EPV Demo
Supplement to "A Multiresolution Stochastic Process Model for Predicting Basketball Possession Outcomes"

Daniel Cervone, Alex D'Amour, Luke Bornn and Kirk Goldsberry

This document provides a demonstration of the code, methodology, and inferential results for the EPV model discussed in our paper.

1 Loading the data

To begin, we must first set the directories containing the supplemental data and code, and install/load all necessary packages.

```
code.dir <- "./code"
data.dir <- "./data"
```

Now we load the `csv` file containing a full game of optical tracking data. As mentioned in the paper, data from this game was not used in parameter inference for any model related to EPV.

```
dat <- read.csv(file=sprintf("%s/2013_11_01_MIA_BKN.csv", data.dir))
```

Each row of `dat` represents a time point (sampled 25 times per second), and columns include

Column	Value	Notes
`time`	Real time (ms)	
`game`	Game ID	
`quarter`	Quarter	
`shot_clock`	Time remaining on shot clock	`NA` for this game
`game_clock`	Time remaining in quarter (s)	
`x, y, z`	Ball position (ft)	Court region is $[0, 94] \times [0, 50]$
`a1_ent`	ID number of player 1 on away team (`a1`)	
`a1_x, a1_y`	Position of `a1`	
`a1_event`	Event code for player `a1`	See Table 5 for reference
`a#_*, h#_*`	As for `a1`	

Table 1: Description of variables in optical tracking data sample.

Let's plot the data for some arbitrary moment in the game in Figure 1.

```
source(sprintf("%s/constants.R", code.dir)) # loads libraries and constants used throughout code
source(sprintf("%s/graphics.R", code.dir))  # graphics/plotting functions
par(mar=c(0, 0, 0, 0))
data.plotter(dat, 1800)
```

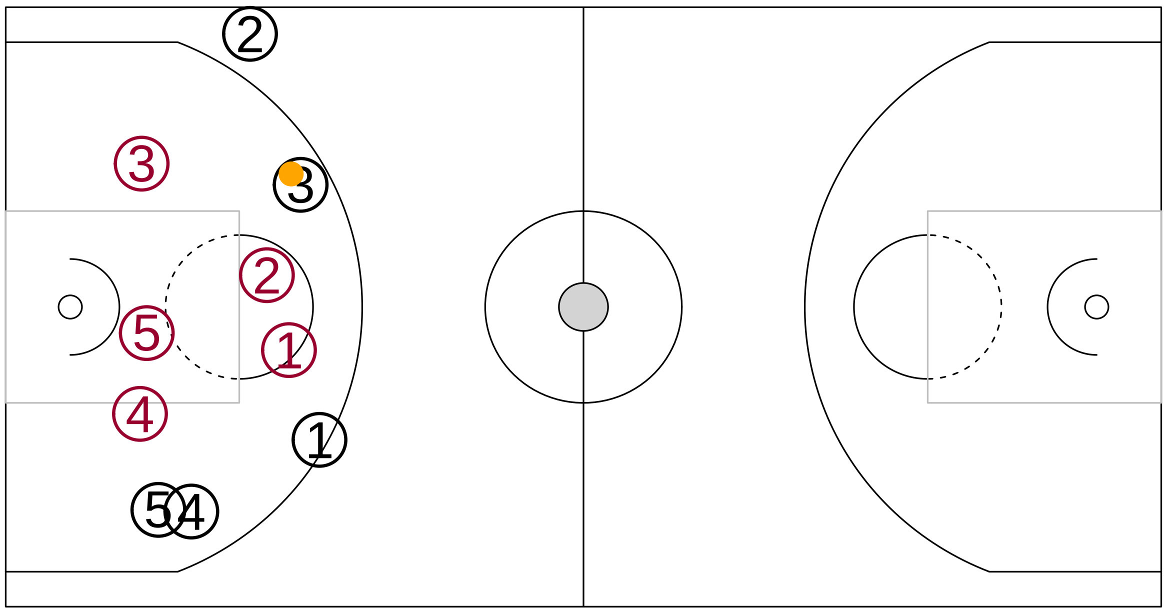

Figure 1: Plotting a single moment of optical tracking data.

1.1 Transformed data

In this format, the data lacks information necessary for computing EPV. Most importantly, the identity of the ballcarrier is not labeled, and most be inferred by the record of game actions (and positional data). We also need to record the covariates used by our multiresolution transition models, and perform some simple data manipulations, such as rotating all data to the offensive half-court and removing moments where the gameplay is suspended. The following code performs these data tasks:

```
source(sprintf("%s/data_formatting.R", code.dir))
source(sprintf("%s/covariates.R", code.dir))

poss <- possession.indicator(dat) # infer ballcarrier... takes about a minute
tdat <- rearrange.data(dat, poss) # re-shuffle columns by to ballcarrier... (2 min)
tdat <- offensive.halfcourt(tdat) # transforming to offensive halfcourt
tdat <- offensive.ballcarrier(tdat)
touchID <- get.touchID(tdat)
covariates <- getAllCovars(tdat) # get covariates... (3 min)
tdat <- data.frame(tdat, touchID=touchID, covariates)
save(tdat, file=sprintf("%s/tdat.Rdata", data.dir))
```

Or, since this takes few minutes to complete, it may be easier to load a pre-computed version of the transformed data set, `tdat`:

```
load(sprintf("%s/tdat.Rdata", data.dir))
```

2 Components of hierarchical models

2.1 Player similarity adjacency matrix, H

The hierarchical models used to estimate parameters for the multiresolution transition models rely on preprocessed data summaries. First, the conditional autoregressive model priors used for many model parameters rely on a graph **H** of player similarity, represented using an adjacency matrix. As discussed in the paper, this graph is constructed based on the similarity in players' court occupancy distributions. We can visualize these court occupancy distributions, as well as the similarity scores we calculate between them.

```
load(sprintf("%s/playerbases.Rdata", data.dir))
players <- read.csv(sprintf("%s/players2013.csv", data.dir))
head(players)

##   player_id firstname lastname       position height weight byear rookie position.number
## 1      3306     Elton    Brand Forward-Center     81    254  1979   1999               8
## 2     58293      Kyle   Korver  Guard-Forward     79    212  1981   2003               4
## 3    292401       Lou Williams          Guard     73    175  1986   2005               2
## 4    237675      Paul  Millsap Forward-Center     80    258  1985   2006               8
## 5    280587        Al  Horford Forward-Center     82    250  1986   2007               8
## 6    398043      Jeff   Teague    Point-Guard     74    181  1988   2009               1
```

`players` is a directory of the 461 NBA players in the 2013-14 season, and `playerbases.Rdata` contains summaries of their court occupancy patterns. `df` is the matrix **G** from the paper: plotting its rows reveals stark differences in players' spatial occupancy patterns:

```
par(mfrow=c(1,5))
for(i in 1:5)
  spatialPlot0(df[i, ], legend=F)
```

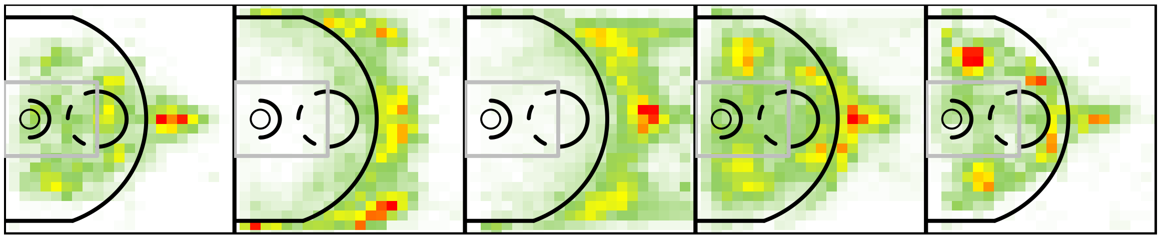

Figure 2: Court occupancy distributions.

In the paper, we use non-negative matrix factorization to obtain a rank 5 approximation of the court occupancy distribution matrix. The basis surfaces of this approximation, given in Figure 8 of the paper, are reproduced here:

```
par(mfrow=c(1,5))
for(i in 1:5)
  spatialPlot0(nmf.basis[i, ], legend=F)
```

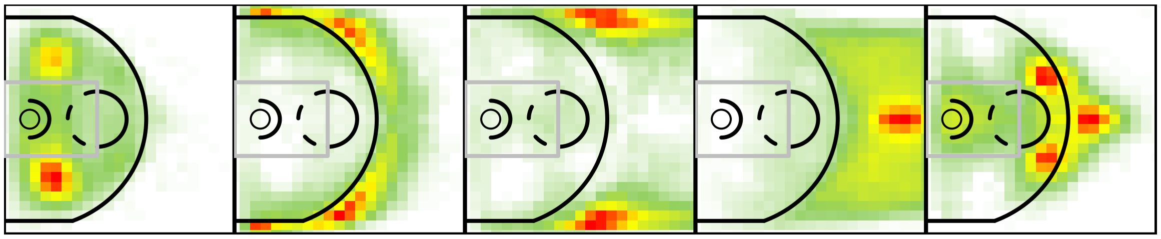

Figure 3: Court occupancy distribution bases.

Projected onto this basis, the court occupancy distributions shown in Figure 2 look like:

```
df.lowrank <- nmf.coef %*% nmf.basis
par(mfrow=c(1,5))
for(i in 1:5)
  spatialPlot0(df.lowrank[i, ], legend=F)
```

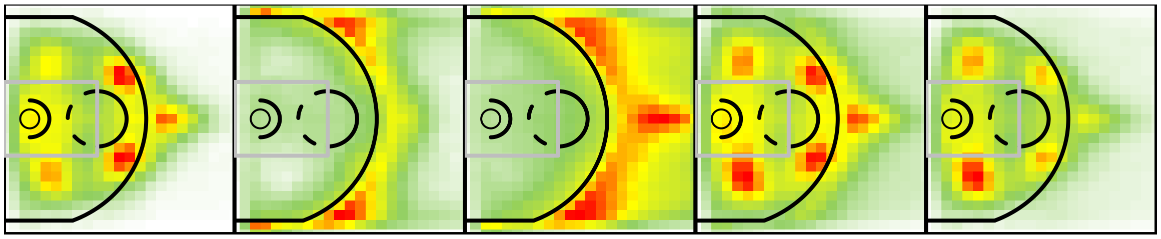

Figure 4: Low rank court occupancy distributions for players shown in Figure 2.

It's better to compute player similarity using distance in the space of basis loadings, rather than the original court occupancy distributions, as such distances are calculated across axes that best describe player variation. We calculate `K`, a distance matrix comparing the loadings for the court occupancy distributions of all 461 players, then map this to a symmetric adjacency matrix `H` based on finding each player's closest eight neighbors:

```
K <- matrix(NA, nrow=nrow(df), ncol=nrow(df))
for(i in 1:nrow(K)){
  this.coef <- nmf.coef[i, ] / sum(nmf.coef[i, ])
  K[i, ] <- apply(nmf.coef, 1, function(r) sum((r / sum(r) - this.coef)^2))
}
H <- 0 * K
for(i in 1:nrow(H)){
  inds <- order(K[i, ])[1:8 + 1]
  H[i,inds] <- H[inds, i] <- 1
}
```

To check any player's "neighbors" according to `H`, we can do (for Al Horford):

```
this.player <- grep("Horford", players$lastname)
paste(players$firstname, players$lastname)[which(H[this.player, ] == 1)]

##  [1] "Brandon Bass"      "J.J. Hickson"      "Andre Drummond"    "Tony Mitchell"
##  [5] "David Lee"         "Dwight Howard"     "Blake Griffin"     "Zach Randolph"
##  [9] "Anthony Davis"     "Amar'e Stoudemire" "Jason Maxiell"     "Glen Davis"
## [13] "DeMarcus Cousins"  "Jonas Valanciunas" "Enes Kanter"
```

2.2 Spatial effect basis functions

Similarly, let's load the basis functions that are used in representing the spatial effects in players' macrotransition entry models: we denote these basis functions ϕ_{ji}, where $i = 1, \dots, 10$, and j indexes shot-taking, four different pass options, and turnovers (recall that for the spatial effects in the shot probability model (Equation 10 in the paper), we use the same basis functions as we do for the shot-taking hazard model). To recreate Figure 6 of the paper, which plots the shot-taking bases, we'd do:

```
par(mfrow = c(2,5))
for(i in 1:10)
  spatialPlot1(take.basis[i, ], legend=F)
```

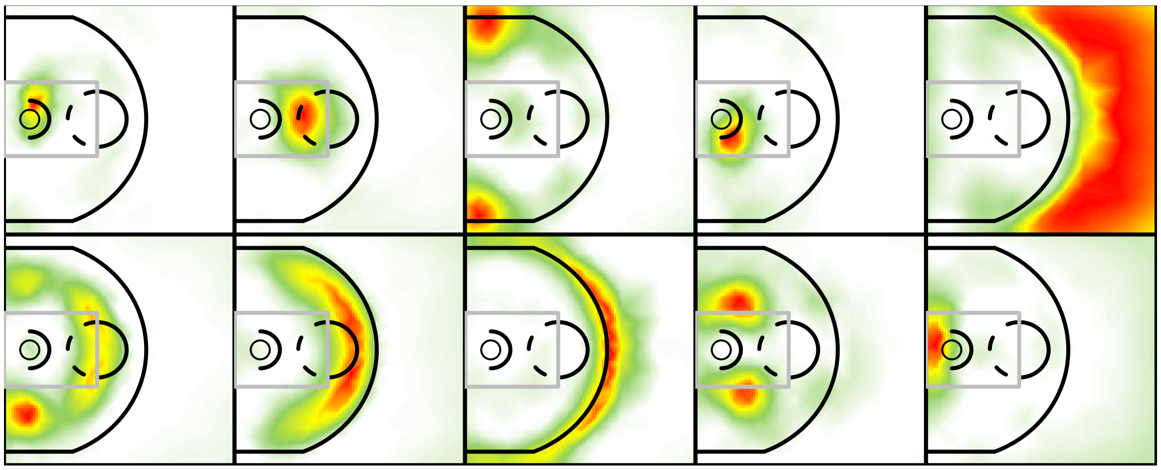

Figure 5: Shot-taking spatial bases; this plot is the same as Figure 6 of the paper (though the ordering is different).

3 Loading parameters and model estimates

3.1 Microtransition model

Here, we will load and illustrate the results of the multiresolution transition models discussed in Section 3 of the paper. First, let's load the (offensive) microtransition model output for LeBron James, print the parameter estimates, and plot of the acceleration effects μ_x^ℓ, μ_y^ℓ, as in Figure 4 of the paper.

```
player.id <- players$player_id[which(players$firstname == "LeBron")]
load(sprintf("%s/micros/%s.Rdata", data.dir, player.id))
# x component of LeBron James' micro model during ball possession
xtable(with.ball$io.x$summary.fixed[, 1:5])
```

	mean	sd	0.025quant	0.5quant	0.975quant
dif	0.98	0.00	0.98	0.98	0.98
intercept	0.00	0.01	-0.03	0.00	0.03

```
par(mfrow=c(1,2), mar=c(0,0,0,0))
vectorPlot(with.ball)
vectorPlot(without.ball)
```

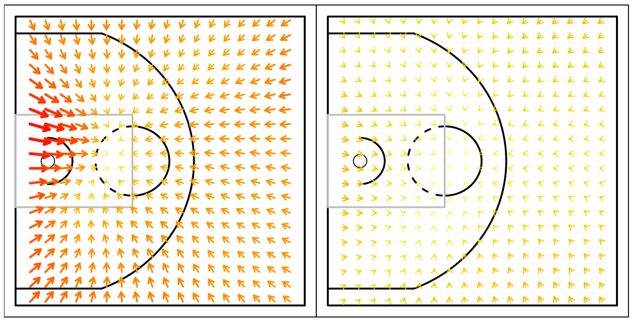

Figure 6: Plots of acceleration effect for LeBron James' offensive microtransition model.

The defensive microtransition model is less complicated, and we can fit it very quickly. The code below estimates the same model parameters for all players on defense:

```
source(sprintf("%s/parameters.R", code.dir)) # loads many modeling functions
def.micro <- microDefModel(tdat)
# coefficients are a_x, c_x, and b_x from Equation 6 in paper
xtable(summary(def.micro$mod.x)$coef[, 1:3])
```

	Estimate	Std. Error	t value
(Intercept)	-0.00	0.00	-29.74
def.eps.x[-length(def.eps.x)]	0.96	0.00	1315.54
residual.x[-length(residual.x)]	-0.00	0.00	-27.12
opt.eps.x[-length(opt.eps.x)]	0.00	0.00	3.99

3.2 Macrotransition entry models

We have six macrotransition entry models (from Section 3.2 of the paper). Each is fit hierarchically for all players in the NBA using the R-INLA software, as discussed in Section 4 of the paper. Let's load the results of the shot-taking macrotransition entry model, and interpret some of the results.

```
load(sprintf("%s/INLA_TAKE.Rdata", data.dir))
# coefficients for time-varying covariates in shot-taking hazard model
xtable(inla.out$summary.fixed[, 1:2])
```

`b1` is the coefficient for the loading on the first basis function (Figure 5). These are fixed effects, so that player-specific coefficient values are represented as random effects. Parameter inference for the random effects are presented somewhat confusingly in the output from R-INLA. Inference for random effects on the situational covariates are stored in matrices where rows represent different players. For instance, for Chris Bosh, we get the mean, SD, and quantiles of his player-specific `dribble` parameter[1] by running:

```
this.player <- grep("Bosh", players$lastname)
xtable(inla.out$summary.random$p.dribble[this.player, 2:6])
```

However, the random effects on the spatial basis coefficients are stacked in a $(1+10)\times 461$ matrix (there are 461 players in our full NBA data), with 11 461-row submatrices giving the random effects on the intercept and each 10 basis function coefficient, in order. This matrix is copied across all 11 corresponding output fields in the `inla.out$summary.random` object:

[1]See Appendix A.1 of the paper for explanations on the meaning of the covariates used

	mean	sd
(Intercept)	-3.30	0.63
dribble	-0.32	0.01
ndef	-0.08	0.01
ball.lastsec	0.06	0.00
b1	1.79	0.63
b2	-1.62	0.63
b3	-0.52	0.64
b4	0.82	0.63
b5	-6.80	0.64
b6	-1.60	0.64
b7	-3.25	0.63
b8	-2.89	0.64
b9	-3.62	0.63
b10	-0.80	0.64

	mean	sd	0.025quant	0.5quant	0.975quant
237	0.31	0.09	0.14	0.31	0.49

```
n.player <- nrow(players)
# inference for Chris Bosh's intercept and first basis coefficient
xtable(inla.out$summary.random$p.int[this.player + 0:1, 2:6])
```

	mean	sd	0.025quant	0.5quant	0.975quant
237	-0.02	0.47	-0.95	-0.02	0.90
238	-0.56	0.49	-1.52	-0.56	0.40

```
xtable(inla.out$summary.random$p.b1[this.player + 0:1, 2:6]) # identical
```

The following code rearranges the output into a single matrix, with each row giving the player-specific parameters' posterior mean (fixed + random effects) for all model components (situational covariates and spatial effects).

```
param.names <- row.names(inla.out$summary.fixed)
n <- nrow(players)
player.params <- matrix(NA, nrow=n, ncol=length(param.names))
y.fix <- inla.out$summary.fixed[, "mean"] # fixed effects
temp <- names(inla.out$summary.random)
basis.inds <- c(which(temp == "p.int"), grep("p.b[0-9][0-9]*", temp))
cov.inds <- setdiff(seq(length(inla.out$summary.random)), basis.inds)
for(pl in 1:n) {
  # add players' random effects to fixed effects
  y.rand <- c(inla.out$summary.random$p.int[pl, "mean"],
    sapply(cov.inds,
      function(k) inla.out$summary.random[[k]][pl, "mean"]),
    inla.out$summary.random$p.b1[pl + n * (1:n.basis), "mean"])
  player.params[pl, ] <- y.fix + y.rand
}
```

For Chris Bosh, for instance, we can view his parameter estimates and see where each ranks relative to the rest of the league:

	mean	sd	0.025quant	0.5quant	0.975quant
237	-0.02	0.47	-0.95	-0.02	0.90
238	-0.56	0.49	-1.52	-0.56	0.40

```r
values <- player.params[this.player, ]
ranks <- apply(player.params, 2, function(col) rank(col)[this.player]) # increasing order
xtable(data.frame(param.names, values, ranks), digits=c(0,0,2,0))
```

	param.names	values	ranks
1	(Intercept)	-3.32	218
2	dribble	-0.00	361
3	ndef	-0.02	251
4	ball.lastsec	0.07	234
5	b1	0.21	57
6	b2	-2.32	110
7	b3	0.10	322
8	b4	0.49	185
9	b5	-5.02	439
10	b6	-2.17	146
11	b7	-2.47	335
12	b8	-1.80	245
13	b9	-3.60	231
14	b10	-2.00	97

The most notable values here a small `b1` coefficient relative to the rest of the league, and a large `b5`. Referring to Figure 5, we see that this means his shot-taking hazard is relatively low in the right-handed layup area, and relatively high in three point range. This suggests that, adjusting for his baseline shooting rate (`intercept`) and other situation covariates, Bosh attempts threes at a high rate (per time controlling the ball from three point range), and right-handed layups/dunks at a low rate. This behavior is generally shared among other stretch-4 type players who are catch-and-shoot three-point shooters, and whose touches near the basket come more from slow-developing plays or those that don't lead to shots—like "isolations" or offensive rebounds—than from layups or attacking (also, note that Bosh is left handed). For instance, players such as Kevin Love and Dirk Nowitzki exhibit similar behavior.

Analagous to Figure 5 in the paper, we can plot players' spatial effect surfaces. It is also helpful to plot only the random effects, to see where players' spatial tendencies differ from typical league behavior. For Chris Bosh's shot-taking hazard, we get these side-by-side with:

```r
vars <- paste0("b", seq(n.basis))
spat.fixed <- as.numeric(inla.out$summary.fixed["(Intercept)", "mean"] +
                         t(take.basis) %*% inla.out$summary.fixed[vars, "mean"])
spat.random <- as.numeric(inla.out$summary.random$p.int[this.player, "mean"] +
                          t(take.basis) %*% inla.out$summary.random$p.int[this.player + n * (1:n.basis), "mean"])

par(mfrow=c(1,2), mar=c(1,4,1,6))
spatialPlot1(spat.fixed + spat.random, axis.args=list(cex.axis=0.75))
spatialPlot1(spat.random, axis.args=list(cex.axis=0.75))
```

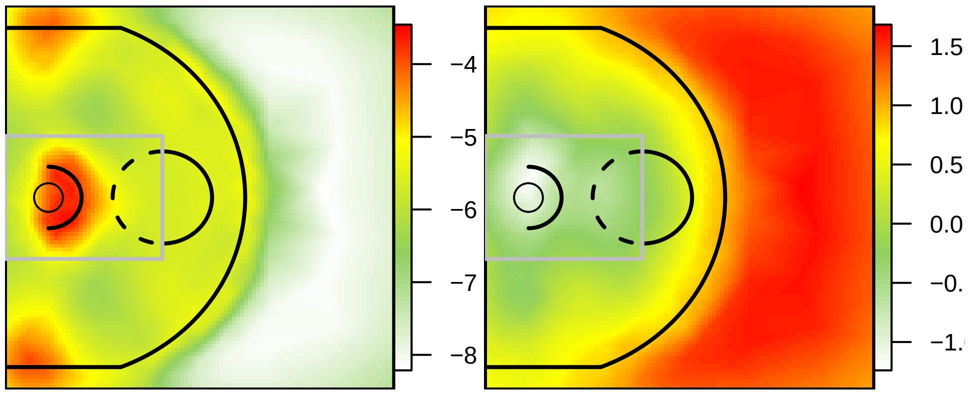

Figure 7: Shot-taking spatial effect for Chris Bosh (left). The difference in this surface relative to the rest of the league is illustrated on the right.

To view the spatial effect on a passing hazard (for instance, to player 1—the point guard), we would do:

```
load(sprintf("%s/INLA_PASS1.Rdata", data.dir))
vars <- paste0("b", seq(n.basis))
spat.fixed <- as.numeric(inla.out$summary.fixed["(Intercept)", "mean"] +
                           t(pass1.basis) %*% inla.out$summary.fixed[vars, "mean"])
spat.random <- as.numeric(inla.out$summary.random$p.int[this.player, "mean"] +
                            t(pass1.basis) %*% inla.out$summary.random$p.int[this.player + n * (1:n.basis), "mean"])

par(mfrow=c(1,2), mar=c(1,4,1,6))
spatialPlot2(head(spat.fixed + spat.random, mesh$n),
             tail(spat.fixed + spat.random, mesh$n),
                 axis.args=list(cex.axis=0.75))
```

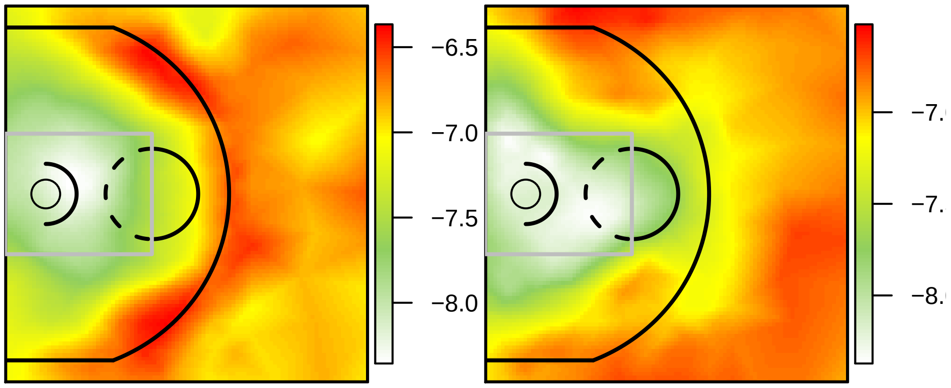

Figure 8: Spatial effect for passes from Chris Bosh to the point guard. The effect of Bosh's location is on the left, and the effect of the PG's location is on the right.

Lastly, it's useful to check the hyperparameter estimates to make sure they are sensible. The hyperparameters for the macrotransition entry models (and shot probability model) and log precision terms for the CAR model, described in Sections 4.1 and 4.2 of the paper. In this implementation, we've fixed the hyperparameters for all spatial basis loadings to be the same within each macrotransition entry model.

```
inla.out$mode$theta         # parameter values

## [1] -2.36010142 -0.04486083  0.53744864  4.00845149 -0.12840774  0.46386675  0.88176478 -0.21073187
```

```
inla.out$mode$theta.tags   # parameter names

## [1] "Log precision for p.int"          "Log precision for p.dribble"
## [3] "Log precision for p.ndef"         "Log precision for p.ball.lastsec"
## [5] "Log precision for p.doff1"        "Log precision for p.doff2"
## [7] "Log precision for p.doff3"        "Log precision for p.ddef"
```

3.3 Transition probability matrices

The last model component needed to calculate EPV are the transition probability matrices for C_t, described in Section 3.4 of the paper. We load these—for instance, for Dwyane Wade, by running:

```
player.id <- players$player_id[grep("Wade", players$lastname)]
load(sprintf("%s/tmats/%s.Rdata", data.dir, player.id))
names(tmat.ind)

## [1] "micros"  "passes1" "passes2" "passes3" "passes4" "absorbs"
```

`tmat.ind` is a list with each element representing blocks (sub-matrices) of $\tilde{\mathbf{N}}$, the transition count matrix for C_t given the players on the court (see Section 3.4 of the paper). The rows in each block represent the 14 $\{$`region`$\} \times \{$defended$\}$ states we use in C_t for a given ballcarrier, as expalined in Section 2.2 of the paper. Columns in these blocks also represent such states, except for the `absorbs` block, where columns represent absorbing states in $\mathcal{C}_{\text{end}}$. Depending on the lineup used, different blocks will be used to construct $\mathbf{P}$. Also note, the `tmat.pos` object contains blocks used in calculating EPV-Added, as discussed in Section A.4 of the paper.

4 Calculating EPV

4.1 Coarsened state expected point values

Given estimates of our parameters, EPV is calculated using Monte Carlo. The general idea, introduced in Section 3 of the paper, is to alternate draws from the micro- and macrotransition entry models until a macrotransition (pass, shot attempt, turnover) occurs. Then, given the predicted outcome of this macrotransition, we calculate EPV using the transition probability matrix of coarsened states. Before actually simulating EPV draws, it's useful to look at what the expected point values are of each coarsened state, as EPV will always be a weighted average of these values:

```
source(sprintf("%s/parameters.R", code.dir))
hyper <- getHyperParams(tdat) # makes sure all parameter inference is loaded
ev.out <- evLineups(tdat) # coarsened state EVs for each offensive lineup in tdat
```

In `ev.out`, `teammates.all` is a matrix of 5-man lineups that appear in `tdat` (there may be duplicate rows). For instance, we have the starting 5 for the Miami Heat:

```
lineup.ids <- ev.out$teammates.all[2, ]
this.lineup <- players[match(lineup.ids, players$player_id), ]
this.lineup[, 2:4]

##     firstname lastname       position
## 243     Mario Chalmers    Point-Guard
## 238    Dwyane     Wade          Guard
## 236    LeBron    James        Forward
## 240    Udonis   Haslem  Power-Forward
## 237     Chris     Bosh Forward-Center
```

For each 5-man lineup, there are 5 $\times$ 2 (defended or not) $\times$ 7 (court regions) = 70 coarsened state expected values. To check these for LeBron James' possession states, for instance, we'd do:

```r
lineup.states <- paste(rep(this.lineup$lastname, each=14), state_nms) # state names
xtable(data.frame(state=lineup.states, EV=ev.out$evs[[2]])[grep("James", lineup.states), ], digits=2)
```

	state	EV
29	James behind-TRUE	1.06
30	James per-TRUE	1.03
31	James rest-TRUE	1.42
32	James key-TRUE	1.24
33	James cor3-TRUE	1.09
34	James cen3-TRUE	1.02
35	James other-TRUE	0.99
36	James behind-FALSE	1.05
37	James per-FALSE	1.03
38	James rest-FALSE	1.60
39	James key-FALSE	1.34
40	James cor3-FALSE	1.21
41	James cen3-FALSE	1.03
42	James other-FALSE	1.00

These results seem pretty sensible, as, for instance, EVs are uniformly higher for uncontested states, with the difference especially great within the restricted area (1.60 versus 1.42) and corner 3 (1.21 versus 1.09). Note that with different teammates, we would see slightly different EVs for these states.

4.2 EPV curves

As mentioned in the paper, given estimates of all parameter values, EPV is computed by Monte Carlo sampling from the multiresolution transition models. This is a computationally expensive procedure, dominated by computing spatial effects for every player-position update from the microtransition model and hazard calculation from the macrotransition entry/exit models. However, it is straightforward to sample multiple time points together.

To supply EPV curves for a full game, it's most efficient to draw a single EPV estimate for all time points in a game, and then parallelize this across multiple machines that don't need to share memory. The code executes an EPV draw at each time point for every offensive possession in our sample game:

```r
source(sprintf("%s/EPV_calcs.R", code.dir))
draw.raw <- multiresDraw(tdat, hyper, def.micro, ev.out, nmic=50, save.positions=F)
draw <- compressEPV(tdat, draw.raw$fv.epv.list)
```

The `nmic` argument specifies 50 iterations (2 seconds) of the microtranistion model, which is usually sufficient to observe τ_t, a macrotransition entry. The `save.positions` argument stores the player-position innovations supplied by the micro model. These are necessary to reproduce Figure 7, which shows players' predicted motion paths, but necessitate lots of additional storage, as they essentially replicate the full positional data `nmic` times for each EPV draw.

We can load a pre-computed version of `draw`:

```r
load(sprintf("%s/draw.Rdata", data.dir))
names(draw)

## [1] "epv"       "probs"     "vals"      "probs.now" "vals.now"
```

Here, **epv** is a vector of EPV values corresponding to each row of **tdat**. **probs** is a data frame where each row gives the probabilities associated with each possible macrotransition event at time t ($\mathbb{P}(C_{\delta_t}|\mathcal{F}_t^{(Z)})$), and **vals** gives the associated expected point values conditional on these macrotransitions: $\mathbb{E}[X|C_{\delta_t}]$ (these probabilities/values are illustrated in Figure 7 of the paper). **probs** and **vals** contain an "**other**" state which represents no macrotransition occurring within the 50 simulated microtransitions. In this case, to calculate the expected value, we use the coarsened state expected value associated with the final microtransition draw, in this case $\mathbb{E}[X|C_{t+2}]$. **probs.now** and **vals.now** are the instantaneous macrotransition probabilities and associated expected values.

For instance, during the first possession in this game, after Udonis Haslem brings the ball into the offensive halfcourt, we see the next action to most likely be pass to Chalmers or Chris Bosh (the next play is a pass to Bosh). A shot attempt is extremely unlikely, and there is a 0.226 probability that Haslem will still possess the ball 2 seconds down the road. Among his passing options, James is the most valuable, though also the least likely to occur (James is near the basket, but the passing lane doesn't appear to be open).

```
transformed.data.plotter(tdat, 30)
```

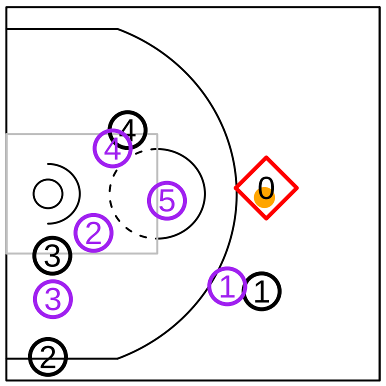

Figure 9: Udonis Haslem with ball possession. His four passing options are 1: Mario Chalmers, 2: Dwyane Wade, 3: LeBron James, 4: Chris Bosh.

```
draw$probs[30, ]

##        pass1      pass2      pass3     pass4        make       miss         TO     other
## 30 0.5747731 0.02065291 0.01900831 0.1448298 0.000981215 0.00157953 0.01181597 0.2263592

draw$vals[30, ]

##        pass1    pass2    pass3    pass4 make miss TO     other
## 30 0.9676442 1.017705 1.105218 1.014151    3 0.15  0 0.9707326
```

Because each EPV draw executes independent multiresolution transition simulations for each time point t, the resultant EPV curve is not very smooth. For instance, at time t, we might simulate a player driving toward the basket and attempting a layup, whereas at time $t + \epsilon$ we simulate the same player passing to a teammate. We see this below:

```
plot(720 - tdat$game_clock[1:100], draw$epv[1:100], xlab="game clock", ylab="EPV")
```

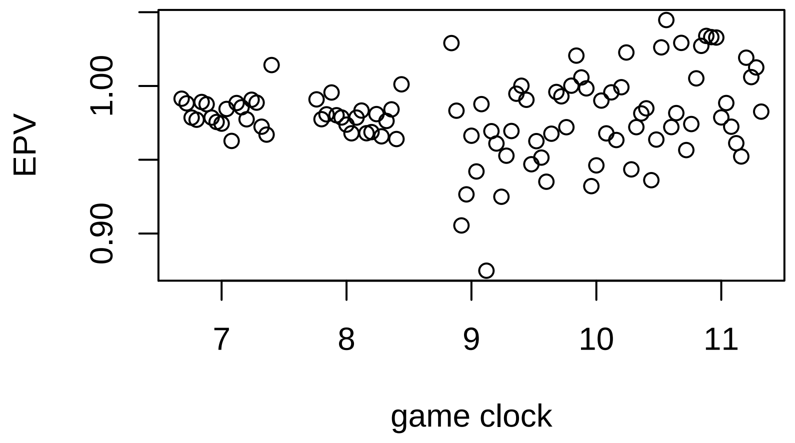

Figure 10: EPV estimates of a single draw

Of course, averaging over multiple EPV draws offers more smoothness—though by design, we see spikes in EPV exactly at moments when passes/shots/turnovers occur. The files `EPV_draw.R` and `combine_draws.R` execute independent EPV draws for this game on a computing cluster. We have combined 200 of these draws to obtain a final Monte Carlo EPV estimate (as well as Monte Carlo estimates of the transition probabilities and values). Below we load this, and merge these EPV estimates into the original full data set `dat`, where EPV is `NA` when the ball is not in the offensive halfcourt with the game clock moving. We also compute a "smoothed EPV" to (very slightly) interpolate the pointwise EPV estimates over time.

```
source(sprintf("%s/EPV_calcs.R", code.dir))
load(sprintf("%s/combined.epv.draws.Rdata", data.dir))
e.dat <- combineDatEPV(dat, epv.table)
```

We can now plot out EPV "tickers", as in Figure 2 of the paper:

```
par(xpd=NA, bty="n", mfrow=c(1, 2))
poss.1 <- which(e.dat$possID == 1)
plot(720 - e.dat$game_clock[poss.1], e.dat$epv.smooth[poss.1],
     xlab="game clock", ylab="EPV", type="l", lwd=2, ylim=c(.5, 1.5))
points(720 - e.dat$game_clock[poss.1], e.dat$epv[poss.1], pch=20, cex=0.5)

poss.90 <- which(e.dat$possID == 90) # possession shown in paper
plot(720 - e.dat$game_clock[poss.90], e.dat$epv.smooth[poss.90],
     xlab="game clock", ylab="EPV", type="l", lwd=2, ylim=c(.5, 1.5))
points(720 - e.dat$game_clock[poss.90], e.dat$epv[poss.90], pch=20, cex=0.5)
```

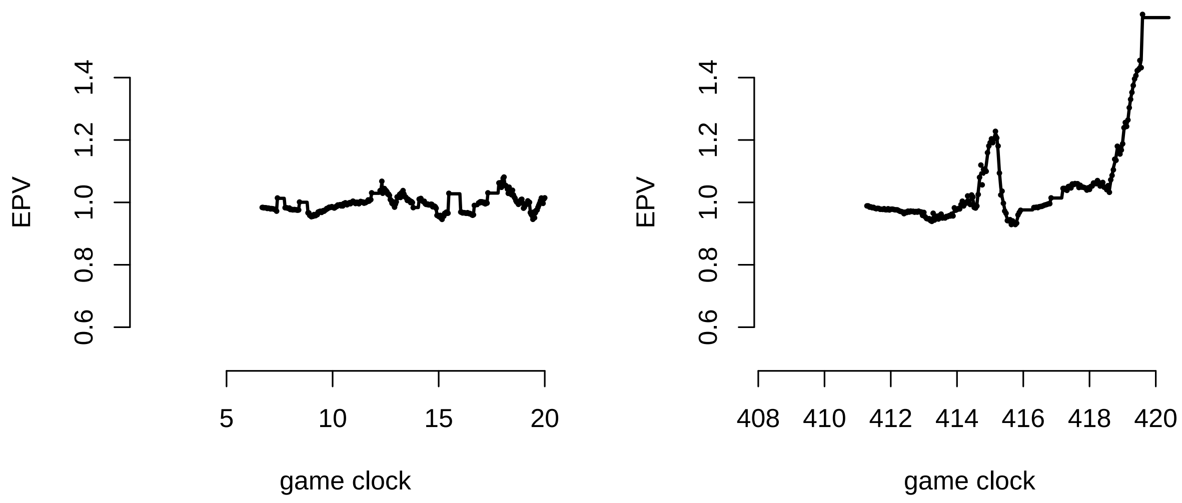

Figure 11: EPV curves for two possessions in this game. The line slightly smooths the actual EPV values (dots).

One of the best ways to view EPV results is by generating gifs that show EPV curves side-by-side with the possession evolution. Below we've generated gifs for a pair of long and interesting-looking possessions (they're located in the `gifs` folder):

```
makeGIF(e.dat, which(e.dat$possID == 12), "poss_12") # takes a few minutes
makeGIF(e.dat, which(e.dat$possID == 24), "poss_24") # takes a few minutes
```

4.3 Derived metrics

The derived metrics presented in the paper, EPV-Added (EPVA) and shot satisfaction, are most meaningful when computed using a large sample of data, such as a full season. However, just as with any other basketball metric, we can calculate per-game versions of these statistics. For instance, to get these metrics for LeBron James and Deron Williams, we'd do:

```
id <- players$player_id[grep("LeBron", players$firstname)]
sum(EPVA(tdat, id))       # sums EPV added on each touch

## [1] 8.336796

mean(shotSatis(tdat, id)) # averages shot satisfaction of each touch

## [1] 0.2229599

id <- players$player_id[grep("Deron", players$firstname)] # Deron Williams
sum(EPVA(tdat, id))       # sums EPV added on each touch

## [1] 0.774197

mean(shotSatis(tdat, id)) # averages shot satisfaction of each touch

## [1] 0.1002303
```

5 Appendix

Raw data `event_id` codes:

Event	ID	Event	ID	Event	ID	Event	ID	Event	ID
FT Made	1	Def. Rebound	6	Timeout	11	Clock Sync	16	Dribble	21
FT Missed	2	Turnover	7	Jump Ball	12	Instant Replay	17	Pass	22
Shot Made	3	Foul	8	Ejection	13	Replay Ruling	18	Possession	23
Shot Missed	4	Violation	9	Start Period	14	Game Over	19	Shot Block	24
Off. Rebound	5	Substitution	10	End Period	15	Stoppage	20	Assist	25

Table 2: Glossary of `event_id` codes in optical tracking data.