Housekeeping

```
var max = function(arr){reduce(function(a,b){if(a>b)return a; else return b;},arr[0],arr)};

//alternative to looping over rows, use map over a countlist(arr.length) index
var countlist = function(n){
    var countdown = function(n){//So ugly! But if you're gonna use this as an index, it makes more sense/keeps correct correspondances in count-up order.
        if(n<=0) return [];
        return [n-1].concat(countdown(n-1));
    }
    countdown(n).reverse()
}
```

Read the stimuli and settings (from R) into lists

```
var ppntid = map(function(arow){arow["ppntid"]},expdf)
var probA = map(function(arow){arow["probA"]},expdf)
var probB = map(function(arow){arow["probB"]},expdf)
var probC = map(function(arow){arow["probC"]},expdf)
var payoffA = map(function(arow){arow["payoffA"]},expdf)
var payoffB = map(function(arow){arow["payoffB"]},expdf)
var payoffC = map(function(arow){arow["payoffC"]},expdf)
//extra agent param setting for choice creation:
var ppnt_calcsd = map(function(arow){arow["calc_sd"]},expdf)
var ppnt_tolerance_prob  = map(function(arow){arow["tolerance_prob"]},expdf)
var ppnt_tolerance_payoff = map(function(arow){arow["tolerance_payoff"]},expdf)
var ppnt_orderror = map(function(arow){arow["p_err"]},expdf)

var hm_ppnts = max(map(function(arow){arow["ppntid"]},expdf))+1;
var hm_trials = expdf.length;
```

Create calc and ord observation for each trial

```
var trial_calcobs = map(function(i){
    var calcobs_arr =
       [probA[i]*payoffA[i]+gaussian({mu:0,sigma:ppnt_calcsd[i]}),
        probB[i]*payoffB[i]+gaussian({mu:0,sigma:ppnt_calcsd[i]}),
        probC[i]*payoffC[i]+gaussian({mu:0,sigma:ppnt_calcsd[i]})
       ];
    return calcobs_arr;
},countlist(hm_trials))

//ordinal observations for individual rows i
var ordrelation = function(a,b,tolerance,orderr){//these args are individual values...
    if(flip(orderr)) return categorical({vs:["<","=",">"],ps:[1,1,1]});
    if(Math.abs(a-b)<tolerance) return "=";
    if(a>b) return ">";
    if(a<b) return "<";
}
//convenient helper to get ord observations for every trial
var ordrelation_list = function(a,b,tolerance){//but these args are lists, entire cols of data.df.
    //Is it interesting that orderr is shared across attribute modality here?
    map(function(i){ordrelation(a[i],b[i],tolerance[i],ppnt_orderror[i])},
       countlist(hm_trials))
}
//ord observations for each trial
var ord_ABprob = ordrelation_list(probA,probB,ppnt_tolerance_prob);
var ord_ACprob = ordrelation_list(probA,probC,ppnt_tolerance_prob);
var ord_BCprob = ordrelation_list(probB,probC,ppnt_tolerance_prob);
var ord_ABpayoff = ordrelation_list(payoffA,payoffB,ppnt_tolerance_payoff);
var ord_ACpayoff = ordrelation_list(payoffA,payoffC,ppnt_tolerance_payoff);
var ord_BCpayoff = ordrelation_list(payoffB,payoffC,ppnt_tolerance_payoff);
```

Create a decision-maker for each trial

Set priors on prob and payoff

```
var agentMaker = function(i){
    return function(){
        var agent_probs = repeat(3,function(){sample(Beta({a:1,b:1}))});
        var agent_payoffs = repeat(3,function(){sample(
            Gaussian({mu:100,sigma:10})
        )});
```

(probabilistically) Constrain inferred prob and payoff to be consistent with ord and calc observations

```
	//observe the calculation observations: calcobs is known to be prob*payoff+noise, calcobs is known, noise is known, prob and payoff are inferred.
	observe(Gaussian({mu:agent_probs[0]*agent_payoffs[0],sigma:ppnt_calcsd[i]}),trial_calcobs[i][0])
	observe(Gaussian({mu:agent_probs[1]*agent_payoffs[1],sigma:ppnt_calcsd[i]}),trial_calcobs[i][1])
	observe(Gaussian({mu:agent_probs[2]*agent_payoffs[2],sigma:ppnt_calcsd[i]}),trial_calcobs[i][2])

	//similarly, agent knows exactly how ordrelation works, needs its prob and payoffs to produce ord relations consistent with the observed ones.
	condition(ordrelation(agent_probs[0],agent_probs[1],ppnt_tolerance_prob[i])==ord_ABprob[i]);
	condition(ordrelation(agent_probs[0],agent_probs[2],ppnt_tolerance_prob[i])==ord_ACprob[i]);
	condition(ordrelation(agent_probs[1],agent_probs[2],ppnt_tolerance_prob[i])==ord_BCprob[i]);
	condition(ordrelation(agent_payoffs[0],agent_payoffs[1],ppnt_tolerance_payoff[i])==ord_ABpayoff[i]);
	condition(ordrelation(agent_payoffs[0],agent_payoffs[2],ppnt_tolerance_payoff[i])==ord_ACpayoff[i]);
	condition(ordrelation(agent_payoffs[1],agent_payoffs[2],ppnt_tolerance_payoff[i])==ord_BCpayoff[i]);
```

Decision maker returns choice with maximum expected return

```
        //return a choice
        var estvalA = agent_probs[0]*agent_payoffs[0]
        var estvalB = agent_probs[1]*agent_payoffs[1]
        var estvalC = agent_probs[2]*agent_payoffs[2]

        if(estvalA>estvalB&&estvalA>estvalC) return 1;
        if(estvalB>estvalA&&estvalB>estvalC) return 2;
        if(estvalC>estvalA&&estvalC>estvalB) return 3;
        }
}
```

Take 500 samples of each row’s decision maker using ‘Infer’: this gives the posterior belief for each option that it is the best one. Take the option with the highest probability of being best.

```
var choices =  map(function(i){
    var agent = Infer({method:"MCMC",samples:499,lag:0,burn:100,model:agentMaker(i)});
    //Take the best:
    var A = Math.exp(agent.score(1)); //exp converts to probability, which I guess is not
necessary here but feels nice.
    var B = Math.exp(agent.score(2));
    var C = Math.exp(agent.score(3));
    if(A>B&&A>C)return 1;
    if(B>A&&B>C)return 2;
    if(C>A&&C>B)return 3;
//Ties rare, but possible if #samples%2==0||#samples%3==0
//Tiebreaker:
    var trialprobs =
[Math.exp(agent.score(1)),Math.exp(agent.score(2)),Math.exp(agent.score(3))];
    return categorical({vs:[1,2,3],ps:Vector(trialprobs)});

},countlist(hm_trials))
```

Return

```
var ret =[trial_calcobs,
ord_Abprob,
ord_Acprob,
ord_Bcprob,
ord_Abpayoff,
ord_Acpayoff,
ord_Bcpayoff,
choices];
ret;
```