Stochastic Models and Simulations of Phototaxis

Link

In [1]:
```
using Random, LinearAlgebra, Plots, Statistics
```

Functions

In [2]:
```
"""
Convert polar coordinates to cartesian
"""
polar2cart(r, θ) = r .* [cos(θ) sin(θ)]

"""
Sample N points from a circle of radius R
https://stackoverflow.com/questions/5837572/generate-a-random-point-within-a-circle-uni

"""
function sampleCircle(N, R)
    # Sample radii
    # PDF = 2x; CDF = x²; CDF⁻¹ = √x
    radii = R * sqrt.(rand(N))

    # Sample angles
    # theta = 0 - 2π
    thetas = rand(N) * 2π

    # points
    X = polar2cart.(radii, thetas)
    return X
end


"""Compute all pairwise distances between each point in X
Store in condensed distance matrix form (flattening of upper triangle)

# Inputs
| X: n (observations) x m (dimensions) matrix
| f(a, b): function to compute distances between a, b
"""
function pairwiseDistance(X, f)
    n = size(X)[1]
    dists = zeros(n, n)
    for i in 1:n
        a = X[i]
        for j in i:n
            b = X[j]
            # Compute distance
            d = f(a, b)
            # Save
            dists[i, j] = dists[j, i] = d
        end
```

```
    end
    return dists
end

"""
Compute Euclidean distance between a and b
"""
euclidDist(a, b) = norm(a - b)

"""
Sample n points from a PDF, p
"""
function sampleDiscrete(n, p)
    # Compute cdf (Normalize to sum to 1)
    cdf = cumsum(p) / sum(p)
    # Sample
    choices = [searchsortedfirst(cdf, rand()) for _ in 1:n]
end

"""
Initialize model with N points in a radius R
"""
function initialize(N, R)
    # Initialize points
    X = sampleCircle(N, R)

    # Initialize directions
    θ = rand(N) * 2pi
    # Convert to polar
    θ = polar2cart.(1, θ)

    # Initialize velocities
    v = rand(N) .> .5

    return X, v, θ
end

"""
Perform one step of the provided model
"""
function stepModel(θ, v, X, p)
    # Select new v and ϑ
    v, θ = phototaxis(θ, v, X, p)

    # Update
    old_X = X
    X += v .* θ

    # Enforce boundary conditions
    outside = norm.(X) .> R
    X[outside] = old_X[outside]
    return X, v, θ
end

"""
Update step for phototaxis model

Local interactions: p["leader"] = zeros(N)
Global forcing model: p["leader"] != zeros(N) AND last point in X is light location

# Inputs
```

```
| θ: Vector of directions (polar coordinates) for each point
| v: Vector of velocities for each point (0 or 1)
| X: Array{Array{Float64, 2}, 1} of points
| p: Dictionary of parameters
    a: P[not changing direction]
    b: P[not changing velocity]
    c: P[moving toward light]
    D: Radius of interaction
    leader: vector of leader labels (0 or 1)
"""
function phototaxis(θ, v, X, p)
    N = size(X)[1]
    """Compute pairwise distances"""
    dists = pairwiseDistance(X, euclidDist)

    """Get new direction"""
    # Find local interactions for each point
    B = dists .< p["D"]
    # Remove self from B
    B[diagind(B)] .= 0

    # Number of neighbhors
    nB = sum(B, dims=2)

    # Probability of changing towards neighbor for leaders
    p_leader = replace((1 - p["c"] - p["a"])./nB, Inf=>0)
    # Probability of changing towards neighbors for normals
    p_normal = replace((1 - p["a"])./nB, Inf=>0)
    # Probability of orienting towards neighbor for each cell
    p_neighbor = @.p["leader"] * p_leader + (1 - p["leader"]) * p_normal

    #  P[swithing to neighbor j]
    PDFs = B .* p_neighbor

    # If any leaders (light is in model) assume last point in X is light location
    if 1 in p["leader"]
        # Add P[toward light] = c.
        PDFs[:, end] .= p["c"] * p["leader"]

        # Set light source to stay in place
        PDFs[end, :] .= 0
    end

    # [same direction] = a = 1 - not same direction;
    PDFs[diagind(PDFs)] = 1 .- sum(PDFs, dims=2)

    # Sample new direction for each point
    new_X_idx = [sampleDiscrete(1, pdf)[1] for pdf = eachrow(PDFs)]

    # Get coresponding point
    new_X = X[new_X_idx]
    # Get new direction (X + new_dir = new_X => new_dir = new_X - X)
    new_dir = new_X - X
    new_dir_mag = norm.(new_X - X)

    # For points without neighbors, θ stays same, otherwise update direction
    new_θ = [new_dir_mag[i] == 0 ? θ[i] : new_dir[i]/new_dir_mag[i] for i=1:N]

    """Get new velocity"""
    change_v = rand(N) .< p["b"]
    new_v = [change_v[i] ? v[i] : Bool(1-v[i]) for i in 1:N]
```

```
    # Boundary condition: avoid overlap by not moving if heading towards cell one away
    not_one_step = [dists[i, n]!=1 for (i, n) = enumerate(new_X_idx)]
    new_v .*= not_one_step

    return new_v, new_θ
end

"""Parameters"""
p = Dict{Any, Any}("a"=>.8, "b"=>.5, "c"=>.005)
```

Out[2]: p

Local interactions - Fig 4

At any given point in time, we assume that every cell may move according to one of the following three options (shown in Figure 2):

1. a cell may continue to move without changing its previous direction
2. a cell can stop moving
3. a cell may orient itself. In this case it will moves in the direction of one of its neighboring cells. The candidate neighboring cells must be within a certain interaction distance (shown as the dotted circle in Figure 2(c)).

In [3]:

```
@time begin
    # Save nested dictionaries by D then N
    data = Dict()
    R = 250

    # Times to run to and save
    tf = 350
    save_ts = [50, 350]

    # Parameters to loop over
    Ds = [5, 10]
    Ns = [250, 500]
    for D = Ds
        D_data = Dict()
        p["D"] = D
        for N = Ns
            # Initialize
            X, v, θ = initialize(N, R)

            # Local interactions model = no leaders
            p["leader"] = zeros(N)

            # Run simulation
            Xs = [X]

            for t = 1:(tf+1)
                # Run one step of model
                X, v, θ = stepModel(θ, v, X, p)

                # Save X at times of interes
                if t in save_ts
```

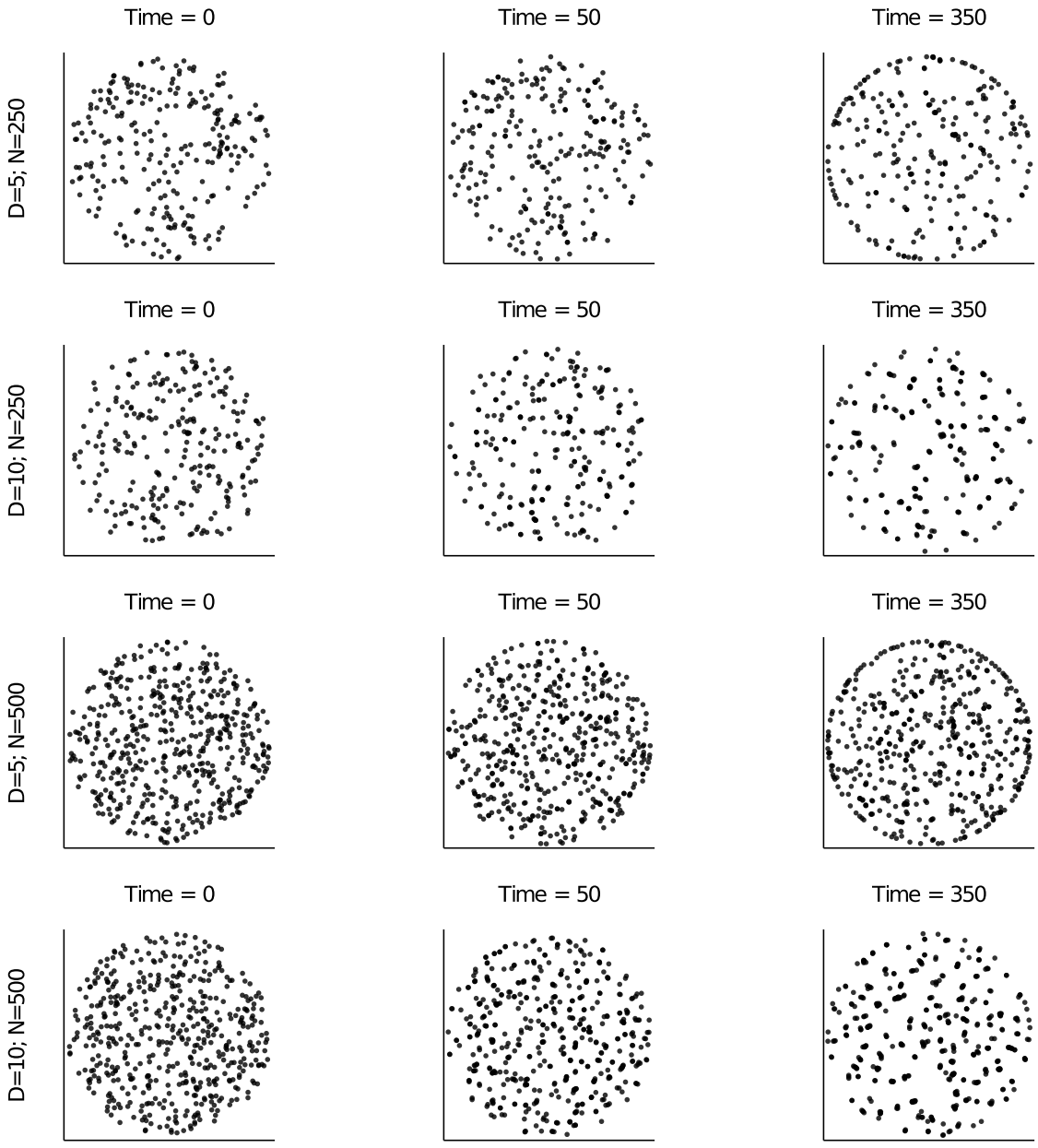
```
                push!(Xs, X)
            end
            if t == tf
                break
            end

            # Recompute pairwise distances
            dists = pairwiseDistance(X, euclidDist)
        end
        # Save simulations for this N
        D_data[N] = Xs
    end
    # Save simulations for this D
    data[D] = D_data
end
end
```

```
87.949740 seconds (233.97 M allocations: 28.792 GiB, 7.69% gc time)
```

In [5]:

```
plts = []
for N = Ns
    for D = Ds
        d = data[D][N]
        for (x, t) in zip(d, [0, 50, 350])
            # Convert data to matrix
            Xmat = vcat(x...)
            # Scatter points
            plt = scatter(Xmat[:, 1], Xmat[:, 2], aspect_ratio=:equal, legend=false, ma
            title!("Time = $t")
            # Add details to left y-axis
            if t == 0
                ylabel!("D=$D; N=$N")
            end
            push!(plts, plt)
        end
    end
end
# Show
lims = (-R-10, R+10)
plt = plot(plts..., titlefontsize=10, yguidefontsize=10, layout=(4, 3), size=(800, 800)
# savefig(plt, "results/fig4")
plt
```

Out[5]:



Global Forcing Model A - Fig 5

Local interaction model with influence of a global attractor that all cells are sensitive to.

In [7]:

```
@time begin
    # Save nested dictionaries by D then N
    data = Dict()
    R = 250
    # Location of light source
    θ₀ = polar2cart(R, -pi/4)

    # Times to run to and save
    tf = 1000
    save_ts = [500, 1000]

    # Parameters to loop over
    Ds = [5, 10]
```

```
    Ns = [250, 500]
    for D = Ds
        D_data = Dict()
        p["D"] = D
        for N = Ns
            # Initialize
            X, v, θ = initialize(N, R)
            # Add light source as final point
            push!(X, θ₀)
            push!(θ, [0 0])
            push!(v, 0)

            # Global forcing model A - all leaders
            p["leader"] = ones(N+1)
            # Run simulation
            Xs = [X]

            for t = 1:(tf+1)
                # Run one step of model
                X, v, θ = stepModel(θ, v, X, p)

                # Save X at times of interes
                if t in save_ts
                    push!(Xs, X)
                end
                if t == tf
                    break
                end

                # Recompute pairwise distances
                dists = pairwiseDistance(X, euclidDist)
            end
            # Save simulations for this N
            D_data[N] = Xs
        end
        # Save simulations for this D
        data[D] = D_data
    end
end
```

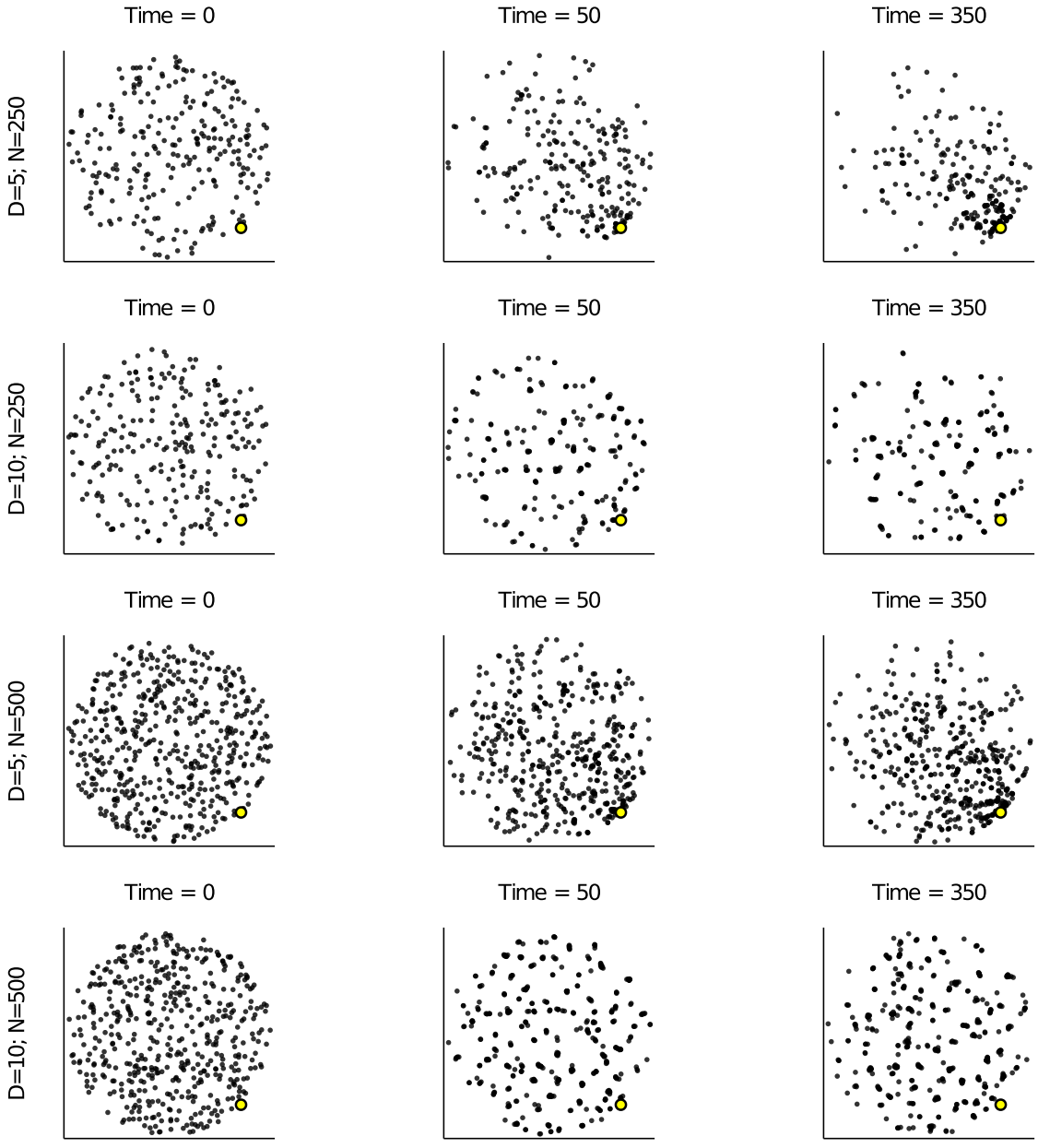
```
154.580707 seconds (641.15 M allocations: 81.149 GiB, 10.56% gc time)
```

In [8]:

```
plts = []
for N = Ns
    for D = Ds
        d = data[D][N]
        for (x, t) in zip(d, [0, 50, 350])
            # Convert data to matrix
            Xmat = vcat(x...)
            # Scatter points
            plt = scatter(Xmat[:, 1], Xmat[:, 2], aspect_ratio=:equal, legend=false, ma
            # Add light
            scatter!([θ₀[1]],[θ₀[2]], color=:yellow)
            title!("Time = $t")
            # Add details to left y-axis
            if t == 0
                ylabel!("D=$D; N=$N")
            end
            push!(plts, plt)
        end
```

```
    end
end
# Show
lims = (-R-10, R+10)
plt = plot(plts..., titlefontsize=10, yguidefontsize=10, layout=(4, 3), size=(800, 800))
# savefig(plt, "results/fig5")
plt
```

Out[8]:



Global forcing B - Fig 6

In this model, only a subset of cells are capable of sensing the direc- tion of the light source. To model this, cells are randomly assigned to be of type leader or type normal. Cells of the leader subtype update directions according to equation (4). Cells of the normal subtype update directions according to equation (1). The ratio of leader to normal cells is a model parameter. Its effect is studied in the numerical simulations.

In [9]:

```
@time begin
    # Save nested dictionaries by D then N
    data = Dict()
    R = 250
    # Location of light source
    θ₀ = polar2cart(R, -pi/4)

    # Times to run to and save
    tf = 1000
    save_ts = [500, 1000]

    # Parameters to loop over
    Ds = [5, 10]
    Ns = [250, 500]
    for D = Ds
        p["D"] = D
        D_data = Dict()
        for N = Ns
            # Initialize
            X, v, θ = initialize(N, R)
            # Add light source as final point
            push!(X, θ₀)
            push!(θ, [0 0])
            push!(v, 0)

            # Global forcing model B - all leaders
            p["leader"] = rand(N+1) .> .5
            # Run simulation
            Xs = [X]

            for t = 1:(tf+1)
                # Run one step of model
                X, v, θ = stepModel(θ, v, X, p)

                # Save X at times of interes
                if t in save_ts
                    push!(Xs, X)
                end
                if t == tf
                    break
                end

                # Recompute pairwise distances
                dists = pairwiseDistance(X, euclidDist)
            end
            # Save simulations for this N along with leader profile
            D_data[N] = Dict("data"=>Xs, "leader"=>p["leader"])
        end
        # Save simulations for this D
        data[D] = D_data
    end
end
```

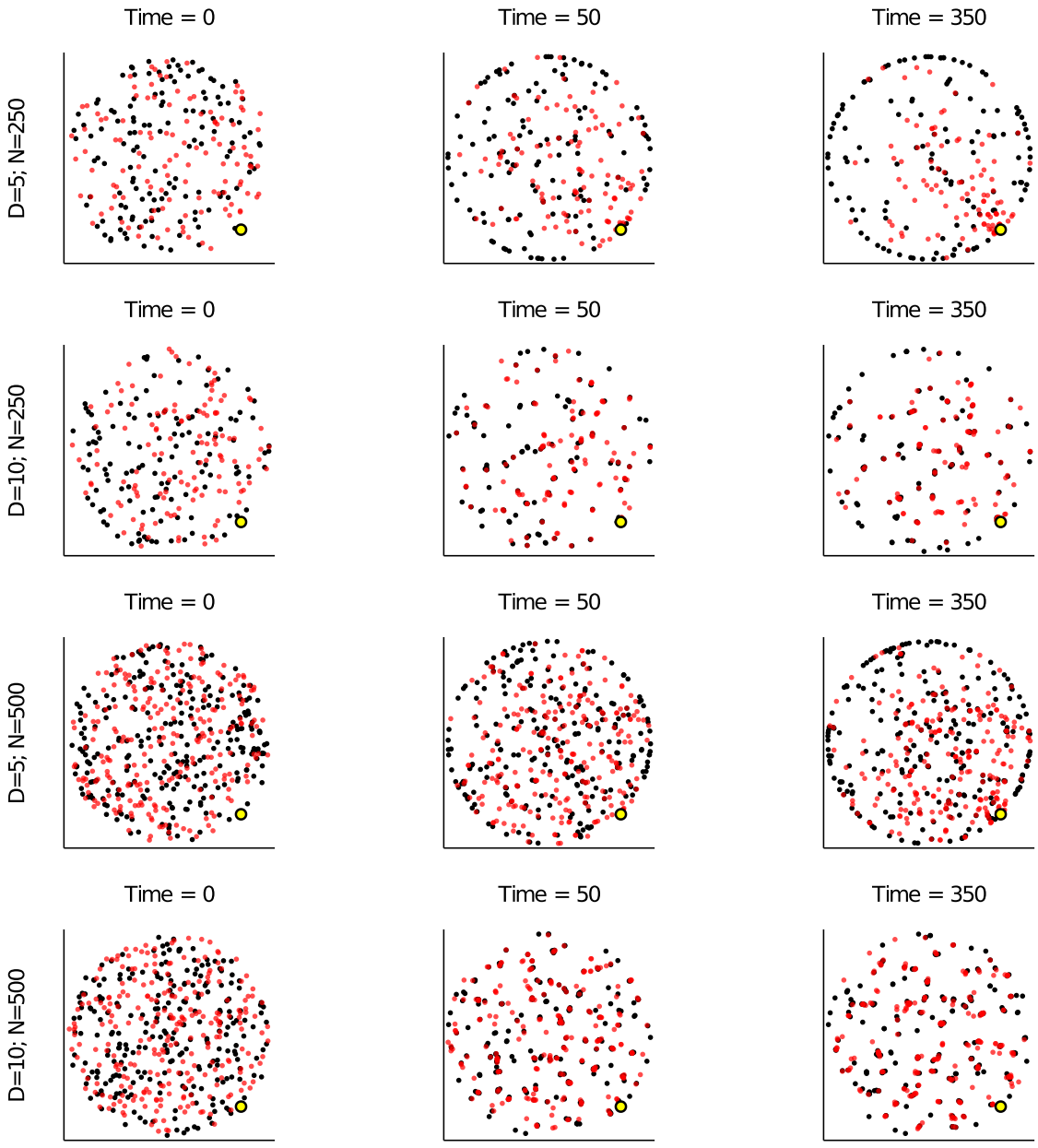
```
160.766933 seconds (643.67 M allocations: 81.270 GiB, 11.25% gc time)
```

In [10]:

```
plts = []
for N = Ns
    for D = Ds
        d = data[D][N]["data"]
        l = data[D][N]["leader"]
```

```
        for (x, t) in zip(d, [0, 50, 350])
            # Convert data to matrix
            Xmat = vcat(x...)
            plt = plot()
            for (label, name, color, alpha) in zip([0, 1], ["normal", "leader"], ["blac
                # Subset to relevant points
                points = Xmat[findall(l .== label), :]
                scatter!(points[:, 1], points[:, 2], aspect_ratio=:equal, legend=false,
            end
            # Add light
            scatter!([θ₀[1]],[θ₀[2]], color=:yellow)
            title!("Time = $t")
            # Add details to left y-axis
            if t == 0
                ylabel!("D=$D; N=$N")
            end
            push!(plts, plt)
        end
    end
end
# Show
lims = (-R-10, R+10)
plt = plot(plts..., titlefontsize=10, yguidefontsize=10, layout=(4, 3), size=(800, 800)
# savefig(plt, "results/fig6")
plt
```

Out[10]:



Long simulation - Fig 7

In [11]:

```
@time begin
    # Save nested dictionaries by D then N
    data = Dict()
    R = 250
    # Location of light source
    θ₀ = polar2cart(R, -pi/4)

    # Times to run to and save
    tf = 9999
    save_ts = [3000, tf]

    # Parameters to loop over
    Ds = [5, 10]
    N = 250
    for (model, model_name) = zip([x->zeros(x), x->ones(x + 1), x->rand(x + 1) .> .5],
        model_data = Dict()
```

```
        for D = Ds
            p["D"] = D
            D_data = Dict()
            # Initialize
            X, v, θ = initialize(N, R)
            if model_name in ["A", "B"]
                # Add light source as final point
                push!(X, θ₀)
                push!(θ, [0 0])
                push!(v, 0)
            end

            # Global forcing model B - all leaders
            p["leader"] = model(N)

            # Run simulation
            Xs = [X]

            for t = 1:(tf+1)
                # Run one step of model
                X, v, θ = stepModel(θ, v, X, p)

                # Save X at times of interes
                if t in save_ts
                    push!(Xs, X)
                end
                    if t == tf
                break
            end

            # Recompute pairwise distances
            dists = pairwiseDistance(X, euclidDist)
            # Save simulations for this D
            model_data[D] = Dict("data"=>Xs, "leader"=>p["leader"])
            end
            # Save simulations for this model
            data[model_name] = model_data
        end
    end
end
```

```
915.675271 seconds (3.90 G allocations: 493.752 GiB, 11.74% gc time)
```

In [12]:

```
plts = []
for D = [5, 10]
    for (t, t_name) = zip([2, 3], [3000, 9999])
        for model = ["local", "A", "B"]
            x = data[model][D]["data"][t]
            Xmat = vcat(x...)
            leaders = data[model][D]["leader"]
            plt = plot()
            for (label, name, color, alpha) in zip([0, 1], ["normal", "leader"], ["blac
                # Subset to relevant points
                points = Xmat[findall(leaders .== label), :]
                scatter!(points[:, 1], points[:, 2], aspect_ratio=:equal, legend=false,
            end
            # Add light
            scatter!([θ₀[1]],[θ₀[2]], markersize=1.5, color=:yellow, markerstrokecolor=
            title!("Model = $model")
            # Add details to left y-axis
```

```
            if model == "local"
                ylabel!("D=$D; T=$t_name")
            end
            push!(plts, plt)
        end
    end
end
# Show
lims = (-R-10, R+10)
plt = plot(plts..., titlefontsize=10, yguidefontsize=10, layout=(4, 3), size=(800, 800))
savefig(plt, "results/fig7")
plt
```

Out[12]:

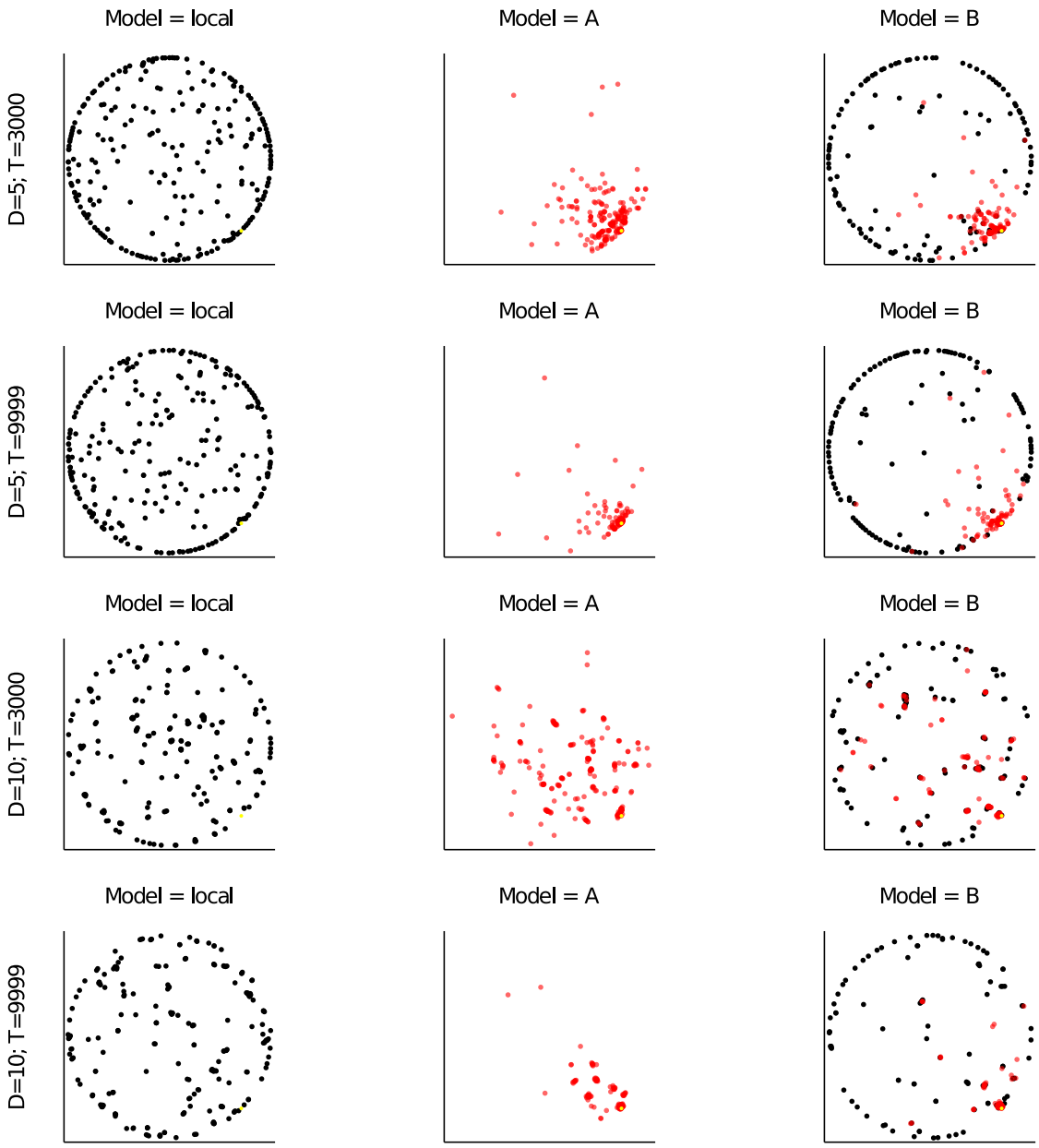


Figure 8

In [33]:

```
@time begin
    # Save data
    data = Dict()
```

```
    R = 250
    # Location of light source
    θ₀ = polar2cart(R, -pi/4)

    # Times to run to and save
    tf = 3000
    save_ts = [1500, tf]

    # Parameters to loop over
    p["D"] = 5
    N = 250
    for model = [1/4, .5, 3/4, 1]
        # Global forcing model B - all leaders
        p["leader"] = rand(N+1) .< model
        # Initialize
        X, v, θ = initialize(N, R)

        # Add light source as final point
        push!(X, θ₀)
        push!(θ, [0 0])
        push!(v, 0)
        # Run simulation
        Xs = [X]
        for t = 1:(tf+1)
            # Run one step of model
            X, v, θ = stepModel(θ, v, X, p)

            # Save X at times of interes
            if t in save_ts
                push!(Xs, X)
            end
            if t == tf
                break
            end
            # Recompute pairwise distances
            dists = pairwiseDistance(X, euclidDist)
        end
        # Save simulations for this D
        data[model] = Dict("data"=>Xs, "leader"=>p["leader"])
    end
end
```

```
110.859762 seconds (781.67 M allocations: 98.949 GiB, 15.29% gc time)
```

In [68]:

```
plts = []
for model = [1/4, .5, 3/4, 1]
    d = data[model]
    xs = d["data"]
    leader = d["leader"]
    for (t, x) = zip([0, 1500, 3000], xs)
        plt = plot()
        for (l, color, alpha) = zip([0, 1], ["black", "red"], [1, .6])
            pts = x[findall(leader.==l)]
            if !isempty(pts)
                Xmat = vcat(pts...)
                scatter!(Xmat[:, 1], Xmat[:, 2], color=color, markeralpha=alpha, marker
            end
        end
        title!("T=$t")
        if t == 0
```

```
            ylabel!("Leaders=$model")
        end
        push!(plts, plt)
    end
end
lims = (-R-10, R+10)
plt = plot(plts..., titlefontsize=10, yguidefontsize=10, layout=(4, 3), size=(800, 800))
savefig(plt, "results/fig8")
plt
```

Out[68]: