[ ]

```
module MyModel

export run_simulation_v1, run_simulation_v2


using Distributions
using DataFrames
# Define the entities = agents and network -------------------------
-----

#Structs for Agents and Beliefs --------------------

abstract type  AbstractAgent end
abstract  type PolAgent <: AbstractAgent end

mutable struct Belief{T<:AbstractFloat}
    o::T
    σ::T
    whichissue::Int
end

mutable struct Agent <: PolAgent
    id::Int
    ideo::Vector
    idealpoint::AbstractFloat
end


# Constructors for Agents, Beliefs and Network --------------------
function create_belief(σ,issue)
    o = rand(Uniform())
    belief = Belief(o, σ, issue)
end

function create_idealpoint(ideology)
    opinions = []
    for issue in ideology
        push!(opinions,issue.o)
    end
    ideal_point = mean(opinions)
end


function create_agent(n_issues,id,σ)
    ideology = [create_belief(σ, issue) for issue in 1:n_issues ]
    idealpoint = create_idealpoint(ideology)
    agent = Agent(id,ideology, idealpoint)
end

# network creation
" Creates an 1-d array of Agents with opinions ∈ [0,1]. Since it's
a complete
graph + pairwise interaction i can create a list of Agents and pick
two of them to
interact!!"

function create_nw(σ, n_issues, size)
```

```
    nw = [create_agent(n_issues,i,σ) for i in 1:size]
end



# Define the processes\actions -------------------------------------
--
#Create i,j with this then pass it to the other fns
function ij_comparison(nw)
    i,j = rand(nw), rand(nw)
    if i==j
        ij_comparison(nw)
    end
    return(i,j)
end


#Input = two agents; Output = a issue and associated beliefs
function pick_issuebelief(i, j, n_issues)
    issue_belief = rand(1:n_issues)
    i_belief = i.ideo[issue_belief]
    j_belief = j.ideo[issue_belief]
    return(issue_belief, i_belief, j_belief)
end


# helper for posterior opinion and uncertainty
function calculate_pstar(i_belief, j_belief, p)
    numerator = p * (1 / (sqrt(2 * π ) * i_belief.σ ) )*
    exp(-((i_belief.o - j_belief.o)^2 / (2*i_belief.σ^2)))
    denominator = numerator + (1 - p)
    pₚ  = numerator / denominator
    return(pₚ)
end


# Helper for update step
#Input = beliefs in an issue and confidence paramater; Output = i
new opinion
function cal_posterior_o(i_belief, j_belief, p)
    pₚ = calculate_pstar(i_belief, j_belief, p)
    posterior_opinion = pₚ * ((i_belief.o + j_belief.o) / 2) +
        (1 - pₚ) * i_belief.o
end


#helper for update_step
function calc_pos_uncertainty(i_belief, j_belief, p)
    pₚ = calculate_pstar(i_belief, j_belief, p)
    posterior_uncertainty = sqrt(i_belief.σ^2 * ( 1 - pₚ/2) + pₚ *
(1 - pₚ) *
                                    ((i_belief.o - j_belief.o)/2)^2)
end



# update_step for changing opinion but not belief
function update_step1!(i, issue_belief, posterior_o)
    i.ideo[issue_belief].o = posterior_o
    newidealpoint = create_idealpoint(i.ideo)
    i.idealpoint = newidealpoint
```

```julia
end


# update_step for the version with changing opinions and changing
uncertainty
function update_step2!(i,issue_belief, posterior_o, posterior_σ)
    i.ideo[issue_belief].o = posterior_o
    i.ideo[issue_belief].σ = posterior_σ
    newidealpoint = create_idealpoint(i.ideo)
    i.idealpoint = newidealpoint
end




# Information Storing ---------------------------------------
# going to think about the dataframe for v2. put mean and std
uncertainty??
# information storing for v1 --------------------
function init_df(nw)
    df = DataFrame(time = [], ideal_point = [], id  = [])
    for agent in nw
        time = 0
        push!(df,[time agent.idealpoint agent.id])
    end
    return df
end

function update_df!(nw,df,time)
    for agent in nw
        push!(df,[time agent.idealpoint agent.id])
    end
end

# Running Commands ---------------------------------------
function run_simulation_v1(; n_issues = n_issues,
                            size_nw = size_nw, p = p, σ = σ, time
= time)
    nw = create_nw(σ, n_issues, size_nw)
    df = init_df(nw)
    for step in 1:time
        i,j = ij_comparison(nw)
        which_issue,i_belief,j_belief = pick_issuebelief(i, j,
n_issues)
        pos_o = cal_posterior_o(i_belief, j_belief, p)
        update_step1!(i, which_issue, pos_o)
        update_df!(nw,df,step)
    end
    return(df)
end

function run_simulation_v2(; n_issues = n_issues,
                            size_nw = size_nw, p = p, σ = σ, time
= time)
    nw = create_nw(σ, n_issues, size_nw)
    df = init_df(nw)
    for step in 1:time
        i,j = ij_comparison(nw)
        which_issue, i_belief, j_belief = pick_issuebelief(i, j,
n_issues)
        pos_o = cal_posterior_o(i_belief, j_belief, p)
```

```
            pos_σ = calc_pos_uncertainty(i_belief, j_belief, p)
            update_step2!(i, which_issue, pos_o, pos_σ)
            update_df!(nw,df,step)
        end
        return(df)
    end

    end
```