Agent-based model using Agents.jl

Simon Frost (@sdwfrost), 2020-04-27

Introduction

The agent-based model approach, implemented using `Agents.jl` taken here is:

- Stochastic
- Discrete in time
- Discrete in state

Libraries

```
using Agents
using Random
using DataFrames
using Distributions
using DrWatson
using StatsPlots
using BenchmarkTools
```

Utility functions

```
function rate_to_proportion(r::Float64,t::Float64)
    1-exp(-r*t)
end;
```

```
rate_to_proportion (generic function with 1 method)
```

Transitions

First, we have to define our agent, which has an `id`, and a `status` (`:S`,`:I`, or `:R`).

```
mutable struct Person <: AbstractAgent
    id::Int64
    status::Symbol
end
```

This utility function sets up the model, by setting parameter fields and adding agents to the model.

```
function init_model(β::Float64,c::Float64,γ::Float64,N::Int64,I0::Int64)
    properties = @dict(β,c,γ)
    model = ABM(Person; properties=properties)
    for i in 1:N
        if i <= I0
            s = :I
        else
            s = :S
        end
        p = Person(i,s)
```

```
        p = add_agent!(p,model)
    end
    return model
end;
```

```
init_model (generic function with 1 method)
```

The following function applies a series of functions to each agent.

```
function agent_step!(agent, model)
    transmit!(agent, model)
    recover!(agent, model)
end;
```

```
agent_step! (generic function with 1 method)
```

This is the transmission function; note that it operates on susceptibles making contact, rather than being focused on infected. This is an inefficient way of doing things, but shows the parallels between the different implementations.

```
function transmit!(agent, model)
    # If I'm not susceptible, I return
    agent.status != :S && return
    ncontacts = rand(Poisson(model.properties[:c]))
    for i in 1:ncontacts
        # Choose random individual
        alter = random_agent(model)
        if alter.status == :I && (rand() ≤ model.properties[:β])
            # An infection occurs
            agent.status = :I
            break
        end
    end
end;
```

```
transmit! (generic function with 1 method)
```

This is the recovery function.

```
function recover!(agent, model)
    agent.status != :I && return
    if rand() ≤ model.properties[:γ]
            agent.status = :R
    end
end;
```

```
recover! (generic function with 1 method)
```

We need some reporting functions.

```
susceptible(x) = count(i == :S for i in x)
infected(x) = count(i == :I for i in x)
recovered(x) = count(i == :R for i in x);
```

```
recovered (generic function with 1 method)
```

Time domain

```
δt = 0.1
nsteps = 400
tf = nsteps*δt
t = 0:δt:tf;
```

```
0.0:0.1:40.0
```

Parameter values

```
β = 0.05
c = 10.0*δt
γ = rate_to_proportion(0.25,δt);
```

```
0.024690087971667385
```

Initial conditions

```
N = 1000
I0 = 10;
```

```
10
```

Random number seed

```
Random.seed!(1234);
```

```
MersenneTwister(UInt32[0x000004d2], Random.DSFMT.DSFMT_state(Int32[-1393240
018, 1073611148, 45497681, 1072875908, 436273599, 1073674613, -2043716458,
1073445557, -254908435, 1072827086  ...  -599655111, 1073144102, 367655457, 1
072985259, -1278750689, 1018350124, -597141475, 249849711, 382, 0]), [0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0  ...  0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0], UInt128[0x00000000000000000000000000000000, 0x00000
000000000000000000000000000, 0x00000000000000000000000000000000, 0x00000000
000000000000000000000000, 0x00000000000000000000000000000000, 0x00000000000
000000000000000000000, 0x00000000000000000000000000000000, 0x00000000000000
000000000000000000, 0x00000000000000000000000000000000, 0x0000000000000000
000000000000000  ...  0x00000000000000000000000000000000, 0x00000000000000000
000000000000000, 0x00000000000000000000000000000000, 0x0000000000000000000
000000000000, 0x00000000000000000000000000000000, 0x0000000000000000000000
000000000, 0x00000000000000000000000000000000, 0x0000000000000000000000000
000000, 0x00000000000000000000000000000000, 0x0000000000000000000000000000
000], 1002, 0)
```

Running the model

```
abm_model = init_model(β,c,γ,N,I0)
```

```
AgentBasedModel with 1000 agents of type Person
 no space
 scheduler: fastest
 properties: Dict(:γ => 0.024690087971667385,:c => 1.0,:β => 0.05)
```

```
to_collect = [(:status, f) for f in (susceptible, infected, recovered)]
abm_data, _ = run!(abm_model, agent_step!, nsteps; adata = to_collect);
```

```
(401×4 DataFrame
 Row   step    susceptible_status   infected_status   recovered_status
       Int64   Int64                Int64             Int64


 1     0       990                  10                0
 2     1       990                  10                0
 3     2       990                  9                 1
 4     3       990                  9                 1
 5     4       990                  9                 1
 6     5       990                  9                 1
 7     6       990                  9                 1
⋮
 394   393     234                  20                746
 395   394     234                  20                746
 396   395     233                  20                747
 397   396     233                  19                748
 398   397     233                  18                749
 399   398     233                  18                749
 400   399     233                  18                749
 401   400     233                  17                750               ,
0×0 DataFrame
)
```

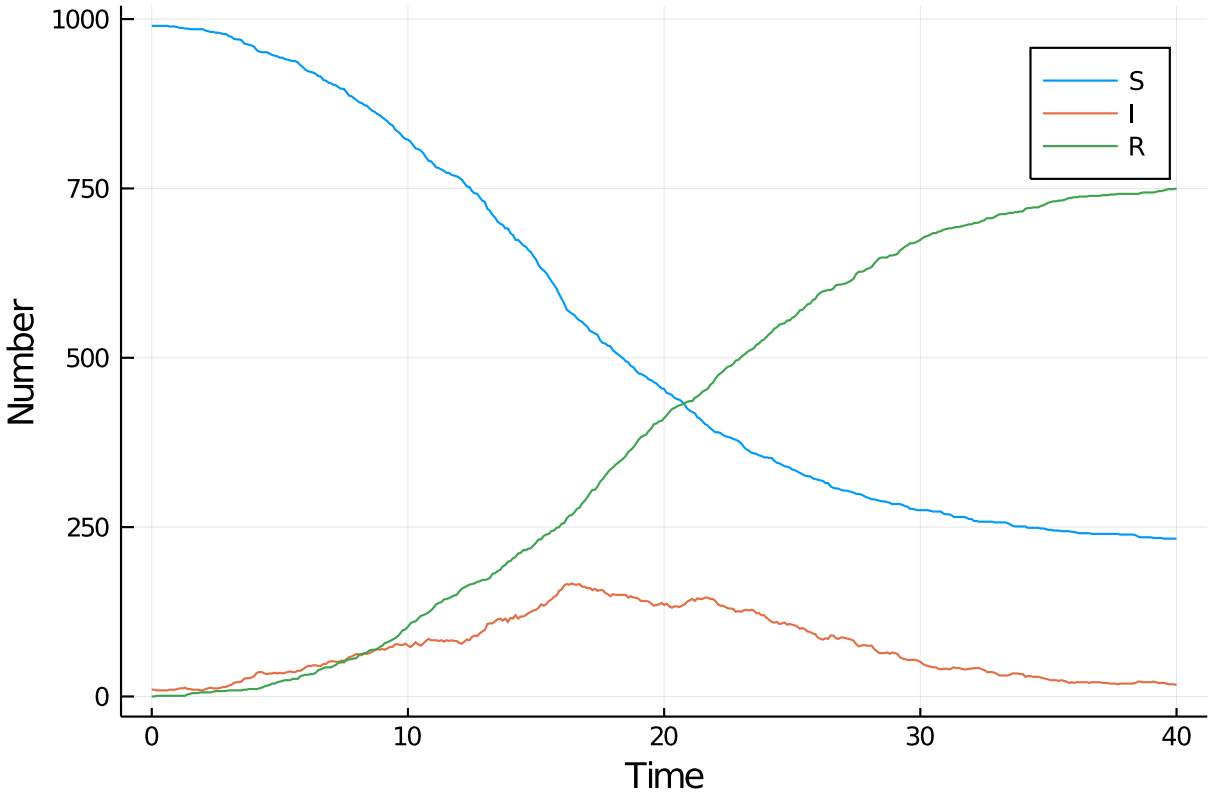
Post-processing

```
abm_data[!,:t] = t;
```

```
0.0:0.1:40.0
```

Plotting

```
plot(t,abm_data[:,2],label="S",xlab="Time",ylabel="Number")
plot!(t,abm_data[:,3],label="I")
plot!(t,abm_data[:,4],label="R")
```



Benchmarking

```
@benchmark begin
abm_model = init_model(β,c,γ,N,I0)
abm_data, _ = run!(abm_model, agent_step!, nsteps; adata = to_collect)
end
```

```
BenchmarkTools.Trial:
  memory estimate:  3.23 MiB
  allocs estimate:  200538
  --------------
  minimum time:     422.242 ms (0.00% GC)
  median time:      476.715 ms (0.00% GC)
  mean time:        497.457 ms (0.00% GC)
  maximum time:     678.454 ms (0.00% GC)
  --------------
  samples:          11
  evals/sample:     1
```

Appendix

Computer Information

```
Julia Version 1.4.1
Commit 381693d3df* (2020-04-14 17:20 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
```

```
  LLVM: libLLVM-8.0.1 (ORCJIT, icelake-client)
Environment:
  JULIA_NUM_THREADS = 4
```

Package Information

```
Status `~/.julia/environments/v1.4/Project.toml`
[46ada45e-f475-11e8-01d0-f70cc89e6671] Agents 3.1.0
[c52e3926-4ff0-5f6e-af25-54175e0327b1] Atom 0.12.11
[6e4b80f9-dd63-53aa-95a3-0cdb28fa8baf] BenchmarkTools 0.5.0
[a134a8b2-14d6-55f6-9291-3336d3ab0209] BlackBoxOptim 0.5.0
[2445eb08-9709-466a-b3fc-47e12bd697a2] DataDrivenDiffEq 0.2.0
[a93c6f00-e57d-5684-b7b6-d8193f3e46c0] DataFrames 0.21.0
[ebbdde9d-f333-5424-9be2-dbf1e9acfb5e] DiffEqBayes 2.14.0
[459566f4-90b8-5000-8ac3-15dfb0a30def] DiffEqCallbacks 2.13.2
[c894b116-72e5-5b58-be3c-e6d8d4ac2b12] DiffEqJump 6.7.5
[1130ab10-4a5a-5621-a13d-e4788d82bd4c] DiffEqParamEstim 1.14.1
[0c46a032-eb83-5123-abaf-570d42b7fbaa] DifferentialEquations 6.14.0
[31c24e10-a181-5473-b8eb-7969acd0382f] Distributions 0.23.2
[634d3b9d-ee7a-5ddf-bec9-22491ea816e1] DrWatson 1.11.0
[587475ba-b771-5e3f-ad9e-33799f191a9c] Flux 0.8.3
[28b8d3ca-fb5f-59d9-8090-bfdbd6d07a71] GR 0.49.1
[523d8e89-b243-5607-941c-87d699ea6713] Gillespie 0.1.0
[7073ff75-c697-5162-941a-fcdaad2a7d2a] IJulia 1.21.2
[4076af6c-e467-56ae-b986-b466b2749572] JuMP 0.21.2
[e5e0dc1b-0480-54bc-9374-aad01c23163d] Juno 0.8.2
[093fc24a-ae57-5d10-9952-331d41423f4d] LightGraphs 1.3.3
[1914dd2f-81c6-5fcd-8719-6d5c9610ff09] MacroTools 0.5.5
[ee78f7c6-11fb-53f2-987a-cfe4a2b5a57a] Makie 0.9.5
[961ee093-0014-501f-94e3-6117800e7a78] ModelingToolkit 3.6.0
[76087f3c-5699-56af-9a33-bf431cd00edd] NLopt 0.6.0
[429524aa-4258-5aef-a3af-852621145aeb] Optim 0.21.0
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.38.1
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 1.3.1
[428bdadb-6287-5aa5-874b-9969638295fd] SimJulia 0.8.0
[05bca326-078c-5bf0-a5bf-ce7c7982d7fd] SimpleDiffEq 1.1.0
[f3b207a7-027a-5e70-b257-86293d7955fd] StatsPlots 0.14.6
[789caeaf-c7a9-5a7d-9973-96adeb23e2a0] StochasticDiffEq 6.23.0
[fce5fe82-541a-59a6-adf8-730c64b5f9a0] Turing 0.12.0
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.10.0
```