Agent-based model using DifferentialEquations

Simon Frost (@sdwfrost), 2020-05-03

Introduction

The agent-based model approach is:

- Stochastic
- Discrete in time
- Discrete in state

There are multiple ways in which the model state can be updated. In this implementation, there is the initial state, `u`, and the next state, `u`, and updates occur by looping through all the agents (in this case, just a vector of states), and determining whether a transition occurs each state. This approach is relatively simple as there is a chain of states that an individual passes through (i.e. only one transition type per state). After all states have been updated in `du`, they are then assigned to the current state, `u`.

Libraries

```
using DifferentialEquations
using DiffEqCallbacks
using Distributions
using StatsBase
```

```
Error: ArgumentError: Package StatsBase not found in current path:
- Run `import Pkg; Pkg.add("StatsBase")` to install the StatsBase package.
```

```
using Random
using DataFrames
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

As this is a simple model, the global state of the system is a vector of infection states, defined using an `@enum`.

```
@enum InfectionStatus Susceptible Infected Recovered
```

```
function sir_abm!(du,u,p,t)
    (β,c,γ,δt) = p
    N = length(u)
    # Initialize du to u
    for i in 1:N
        du[i] = u[i]
    end
    for i in 1:N # loop through agents
        # If recovered
        if u[i]==Recovered
            continue
        # If susceptible
        elseif u[i]==Susceptible
            ncontacts = rand(Poisson(c*δt))
            while ncontacts > 0
                j = sample(1:N)
                if j==i
                    continue
                end
                a = u[j]
                if a==Infected && rand() < β
                    du[i] = Infected
                    break
                end
                ncontacts -= 1
            end
        # If infected
        else u[i]==Infected
            if rand() < γ
                du[i] = Recovered
            end
        end
    end
    nothing
end;
```

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

Time domain

```
δt = 0.1
tf = 40.0
tspan = (0.0,tf);
```

```
(0.0, 40.0)
```

Parameter values

```
β = 0.05
c = 10.0
γ = rate_to_proportion(0.25,δt)
p = [β,c,γ,δt]
```

```
4-element Array{Float64,1}:
  0.05
 10.0
  0.024690087971667385
  0.1
```

Initial conditions

```
N = 1000
I0 = 10
u0 = Array{InfectionStatus}(undef,N)
for i in 1:N
    if i <= I0
        s = Infected
    else
        s = Susceptible
    end
    u0[i] = s
end
```

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, 0x00000000000000000
000000000000000  ...  0x00000000000000000000000000000000, 0x00000000000000000
000000000000000, 0x00000000000000000000000000000000, 0x00000000000000000000
000000000000, 0x00000000000000000000000000000000, 0x00000000000000000000000
000000000, 0x00000000000000000000000000000000, 0x00000000000000000000000000
000000, 0x00000000000000000000000000000000, 0x00000000000000000000000000000
000], 1002, 0)
```

Running the model

We need some reporting functions.

```
susceptible(u) = count(i == Susceptible for i in u)
infected(u) = count(i == Infected for i in u)
recovered(u) = count(i == Recovered for i in u);
```

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

```
saved_values = SavedValues(Float64, Tuple{Int64,Int64,Int64})
cb = SavingCallback((u,t,integrator)->(susceptible(u),infected(u),recovered(u)),
    saved_values,
    saveat=0:δt:tf)
```

```
DiscreteCallback{DiffEqCallbacks.var"#30#31",DiffEqCallbacks.SavingAffect{M
ain.##WeaveSandBox#420.var"#7#8",Float64,Tuple{Int64,Int64,Int64},DataStruc
tures.BinaryHeap{Float64,DataStructures.LessThan},Array{Float64,1}},typeof(
DiffEqCallbacks.saving_initialize)}(DiffEqCallbacks.var"#30#31"(), DiffEqCa
llbacks.SavingAffect{Main.##WeaveSandBox#420.var"#7#8",Float64,Tuple{Int64,
Int64,Int64},DataStructures.BinaryHeap{Float64,DataStructures.LessThan},Arr
ay{Float64,1}}(Main.##WeaveSandBox#420.var"#7#8"(), SavedValues{tType=Float
64, savevalType=Tuple{Int64,Int64,Int64}}
t:
```

```
Float64[]
saveval:
Tuple{Int64,Int64,Int64}[], DataStructures.BinaryHeap{Float64,DataStructure
s.LessThan}(DataStructures.LessThan(), [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
0.7, 0.8, 0.9  ...  39.1, 39.2, 39.3, 39.4, 39.5, 39.6, 39.7, 39.8, 39.9, 40.
0]), [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9  ...  39.1, 39.2, 39.3
, 39.4, 39.5, 39.6, 39.7, 39.8, 39.9, 40.0], false, true, 0), DiffEqCallbac
ks.saving_initialize, Bool[0, 0])
```

```julia
prob_abm = DiscreteProblem(sir_abm!,u0,tspan,p)
```

```
DiscreteProblem with uType Array{Main.##WeaveSandBox#420.InfectionStatus,1}
 and tType Float64. In-place: true
timespan: (0.0, 40.0)
u0: Main.##WeaveSandBox#420.InfectionStatus[Main.##WeaveSandBox#420.Infecte
d, Main.##WeaveSandBox#420.Infected, Main.##WeaveSandBox#420.Infected, Main
.##WeaveSandBox#420.Infected, Main.##WeaveSandBox#420.Infected, Main.##Weav
eSandBox#420.Infected, Main.##WeaveSandBox#420.Infected, Main.##WeaveSandBo
x#420.Infected, Main.##WeaveSandBox#420.Infected, Main.##WeaveSandBox#420.I
nfected  ...  Main.##WeaveSandBox#420.Susceptible, Main.##WeaveSandBox#420.Su
sceptible, Main.##WeaveSandBox#420.Susceptible, Main.##WeaveSandBox#420.Sus
ceptible, Main.##WeaveSandBox#420.Susceptible, Main.##WeaveSandBox#420.Susc
eptible, Main.##WeaveSandBox#420.Susceptible, Main.##WeaveSandBox#420.Susce
ptible, Main.##WeaveSandBox#420.Susceptible, Main.##WeaveSandBox#420.Suscep
tible]
```

```julia
sol_abm = solve(prob_abm,
    solver = FunctionMap(),
    dt = δt,
    callback = cb,
    dense = false,
    save_on = false);
```

```
retcode: Success
Interpolation: left-endpoint piecewise constant
t: 2-element Array{Float64,1}:
  0.0
 40.0
u: 2-element Array{Array{Main.##WeaveSandBox#420.InfectionStatus,1},1}:
 [Main.##WeaveSandBox#420.Infected, Main.##WeaveSandBox#420.Infected, Main.
##WeaveSandBox#420.Infected, Main.##WeaveSandBox#420.Infected, Main.##Weave
SandBox#420.Infected, Main.##WeaveSandBox#420.Infected, Main.##WeaveSandBox
#420.Infected, Main.##WeaveSandBox#420.Infected, Main.##WeaveSandBox#420.In
fected, Main.##WeaveSandBox#420.Infected  ...  Main.##WeaveSandBox#420.Suscep
tible, Main.##WeaveSandBox#420.Susceptible, Main.##WeaveSandBox#420.Suscept
ible, Main.##WeaveSandBox#420.Susceptible, Main.##WeaveSandBox#420.Suscepti
ble, Main.##WeaveSandBox#420.Susceptible, Main.##WeaveSandBox#420.Susceptib
le, Main.##WeaveSandBox#420.Susceptible, Main.##WeaveSandBox#420.Susceptibl
e, Main.##WeaveSandBox#420.Susceptible]
 [Main.##WeaveSandBox#420.Recovered, Main.##WeaveSandBox#420.Recovered, Mai
n.##WeaveSandBox#420.Recovered, Main.##WeaveSandBox#420.Recovered, Main.##W
eaveSandBox#420.Recovered, Main.##WeaveSandBox#420.Recovered, Main.##WeaveS
andBox#420.Recovered, Main.##WeaveSandBox#420.Recovered, Main.##WeaveSandBo
x#420.Recovered, Main.##WeaveSandBox#420.Recovered  ...  Main.##WeaveSandBox#
420.Susceptible, Main.##WeaveSandBox#420.Recovered, Main.##WeaveSandBox#420
.Recovered, Main.##WeaveSandBox#420.Recovered, Main.##WeaveSandBox#420.Susc
eptible, Main.##WeaveSandBox#420.Recovered, Main.##WeaveSandBox#420.Recover
ed, Main.##WeaveSandBox#420.Recovered, Main.##WeaveSandBox#420.Recovered, M
ain.##WeaveSandBox#420.Susceptible]
```

Post-processing

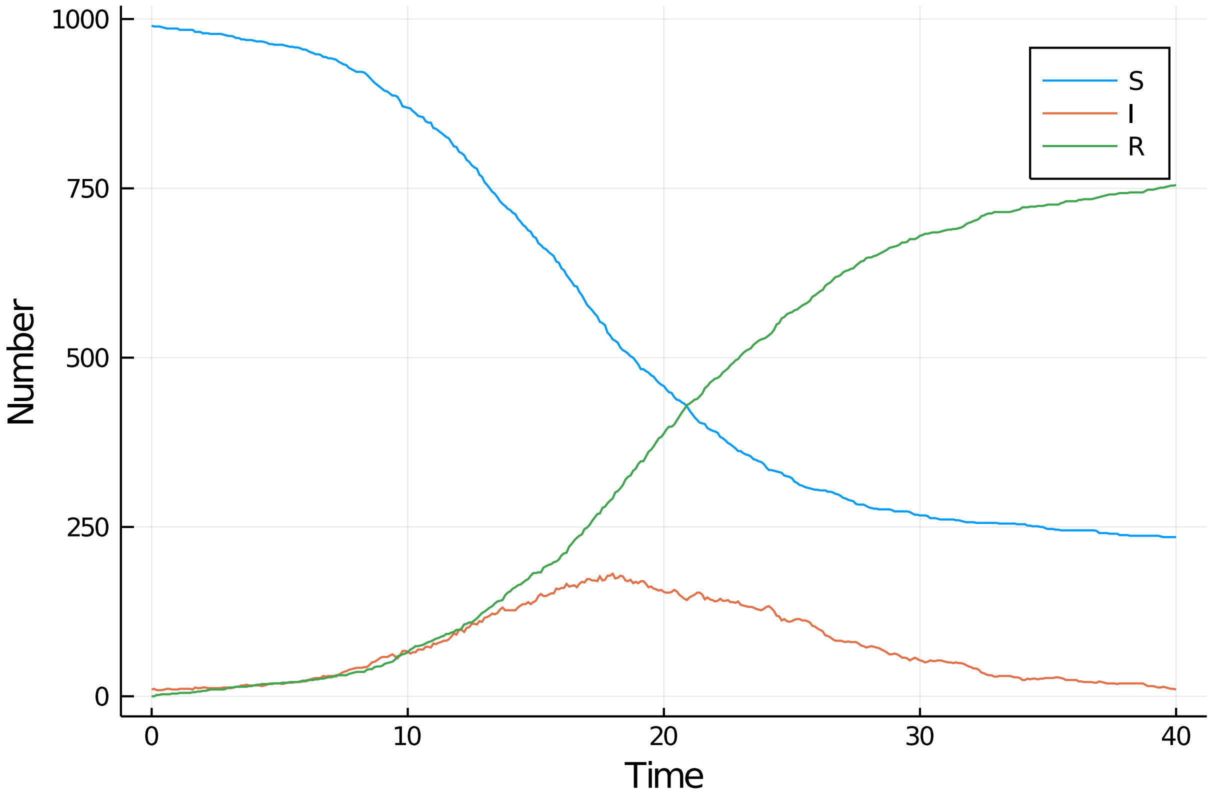
We can convert the output to a dataframe for convenience.

```
df_abm = DataFrame(saved_values.saveval)
rename!(df_abm,[:S,:I,:R])
df_abm[!,:t] = saved_values.t;
```

```
401-element Array{Float64,1}:
  0.0
  0.1
  0.2
  0.3
  0.4
  0.5
  0.6
  0.7
  0.8
  0.9
  ⋮
 39.2
 39.3
 39.4
 39.5
 39.6
 39.7
 39.8
 39.9
 40.0
```

Plotting

```
@df df_abm plot(:t,
    [:S :I :R],
    label=["S" "I" "R"],
    xlabel="Time",
    ylabel="Number")
```



Benchmarking

```
@benchmark solve(prob_abm,
    solver=FunctionMap,
    dt=δt,
    callback=cb,
    dense=false,
    save_on=false)
```

```
BenchmarkTools.Trial:
  memory estimate:  43.41 KiB
  allocs estimate:  79
  --------------
  minimum time:     46.057 ms (0.00% GC)
  median time:      59.772 ms (0.00% GC)
  mean time:        60.869 ms (0.00% GC)
  maximum time:     83.683 ms (0.00% GC)
  --------------
  samples:          83
  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
```