Stan code for Husain et al paper, PLoS ONE 2014

Shravan Vasishth

September 12, 2014

1 Paper reference

This code is for the paper:

Samar Husain, Shravan Vasishth, and Narayanan Srinivasan. Strong Expectations Cancel Locality Effects: Evidence from Hindi. PLoS ONE, 9(7):1-14, 2014.

Available from: http://tinyurl.com/pnaykvs.

2 Abstract

Expectation-driven facilitation (Hale, 2001; Levy, 2008) and locality-driven retrieval difficulty (Gibson, 1998, 2000; Lewis & Vasishth, 2005) are widely recognized to be two critical factors in incremental sentence processing; there is accumulating evidence that both can influence processing difficulty. However, it is unclear whether and how expectations and memory interact. We first confirm a key prediction of the expectation account: a Hindi self-paced reading study shows that when an expectation for an upcoming part of speech is dashed, building a rarer structure consumes more processing time than building a less rare structure. This is a strong validation of the expectation-based account. In a second study, we show that when expectation is strong, i.e., when a particular verb is predicted, strong facilitation effects are seen when the appearance of the verb is delayed; however, when expectation is weak, i.e., when only the part of speech âĂIJverbâĂİ is predicted but a particular verb is not predicted, the facilitation disappears and a tendency towards a locality effect is seen. The interaction seen between expectation strength and distance shows that strong expectations cancel locality effects, and that weak expectations allow locality effects to emerge.

3 Session Info

- R version 3.1.1 (2014-07-10), `x86_64-apple-darwin13.1.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`

- Base packages:
- Other packages: rstan 2.4.0
- Loaded via a namespace (and not attached): base 3.1.1, datasets 3.1.1, graphics 3.1.1, grDevices 3.1.1, inline 0.3.13, methods 3.1.1, Rcpp 0.11.2, stats 3.1.1, stats4 3.1.1, tools 3.1.1, utils 3.1.1

4 Experiment 1

4.1 Data preparation

Load data and format for Stan:

```
> RC1<-read.table("expt1critdata.txt",header=T)
> e1data <- list(mu_prior=c(0,0,0,0),
+              subj=sort(as.integer(factor(RC1$subj))),
+              item=sort(as.integer(factor(RC1$item))),
+              lrt=RC1$lrt,
+              dist = RC1$dist,
+              rctype = RC1$RCType,
+              interaction = RC1$int,
+              N = nrow(RC1),
+              I = length(unique(RC1$subj)),
+              K = length(unique(RC1$item))
+ )
```

4.2 Define model (method 1)

```
> expt1_code <-
+ 'data {
+ int<lower=1> N;
+ real lrt[N];                      //outcome
+ real<lower=-1,upper=1> dist[N];  //predictor
+ real rctype[N];                   //predictor
+ real interaction[N];              //predictor
+ int<lower=1> I;                   //number of subjects
+ int<lower=1> K;                   //number of items
+ int<lower=1, upper=I> subj[N];   //subject id
+ int<lower=1, upper=K> item[N];   //item id
+ vector[4] mu_prior;               //vector of zeros passed in from R
+ }
+ transformed data {
+ real ZERO;                       // like #define ZERO 0 in C/C++
+ ZERO <- 0.0;
+ }
```

```
+ parameters {
+ vector[4] beta;                  // intercept and slope
+ vector[4] u[I];                  // random intercept and slopes subj
+ vector[4] w[K];
+ real<lower=0> sigma_e;          // residual sd
+ vector<lower=0>[4] sigma_u;     // subj sd
+ vector<lower=0>[4] sigma_w;     // item sd
+ corr_matrix[4] Omega_u;           // correlation matrix for random intercepts and slopes s
+ corr_matrix[4] Omega_w;           // correlation matrix for random intercepts and slopes i
+ }
+ transformed parameters {
+ matrix[4,4] D_u;
+ matrix[4,4] D_w;
+ D_u <- diag_matrix(sigma_u);
+ D_w <- diag_matrix(sigma_w);
+ }
+ model {
+ matrix[4,4] L_u;
+ matrix[4,4] DL_u;
+ matrix[4,4] L_w;
+ matrix[4,4] DL_w;
+ real mu[N]; // mu for likelihood
+ //priors:
+ beta ~ normal(0,10);
+ sigma_e ~ normal(0,10);
+ sigma_u ~ normal(0,10);
+ sigma_w ~ normal(0,10);
+ Omega_u ~ lkj_corr(4.0);
+ Omega_w ~ lkj_corr(4.0);
+ L_u <- cholesky_decompose(Omega_u);
+ L_w <- cholesky_decompose(Omega_w);
+ for (m in 1:4) {
+ for (n in 1:m) {
+ DL_u[m,n] <- L_u[m,n] * sigma_u[m];
+ }
+ }
+ for (m in 1:4){
+ for (n in (m+1):4){
+ DL_u[m,n] <- ZERO;
+ }
+ }
+ for (m in 1:4){
+ for (n in 1:m){
+ DL_w[m,n] <- L_w[m,n] * sigma_w[m];
+ }
+ }
```

```
+ for (m in 1:4){
+ for (n in (m+1):4){
+ DL_w[m,n] <- ZERO;
+ }
+ }
+ for (i in 1:I)                  // loop for subj random effects
+ u[i] ~ multi_normal_cholesky(mu_prior, DL_u);
+ for (k in 1:K)                  // loop for item random effects
+ w[k] ~ multi_normal_cholesky(mu_prior, DL_w);
+ for (n in 1:N) {
+ mu[n] <- beta[1] + beta[2]*dist[n] + beta[3]*rctype[n] + beta[4]*interaction[n] + u[subj[n
+ }
+ lrt ~ normal(mu,sigma_e);        // likelihood
+ }
+ generated quantities {
+ cov_matrix[4] Sigma_u;
+ cov_matrix[4] Sigma_w;
+ Sigma_u <- D_u * Omega_u * D_u;
+ Sigma_w <- D_w * Omega_w * D_w;
+ }
+ '
```

Fit data:

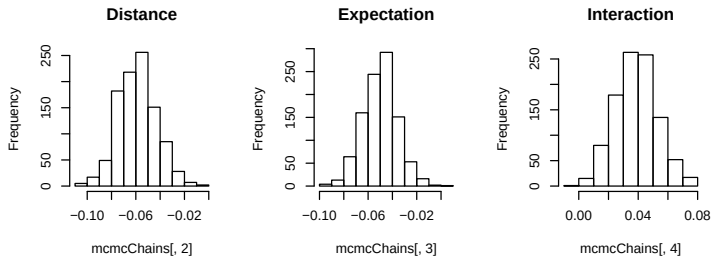
```
> set_cppo('fast')
> fit <- stan(model_code = expt1_code, data = e1data,
+             iter = 500, chains = 4)

TRANSLATING MODEL 'expt1_code' FROM Stan CODE TO C++ CODE NOW.
COMPILING THE C++ CODE FOR MODEL 'expt1_code' NOW.

SAMPLING FOR MODEL 'expt1_code' NOW (CHAIN 1).

Iteration:   1 / 500 [  0%]  (Warmup)
Iteration:  50 / 500 [ 10%]  (Warmup)
Iteration: 100 / 500 [ 20%]  (Warmup)
Iteration: 150 / 500 [ 30%]  (Warmup)
Iteration: 200 / 500 [ 40%]  (Warmup)
Iteration: 250 / 500 [ 50%]  (Warmup)
Iteration: 251 / 500 [ 50%]  (Sampling)
Iteration: 300 / 500 [ 60%]  (Sampling)
Iteration: 350 / 500 [ 70%]  (Sampling)
Iteration: 400 / 500 [ 80%]  (Sampling)
Iteration: 450 / 500 [ 90%]  (Sampling)
Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 12.7522 seconds (Warm-up)
#                5.3349 seconds (Sampling)
```

```
#                18.0871 seconds (Total)


SAMPLING FOR MODEL 'expt1_code' NOW (CHAIN 2).

Iteration:   1 / 500 [  0%]  (Warmup)
Iteration:  50 / 500 [ 10%]  (Warmup)
Iteration: 100 / 500 [ 20%]  (Warmup)
Iteration: 150 / 500 [ 30%]  (Warmup)
Iteration: 200 / 500 [ 40%]  (Warmup)
Iteration: 250 / 500 [ 50%]  (Warmup)
Iteration: 251 / 500 [ 50%]  (Sampling)
Iteration: 300 / 500 [ 60%]  (Sampling)
Iteration: 350 / 500 [ 70%]  (Sampling)
Iteration: 400 / 500 [ 80%]  (Sampling)
Iteration: 450 / 500 [ 90%]  (Sampling)
Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 14.4072 seconds (Warm-up)
#                24.3153 seconds (Sampling)
#                38.7225 seconds (Total)


SAMPLING FOR MODEL 'expt1_code' NOW (CHAIN 3).

Iteration:   1 / 500 [  0%]  (Warmup)
Iteration:  50 / 500 [ 10%]  (Warmup)
Iteration: 100 / 500 [ 20%]  (Warmup)
Iteration: 150 / 500 [ 30%]  (Warmup)
Iteration: 200 / 500 [ 40%]  (Warmup)
Iteration: 250 / 500 [ 50%]  (Warmup)
Iteration: 251 / 500 [ 50%]  (Sampling)
Iteration: 300 / 500 [ 60%]  (Sampling)
Iteration: 350 / 500 [ 70%]  (Sampling)
Iteration: 400 / 500 [ 80%]  (Sampling)
Iteration: 450 / 500 [ 90%]  (Sampling)
Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 10.2164 seconds (Warm-up)
#                10.2089 seconds (Sampling)
#                20.4253 seconds (Total)


SAMPLING FOR MODEL 'expt1_code' NOW (CHAIN 4).

Iteration:   1 / 500 [  0%]  (Warmup)
Iteration:  50 / 500 [ 10%]  (Warmup)
Iteration: 100 / 500 [ 20%]  (Warmup)
```

```
Iteration: 150 / 500 [ 30%]  (Warmup)
Iteration: 200 / 500 [ 40%]  (Warmup)
Iteration: 250 / 500 [ 50%]  (Warmup)
Iteration: 251 / 500 [ 50%]  (Sampling)
Iteration: 300 / 500 [ 60%]  (Sampling)
Iteration: 350 / 500 [ 70%]  (Sampling)
Iteration: 400 / 500 [ 80%]  (Sampling)
Iteration: 450 / 500 [ 90%]  (Sampling)
Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 10.877 seconds (Warm-up)
#                10.9669 seconds (Sampling)
#                21.844 seconds (Total)
```

```
> ## not run, too verbose:
> #print(fit)
```

Plot the posterior distributions of the three coefficients of interest:



4.3 Run model (Method 2)

```
> library(parallel)
> ## uncomment to save results to a file:
> #sink("expt1resultsrun2.txt")
```

```
>
> ## the model is defined in a separate file:
> e1.sm <- stan_model("expt1subjitem.stan", model_name = "e1subjitem")

TRANSLATING MODEL 'e1subjitem' FROM Stan CODE TO C++ CODE NOW.
COMPILING THE C++ CODE FOR MODEL 'e1subjitem' NOW.

> sflist <- mclapply(1:4, mc.cores = detectCores(),
+                    function(i) sampling(e1.sm, data = e1data,
+                                         chains = 1, chain_id = i,
+                                         seed = 12345))
> e1.sf <- sflist2stanfit(sflist)
> #print(e1.sf,digits=4)
```

Comparison with lmer:

```
> library(lme4)
> m1<-lmer(lrt~dist+RCType+int+(1+dist+RCType+int|subj)+(1+dist+RCType+int|item),RC1)
> summary(m1)

Linear mixed model fit by REML ['lmerMod']
Formula: lrt ~ dist + RCType + int + (1 + dist + RCType + int | subj) +
    (1 + dist + RCType + int | item)
   Data: RC1

REML criterion at convergence: 2051.9

Scaled residuals:
    Min      1Q  Median      3Q     Max
-5.6366 -0.6089 -0.1853  0.4630  4.6925

Random effects:
 Groups   Name        Variance  Std.Dev. Corr
 subj     (Intercept) 0.0901560 0.30026
          dist        0.0031861 0.05645  -0.60
          RCType      0.0005826 0.02414  -0.56  0.70
          int         0.0018562 0.04308   0.74 -0.94 -0.88
 item     (Intercept) 0.0034894 0.05907
          dist        0.0018261 0.04273  -0.39
          RCType      0.0007440 0.02728   0.60 -0.35
          int         0.0010803 0.03287  -0.61 -0.38  0.05
 Residual             0.2089150 0.45707
Number of obs: 1440, groups:  subj, 60; item, 24

Fixed effects:
            Estimate Std. Error t value
(Intercept)  6.65832    0.04234  157.24
```

```
dist        -0.05746    0.01656   -3.47
RCType      -0.04974    0.01363   -3.65
int          0.03712    0.01487    2.50

Correlation of Fixed Effects:
       (Intr) dist   RCType
dist   -0.301
RCType -0.047 -0.005
int     0.176 -0.245 -0.067
```

5 Experiment 2

5.1 Data preparation

Load data:

```
> CP1<-read.table("expt2critdata.txt",header=T)
> e2data <- list(mu_prior=c(0,0,0,0),
+                subj=sort(as.integer(factor(CP1$subj))),
+                item=sort(as.integer(factor(CP1$item))),
+                lrt = log(CP1$rt),
+                dist = CP1$dist,
+                expectation = CP1$exp,
+                interaction = CP1$int,
+                N = nrow(CP1),
+                I = length(unique(CP1$subj)),
+                K = length(unique(CP1$item))
+ )
```

5.2 Define model

```
> expt2_code <-'
+ data {
+     int<lower=0> N;
+     real lrt[N];                    //outcome
+ real dist[N];                     //predictor
+ real expectation[N];                     //predictor
+ real interaction[N];                     //predictor
+ int<lower=1> I;                 //number of subjects
+ int<lower=1> K;                 //number of items
+ int<lower=1, upper=I> subj[N];    //subject id
+ int<lower=1, upper=K> item[N];    //item id
+ vector[4] mu_prior;             //vector of zeros passed in from R
+ }
+ transformed data {
+ real ZERO;                      // like #define ZERO 0 in C/C++
```

```
+ ZERO <- 0.0;
+ }
+ parameters {
+ vector[4] beta;                 // intercept and slope
+ vector[4] u[I];                 // random intercept and slopes subj
+ vector[4] w[K];
+ real<lower=0> sigma_e;          // residual sd
+ vector<lower=0>[4] sigma_u;     // subj sd
+ vector<lower=0>[4] sigma_w;     // item sd
+ corr_matrix[4] Omega_u;           // correlation matrix for random intercepts and slopes s
+ corr_matrix[4] Omega_w;           // correlation matrix for random intercepts and slopes i
+ }
+ transformed parameters {
+ matrix[4,4] D_u;
+ matrix[4,4] D_w;
+ D_u <- diag_matrix(sigma_u);
+ D_w <- diag_matrix(sigma_w);
+ }
+ model {
+ matrix[4,4] L_u;
+ matrix[4,4] DL_u;
+ matrix[4,4] L_w;
+ matrix[4,4] DL_w;
+ real mu[N]; // mu for likelihood
+ //priors:
+ beta ~ normal(0,10);
+ sigma_e ~ normal(0,10);
+ sigma_u ~ normal(0,10);
+ sigma_w ~ normal(0,10);
+ Omega_u ~ lkj_corr(4.0);
+ Omega_w ~ lkj_corr(4.0);
+ L_u <- cholesky_decompose(Omega_u);
+ L_w <- cholesky_decompose(Omega_w);
+ for (m in 1:4) {
+ for (n in 1:m) {
+ DL_u[m,n] <- L_u[m,n] * sigma_u[m];
+ }
+ }
+ for (m in 1:4){
+ for (n in (m+1):4){
+ DL_u[m,n] <- ZERO;
+ }
+ }
+ for (m in 1:4){
+ for (n in 1:m){
+ DL_w[m,n] <- L_w[m,n] * sigma_w[m];
```

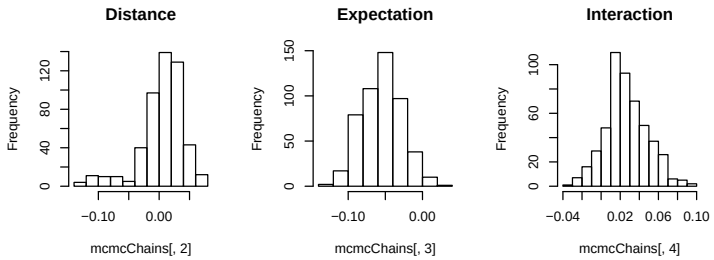
```
+ }
+ }
+ for (m in 1:4){
+ for (n in (m+1):4){
+ DL_w[m,n] <- ZERO;
+ }
+ }
+ for (i in 1:I)                  // loop for subj random effects
+ u[i] ~ multi_normal_cholesky(mu_prior, DL_u);
+ for (k in 1:K)                  // loop for item random effects
+ w[k] ~ multi_normal_cholesky(mu_prior, DL_w);
+ for (n in 1:N) {
+ mu[n] <- beta[1] + beta[2]*dist[n] + beta[3]*expectation[n] + beta[4]*interaction[n]
+ + u[subj[n], 1] + u[subj[n], 2]*dist[n] + u[subj[n], 3]*expectation[n] + u[subj[n], 4]*int
+ }
+ lrt ~ normal(mu,sigma_e);        // likelihood
+ }
+ generated quantities {
+ cov_matrix[4] Sigma_u;
+ cov_matrix[4] Sigma_w;
+ Sigma_u <- D_u * Omega_u * D_u;
+ Sigma_w <- D_w * Omega_w * D_w;
+ }
+ '
```

5.3 Run model (Method 1)

```
> set_cppo('fast')
> fit <- stan(model_code = expt2_code, data = e2data,
+             iter = 500, chains = 2)

TRANSLATING MODEL 'expt2_code' FROM Stan CODE TO C++ CODE NOW.
COMPILING THE C++ CODE FOR MODEL 'expt2_code' NOW.

SAMPLING FOR MODEL 'expt2_code' NOW (CHAIN 1).

Iteration:   1 / 500 [  0%]  (Warmup)
Iteration:  50 / 500 [ 10%]  (Warmup)
Iteration: 100 / 500 [ 20%]  (Warmup)
Iteration: 150 / 500 [ 30%]  (Warmup)
Iteration: 200 / 500 [ 40%]  (Warmup)
Iteration: 250 / 500 [ 50%]  (Warmup)
Iteration: 251 / 500 [ 50%]  (Sampling)
Iteration: 300 / 500 [ 60%]  (Sampling)
Iteration: 350 / 500 [ 70%]  (Sampling)
Iteration: 400 / 500 [ 80%]  (Sampling)
```

```
Iteration: 450 / 500 [ 90%]  (Sampling)
Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 5.85641 seconds (Warm-up)
#                4.26557 seconds (Sampling)
#                10.122 seconds (Total)


SAMPLING FOR MODEL 'expt2_code' NOW (CHAIN 2).

Iteration:   1 / 500 [  0%]  (Warmup)
Iteration:  50 / 500 [ 10%]  (Warmup)
Iteration: 100 / 500 [ 20%]  (Warmup)
Iteration: 150 / 500 [ 30%]  (Warmup)
Iteration: 200 / 500 [ 40%]  (Warmup)
Iteration: 250 / 500 [ 50%]  (Warmup)
Iteration: 251 / 500 [ 50%]  (Sampling)
Iteration: 300 / 500 [ 60%]  (Sampling)
Iteration: 350 / 500 [ 70%]  (Sampling)
Iteration: 400 / 500 [ 80%]  (Sampling)
Iteration: 450 / 500 [ 90%]  (Sampling)
Iteration: 500 / 500 [100%]  (Sampling)
#  Elapsed Time: 5.76126 seconds (Warm-up)
#                3.07824 seconds (Sampling)
#                8.8395 seconds (Total)
```

Plot posterior distributions:



5.4 Run model (Method 2)

```
> #sink("expt2resultsrun2.txt")
>
> e2.sm <- stan_model("expt2subjitem.stan", model_name = "e2subjitem")

TRANSLATING MODEL 'e2subjitem' FROM Stan CODE TO C++ CODE NOW.
COMPILING THE C++ CODE FOR MODEL 'e2subjitem' NOW.

> sflist <- mclapply(1:4, mc.cores = detectCores(),
+                    function(i) sampling(e2.sm, data = e2data,chains = 1, chain_id = i, see
> e2.sf <- sflist2stanfit(sflist)
> #print(e2.sf,digits=4)
> #sink()
```