 2    2nd      133        9       121
## 3    1st      196        5       120
## 4    2nd       80       10       104
## 5    1st      111        7       218
## 6    2nd      101        1       109
## 7    1st      153        8       120
## 8    2nd      154        3       108
## 9    1st      161        7       120
## 10   2nd      162        5       121
## 11   1st      115        8       120
## 12   2nd      116        3        89
## 13   1st      159        5       121
## 14   2nd      136       10       117
```

In each case, a match takes up two lines; we only need the line starting with "1st". You see that each first innings is not done yet (less than 10 wickets) but the allotted 300 balls have not been used. I guess that these matches were interrupted, with a revised number of overs for at least the team batting second. In any case, we don't want to include these in our analysis.

In addition, we need to check whether each match is a ODI (which we want to assess) or something else.

That's another application of `atop`:

```
v=atop(files,matchInfo)
```

```
## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character
```

```
head(v)
```

```
##                                     fname       date type
## 1  Afghanistan-Australia-2012-08-25.RData 2012-08-25  ODI
## 2 Afghanistan-Bangladesh-2015-02-18.RData 2015-02-18  ODI
## 3     Afghanistan-Canada-2012-03-18.RData 2012-03-18  T20
## 4    Afghanistan-England-2012-09-21.RData 2012-09-21  T20
## 5    Afghanistan-England-2015-03-13.RData 2015-03-13  ODI
## 6  Afghanistan-Hong Kong-2014-03-18.RData 2014-03-18  T20
```

```
tab=table(v$type)
tab
```

```
##
##  ODI  T20
## 1139  960
```

There are 1139 one-day internationals, and 960 Twenty-20 matches, which are a mixture of internationals and Indian Premier League.

Getting all the matches we want

To get just the matches we want, we do this:

- read all the matches' info
- select only the ones whose type is ODI
- of those, select the ones that have complete first innings
- construct the desired (big) data for these (just first innings)

Since `atop` returns a data frame, we can use it as the *start* of a pipe:

```
atop(files,matchInfo) %>% filter(type=="ODI") -> odis
```

```
## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character
```

```
atop(odis$fname,firstComplete) %>% filter(complete) -> complete.odis
complete.odis
```

```
## # A tibble: 1,040 x 2
##                                        fname complete
##                                        <chr>    <lgl>
## 1    Afghanistan-Australia-2012-08-25.RData     TRUE
## 2   Afghanistan-Bangladesh-2015-02-18.RData     TRUE
## 3        Afghanistan-India-2014-03-05.RData     TRUE
## 4      Afghanistan-Ireland-2015-01-17.RData     TRUE
## 5  Afghanistan-Netherlands-2012-03-29.RData     TRUE
## 6     Afghanistan-Pakistan-2012-02-10.RData     TRUE
## 7     Afghanistan-Pakistan-2014-02-27.RData     TRUE
## 8     Afghanistan-Scotland-2009-04-19.RData     TRUE
```

```
## 9     Afghanistan-Scotland-2015-01-14.RData     TRUE
## 10    Afghanistan-Scotland-2015-02-26.RData     TRUE
## # ... with 1,030 more rows
```

There are 1040 complete ODIs. Now to get the actual information, which we can expect to be slow:

```
atop(complete.odis$fname,processMatch) %>%
  filter(inns=="1st") -> d
d
```

```
## Source: local data frame [307,956 x 6]
## Groups: inns [1]
##
##     inns totalBalls isExtra isWkt wktDown totalRuns
##    <chr>      <dbl>   <lgl> <lgl>   <int>     <dbl>
## 1    1st          1   FALSE FALSE       0         0
## 2    1st          2   FALSE FALSE       0         0
## 3    1st          3   FALSE FALSE       0         0
## 4    1st          4   FALSE FALSE       0         1
## 5    1st          5   FALSE FALSE       0         0
## 6    1st          6   FALSE FALSE       0         0
## 7    1st          7   FALSE FALSE       0         0
## 8    1st          8   FALSE FALSE       0         0
## 9    1st          9   FALSE FALSE       0         0
## 10   1st         10   FALSE FALSE       0         0
## # ... with 307,946 more rows
```

307956 rows! But that's what we need.

Estimating the parameters

The model

The Carter and Guthrie (2004) model is a simplified version of cricketing reality, thus:

- with a certain probability (which is fixed), a ball is an extra (wide or no ball), which counts one run and leads to an extra ball being bowled. (The assumption is that no additional runs are scored, for example runs scored off a no ball.)
- otherwise, with a certain probability, which depends on the number of balls and wickets left, a ball is a wicket. This reduces the number of balls and wickets left by 1 without changing the number of runs. (This assumes that no runs are made on a ball with a wicket, such as when a high near-six is caught near the boundary and the batsmen have already completed one or more runs.)
- otherwise, with certain probabilities, which depend on the number of balls and wickets left, a certain number of runs is scored.

Probability of extra

Estimating the "extra" probability is the easiest, since that doesn't (by hypothesis) depend on anything else:

```
d %>% summarize(pEx=sum(isExtra)/nrow(d)) %>% select(pEx) -> pExtra
pExtra
```

```
## # A tibble: 1 x 1
##          pEx
```

```
##        <dbl>
## 1 0.02616932
```

This is about half the corresponding value in Carter and Guthrie (2004).

Probability of wicket

The probability of a wicket is modelled using a logistic model (a probit model in Carter and Guthrie (2004)); the log-odds of a wicket is modelled as a quadratic function of the number of balls and wickets left.

Thus our first step is to calculate the number of balls and wickets left, and then to remove the rows with extras:

```
d %>% mutate(ballsLeft=300-totalBalls,
            wktsLeft=10-wktDown) %>%
  filter(!isExtra) -> d1
```

and now we can model:

```
wModel=glm(isWkt~ballsLeft+wktsLeft+I(ballsLeft^2),data=d1,family="binomial")
summary(wModel)
```

```
##
## Call:
## glm(formula = isWkt ~ ballsLeft + wktsLeft + I(ballsLeft^2),
##     family = "binomial", data = d1)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.7582  -0.2410  -0.1954  -0.1590   3.2174
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -1.109e+00  3.091e-02  -35.87   <2e-16 ***
## ballsLeft      -9.177e-03  4.845e-04  -18.94   <2e-16 ***
## wktsLeft       -4.029e-01  6.681e-03  -60.30   <2e-16 ***
## I(ballsLeft^2)  4.825e-05  1.613e-06   29.91   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 76107  on 299896  degrees of freedom
## Residual deviance: 70377  on 299893  degrees of freedom
## AIC: 70385
##
## Number of Fisher Scoring iterations: 7
```

Everything here is strongly significant, as would be expected with so much data. (These are the same explanatory variables as used by Carter and Guthrie (2004).)

We can make a table of predicted probability of a ball being a wicket as a function of the number of balls and wickets left:

```
new=expand.grid(wktsLeft=c(1,5,10),ballsLeft=c(1,100,200,300))
p=predict(wModel,new,type="response")
data.frame(new,p)
```

```
##    wktsLeft ballsLeft           p
## 1         1         1 0.179338444
## 2         5         1 0.041787135
## 3        10         1 0.005782970
## 4         1       100 0.124889986
## 5         5       100 0.027691119
## 6        10       100 0.003784241
## 7         1       200 0.195104269
## 8         5       200 0.046140556
## 9        10       200 0.006410539
## 10        1       300 0.519366763
## 11        5       300 0.177388823
## 12       10       300 0.027957981
```

For a fixed number of balls left, the probability of a wicket is higher if there are fewer wickets left. The pattern for a fixed number of balls left is unclear, but then in practice if there are many balls left, there are unlikely to be few wickets left.

Modelling the number of runs

Carter and Guthrie (2004) consider the number of runs on any ball as an ordered response, and model the probit of the probability of landing in each category as a quadratic function of the number of balls and wickets left. We use an ordered logistic model, using the `polr` function from library `MASS`. First, however, we need to remove the wicket balls from the data frame used to estimate the wickets process, and we need to turn the `totalRuns` on each ball into an ordered factor:

```
d1 %>% mutate(rf=ordered(totalRuns)) %>%
  filter(!isWkt) -> d2
```

and then down to business:

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
rModel=polr(rf~ballsLeft+wktsLeft+I(ballsLeft^2),data=d2)
```

The summary of the model is not very illuminating, so again we show predictions. In the output below, the number of runs is turned into column names for a data frame by appending an X:

```
p=predict(rModel,new,type="probs")
data.frame(new,round(p,3))
```

```
##    wktsLeft ballsLeft    X0    X1    X2    X3    X4    X5    X6
## 1         1         1 0.399 0.398 0.073 0.009 0.102 0.000 0.019
## 2         5         1 0.242 0.411 0.109 0.015 0.183 0.001 0.039
## 3        10         1 0.114 0.317 0.132 0.021 0.323 0.001 0.092
## 4         1       100 0.680 0.246 0.029 0.004 0.035 0.000 0.006
## 5         5       100 0.506 0.352 0.053 0.007 0.069 0.000 0.013
## 6        10       100 0.291 0.417 0.097 0.013 0.151 0.000 0.031
## 7         1       200 0.836 0.132 0.013 0.002 0.015 0.000 0.003
## 8         5       200 0.711 0.225 0.025 0.003 0.031 0.000 0.005
```

```
## 9         10        200 0.497 0.357 0.055 0.007 0.072 0.000 0.013
## 10         1        300 0.901 0.081 0.007 0.001 0.008 0.000 0.001
## 11         5        300 0.814 0.148 0.015 0.002 0.017 0.000 0.003
## 12        10        300 0.638 0.274 0.034 0.004 0.042 0.000 0.007
```

For any fixed number of balls left, if there are fewer wickets left, there is a higher chance of a "dot ball" (no runs), and a lower chance of a boundary (4 or 6 runs, in the columns labelled `X4` and `X6`). Also, for a fixed number of wickets left, there is a higher chance of a dot ball when many balls are left and a higher chance of a boundary with few balls left. These accord with intuition.

Constructing the recursion

Let $F(r;b,w)$ denote the probability of a team scoring r runs or fewer with b balls and w wickets left. There are some boundary cases: if there are no balls or wickets left, then a team scores 0 (more) runs with probability 1. Thus $F(r;0,w)=1, F(r;b,0)=1$ for $r\geq 0$, and (for completeness) $F(r;b,w)=0$ for $r<0$. These form the base cases for a recursion. To evaluate $F(r;b,w)$ in general, consider what might happen on the next ball: an extra, in which case the team needs $r-1$ runs from b balls with w wickets left; a wicket, in which case the team needs r runs from $b-1$ balls with $w-1$ wickets left, or j runs are scored, in which case the team needs $r-j$ runs from $b-1$ balls with w wickets left. Details can be found in Carter and Guthrie (2004, 826).

As an example, suppose we wish to find $F(4;3,2)$. We need the probability of an extra, a wicket and each number of runs at this point:

```
new=data.frame(ballsLeft=3,wktsLeft=2)
pExtra
```

```
## # A tibble: 1 x 1
##          pEx
##        <dbl>
## 1 0.02616932
```

```
predict(wModel,new,type="response")
```

```
##         1
## 0.1254596
```

```
predict(rModel,new,type="probs")
```

```
##            0            1            2            3            4
## 0.3618481772 0.4080933004 0.0804909684 0.0106283094 0.1161361034
##            5            6
## 0.0003392157 0.0224639254
```

Thus, the probability of an extra is 0.026, the probability of a wicket given a non-extra is 0.126, and the probabilities of $0,1,2,\ldots$ runs, given that there was neither an extra nor a wicket, are $0.363, 0.408, 0.080, \ldots$. In detail, therefore, the recursion says that

$$\begin{aligned} F(4;3,2) \quad &= \quad 0.026F(3;3,2)+(1-0.026)(0.126)F(4;2,1)+ \\ &\quad\ (1-0.026)(1-0.126)\left[0.363F(4;2,2)+0.408F(3;2,2)+\right. \\ &\quad\ 0.080F(2;2,2)+0.011F(1;2,2)+0.116F(0;2,2)+ \\ &\quad\ \left.0.0003F(-1;2,2)+0.022F(-2;2,2)\right] \end{aligned}$$

We note that $F(-1;2,2) = F(-2;2,2) = 0$ (in words, the probability of getting 4 runs or less in 3 balls is 0 if 5 or 6 runs are scored off the first ball). The remaining $F(r;b,w)$ need to be calculated, and this is done using a similar recursion. Note that, for example, $F(3;3,2)$ will also need $F(2;2,2)$ (if one run is scored off the next ball), and thus a naive recursion would do a lot of unnecessary re-calculation. It is important, therefore, to create a look-up table of values already calculated, and to check it before starting on a calculation. In pseudocode, the procedure is therefore this:

```
bigF=function(r,b,w) {
  # base cases
  if (r<0) return 0
  if (b==0) return 1
  if (w==0) return 1
  # lookup table
  if (r,b,w) in lookup table, return value
  # else recurse
  obtain probability of extra PE, wicket PW, and array P[j] each number of runs j this ball
  sum=0
  for (j in 0:6) {
    sum=sum+P[j]*bigF(r-j,b-1,w)
  }
  ans=PE*bigF(r-1,b,w)+(1-PE)*PW*bigF(r,b-1,w-1)+(1-PE)*(1-PW)*sum
  save ans in lookup table
  return ans
}
```

Let F be the probability of scoring r runs or less with b balls and w wickets remaining. There are two choices for the data structure for the lookup table. One is a three-dimensional array (dimensions runs, balls and wickets, with the value of the array being F), and the other is a data frame, with columns for runs, balls, wickets and F. I chose the latter, because of the simplicity of using `dplyr::filter` to see whether a row exists yet. (The notation F is a reminder that this is a *cumulative* probability, that many runs or less.)

We will keep the lookup table as a data frame called `lookupTable`, dimensions runs, balls and wickets left in that order, initialized as empty and kept in an environment called `env` (so that we can access and change it from within a function):

```
max.runs=400
max.balls=300
max.wickets=10
env=new.env(parent=emptyenv())
aa=data.frame(rr=integer(),bb=integer(),ww=integer(),F=double())
env$lookupTable=aa
str(env$lookupTable)
```

```
## 'data.frame':    0 obs. of  4 variables:
##  $ rr: int
##  $ bb: int
##  $ ww: int
##  $ F : num
```

Then we implement the function outlined in pseudocode above:

```
bigF=function(r,b,w) {
  # base cases
  if (r<0) return(0)
  if (b==0) return(1)
  if (w==0) return(1)
  # return value if in lookup table
```

```
  tab=get("lookupTable",envir=env)
  tab %>% filter(rr==r, bb==b, ww==w) -> x
  if (nrow(x)>0) return(x[1,4])
  # recursion
  new=data.frame(ballsLeft=b,wktsLeft=w)
  pW=predict(wModel,new,type="response")
  p=predict(rModel,new,type="probs")
  pE=as.numeric(pExtra[1,1]) # global variable
  sum=0
  for (j in 0:6) {
    sum=sum+p[j+1]*bigF(r-j,b-1,w)
  }
  ans=pE*bigF(r-1,b,w)+(1-pE)*pW*bigF(r,b-1,w-1)+(1-pE)*(1-pW)*sum
  names(ans)=NULL
  # lookup table might have changed since earlier, so get again before altering
  tab=get("lookupTable",envir=env)
  tab=rbind(tab,data.frame(rr=r,bb=b,ww=w,F=ans))
  assign("lookupTable",value=tab,envir=env)
  return(ans)
}
```

Let's test it with the example we used above, and at the same time, time it:

```
system.time(ans <- bigF(4,3,2))
```

```
##    user  system elapsed
##   0.128   0.000   0.127
```

```
ans
```

```
## [1] 0.721199
```

This result is reasonable, since with only three balls and two wickets left, it is most likely that four runs or fewer will be scored. I had to use the "alternative" arrow-like assignment operator here, because an equals sign inside `system.time` has a different meaning. We will assess the times in a moment.

It would be interesting to see what the lookup table looks like (that is, which values have been calculated). This is a data frame, so can just be displayed, or we can sort by something. I wanted to see the cumulative distribution of runs given a number of balls and wickets remaining:

```
env$lookupTable %>% arrange(bb,ww,rr)
```

```
##    rr bb ww          F
## 1   0  1  1 0.49321063
## 2   1  1  1 0.82404148
## 3   2  1  1 0.89087956
## 4   3  1  1 0.90020778
## 5   4  1  1 0.98174689
## 6   0  1  2 0.42641663
## 7   1  1  2 0.78550080
## 8   2  1  2 0.86441506
## 9   3  1  2 0.87568180
## 10  4  1  2 0.97686396
## 11  0  2  1 0.33198089
## 12  1  2  1 0.60371717
## 13  2  2  1 0.76588781
## 14  3  2  1 0.81717688
```

```
## 15   4   2   1 0.89376016
## 16   0   2   2 0.19088917
## 17   1   2   2 0.49451548
## 18   2   2   2 0.68908943
## 19   3   2   2 0.75489255
## 20   4   2   2 0.85277871
## 21   0   3   2 0.09938645
## 22   1   3   2 0.29509984
## 23   2   3   2 0.49860893
## 24   3   3   2 0.62064608
## 25   4   3   2 0.72119896
```

Because the lookup table has now been populated, running the same calculation again should be a lot quicker:

```r
system.time(ans <- bigF(4,3,2))
```

```
##    user  system elapsed
##   0.004   0.000   0.003
```

```r
ans
```

```
## [1] 0.721199
```

And so it is.

Applications

The nature of this recursion, with 8 recursive calls to `bigF` inside one original call, means that we threaten to branch wildly and will need to keep a large number of (r, b, w) triples on the stack. This is obviated by calling `bigF` repeatedly with values of r, b, w only a little bigger than those already in the lookup table, taking advantage of the calculations already done (which will be most of the ones we need). For example:

```r
system.time(bigF(10,10,10))
```

```
##    user  system elapsed
##   4.432   0.000   4.435
```

showing, for example, the cumulative distribution of runs with 8 balls and 9 wickets left (as calculated so far):

```r
env$lookupTable
```

```
##      rr bb  ww            F
## 1     0  1   2 4.264166e-01
## 2     1  1   2 7.855008e-01
## 3     2  1   2 8.644151e-01
## 4     3  1   2 8.756818e-01
## 5     4  1   2 9.768640e-01
## 6     0  1   1 4.932106e-01
## 7     0  2   2 1.908892e-01
## 8     1  1   1 8.240415e-01
## 9     1  2   2 4.945155e-01
## 10    2  1   1 8.908796e-01
## 11    2  2   2 6.890894e-01
## 12    3  1   1 9.002078e-01
## 13    3  2   2 7.548925e-01
## 14    4  1   1 9.817469e-01
## 15    4  2   2 8.527787e-01
```

```
## 16   0  2  1 3.319809e-01
## 17   0  3  2 9.938645e-02
## 18   1  2  1 6.037172e-01
## 19   1  3  2 2.950998e-01
## 20   2  2  1 7.658878e-01
## 21   2  3  2 4.986089e-01
## 22   3  2  1 8.171769e-01
## 23   3  3  2 6.206461e-01
## 24   4  2  1 8.937602e-01
## 25   4  3  2 7.211990e-01
## 26   0  1 10 1.156808e-01
## 27   1  1 10 4.257753e-01
## 28   2  1 10 5.614292e-01
## 29   3  1 10 5.853785e-01
## 30   4  1 10 8.985951e-01
## 31   5  1 10 9.080427e-01
## 32   6  1 10 9.975935e-01
## 33   7  1 10 9.999370e-01
## 34   8  1 10 9.999984e-01
## 35   9  1 10 1.000000e+00
## 36  10  1 10 1.000000e+00
## 37   0  1  9 1.371772e-01
## 38   0  2 10 1.364679e-02
## 39   1  1  9 4.715516e-01
## 40   1  2 10 8.613657e-02
## 41   2  1  9 6.065935e-01
## 42   2  2 10 2.144448e-01
## 43   3  1  9 6.297739e-01
## 44   3  2 10 3.044143e-01
## 45   4  1  9 9.138871e-01
## 46   4  2 10 4.101744e-01
## 47   5  1  9 9.223943e-01
## 48   5  2 10 6.130717e-01
## 49   6  1  9 9.979691e-01
## 50   6  2 10 7.248551e-01
## 51   7  1  9 9.999469e-01
## 52   7  2 10 7.982689e-01
## 53   8  1  9 9.999986e-01
## 54   8  2 10 9.218142e-01
## 55   9  1  9 1.000000e+00
## 56   9  2 10 9.326238e-01
## 57  10  1  9 1.000000e+00
## 58  10  2 10 9.884259e-01
## 59   0  1  8 1.624145e-01
## 60   0  2  9 1.922219e-02
## 61   0  3 10 1.643825e-03
## 62   1  1  8 5.182003e-01
## 63   1  2  9 1.118591e-01
## 64   1  3 10 1.460735e-02
## 65   2  1  8 6.503805e-01
## 66   2  2  9 2.614652e-01
## 67   2  3 10 5.449871e-02
## 68   3  1  8 6.724397e-01
## 69   3  2  9 3.583574e-01
```

```
## 70    3  3 10 1.156138e-01
## 71    4  1  8 9.271760e-01
## 72    4  2  9 4.699305e-01
## 73    4  3 10 1.797561e-01
## 74    5  1  8 9.347556e-01
## 75    5  2  9 6.683254e-01
## 76    5  3 10 2.747268e-01
## 77    6  1  8 9.982926e-01
## 78    6  2  9 7.715947e-01
## 79    6  3 10 4.096216e-01
## 80    7  1  8 9.999553e-01
## 81    7  2  9 8.378965e-01
## 82    7  3 10 5.187978e-01
## 83    8  1  8 9.999988e-01
## 84    8  2  9 9.400456e-01
## 85    8  3 10 6.212674e-01
## 86    9  1  8 1.000000e+00
## 87    9  2  9 9.488874e-01
## 88    9  3 10 7.458280e-01
## 89   10  1  8 1.000000e+00
## 90   10  2  9 9.915966e-01
## 91   10  3 10 8.201832e-01
## 92    0  1  7 1.919674e-01
## 93    0  2  8 2.700279e-02
## 94    0  3  9 2.755803e-03
## 95    0  4 10 2.024838e-04
## 96    1  1  7 5.651009e-01
## 97    1  2  8 1.436170e-01
## 98    1  3  9 2.242180e-02
## 99    1  4 10 2.307160e-03
## 100   2  1  7 6.922777e-01
## 101   2  2  8 3.138125e-01
## 102   2  3  9 7.743862e-02
## 103   2  4 10 1.147321e-02
## 104   3  1  7 7.129180e-01
## 105   3  2  8 4.151631e-01
## 106   3  3  9 1.545331e-01
## 107   3  4 10 3.325944e-02
## 108   4  1  7 9.386935e-01
## 109   4  2  8 5.307730e-01
## 110   4  3  9 2.306233e-01
## 111   4  4 10 6.651824e-02
## 112   5  1  7 9.453751e-01
## 113   5  2  8 7.199863e-01
## 114   5  3  9 3.386473e-01
## 115   5  4 10 1.114378e-01
## 116   6  1  7 9.985705e-01
## 117   6  2  8 8.133602e-01
## 118   6  3  9 4.814303e-01
## 119   6  4 10 1.802679e-01
## 120   7  1  7 9.999626e-01
## 121   7  2  8 8.720966e-01
## 122   7  3  9 5.901286e-01
## 123   7  4 10 2.705937e-01
```

```
## 124  8  1  7 9.999990e-01
## 125  8  2  8 9.546843e-01
## 126  8  3  9 6.884454e-01
## 127  8  4 10 3.616116e-01
## 128  9  1  7 1.000000e+00
## 129  9  2  8 9.617635e-01
## 130  9  3  9 7.994377e-01
## 131  9  4 10 4.619508e-01
## 132 10  1  7 1.000000e+00
## 133 10  2  8 9.939525e-01
## 134 10  3  9 8.626301e-01
## 135 10  4 10 5.746153e-01
## 136  0  1  6 2.264689e-01
## 137  0  2  7 3.782375e-02
## 138  0  3  8 4.604763e-03
## 139  0  4  9 4.049835e-04
## 140  0  5 10 2.555159e-05
## 141  1  1  6 6.116678e-01
## 142  1  2  7 1.822288e-01
## 143  1  3  8 3.395302e-02
## 144  1  4  9 4.202616e-03
## 145  1  5 10 3.532409e-04
## 146  2  1  6 7.319032e-01
## 147  2  2  7 3.708128e-01
## 148  2  3  8 1.077834e-01
## 149  2  4  9 1.919200e-02
## 150  2  5 10 2.187927e-03
## 151  3  1  6 7.508900e-01
## 152  3  2  7 4.736674e-01
## 153  3  3  8 2.016278e-01
## 154  3  4  9 5.163752e-02
## 155  3  5 10 8.079322e-03
## 156  4  1  6 9.486613e-01
## 157  4  2  7 5.913870e-01
## 158  4  3  8 2.889168e-01
## 159  4  4  9 9.731150e-02
## 160  4  5 10 2.044070e-02
## 161  5  1  6 9.544870e-01
## 162  5  2  7 7.672320e-01
## 163  5  3  8 4.080992e-01
## 164  5  4  9 1.556724e-01
## 165  5  5 10 4.032426e-02
## 166  6  1  6 9.988090e-01
## 167  6  2  7 8.499631e-01
## 168  6  3  8 5.540050e-01
## 169  6  4  9 2.402720e-01
## 170  6  5 10 7.085960e-02
## 171  7  1  6 9.999688e-01
## 172  7  2  7 9.009623e-01
## 173  7  3  8 6.585567e-01
## 174  7  4  9 3.437431e-01
## 175  7  5 10 1.175065e-01
## 176  8  1  6 9.999992e-01
## 177  8  2  7 9.662589e-01
```

```
## 178  8  3  8 7.500289e-01
## 179  8  4  9 4.420769e-01
## 180  8  5 10 1.787813e-01
## 181  9  1  6 1.000000e+00
## 182  9  2  7 9.718062e-01
## 183  9  3  8 8.454724e-01
## 184  9  4  9 5.453600e-01
## 185  9  5 10 2.496227e-01
## 186 10  1  6 1.000000e+00
## 187 10  2  7 9.956897e-01
## 188 10  3  8 8.975696e-01
## 189 10  4  9 6.539378e-01
## 190 10  5 10 3.338913e-01
## 191  0  1  5 2.665899e-01
## 192  0  2  6 5.281414e-02
## 193  0  3  7 7.667926e-03
## 194  0  4  8 8.073261e-04
## 195  0  5  9 6.114452e-05
## 196  0  6 10 3.310421e-06
## 197  1  1  5 6.573859e-01
## 198  1  2  6 2.284302e-01
## 199  1  3  7 5.068675e-02
## 200  1  4  8 7.542161e-03
## 201  1  5  9 7.660542e-04
## 202  1  6 10 5.350554e-05
## 203  2  1  5 7.690113e-01
## 204  2  2  6 4.315619e-01
## 205  2  3  7 1.468772e-01
## 206  2  4  8 3.134835e-02
## 207  2  5  9 4.331292e-03
## 208  2  6 10 3.949604e-04
## 209  3  1  5 7.861753e-01
## 210  3  2  6 5.327008e-01
## 211  3  3  7 2.568575e-01
## 212  3  4  8 7.781474e-02
## 213  3  5  9 1.472102e-02
## 214  3  6 10 1.769703e-03
## 215  4  1  5 9.572859e-01
## 216  4  2  6 6.505329e-01
## 217  4  3  7 3.537185e-01
## 218  4  4  8 1.378501e-01
## 219  4  5  9 3.465992e-02
## 220  4  6 10 5.466645e-03
## 221  5  1  5 9.623064e-01
## 222  5  2  6 8.095601e-01
## 223  5  3  7 4.811869e-01
## 224  5  4  8 2.106288e-01
## 225  5  5  9 6.447426e-02
## 226  5  6 10 1.284032e-02
## 227  6  1  5 9.990136e-01
## 228  6  2  6 8.814773e-01
## 229  6  3  7 6.249930e-01
## 230  6  4  8 3.103504e-01
## 231  6  5  9 1.077020e-01
```

```
## 232  6  6 10 2.536923e-02
## 233  7  1  5 9.999742e-01
## 234  7  2  6 9.248199e-01
## 235  7  3  7 7.221665e-01
## 236  7  4  8 4.236023e-01
## 237  7  5  9 1.697065e-01
## 238  7  6 10 4.572660e-02
## 239  8  1  5 9.999993e-01
## 240  8  2  6 9.752796e-01
## 241  8  3  7 8.046697e-01
## 242  8  4  8 5.252308e-01
## 243  8  5  9 2.457951e-01
## 244  8  6 10 7.676537e-02
## 245  9  1  5 1.000000e+00
## 246  9  2  6 9.795313e-01
## 247  9  3  7 8.838073e-01
## 248  9  4  8 6.270490e-01
## 249  9  5  9 3.288285e-01
## 250  9  6 10 1.187570e-01
## 251 10  1  5 1.000000e+00
## 252 10  2  6 9.969619e-01
## 253 10  3  7 9.255185e-01
## 254 10  4  8 7.270154e-01
## 255 10  5  9 4.223227e-01
## 256 10  6 10 1.716495e-01
## 257  0  1  4 3.129879e-01
## 258  0  2  5 7.347681e-02
## 259  0  3  6 1.272175e-02
## 260  0  4  7 1.604184e-03
## 261  0  5  8 1.459411e-04
## 262  0  6  9 9.510528e-06
## 263  0  7 10 4.415013e-07
## 264  1  1  4 7.018273e-01
## 265  1  2  5 2.827857e-01
## 266  1  3  6 7.454078e-02
## 267  1  4  7 1.332430e-02
## 268  1  5  8 1.635778e-03
## 269  1  6  9 1.385965e-04
## 270  1  7 10 8.119510e-06
## 271  2  1  4 8.034799e-01
## 272  2  2  5 4.949894e-01
## 273  2  3  6 1.958943e-01
## 274  2  4  7 4.995031e-02
## 275  2  5  8 8.355386e-03
## 276  2  6  9 9.289414e-04
## 277  2  7 10 6.922704e-05
## 278  3  1  4 8.187135e-01
## 279  3  2  5 5.911866e-01
## 280  3  3  6 3.196671e-01
## 281  3  4  7 1.137549e-01
## 282  3  5  8 2.593239e-02
## 283  3  6  9 3.806770e-03
## 284  3  7 10 3.635081e-04
## 285  4  1  4 9.647555e-01
```

```
## 286  4  2  5 7.071117e-01
## 287  4  3  6 4.237092e-01
## 288  4  4  7 1.891581e-01
## 289  4  5  8 5.654792e-02
## 290  4  6  9 1.084951e-02
## 291  4  7 10 1.327897e-03
## 292  5  1  4 9.690262e-01
## 293  5  2  5 8.467762e-01
## 294  5  3  6 5.557308e-01
## 295  5  4  7 2.762531e-01
## 296  5  5  8 9.902605e-02
## 297  5  6  9 2.377068e-02
## 298  5  7 10 3.665369e-03
## 299  6  1  4 9.991894e-01
## 300  6  2  5 9.081737e-01
## 301  6  3  6 6.922812e-01
## 302  6  4  7 3.888477e-01
## 303  6  5  8 1.572262e-01
## 304  6  6  9 4.426880e-02
## 305  6  7 10 8.261671e-03
## 306  7  1  4 9.999788e-01
## 307  7  2  5 9.441521e-01
## 308  7  3  6 7.795564e-01
## 309  7  4  7 5.071467e-01
## 310  7  5  8 2.354124e-01
## 311  7  6  9 7.557512e-02
## 312  7  7 10 1.636847e-02
## 313  8  1  4 9.999994e-01
## 314  8  2  5 9.822144e-01
## 315  8  3  6 8.516382e-01
## 316  8  4  7 6.076181e-01
## 317  8  5  8 3.248254e-01
## 318  8  6  9 1.202400e-01
## 319  8  7 10 2.984330e-02
## 320  9  1  4 1.000000e+00
## 321  9  2  5 9.853972e-01
## 322  9  3  6 9.148154e-01
## 323  9  4  7 7.036763e-01
## 324  9  5  8 4.170093e-01
## 325  9  6  9 1.768115e-01
## 326  9  7 10 5.050717e-02
## 327 10  1  4 1.000000e+00
## 328 10  2  5 9.978873e-01
## 329 10  3  6 9.472719e-01
## 330 10  4  7 7.915458e-01
## 331 10  5  8 5.152407e-01
## 332 10  6  9 2.440108e-01
## 333 10  7 10 7.945164e-02
## 334  0  1  3 3.661955e-01
## 335  0  2  4 1.017548e-01
## 336  0  3  5 2.101530e-02
## 337  0  4  6 3.176853e-03
## 338  0  5  7 3.475818e-04
## 339  0  6  8 2.729662e-05
```

```
## 340  0  7  9 1.529112e-06
## 341  0  8 10 6.080863e-08
## 342  1  1  3 7.446398e-01
## 343  1  2  4 3.455540e-01
## 344  1  3  5 1.078846e-01
## 345  1  4  6 2.314889e-02
## 346  1  5  7 3.436380e-03
## 347  1  6  8 3.535516e-04
## 348  1  7  9 2.521303e-05
## 349  1  8 10 1.245699e-06
## 350  2  1  3 8.352780e-01
## 351  2  2  4 5.599157e-01
## 352  2  3  5 2.556514e-01
## 353  2  4  6 7.756588e-02
## 354  2  5  7 1.568572e-02
## 355  2  6  8 2.126466e-03
## 356  2  7  9 1.941511e-04
## 357  2  8 10 1.197877e-05
## 358  3  1  3 8.485291e-01
## 359  3  2  4 6.482025e-01
## 360  3  3  5 3.889940e-01
## 361  3  4  6 1.612798e-01
## 362  3  5  7 4.411350e-02
## 363  3  6  8 7.895216e-03
## 364  3  7  9 9.271960e-04
## 365  3  8 10 7.180618e-05
## 366  4  1  3 9.712354e-01
## 367  4  2  4 7.601892e-01
## 368  4  3  5 4.972861e-01
## 369  4  4  6 2.516404e-01
## 370  4  5  7 8.872970e-02
## 371  4  6  8 2.062187e-02
## 372  4  7  9 3.104181e-03
## 373  4  8 10 3.022270e-04
## 374  5  1  3 9.748136e-01
## 375  5  2  4 8.789434e-01
## 376  5  3  5 6.294532e-01
## 377  5  4  6 3.516465e-01
## 378  5  5  7 1.461611e-01
## 379  5  6  8 4.198703e-02
## 380  5  7  9 7.925072e-03
## 381  5  8 10 9.623015e-04
## 382  6  1  3 9.993409e-01
## 383  6  2  4 9.304447e-01
## 384  6  3  5 7.541760e-01
## 385  6  4  6 4.732229e-01
## 386  6  5  7 2.206098e-01
## 387  6  6  8 7.360091e-02
## 388  6  7  9 1.668809e-02
## 389  6  8 10 2.469371e-03
## 390  7  1  3 9.999828e-01
## 391  7  2  4 9.595209e-01
## 392  7  3  5 8.298827e-01
## 393  7  4  6 5.909867e-01
```

```
## 394  7  5  7 3.139783e-01
## 395  7  6  8 1.189361e-01
## 396  7  7  9 3.112959e-02
## 397  7  8 10 5.428876e-03
## 398  8  1  3 9.999995e-01
## 399  8  2  4 9.874742e-01
## 400  8  3  5 8.907782e-01
## 401  8  4  6 6.859782e-01
## 402  8  5  7 4.132867e-01
## 403  8  6  8 1.792867e-01
## 404  8  7  9 5.360757e-02
## 405  8  8 10 1.075263e-02
## 406  9  1  3 1.000000e+00
## 407  9  2  4 9.897950e-01
## 408  9  3  5 9.392156e-01
## 409  9  4  6 7.725804e-01
## 410  9  5  7 5.101685e-01
## 411  9  6  8 2.507398e-01
## 412  9  7  9 8.582651e-02
## 413  9  8 10 1.968740e-02
## 414 10  1  3 1.000000e+00
## 415 10  2  4 9.985557e-01
## 416 10  3  5 9.637588e-01
## 417 10  4  6 8.462393e-01
## 418 10  5  7 6.078020e-01
## 419 10  6  8 3.308006e-01
## 420 10  7  9 1.281300e-01
## 421 10  8 10 3.354326e-02
## 422  0  2  3 1.400241e-01
## 423  0  3  4 3.451464e-02
## 424  0  4  5 6.265399e-03
## 425  0  5  6 8.261245e-04
## 426  0  6  7 7.834312e-05
## 427  0  7  8 5.305272e-06
## 428  0  8  9 2.551895e-07
## 429  0  9 10 8.683863e-09
## 430  1  2  3 4.164833e-01
## 431  1  3  4 1.534506e-01
## 432  1  4  5 3.949267e-02
## 433  1  5  6 7.094023e-03
## 434  1  6  7 8.875446e-04
## 435  1  7  8 7.718049e-05
## 436  1  8  9 4.656748e-06
## 437  1  9 10 1.946768e-07
## 438  2  2  3 6.250756e-01
## 439  2  3  4 3.263666e-01
## 440  2  4  5 1.172539e-01
## 441  2  5  6 2.861505e-02
## 442  2  6  7 4.730448e-03
## 443  2  7  8 5.297285e-04
## 444  2  8  9 4.021283e-05
## 445  2  9 10 2.071252e-06
## 446  3  2  3 7.029887e-01
## 447  3  3  4 4.633181e-01
```

```
## 448  3  4  5 2.217494e-01
## 449  3  5  6 7.238714e-02
## 450  3  6  7 1.576114e-02
## 451  3  7  8 2.276075e-03
## 452  3  8  9 2.180352e-04
## 453  3  9 10 1.388305e-05
## 454  4  2  3 8.089758e-01
## 455  4  3  4 5.726616e-01
## 456  4  4  5 3.249338e-01
## 457  4  5  6 1.339001e-01
## 458  4  6  7 3.749257e-02
## 459  4  7  8 6.925816e-03
## 460  4  8  9 8.361319e-04
## 461  4  9 10 6.591345e-05
## 462  5  1  2 9.798074e-01
## 463  5  2  3 9.063066e-01
## 464  5  3  4 7.001347e-01
## 465  5  4  5 4.350501e-01
## 466  5  5  6 2.075110e-01
## 467  5  6  7 7.073761e-02
## 468  5  7  8 1.626301e-02
## 469  5  8  9 2.445443e-03
## 470  5  9 10 2.377267e-04
## 471  6  1  2 9.994716e-01
## 472  6  2  3 9.487323e-01
## 473  6  3  4 8.094805e-01
## 474  6  4  5 5.602800e-01
## 475  6  5  6 2.978824e-01
## 476  6  6  7 1.166315e-01
## 477  6  7  8 3.189649e-02
## 478  6  8  9 5.816592e-03
## 479  6  9 10 6.879322e-04
## 480  7  1  2 9.999862e-01
## 481  7  2  3 9.715023e-01
## 482  7  3  4 8.728075e-01
## 483  7  4  5 6.717821e-01
## 484  7  5  6 4.032468e-01
## 485  7  6  7 1.783429e-01
## 486  7  7  8 5.593479e-02
## 487  7  8  9 1.195466e-02
## 488  7  9 10 1.680533e-03
## 489  8  1  2 9.999996e-01
## 490  8  2  3 9.914065e-01
## 491  8  3  4 9.223926e-01
## 492  8  4  5 7.575959e-01
## 493  8  5  6 5.073488e-01
## 494  8  6  7 2.547567e-01
## 495  8  7  8 9.089175e-02
## 496  8  8  9 2.226083e-02
## 497  8  9 10 3.629707e-03
## 498  9  1  2 1.000000e+00
## 499  9  2  3 9.930479e-01
## 500  9  3  4 9.579128e-01
## 501  9  4  5 8.320066e-01
```

```
## 502  9  5  6 6.036429e-01
## 503  9  6  7 3.392503e-01
## 504  9  7  8 1.376034e-01
## 505  9  8  9 3.841906e-02
## 506  9  9 10 7.164555e-03
## 507 10  1  2 1.000000e+00
## 508 10  2  3 9.990344e-01
## 509 10  3  4 9.759226e-01
## 510 10  4  5 8.907839e-01
## 511 10  5  6 6.952830e-01
## 512 10  6  7 4.285962e-01
## 513 10  7  8 1.950291e-01
## 514 10  8  9 6.183717e-02
## 515 10  9 10 1.313025e-02
## 516  0  3  3 5.618778e-02
## 517  0  4  4 1.227897e-02
## 518  0  5  5 1.957708e-03
## 519  0  6  6 2.249440e-04
## 520  0  7  7 1.846935e-05
## 521  0  8  8 1.077188e-06
## 522  0  9  9 4.443463e-08
## 523  0 10 10 1.292220e-09
## 524  1  3  3 2.140128e-01
## 525  1  4  4 6.600139e-02
## 526  1  5  5 1.436324e-02
## 527  1  6  6 2.189978e-03
## 528  1  7  7 2.327982e-04
## 529  1  8  8 1.719553e-05
## 530  1  9  9 8.805313e-07
## 531  1 10 10 3.120852e-08
## 532  2  3  3 4.073501e-01
## 533  2  4  4 1.722818e-01
## 534  2  5  5 5.062892e-02
## 535  2  6  6 1.020729e-02
## 536  2  7  7 1.403845e-03
## 537  2  8  8 1.313832e-04
## 538  2  9  9 8.358630e-06
## 539  2 10 10 3.614152e-07
## 540  3  3  3 5.407177e-01
## 541  3  4  4 2.956678e-01
## 542  3  5  5 1.144512e-01
## 543  3  6  6 3.023131e-02
## 544  3  7  7 5.366139e-03
## 545  3  8  8 6.365137e-04
## 546  3  9  9 5.038295e-05
## 547  3 10 10 2.662403e-06
## 548  4  3  3 6.478918e-01
## 549  4  4  4 4.078172e-01
## 550  4  5  5 1.943963e-01
## 551  4  6  6 6.514051e-02
## 552  4  7  7 1.472048e-02
## 553  4  8  8 2.202378e-03
## 554  4  9  9 2.167152e-04
## 555  4 10 10 1.400895e-05
```

```
## 556  5  1  1 9.841142e-01
## 557  5  2  2 9.292097e-01
## 558  5  3  3 7.657203e-01
## 559  5  4  4 5.238799e-01
## 560  5  5  5 2.837483e-01
## 561  5  6  6 1.136939e-01
## 562  5  7  7 3.163821e-02
## 563  5  8  8 5.874203e-03
## 564  5  9  9 7.135869e-04
## 565  5 10 10 5.630044e-05
## 566  6  1  1 9.995843e-01
## 567  6  2  2 9.634710e-01
## 568  6  3  3 8.574712e-01
## 569  6  4  4 6.464633e-01
## 570  6  5  5 3.876296e-01
## 571  6  6  6 1.763143e-01
## 572  6  7  7 5.766963e-02
## 573  6  8  8 1.289053e-02
## 574  6  9  9 1.900975e-03
## 575  6 10 10 1.815514e-04
## 576  7  1  1 9.999891e-01
## 577  7  2  2 9.806391e-01
## 578  7  3  3 9.083859e-01
## 579  7  4  4 7.465818e-01
## 580  7  5  5 4.996401e-01
## 581  7  6  6 2.551031e-01
## 582  7  7  7 9.498001e-02
## 583  7  8  8 2.470088e-02
## 584  7  9  9 4.311129e-03
## 585  7 10 10 4.906039e-04
## 586  8  1  1 9.999997e-01
## 587  8  2  2 9.942955e-01
## 588  8  3  3 9.470986e-01
## 589  8  4  4 8.204919e-01
## 590  8  5  5 6.024288e-01
## 591  8  6  6 3.455469e-01
## 592  8  7  7 1.455418e-01
## 593  8  8  8 4.316906e-02
## 594  8  9  9 8.704522e-03
## 595  8 10 10 1.156840e-03
## 596  9  1  1 1.000000e+00
## 597  9  2  2 9.954144e-01
## 598  9  3  3 9.718598e-01
## 599  9  4  4 8.811420e-01
## 600  9  5  5 6.927634e-01
## 601  9  6  6 4.389344e-01
## 602  9  7  7 2.083734e-01
## 603  9  8  8 7.014434e-02
## 604  9  9  9 1.613073e-02
## 605  9 10 10 2.461927e-03
## 606 10  1  1 1.000000e+00
## 607 10  2  2 9.993733e-01
## 608 10  3  3 9.846399e-01
## 609 10  4  4 9.256530e-01
```

```
## 610 10  5  5 7.737536e-01
## 611 10  6  6 5.321777e-01
## 612 10  7  7 2.806527e-01
## 613 10  8  8 1.065993e-01
## 614 10  9  9 2.782413e-02
## 615 10 10 10 4.833542e-03
```

```
env$lookupTable %>% filter(bb==8, ww==9) %>% arrange(rr)
```

```
##    rr bb ww            F
## 1   0  8  9 2.551895e-07
## 2   1  8  9 4.656748e-06
## 3   2  8  9 4.021283e-05
## 4   3  8  9 2.180352e-04
## 5   4  8  9 8.361319e-04
## 6   5  8  9 2.445443e-03
## 7   6  8  9 5.816592e-03
## 8   7  8  9 1.195466e-02
## 9   8  8  9 2.226083e-02
## 10  9  8  9 3.841906e-02
## 11 10  8  9 6.183717e-02
```

and then

```
system.time(bigF(20,15,10))
```

```
##    user  system elapsed
##  15.588   0.004  15.591
```

after which, one can extract the more complete cumulative distribution of runs with 8 balls and 9 wickets left, thus:

```
env$lookupTable %>% filter(bb==8, ww==3) %>% arrange(rr)
```

```
##    rr bb ww           F
## 1   0  8  3 0.005571794
## 2   1  8  3 0.017038160
## 3   2  8  3 0.038625417
## 4   3  8  3 0.074925238
## 5   4  8  3 0.128495584
## 6   5  8  3 0.195911024
## 7   6  8  3 0.274397007
## 8   7  8  3 0.361791406
## 9   8  8  3 0.453967114
## 10  9  8  3 0.544616343
## 11 10  8  3 0.629506384
## 12 11  8  3 0.706438483
## 13 12  8  3 0.773569514
## 14 13  8  3 0.829666449
## 15 14  8  3 0.875007917
## 16 15  8  3 0.910591963
## 17 16  8  3 0.937609331
## 18 17  8  3 0.957501141
## 19 18  8  3 0.971764816
## 20 19  8  3 0.981690768
## 21 20  8  3 0.988404011
```

```r
system.time(bigF(50,20,10))
```

```
##    user  system elapsed
##  66.828   0.000  66.844
```

```r
str(env$lookupTable)
```

```
## 'data.frame':    7905 obs. of  4 variables:
##  $ rr: num  0 1 2 3 4 0 0 1 1 2 ...
##  $ bb: num  1 1 1 1 1 1 2 1 2 1 ...
##  $ ww: num  2 2 2 2 2 1 2 1 2 1 ...
##  $ F : num  0.426 0.786 0.864 0.876 0.977 ...
```

```r
env$lookupTable %>% filter(bb==20)
```

```
##    rr bb ww            F
## 1   0 20 10 8.433660e-15
## 2   1 20 10 6.043292e-14
## 3   2 20 10 3.090972e-13
## 4   3 20 10 1.610218e-12
## 5   4 20 10 8.871184e-12
## 6   5 20 10 4.727149e-11
## 7   6 20 10 2.302800e-10
## 8   7 20 10 1.009002e-09
## 9   8 20 10 3.977929e-09
## 10  9 20 10 1.418988e-08
## 11 10 20 10 4.613015e-08
## 12 11 20 10 1.377285e-07
## 13 12 20 10 3.806010e-07
## 14 13 20 10 9.808029e-07
## 15 14 20 10 2.373436e-06
## 16 15 20 10 5.426839e-06
## 17 16 20 10 1.178749e-05
## 18 17 20 10 2.443415e-05
## 19 18 20 10 4.852751e-05
## 20 19 20 10 9.265595e-05
## 21 20 20 10 1.705831e-04
## 22 21 20 10 3.035929e-04
## 23 22 20 10 5.234973e-04
## 24 23 20 10 8.763218e-04
## 25 24 20 10 1.426594e-03
## 26 25 20 10 2.262052e-03
## 27 26 20 10 3.498464e-03
## 28 27 20 10 5.284114e-03
## 29 28 20 10 7.803390e-03
## 30 29 20 10 1.127880e-02
## 31 30 20 10 1.597072e-02
## 32 31 20 10 2.217432e-02
## 33 32 20 10 3.021299e-02
## 34 33 20 10 4.042815e-02
## 35 34 20 10 5.316539e-02
## 36 35 20 10 6.875748e-02
## 37 36 20 10 8.750482e-02
## 38 37 20 10 1.096548e-01
## 39 38 20 10 1.353812e-01
```

```
## 40 39 20 10 1.647659e-01
## 41 40 20 10 1.977836e-01
## 42 41 20 10 2.342919e-01
## 43 42 20 10 2.740277e-01
## 44 43 20 10 3.166105e-01
## 45 44 20 10 3.615527e-01
## 46 45 20 10 4.082765e-01
## 47 46 20 10 4.561370e-01
## 48 47 20 10 5.044488e-01
## 49 48 20 10 5.525152e-01
## 50 49 20 10 5.996572e-01
## 51 50 20 10 6.452417e-01
```

```
env$lookupTable %>% filter(bb==20,ww==5) %>% arrange(rr)
```

```
## [1] rr bb ww F
## <0 rows> (or 0-length row.names)
```

References

Carter, M, and G Guthrie. 2004. "Cricket Interruptus: Fairness and Incentive in Limited Overs Cricket Matches." *Journal of the Operational Research Society* 55 (8). Nature Publishing Group: 822–29.

Duckworth, Frank C, and Anthony J Lewis. 1998. "A Fair Method for Resetting the Target in Interrupted One-Day Cricket Matches." *Journal of the Operational Research Society* 49 (3). Springer: 220–27.

Ganesh, Tinniam V. 2016a. *Yorkr: Analyze Cricket Performances Based on Data from Cricsheet*. https://CRAN.R-project.org/package=yorkr.

———. 2016b. "Yorkrdata." *GitHub Repository*. https://github.com/tvganesh/yorkrdata; GitHub.

Wickham, Hadley. 2016. *Tidyr: Easily Tidy Data with 'Spread()' and 'Gather()' Functions*. https://CRAN.R-project.org/package=tidyr.

Wickham, Hadley, and Romain Francois. 2016. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.