Taming PITCHf/x Data with XML2R and pitchRx

by Carson Sievert

Abstract XML2R is a framework that reduces the effort required to transform XML content into tables in a way that preserves parent to child relationships. pitchRx applies XML2R's grammar for XML manipulation to Major League Baseball Advanced Media (MLBAM)'s Gameday data. With pitchRx, one can easily obtain and store Gameday data in a remote database. The Gameday website hosts a wealth of XML data, but perhaps most interesting is PITCHf/x. Among other things, PITCHf/x data can be used to recreate a baseball's flight path from a pitcher's hand to home plate. With pitchRx, one can easily create animations and interactive 3D scatterplots of the baseball's flight path. PITCHf/x data is also commonly used to generate a static plot of baseball locations at the moment they cross home plate. These plots, sometimes called *strike-zone plots*, can also refer to a plot of event probabilities over the same region. pitchRx provides an easy and robust way to generate strike-zone plots using the ggplot2 package.

Introduction

What is PITCHf/x?

PITCHf/x is a general term for a system of cameras which tracks the flight of a baseball with a series of 3D measurements. These measurements define a baseball's flight path from a pitcher's hand to home plate.[1] A best fitting parametric curve is fit to these measurements under the assumption of constant acceleration Alt and White (2008). There are studies that suggest that this assumption is quite reasonable - especially for non-knuckleballs Nathan (2008). In other words, the smoothed flight paths are often a reasonable approximation (within a couple inches) of the real path. The parameters used to fit this curve are made available in XML format on a publicly accessible website. This website, maintained by MLBAM, also hosts a wealth of other baseball related data used to inform MLB's Gameday webcast in near real time.

Why is PITCHf/x important?

On the business side of baseball, using statistical analysis to scout and evaluate players has become mainstream. When PITCHf/x was first introduced, (DiMeo, 2007) proclaimed it as,

> "The new technology that will change statistical analysis [of baseball] forever."

PITCHf/x has yet to fully deliver this claim, partially due to the difficulty in accessing and deriving insight from the large amount of complex information. By providing better tools to collect and visualize this data, **pitchRx** makes PITCHf/x analysis more accessible to the general public.

PITCHf/x applications

PITCHf/x data is and can be used for many different projects. It can also complement other baseball data sources, which poses an interesting database management problem. Statistical analysis of PITCHf/x and baseball data in general has become so popular that it has helped expose statistical ideas and practice to the general public. If you have witnessed television broadcasts of MLB games, you know one obvious application of PITCHf/x is locating pitches in the strike-zone as well as recreating flight trajectories, tracking pitch speed, etc. Some well-known and statistically intriguing problems related to PITCHf/x include: classifying pitch types, predicting pitch sequences, and clustering pitchers with similar tendencies Pane et al. (2013).

Contributions of pitchRx and XML2R

pitchRx has two main focuses Sievert (2014b). The first focus is to provide easy access to Gameday data. Not only is **pitchRx** helpful for collecting this data in bulk, but it has served as a helpful teaching and research aide (baseballwithr.wordpress.com is one such example). Methods for collecting Gameday

[1] A *pitcher* throws a ball to the opposing *batter*, who stands besides home plate and tries to hit the ball into the field of play.

data existed prior to **pitchRx**; however, these methods are not easily extensible and require juggling technologies that may not be familiar or accessible Fast (2007). Moreover, these working environments are less desirable than R for data analysis and visualization. Since **pitchRx** is built upon **XML2R**'s united framework, it can be easily modified and/or extended Sievert (2014a). For this same reason, **pitchRx** serves as a model for building XML data collection tools with **XML2R**.

The other main focus of **pitchRx** is to simplify the process creating popular PITCHf/x graphics. Strike-zone plots and animations made via **pitchRx** utilize the extensible **ggplot2** framework as well as various customized options Wickham (2009). **ggplot2** is a convenient framework for generating strike-zone plots primarily because of its facetting schema which allows one to make visual comparisons across any combination of discrete variable(s). Interactive 3D scatterplots are based on the **rgl** package and useful for gaining a new perspective on flight trajectories.

Getting familiar with Gameday

Gameday data is hosted and made available for free thanks to MLBAM via `http://gd2.mlb.com/components/game/mlb/`.[2] From this website, one can obtain many different types of data besides PITCHf/x. For example, one can obtain everything from structured media metadata to insider tweets. In fact, this website's purpose is to serve data to various mlb.com web pages and applications. As a result, some data is redundant and the format may not be optimal for statistical analysis. For these reasons, `scrape` is focused on retrieving data that is useful for PITCHf/x analysis and providing it in a convenient format for data analysis.

Nativigating through the MLBAM website can be overwhelming, but it helps to recognize that a homepage exists for nearly every day and every game. For example, `http://gd2.mlb.com/components/game/mlb/year_2011/month_02/day_26/` displays numerous hyperlinks to various files specific to February 26th, 2011. On this page is a hyperlink to a miniscoreboard.xml file which contains information on every game played on that date. This page also has numerous hyperlinks to game specific homepages. For example, gid_2011_02_26_phimlb_nyamlb_1/ points to the homepage for that day's game between the NY Yankees and Philadelphia Phillies. On this page is a hyperlink to the players.xml file which contains information about the players, umpires, and coaches (positions, names, batting average, etc.) coming into that game.

Starting from a particular game's homepage and clicking on the inning/ directory, we *should* see another page with links to the inning_all.xml file and the inning_hit.xml file. If it is available, the inning_all.xml file contains the PITCHf/x data for that game. It's important to note that this file won't exist for some games, because some games are played in venues that do not have a working PITCHf/x system in place. This is especially true for preseason games and games played prior to the 2008 season when the PITCHf/x system became widely adopted.[3] The inning_hit.xml files have manually recorded spatial coordinates of where a home run landed or where the baseball made initial contact with a defender after it was hit into play.

The relationship between these XML files and the tables returned by `scrape` is presented in table 1. The `pitch` table is extracted from file(s) whose name ends in inning_all.xml. This is the only table returned by `scrape` that contains data on the pitch-by-pitch level. The `atbat`, `runner`, `action` and `hip` tables from this same file are commonly labeled somewhat ambiguously as play-by-play data. The `player`, `coach`, and `umpire` tables are extracted from players.xml and are classified as game-by-game since there is one record per person per game. Figure 1 shows how these tables can be connected to one another in a database setting. The direction of the arrows represent a one to possibly many relationship. For example, at least one pitch is thrown for each *at bat* (that is, each bout between pitcher and batter) and there are numerous at bats within each game.

In a rough sense, one can relate tables returned by `scrape` back to XML nodes in the XML files. For convenience, some information in certain XML nodes are combined into one table. For example, information gleaned from the 'top', 'bottom', and 'inning' XML nodes within inning_all.xml are included as `inning` and `inning_side` fields in the `pitch`, `po`, `atbat`, `runner`, and `action` tables. This helps reduce the burden of merging many tables together in order to have inning information on the play-by-play and/or pitch-by-pitch level. Other information is simply ignored simply because it is redundant. For example, the 'game' node within the players.xml file contains information that can be recovered from the `game` table extracted from the miniscoreboard.xml file. If the reader wants a more detailed explanation of fields in these tables, Marchi and Albert (2013) provide nice overview.

[2]Please be respectful of this service and store any information after you extract it instead of repeatedly querying the website. Before using any content from this website, please also read the copyright

[3]In this case, `scrape` will print "failed to load HTTP resource" in the R console (after the relevant file name) to indicate that no data was available.

Source file suffix	Information level	XML nodes	Tables returned by `scrape`
miniscoreboard.xml	game-by-game	games, game, game_media, media	game, media
players.xml	game-by-game	game, team, player, coach, umpire	player, coach, umpire
inning_all.xml	play-by-play, pitch-by-pitch	game, inning, bottom, top, atbat, po, pitch, runner, action	atbat, po, pitch, runner, action
inning_hit.xml	play-by-play	hitchart, hip	hip

Table 1: Structure of PITCHf/x and related Gameday data sources accessible to `scrape`

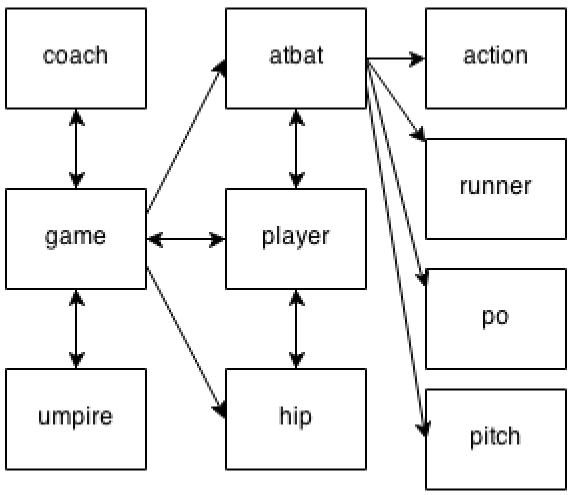

Figure 1: Table relations between Gameday data accessible via `scrape`. Direction of arrows indicate a one to possibly many relationship.

Introducing XML2R

The R package **XML2R** significantly reduces the amount of coding and cognitive effort required to extract and manipulate XML content. It was designed specifically for XML content with a relational structure in mind. That does not imply the XML content has to be relational in order for **XML2R** to be useful, but the package conventions are better suited for that case. This section shows how the scraping functionality of **pitchRx** uses ideas and tools from **XML2R** to produce a collection of tables from inning_all.xml files. A similar approach is used by `pitchRx::scrape` to construct tables from the other Gameday files in table 1. In fact, **XML2R** has also been implemented in the R package bbscrapeR which collects data from nba.com and wnba.com.

Constructing file names

Perhaps the most frustrating part of obtaining data in bulk off of the web is finding the proper collection of urls or file names of interest. Since files on the Gameday website are fairly well organized, the `makeUrls` function can be used to construct `urls` that point to every game's homepage within a window of dates.

```
urls <- makeUrls(start="2011-06-01", end="2011-06-01")
sub("http://gd2.mlb.com/components/game/mlb/", "", head(urls))
```

```
[1] "year_2011/month_06/day_01/gid_2011_06_01_anamlb_kcamlb_1"
[2] "year_2011/month_06/day_01/gid_2011_06_01_balmlb_seamlb_1"
[3] "year_2011/month_06/day_01/gid_2011_06_01_chamlb_bosmlb_1"
[4] "year_2011/month_06/day_01/gid_2011_06_01_clemlb_tormlb_1"
[5] "year_2011/month_06/day_01/gid_2011_06_01_colmlb_lanmlb_1"
[6] "year_2011/month_06/day_01/gid_2011_06_01_flomlb_arimlb_1"
```

Extracting observations

Once we have a collection of `files`, the next step is to parse XML content into a list of *observations*. An observation is technically defined as a matrix with one row and any number of columns. The columns are comprised of XML attributes and the XML value for a particular XML lineage. The name of each list element (or each observation) tracks the XML hierarchy so observations can be grouped together in a sensible fashion at a later point.

```
library(XML2R)
files <- paste0(urls, "/inning/inning_all.xml")
obs <- XML2Obs(files, url.map=TRUE, quiet=TRUE)
table(names(obs))
```

```
                                 game                          game//inning
                                   15                                   137
         game//inning//bottom//action          game//inning//bottom//atbat
                                  116                                   532
   game//inning//bottom//atbat//pitch      game//inning//bottom//atbat//po
                                 1978                                    61
  game//inning//bottom//atbat//runner             game//inning//top//action
                                  373                                   150
              game//inning//top//atbat       game//inning//top//atbat//pitch
                                  593                                  2183
          game//inning//top//atbat//po      game//inning//top//atbat//runner
                                   75                                   509
                              url_map
                                    1
```

This output tells us that 1978 pitches were thrown in the bottom inning and 2183 were thrown in the top inning on June 1st, 2011. Also, there are 12 different levels of observations. The list element named `url_map` is not considered an observation and was included since `url.map=TRUE`. This helps avoid repeating long file names in the `url_key` column which tracks the mapping between observations and file names.

```
obs[c(1, 2500)]
```

```
$`game//inning//top//atbat//pitch`
     des             id  type tfs      tfs_zulu               x        y
[1,] "Called Strike" "3" "S"  "161107" "2011-06-01T20:11:07Z" "103.00" "149.38"
     sv_id           start_speed end_speed sz_top sz_bot pfx_x   pfx_z
[1,] "110601_151108" "94.0"      "86.1"    "2.85" "1.36" "-8.12" "11.0"
     px       pz      x0       y0     z0      vx0     vy0        vz0
[1,] "-0.143" "2.376" "-2.435" "50.0" "5.831" "9.058" "-137.334" "-7.288"
     ax        ay       az        break_y break_angle break_length pitch_type
[1,] "-15.446" "31.474" "-11.175" "23.8"  "46.3"      "4.0"        "FT"
     type_confidence zone nasty spin_dir  spin_rate  cc mt url_key
[1,] ".909"          "2"  "39"  "216.336" "2753.789" "" "" "url1"

$`game//inning//bottom//atbat//runner`
     id       start end  event         url_key
[1,] "471083" "2B"  "3B" "Field Error" "url6"
```

Renaming observations

Before grouping observations into a collection tables based on their names, one may want to `re_name` observations. Observations named `'game//inning//bottom//atbat'` and `'game//inning//top//atbat'` should be included the same table since they share XML attributes (in other words, the observations share variables).

```
tmp <- re_name(obs, equiv=c("game//inning//top//atbat",
                            "game//inning//bottom//atbat"), diff.name="inning_side")
```

By passing these names to the `equiv` argument, `re_name` determines the difference in the naming scheme and suppresses that difference. In other words, observations names that match the `equiv` argument will be renamed to `'game//inning//atbat'`. The information removed from the name is not lost; however, as a new column is appended to the end of each relevant observation. For example, notice how the `inning_side` column contains the part of the name we just removed:

```
tmp[grep("game//inning//atbat", names(tmp))][1:2]

$`game//inning//atbat`
     num b   s   o   start_tfs start_tfs_zulu         batter   stand b_height
[1,] "1" "3" "2" "0" "161034"  "2011-06-01T20:10:34Z" "430947" "L"   "5-10"
     pitcher  p_throws
[1,] "462956" "R"
     des
[1,] "Erick Aybar singles on a line drive to center fielder Melky Cabrera.  "
     event    url_key inning_side
[1,] "Single" "url1"  "top"

$`game//inning//atbat`
     num b   s   o   start_tfs start_tfs_zulu         batter   stand b_height
[1,] "2" "2" "3" "1" "161412"  "2011-06-01T20:14:12Z" "110029" "L"   "6-0"
     pitcher  p_throws des                                    event
[1,] "462956" "R"      "Bobby Abreu called out on strikes.  " "Strikeout"
     url_key inning_side
[1,] "url1"  "top"
```

For similar reasons, other observation name pairs are renamed in a similar fashion.

```
tmp <- re_name(tmp, equiv=c("game//inning//top//atbat//runner",
                            "game//inning//bottom//atbat//runner"),
                diff.name="inning_side")
tmp <- re_name(tmp, equiv=c("game//inning//top//action",
                            "game//inning//bottom//action"),
                diff.name="inning_side")
tmp <- re_name(tmp, equiv=c("game//inning//top//atbat//po",
                            "game//inning//bottom//atbat//po"),
                diff.name="inning_side")
obs2 <- re_name(tmp, equiv=c("game//inning//top//atbat//pitch",
                            "game//inning//bottom//atbat//pitch"),
```

```
                diff.name="inning_side")
table(names(obs2))
```

```
                         game                 game//inning
                           15                          137
        game//inning//action          game//inning//atbat
                          266                         1125
  game//inning//atbat//pitch     game//inning//atbat//po
                         4161                          136
game//inning//atbat//runner                       url_map
                          882                            1
```

Linking observations

After all that renaming, we now have 7 different levels of observations. Let's examine observations on the game//inning level:

```
obs2[grep("^game//inning$", names(obs2))][1:3]
```

```
$`game//inning`
     num away_team home_team next url_key
[1,] "1" "ana"     "kca"     "Y"  "url1"

$`game//inning`
     num away_team home_team next url_key
[1,] "2" "ana"     "kca"     "Y"  "url1"

$`game//inning`
     num away_team home_team next url_key
[1,] "3" "ana"     "kca"     "Y"  "url1"
```

Before we grouping observations into tables, it is important preserve the parent-to-child relationships in the XML lineage. For example, one may want to map a particular pitch back to the inning in which it was thrown. Using the add_key function, the relevant value of num for game//inning observations can be recycled to its XML descendents.

```
obswkey <- add_key(obs2, parent="game//inning", recycle="num", key.name="inning")
```

```
A key for the following children will be generated for the game//inning node:
game//inning//atbat//pitch
game//inning//atbat//runner
game//inning//atbat
game//inning//action
game//inning//atbat//po
```

As it turns out, the away_team and home_team columns are redundant as this information is embeded in the url column. Thus, there is only one other informative attribute on this level which is next. By recycling this value to all of its descendants as well, we remove any need to retain a game//inning table.

```
obswkey <- add_key(obswkey, parent="game//inning", recycle="next")
```

```
A key for the following children will be generated for the game//inning node:
game//inning//atbat//pitch
game//inning//atbat//runner
game//inning//atbat
game//inning//action
game//inning//atbat//po
```

It is also imperative that we can identify which atbat a particular pitch, runner, and po belongs to. This can be done as follows:

```
obswkey <- add_key(obswkey, parent="game//inning//atbat", recycle="num")
```

```
A key for the following children will be generated for the game//inning//atbat node:
game//inning//atbat//pitch
game//inning//atbat//runner
game//inning//atbat//po
```

Collapsing observations

Finally, we are in a position to pool together observations that have a common name. The collapse_obs function achieves this by row binding observations together and returning a list of matrices. Note that collapse_obs does not require that observations from the same level to have the same set of variables in order to be binded into a common table. In the case where variables are missing, NAs will be used as the value.

```
tables <- collapse_obs(obswkey)
#As mentioned before, we don't need the 'inning' table
tables <- tables[-grep("^game//inning$", names(tables))]
table.names <- c("game", "action", "atbat", "pitch", "po", "runner")
tables <- setNames(tables, table.names)
head(tables[["runner"]])
```

```
     id       start end  event                url_key inning_side inning next
[1,] "430947" ""    "1B" "Single"             "url1"  "top"       "1"    "Y"
[2,] "430947" "1B"  "2B" "Stolen Base 2B"     "url1"  "top"       "1"    "Y"
[3,] "430947" "2B"  "3B" "Groundout"          "url1"  "top"       "1"    "Y"
[4,] "430947" "3B"  ""   "Groundout"          "url1"  "top"       "1"    "Y"
[5,] "543333" ""    "1B" "Single"             "url1"  "bottom"    "1"    "Y"
[6,] "543333" "1B"  ""   "Pickoff Attempt 1B" "url1"  "bottom"    "1"    "Y"
     num score rbi earned
[1,] "1" NA    NA  NA
[2,] "2" NA    NA  NA
[3,] "3" NA    NA  NA
[4,] "4" NA    NA  NA
[5,] "7" NA    NA  NA
[6,] "8" NA    NA  NA
```

Collecting Gameday data with pitchRx

The main scraping function in **pitchRx**, scrape, can be used to easily obtain data from the files listed in table 1. In fact, any combination of these file(s) can be queried using the suffix option. In the example below, the start and end options are also used so that all available file types for June 1st, 2011 are queried.

```
library(pitchRx)
files <- c("inning/inning_all.xml", "inning/inning_hit.xml",
                "miniscoreboard.xml", "players.xml")
dat <- scrape(start = "2011-06-01", end = "2011-06-01", suffix = files)
```

The game.ids option can be used instead of start and end to obtain an equivalent dat object. This option can be useful if the user wants to query specific games rather than all games played over a particular time span. When using this game.ids option, the built-in gids object, is quite convenient.

```
data(gids, package = "pitchRx")
gids11 <- gids[grep("2011_06_01", gids)]
head(gids11)
```

```
[1] "gid_2011_06_01_anamlb_kcamlb_1" "gid_2011_06_01_balmlb_seamlb_1"
[3] "gid_2011_06_01_chamlb_bosmlb_1" "gid_2011_06_01_clemlb_tormlb_1"
[5] "gid_2011_06_01_colmlb_lanmlb_1" "gid_2011_06_01_flomlb_arimlb_1"
```

```
dat <- scrape(game.ids = gids11, suffix = files)
```

The object dat is a list of data frames containing all data available for June 1st, 2011 using scrape. The list names match the table names provided in table 1. For example, dat$atbat is data frame with every at bat on June 1st, 2011 and dat$pitch has information related to the outcome of each pitch (including PITCHf/x parameters). Just one day's worth of data will exhaust nearly 300MB of memory. Multiply this by 100 games and this starts to exceed the RAM limitations on most machines. For this and other reasons, it is recommended that the user exploits R's database interface and related facilities on Databases (2013).

Storing and querying Gameday data

Since collecting PITCHf/x and related Gameday data can easily exhaust virtual memory, one should consider establishing a database instance before using `scrape`. By passing a database connection to the `connect` argument, `scrape` will try to create (and/or append to existing) tables using that connection. If the connection fails for some reason, tables will be written as csv files in the current working directory. The benefits of using the `connect` argument includes improved virtual memory management which can greatly reduce run time. `connect`will support a MySQL connection, but creating a SQLite database is quite easy with **dplyr** Wickham and Francois (2014).

```
library(dplyr)
my_db <- src_sqlite("GamedayDB.sqlite3", create = TRUE)
#Collect and store all PITCHf/x data from 2008 to 2013
scrape(start = "2008-01-01", end = "2014-01-01",
       suffix = "inning/inning_all.xml", connect = my_db$con)
```

Later on, in the PITCHf/x animations, four-seam and cut fastballs thrown by Mariano Rivera and Phil Hughes during the 2011 season are used to demonstrate PITCHf/x animations. In order to obtain such a data set, one must set criteria on: values of the `pitcher_name` field in the `pitch` table, values of the `des` field in the `atbat` table, and the `url` field in both tables. Thus, to speed the time to execute such a query, one should create an index on these three fields.

```
dbSendQuery(my_db$con, "CREATE INDEX url_atbat ON atbat(url)")
dbSendQuery(my_db$con, "CREATE INDEX url_pitch ON pitch(url)")
dbSendQuery(my_db$con, "CREATE INDEX pitcher_index ON atbat(pitcher_name)")
dbSendQuery(my_db$con, "CREATE INDEX des_index ON pitch(des)")
```

Although our desired query could be expressed entirely in SQL, **dplyr**'s grammar for data manipulation (which is database agnostic) can be help simply the task. First, create `pitch11` and `atbat11` which are *representations* of 2011 data in `my_db`. That is, `pitch11` does not contain actual data from every pitch thrown during 2011 into memory, but is a portrayal of the relevant data sitting in `my_db`.

```
pitch11 <- tbl(my_db, sql("SELECT * FROM pitch WHERE pitch.url LIKE '%year_2011%'"))
atbat11 <- tbl(my_db, sql("SELECT * FROM atbat WHERE atbat.url LIKE '%year_2011%'"))
```

Next, `filter` the `atbat11` table to restrict to at bats in 2011 where either Rivera or Hughes was the pitcher. Then `filter` the `pitch11` table to include only four-seam (FF) and cut (FC) fastballs from 2011. By taking an `inner_join` of these two filtered tables, we are left with one table representing the data of interest. Lastly, `collect` the resulting database query in order to bring the actual data into the R session.

```
bats <- filter(atbat11, pitcher_name == "Mariano Rivera" | pitcher_name == "Phil Hughes")
FBs <- filter(pitch11, pitch_type == "FF" | pitch_type == "FC")
pitches <- collect(inner_join(FBs, bats))
```

Visualizing PITCHf/x

Strike-zone plots and umpire bias

Amongst the most common PITCHf/x graphics are strike-zone plots. Such a plot has two axes and the coordinates represent the location of baseballs as they cross home plate. The term strike-zone plot is used here to refer to both pitch *density* and event *probability* plots. Thanks to their nice interpretation, event probability plots can address much more interesting questions. To demonstrate, all available pitches from 2008 to 2013 were obtained using `scrapeFX`. Strike-zone plots of this data are used to explore the strike-zone and also address whether umpires are biased toward home pitchers.

In the world of sports, it is a common belief that umpires (or referees) have a tendency to favor the home team. PITCHf/x provides a unique opportunity to validate such a hypothesis by looking at the probability of a home pitcher receiving a called strike at a specific location minus the probability of an away pitcher receiving a called strike at that same location (given the umpire has to make the decision between strike or ball). There are many different possible outcomes of each pitch, but we can condition on the umpire making a decision by limiting to the following two cases. A *called strike* is an outcome of a pitch where the batter does not swing and the umpire declares the pitch a strike (which is a favorable outcome for the pitcher). A *ball* is another outcome where the batter doesn't swing and the umpire declares the pitch a ball (which is a favorable outcome for the batter). Data from every called strike and ball from 2008 to 2013 is saved as a data frame named decisions. The following sections use this data to showcase strikeFX's capability to generate frequency and probability plots while providing evidence for our umpire bias hypothesis.

Density plots

The decisions data frame contains data on over 2.5 million pitches thrown from 2008 to 2013. About a third of them are called strikes and two-thirds balls. Figure 2 shows the density of all called strikes. Clearly, most called strikes occur on the outer region of the strike-zone. Many factors could contribute to this phenomenon; which we won't investigate here.

```
decisions$strike <- as.numeric(decisions$des %in% "Called Strike")
strikes <- subset(decisions, strike == 1)
strikeFX(strikes, geom="tile", layer=facet_grid(.~stand, labeller = label_both))
```

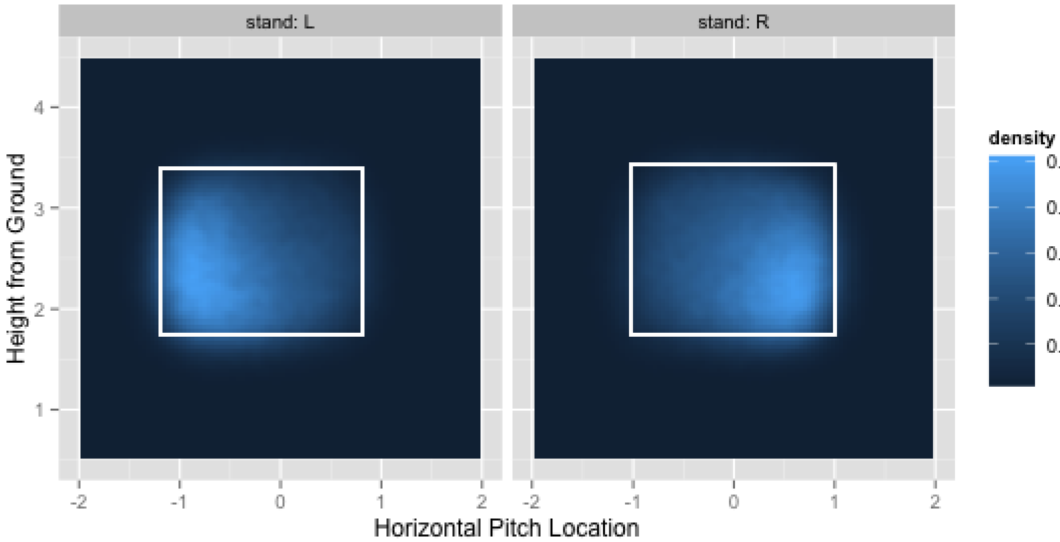

Figure 2: Density of called strikes for right-handed batters and left-handed batters (from 2008 to 2013).

Figure 2 shows one static rectangle (or strike-zone) per plot automatically generated by strikeFX. The definition of the strike-zone is notoriously ambiguous. As a result, the boundaries of the strike-zone may be noticeably different in some situations. However, we can achieve a fairly accurate representation of strike-zones using a rectangle defined by batters' average height and stance Fast (2011). As figure 4 reinforces, batter stance makes an important difference since the strike-zone seems to be horizontally shifted away from the batter. The batter's height is also important since the strike-zone is classically defined as approximately between the batter's knees and armpits.

Figure 2 has is one strike-zone per plot since the layer option contains a **ggplot2** argument that facets according to batter stance. Facet layers are a powerful tool for analyzing PITCHf/x because they help produce quick and insightful comparisons. In addition to using the layer option, one can add layers to a graphic returned by strikeFX using **ggplot2** arithmetic. It is also worth pointing out that figure 2 could have been created without introducing the strikes data frame by using the density1 and density2 options.

```
strikeFX(decisions, geom="tile", density1=list(des="Called Strike"),
         density2=list(des="Called Strike"))+facet_grid(.~stand)
```

In general, when density1 and density2 are identical, the result is equivalent to subsetting the data frame appropriately beforehand. More importantly, by specifying *different* values for density1 and density2, differenced densities are easily generated. In this case, a grid of density estimates for density2 are subtracted from the corresponding grid of density estimates for density1. Note that the default NULL value for either density option infers that the entire data set defines the relevant density. Thus, if density2 was NULL (when density1=list(des='Called Strike')), we would obtain the density of called strikes minus the density of *both* called strikes and balls. In figure 3, we define density1 as called strikes and define density2 as balls. As expected, we see positive density values (in blue) inside the strike-zone and negative density values (in red) outside of the strike-zone.

```
strikeFX(decisions, geom="tile", density1=list(des="Called Strike"),
        density2=list(des="Ball"), layer=facet_grid(.~stand, labeller = label_both))
```

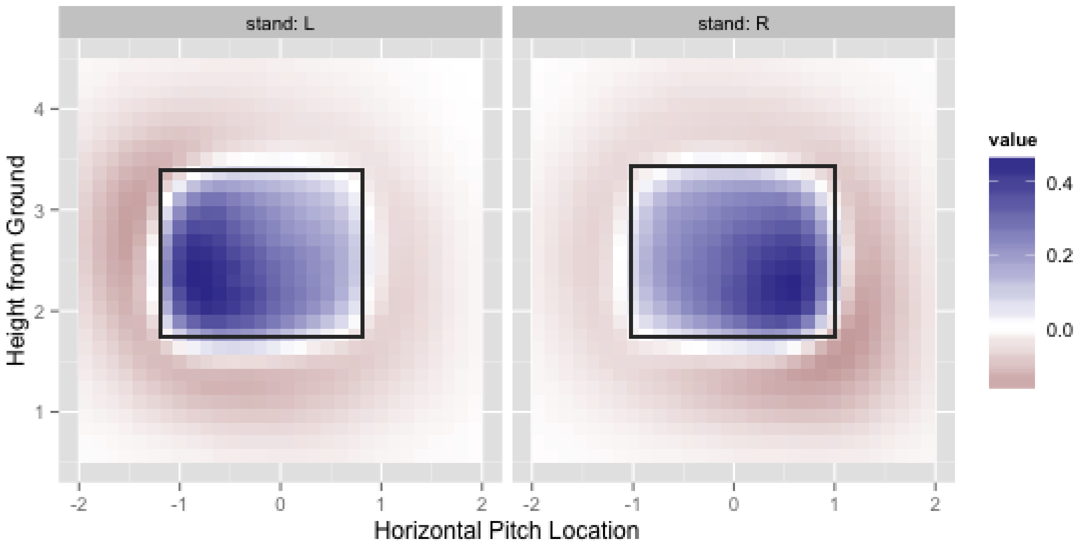

Figure 3: Density of called strikes minus density of balls for both right-handed batters and left-handed batters (from 2008 to 2013). The blue region indicates a higher frequency of called strikes and the red region indicates a higher frequency of balls.

These density plots are helpful for visualizing the observed frequency of events; however, they aren't very useful for addressing our umpire bias hypothesis. Instead of looking simply at the *density*, we want to model the *probability* of a strike called at each coordinate given the umpire has to make a decision.

Probabilistic plots

There are many ways to approach probabilistic modeling over a two dimensional spatial region. Since this data is binomial (each pitch is either a called strike or ball), generalized additive models (GAMs) are perhaps most appropriate for modeling the probability of a certain event at each coordinate Mills (2010). There are numerous R package implementations of GAMs, but the bam function from the **mgcv** package has several desirable properties Wood (2006). Most importantly, the smoothing parameter can be estimated using several different methods. In order to have a reasonable estimate of the smooth 2D surface, GAMs require fairly large amount of observations. As a result, run time can be an issue - especially when modeling 2.5 million observations! Thankfully, the bam function has a cluster option which allows one to distribute computations across multiple cores using the **parallel** package R Core Team (2013).

```
library(parallel)
cl <- makeCluster(detectCores()-1)
library(mgcv)
m <- bam(strike ~ interaction(stand, p_throws, inning_side) +
                        s(px, pz, by=interaction(stand, p_throws, inning_side)),
                        data=decisions, family = binomial(link='logit'), cluster=cl)
```

This formula models the probability of a strike as a function of the batter's stance, the pitcher's throwing arm, and the side of the inning. Since home pitchers always pitch during the top of the inning, `inning_side` serves as an indication of whether the pitch is thrown by a home pitcher. Note that `interaction(stand,p_throws,inning_side)` yields a factor with 8 different levels. As a result, there are 8 different levels of smooth surfaces over the spatial region defined by `px` and `pz`. This model may not be optimal for explaining the probability of a called strike, but it serves as a nice example to demonstrate different options for visualizing these surfaces using `strikeFX`.

The `gamObject`, `m`, contains a lot of information which `strikeFX` uses in conjunction with any existing facet commands to infer which surfaces should be plotted. In particular, the `var.summary` value of `m` is used to identify the covariates as well as what value should be used in case any have to be conditioned upon. In our case, the majority of `decisions` are from right-handed pitchers and the top of the inning. Thus, the default conditioning values are `"top"` for `inning_side` and `"R"` for `p_throws`. If different conditioning values are desired, `var.summary` can be modified accordingly. To demonstrate, figure 4 shows 2 of the 8 possible surfaces that correspond to a right-handed *away* pitcher.

```
away <- list(inning_side = factor("bottom", levels=c("top", "bottom")))
m$var.summary <- modifyList(m$var.summary, away)
strikeFX(decisions, model=m, layer=facet_grid(.~stand, labeller = label_both))
```

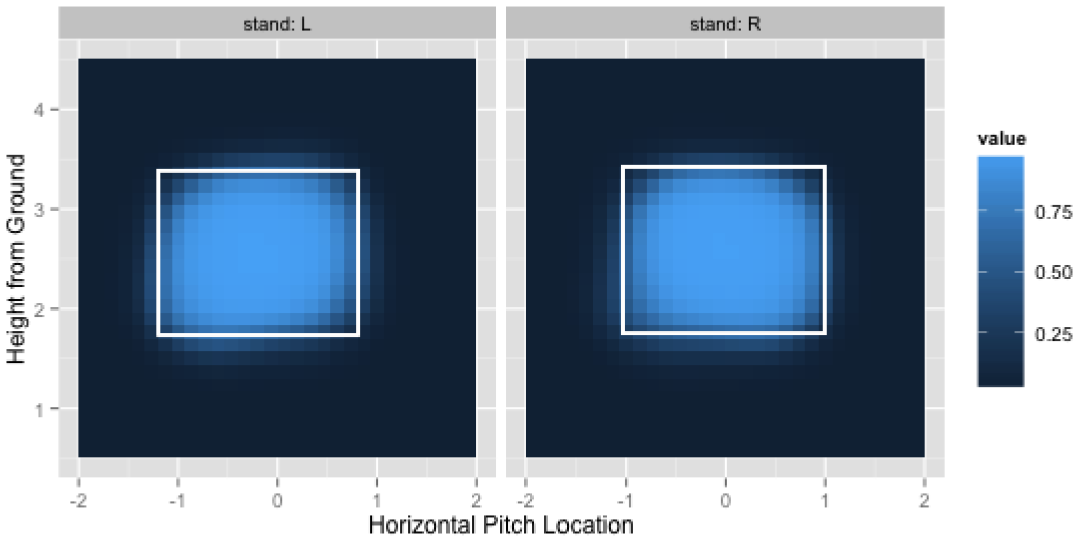

Figure 4: Probability that a right-handed away pitcher receives a called strike (provided the umpire has to make a decision). Plots are faceted by the handedness of the batter.

Using the same intuition exploited earlier to obtain differenced density plots, we can easily obtain differenced probability plots. To obtain figure 5, we simply add `p_throws` as another facet variable and `inning_side` as a differencing variable. In this case, conditioning values do not matter since every one of the 8 surfaces are required in order to produce figure 5.

```
strikeFX(decisions, model=m, layer=facet_grid(p_throws~stand, labeller = label_both),
                 density1=list(inning_side="top"), density2=list(inning_side="bottom"))
```

The four different plots in figure 5 represent the four different combination of values between `p_throws` and `stand`. In general, provided that a pitcher throws to a batter in the blue region, the pitch is more likely to be called a strike if the pitcher is on their home turf. Interestingly, there is a well-defined blue elliptical band around the boundaries of the typical strike-zone. Thus, home pitchers are more likely to receive a favorable call - especially when the classification of the pitch is in question. In some situations, the home pitcher has up to a 6 percent higher probability of receiving a called strike than an away pitcher. The subtle differences in spatial patterns across the different values of `p_throws` and `stand` are interesting as well. For instance, pitching at home has a large positive impact for a right-handed pitcher throwing in the lower inside portion of the strike-zone to a left-handed batter,

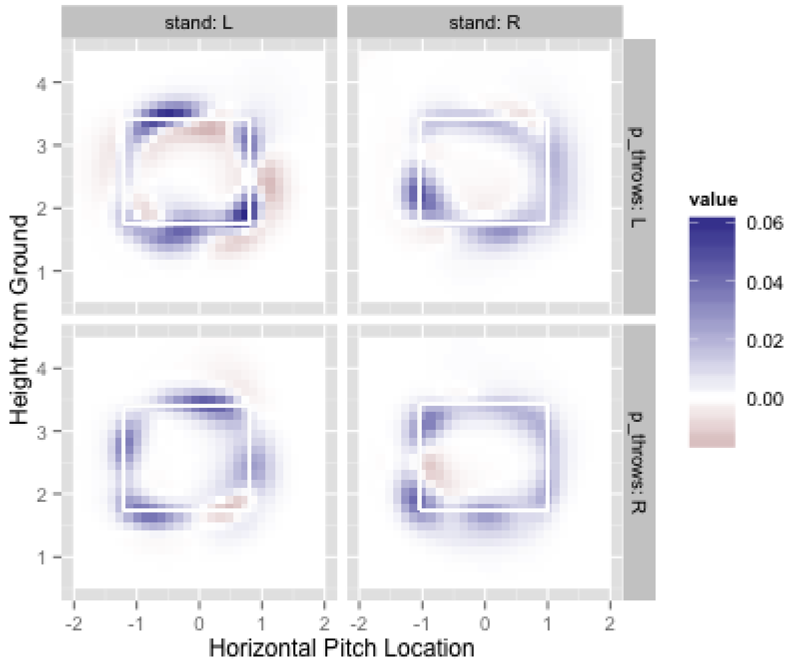

Figure 5: Difference between home and away pitchers in the probability of a strike (provided the umpire has to make a decision). The blue regions indicate a higher probability of a strike for home pitchers and red regions indicate a higher probability of a strike for away pitchers. Plots are faceted by the handedness of both the pitcher and the batter.

but the impact seems negligible in the mirror opposite case (i.e., right-handed pitcher and left-handed batter).

Differenced probabilistic densities are clearly an interesting visual tool for analyzing PITCHf/x data. With `strikeFX`, one can quickly and easily make all sorts of visual comparisons for various situations. In fact, one can explore and compare the probabilistic structure of any well-defined event over a strike-zone region (for example, the probability a batter reaches base) using a similar approach.

2D animation

`animateFX` provides convenient and flexible functionality for animating the trajectory of any desired set of pitches. For demonstration purposes, this section animates every four-seam and cut fastball thrown by Mariano Rivera and Phil Hughes during the 2011 season. These pitches provide a good example of how facets plays an important role in extracting new insights. Similar methods can be used to analyze any MLB player (or combination of players) in greater detail.

`animateFX` tracks three dimensional pitch locations over a sequence of two dimensional plots. The animation takes on the viewpoint of the umpire; that is, each time the plot refreshes, the balls are getting closer to the viewer. This is reflected with the increase in size of the points as the animation progresses. Obviously, some pitches travel faster than others, which explains the different sizes within the same frame. Note that animations in this paper revert to the initial point of release once *all* of the baseballs have reached home plate. During an interactive session, `animateFX` produces a series of plots that may not viewed easily. One option available to the user is to wrap `animation::saveHTML` around `animateFX` to view the animation in a browser with proper looping controls Xie (2013a).

To reduce the time and thinking required to produce these animations, `animateFX` has default settings for the geometry, color, opacity and size associated with each plot. Any of these assumptions can be altered - except for the point geometry. In order for animations to work, a data frame with the appropriately named PITCHf/x parameters (that is, x0, y0, z0, vx0, vy0, vz0, ax0, ay0 and az0) is required. In figure 6, the data frame `pitches` with data on every four-seam and cut fastball thrown by Rivera and Hughes during the 2011 season is used.

The upper right-hand plot of figure 6 (Rivera throwing to right-handed batters) reveals the clearest pattern in flight trajectories. Around the point of release, Rivera's two pitch types are hard to distinguish. However, after a certain point, there is a very different flight path among the two pitch types. Specifically, the drastic left-to-right movement of the cut fastball is noticeably different from the slight right-to-left movement of the four-seam fastball. In recent years, cut fastballs have gained notoriety among the baseball community as a coveted pitch for pitchers have at their disposal. This is largely due to the difficulty that a batter has in distinguishing the cut fastball from another fastball as the ball travels toward home plate. Clearly, this presents an advantage for the pitcher since they can use deception to reduce batter's ability to predict where the ball will cross home plate. This deception factor combined with Rivera's ability to locate his pitches explain his accolades as one of the greatest pitchers of all time Traub (2010).

Although we see a clear pattern in Rivera's pitches, MLB pitchers are hardly ever that predictable. Animating that many pitches for another pitcher can produce a very cluttered graphic which is hard to interpret (especially when many pitch types are considered). However, we may still want to obtain an indication of pitch trajectory over a set of many pitches. A way to achieve this is to average over the PITCHf/x parameters to produce an overall sense of pitch type behavior (via the `avg.by` option). Note that the facet variables are automatically considered indexing variables. That is, in figure 7, there are eight 'average' pitches since there are two pitch types, two pitchers, and two types of batting stance.

Interactive 3D graphics

rgl is an R package that utilizes OpenGL for graphics rendering Adler et al.. `interactiveFX` utilizes **rgl** functionality to reproduce flight paths on an interactive 3D platform. Figure 8 has two static pictures of Mariano Rivera's 2011 fastballs on this interactive platform. This is great for gaining new perspectives on a certain set of pitches, since the trajectories can be viewed from any angle. Figure 8 showcases the difference in trajectory between Rivera's pitch types.

```
Rivera <- subset(pitches, pitcher_name == "Mariano Rivera")
interactiveFX(Rivera, avg.by="pitch_types")
```

Conclusion

pitchRx utilizes **XML2R**'s convenient framework for manipulating XML content in order to provide easy access to PITCHf/x and related Gameday data. **pitchRx** removes access barriers which allows the average R user and baseball fan to spend their valuable time analyzing Gameday's enormous source of baseball information. **pitchRx** also provides a suite of functions that greatly reduce the amount of work involved to create popular PITCHf/x graphics. For those interested in obtaining other XML data, **pitchRx** serves as a nice example of leveraging **XML2R** to quickly assemble custom XML data collection mechanisms.

Acknowledgements

Many thanks to my major professor, Dr. Heike Hofmann, for her direction and support throughout this project. Thanks also to the anonymous reviewers for helpful feedback.

Bibliography

D. Adler, D. Murdoch, and others. *rgl: 3D visualization device system (OpenGL)*. URL http://rgl.neoscientists.org. R package version 0.93.963. [p]

A. Alt and M. S. White. Tracking an object with multiple asynchronous cameras, 09 2008. URL http://www.patentlens.net/patentlens/patent/US_7062320/. [p]

N. DiMeo. Baseball's particle accelerator, 2007. URL http://www.slate.com/articles/sports/sports_nut/2007/08/baseballs_particle_accelerator.html. Last visited on 07/12/2012. [p]

M. Fast. How to build a pitch database, 2007. URL http://fastballs.wordpress.com/2007/08/23/how-to-build-a-pitch-database/. Last visited on 12/15/2012. [p]

M. Fast. Spinning yarn: A zone of their own, 2011. URL http://www.baseballprospectus.com/article.php?articleid=14572. Last visited on 06/17/2013. [p]

M. Marchi and J. Albert. *Analyzing Baseball Data with R*. Chapman and Hall/CRC, 2013. ISBN 9781466570221. URL http://baseballwithr.wordpress.com/. [p]

B. Mills. Rethinking 'loess' for binomial-response pitch f/x strike zone maps, 2010. URL http://princeofslides.blogspot.com/2010/12/rethinking-loess-for-binomial-response.html. Last visited on 06/20/2013. [p]

A. Nathan. A statistical study of PITCHf/x pitched baseball trajectories, Feburary 2008. URL http://webusers.npl.illinois.edu/~a-nathan/pob/MCAnalysis.pdf. Last visited on 07/25/2012. [p]

R. S. I. G. on Databases. *DBI: R Database Interface*, 2013. URL http://CRAN.R-project.org/package=DBI. R package version 0.2-7. [p]

M. A. Pane, S. L. Ventura, R. C. Steorts, and A. C. Thomas. Trouble With The Curve: Improving MLB Pitch Classification. *ArXiv e-prints*, Apr. 2013. [p]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL http://www.R-project.org/. [p]

C. Sievert. *XML2R: EasieR XML data collection*, 2014a. URL http://cpsievert.github.com/XML2R. R package version 0.0.4. [p]

C. Sievert. *pitchRx: Tools for Scraping XML Data and Visualizing Major League Baseball PITCHf/x*, 2014b. URL http://cpsievert.github.com/pitchRx. R package version 1.0. [p]

J. Traub. Mariano rivera, king of the closers, 2010. URL http://www.nytimes.com/2010/07/04/magazine/04Rivera-t.html. Last visited on 02/04/2013. [p]

H. Wickham. *ggplot2: elegant graphics for data analysis.* Springer New York, 2009. ISBN 978-0-387-98140-6. URL http://had.co.nz/ggplot2/book. [p]

H. Wickham and R. Francois. *dplyr: dplyr: a grammar of data manipulation*, 2014. R package version 0.1. [p]

S. Wood. *Generalized Additive Models: An Introduction with R.* Chapman and Hall/CRC, 2006. [p]

Y. Xie. animation: An R package for creating animations and demonstrating statistical methods. *Journal of Statistical Software*, 53(1):1–27, 2013a. URL http://www.jstatsoft.org/v53/i01/. [p]

Y. Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, 2013b. ISBN 978-1482203530. [p]

Carson Sievert
Department of Statistics
Iowa State University
sievert@iastate.edu

```
animateFX(pitches, layer=list(coord_equal(), theme_bw(),
        facet_grid(pitcher_name~stand, labeller = label_both)))
```

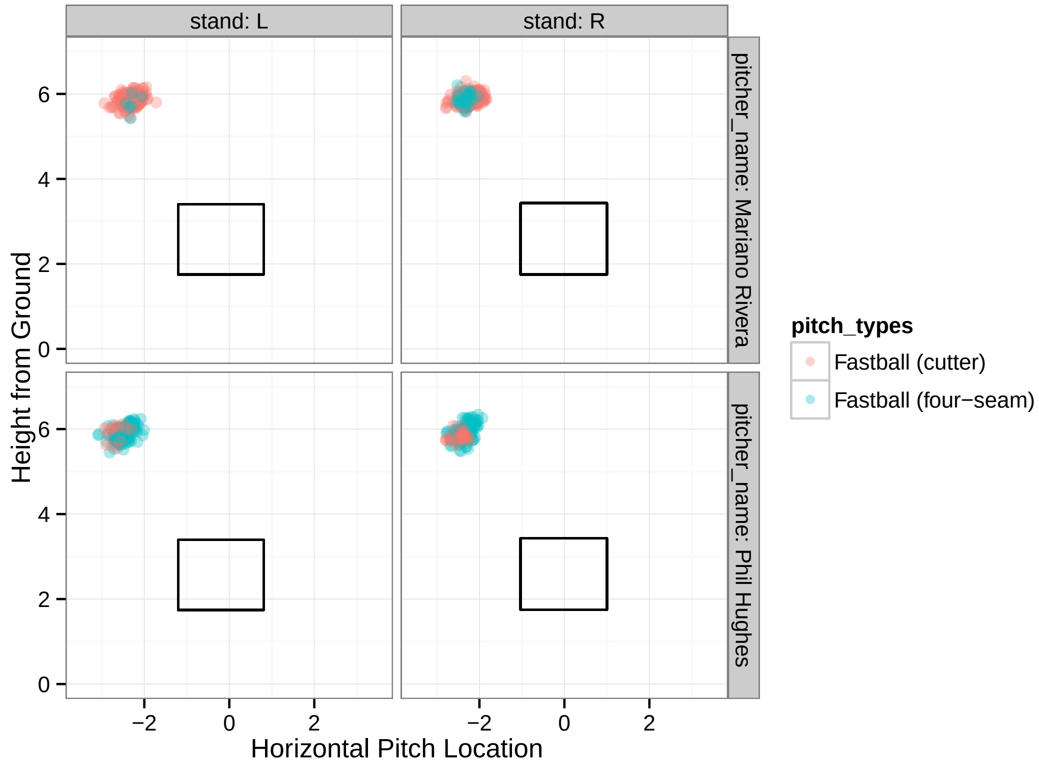

Figure 6: Animation of every four-seam and cutting fastballs thrown by NY Yankee pitchers Mariano Rivera and Phil Hughes during the 2011 season. Pitches are faceted by pitcher and batting stance. For instance, the top left plot portrays pitches thrown by Rivera to left-handed batters. Animations are intentionally slower than real-time for visual recognition and should be viewed within Abode Reader. Xie (2013b)

```
animateFX(pitches, avg.by="pitch_types", layer=list(coord_equal(), theme_bw(),
        facet_grid(pitcher_name~stand, labeller = label_both)))
```

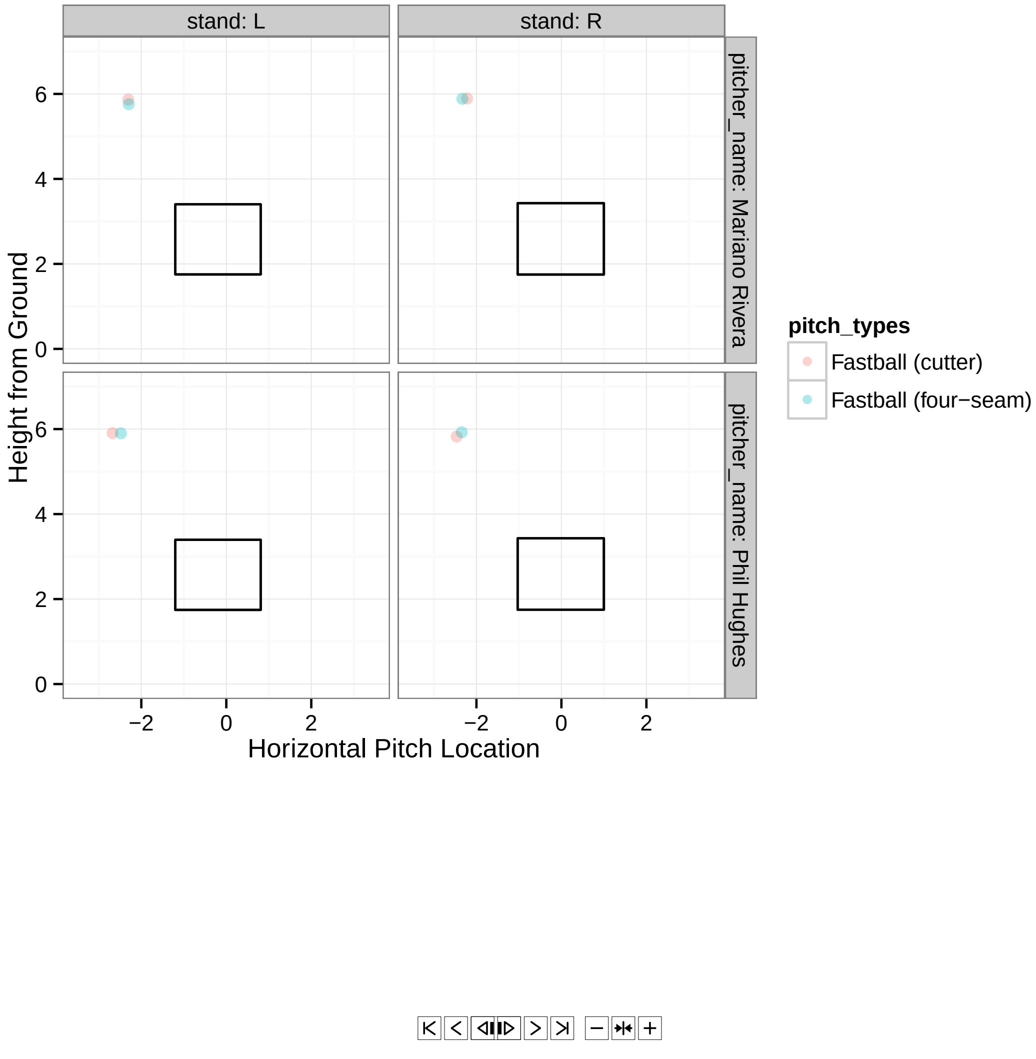

Figure 7: Animation of 'average' four-seam and cutting fastballs thrown by NY Yankee pitchers Mariano Rivera and Phil Hughes during the 2011 season. PITCHf/x parameters are averaged over pitch type, pitcher and batting stance. For instance, the bottom right plot portrays an "average four-seam" and "average cutter" thrown by Hughes to right-handed batters. Animations are intentionally slower than real-time for visual recognition and should be viewed within Abode Reader.

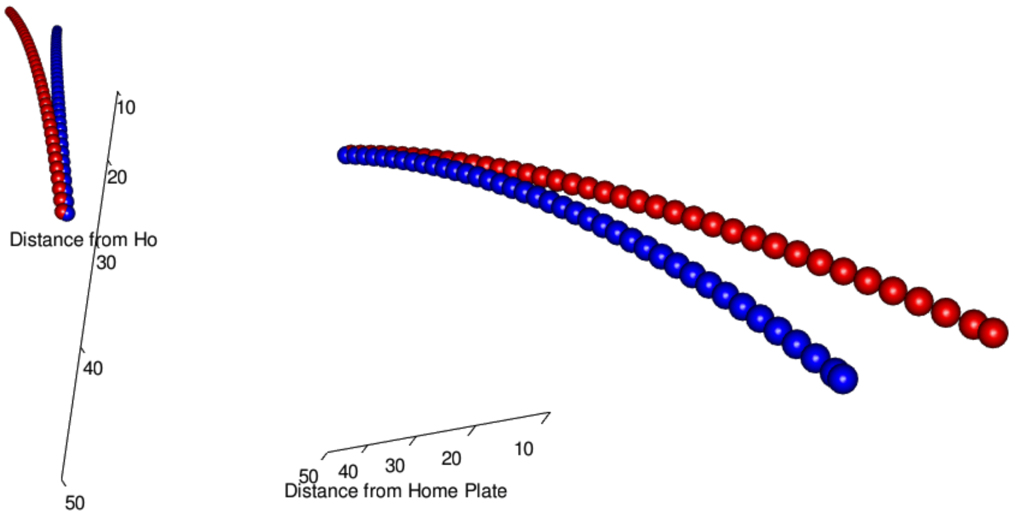

Figure 8: 3D scatterplot of pitches from Rivera. Pitches are plotted every one-hundredth of a second. Cutting fastballs are shown in red and four-seam fastballs are shown in blue. The left hand plot takes a viewpoint of Rivera and the right hand plot takes a viewpoint near the umpire. Note these are static pictures of an interactive object.