SoDA

Supercomputing with R part 2

Agent-based model of all neighbourhoods in the Netherlands

Erik-Jan van Kesteren

About me

Background

- PhD in **Statistics** (UU)
  Structural Equation Models, high-dimensional data, regularization & penalization, algorithms & optimization
- Assistant professor at **Methodology & Statistics**, UU
  Human Data Science group, teaching Data Science master courses
- Team lead for the **ODISSEI Social Data Science (SoDa) team**
  Advancing data- & computation-intensive research in social science

https://odissei-soda.nl/

SoDA
ODISSEI Social Data Science Team

Home About Projects Workshops Team Fellowship Tutorials Contact us

We help social scientists with data intensive & computational research

Our goal is to enhance the evidence base and impact of social science by bringing the added value of new data sources and new data analysis techniques into social research in the Netherlands

part of ODISSEI

Contact us Get more info →

About me

Relevant experience

- Experience with **parallel programming, supercomputing, large simulations**
  statistics, social sciences, a bit of neuroscience (structural MRI), and a bit of bioinformatics (microarrays, epigenetics)
- Native in **R**, capable in **Python**
  dabbled in C++, C#, web languages, Julia, and more
- Many research **consultations**
- Strongly advocating for **open science**
  Make everything available all the time!
- **Almost no experience with agent-based models!!!**

About you

Write down in one short sentence why you are here / what you hope to learn

The remainder of today

How to structure R projects for running analyses on a SURF supercomputer

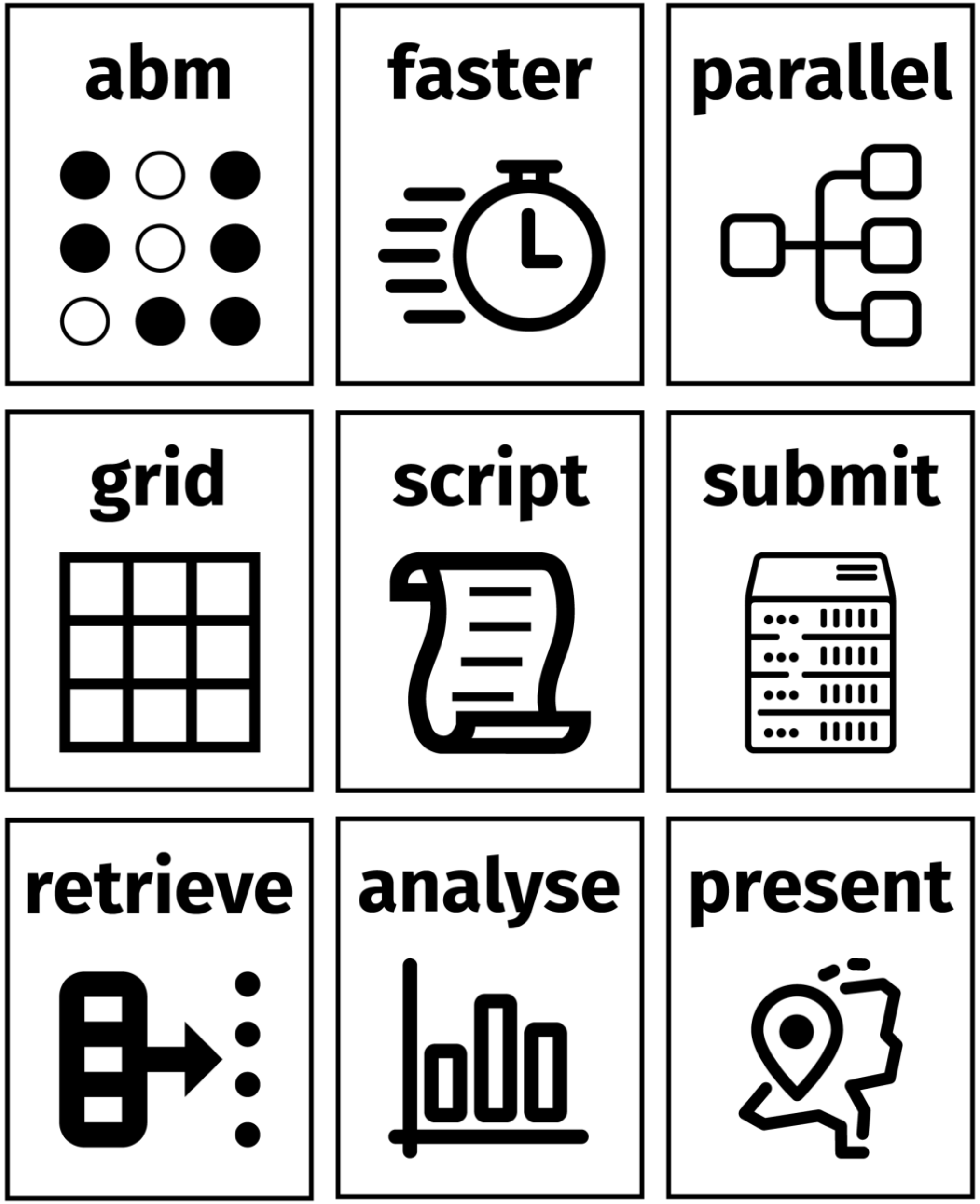
The remainder of today

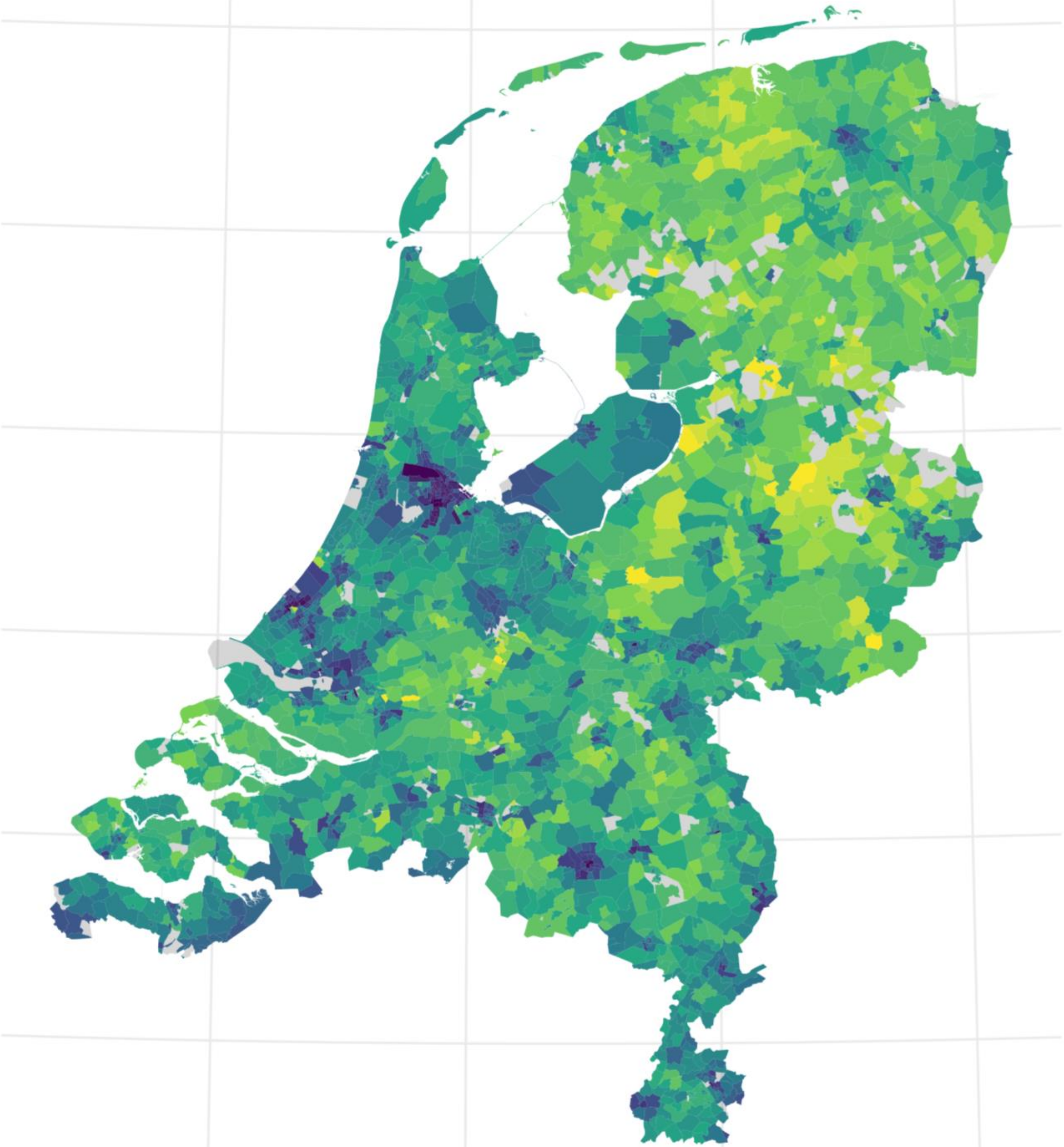
Time	Title
11:00	Lecture: computational limits in social science
11:45	Hands-on: a parallel agent-based model in R
12:30	Break
13:30	Lecture: supercomputing with R
14:15	Hands-on: submitting an R array job
15:15	Break
15:30	Lecture: combining & analysing the results
16:15	Conclusion & Q&A

Hands-on 1

Hands-on 2

Lecture 3





Computational limits in social science

Experimental research in soc. sci.

- Come up with research question
- Design experiment
- Run experiment
- Analyze results (perform statistical test)
- Make inferences about found effect

Observational research in soc. sci.

- Come up with research question
- Collect data
- Create statistical model
- Make inferences about model (pay attention to assumptions)

Computational research in soc. sci.?

- Come up with research question
- Create generative / computational model
- Generate data from computational model
- (compare computational model data with real data)
- Make conclusions about computational model (pay attention to assumptions)

Psych trend: theory construction

*A **formal model** captures the principles of the explanatory theory in a set of equations or rules (as **implemented in a computer program or simulation**).*

*The theorist can then examine whether the theory, as implemented in the formal model, does in fact **generate the phenomena** as a matter of course, either **in a simulation** study or through analytic derivations.*

Borsboom et al. (2021), Theory Construction Methodology: A Practical Framework for Building Theories in Psychology doi.org/10.1177/1745691620969647

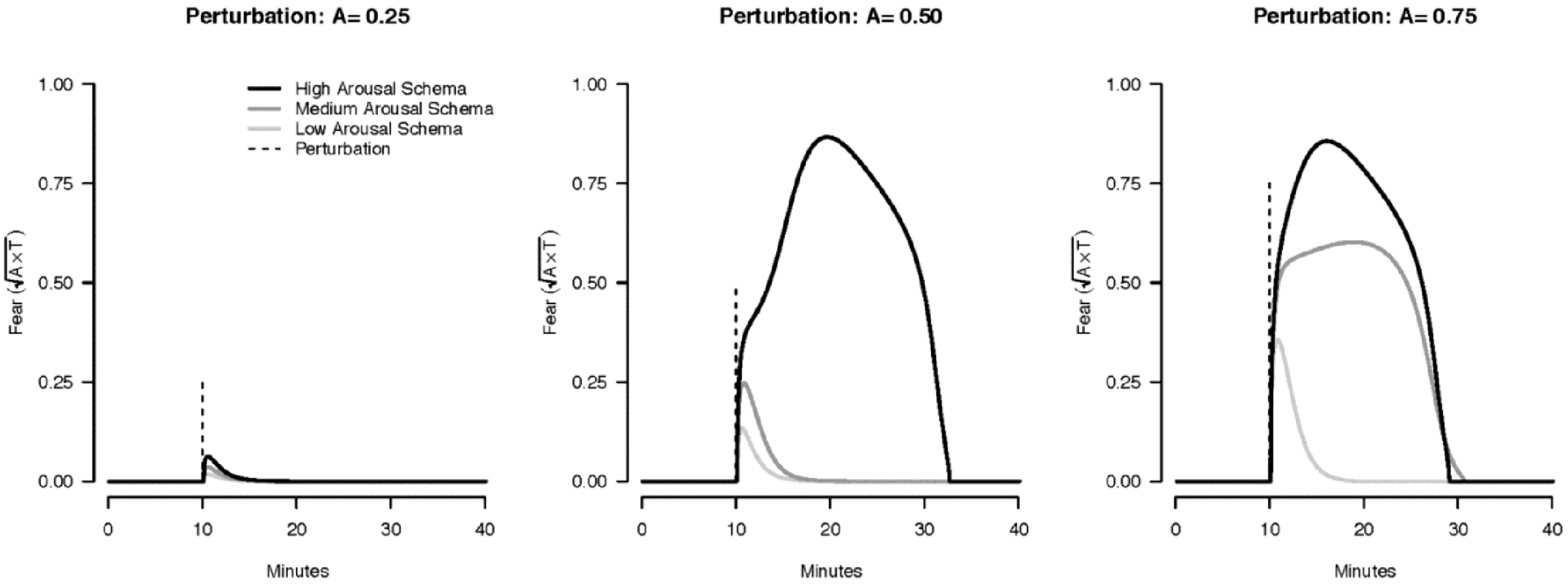
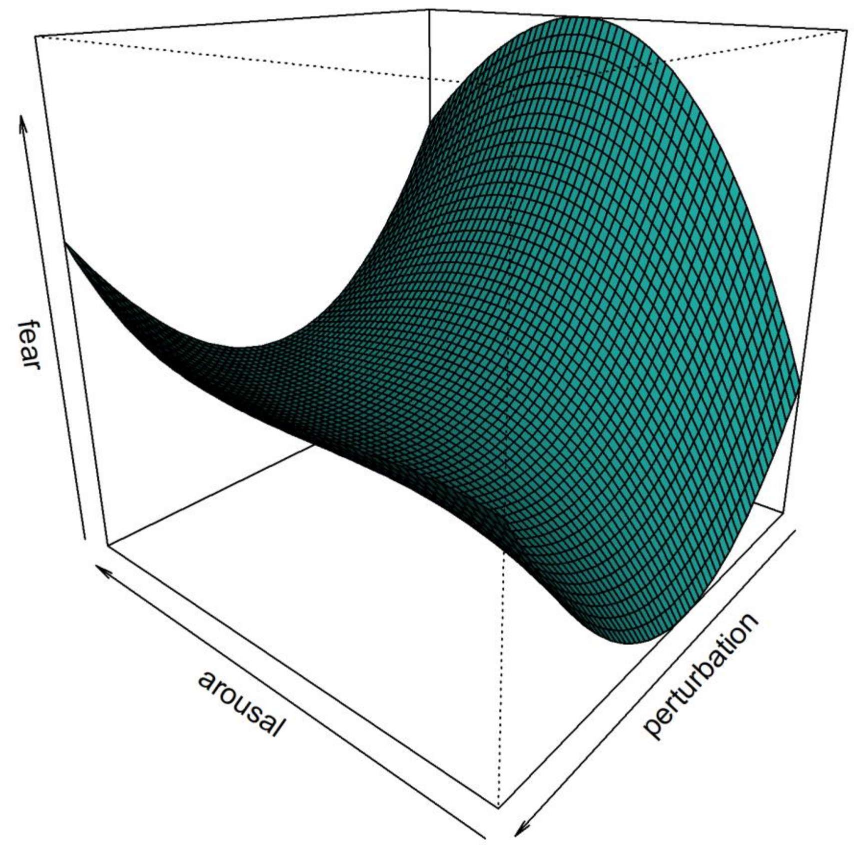


Figure 4. Individual Differences in Vulnerability to Panic Attacks. We simulated perturbations to arousal of varying strength (inducing arousal of .25, .50, and .75) in three conditions: low, medium, and high arousal schema (*S*=.25, .50, and .75, respectively). To

With more parameter settings?



(this is not real, just for illustration)

Agent-based models

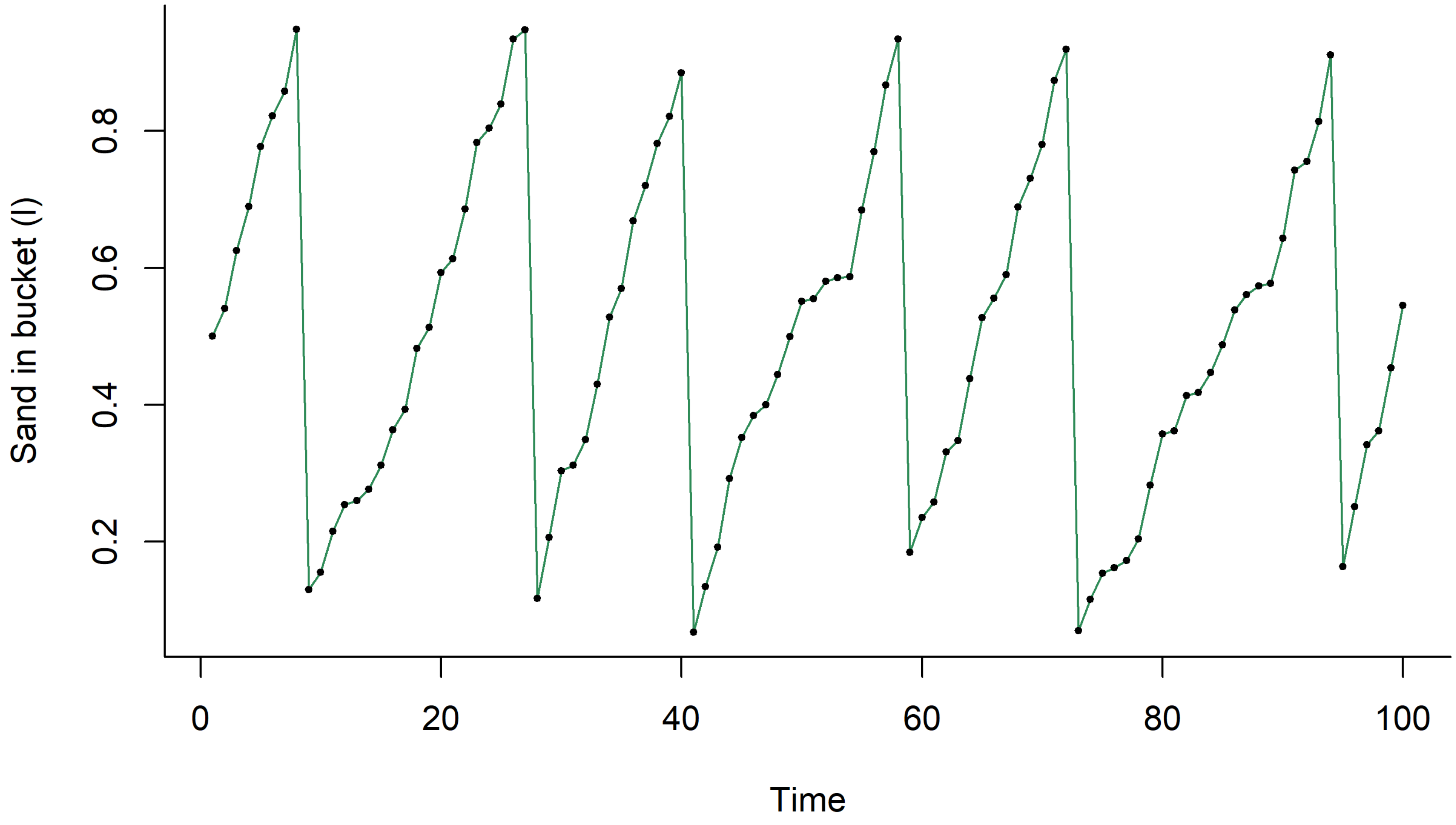
- Used in economics, sociology, ecology, finance, spatial planning, social psychology, and more
- Create **agents** who interact in an **environment**
- Each agent has **rules** based on theory
- Simulating the system means applying these rules repeatedly
- Then you can investigate the system

Agent-based models

Agent-based models

```r
child_a <- function(sand) sand + runif(1, 0, 0.1)
child_b <- function(sand) if (sand > 0.95) runif(1, 0.05, 0.2) else sand

sand_vec <- numeric(100)
sand_vec[1] <- 0.5
for (i in 2:100) {
  sand_vec[i] <- child_a(sand_vec[i-1])
  sand_vec[i] <- child_b(sand_vec[i])
}
plot(sand_vec, type = "l")
```

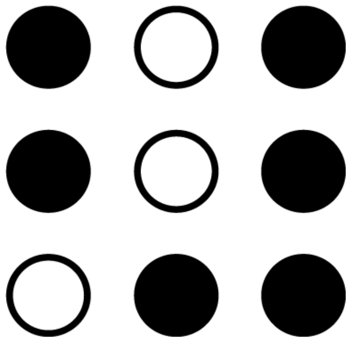
Agent-based models



Interim conclusion

1. Computational methods used by social scientists to formalize & investigate theories
2. Simulation from computational models to inspect phenomena following from model
3. Do this for different parameter settings
4. (2) and (3) may take a long time -> computational limits reached!

abm



Schelling’s model of segregation

Schelling segregation model

- Famous example of ABM in social behaviour

- What are the causes of de facto segregation in society?
- Theoretical / formal model of population dynamics
- Implemented as an agent-based model
- Conclusions drawn based on phenomena resulting from this model

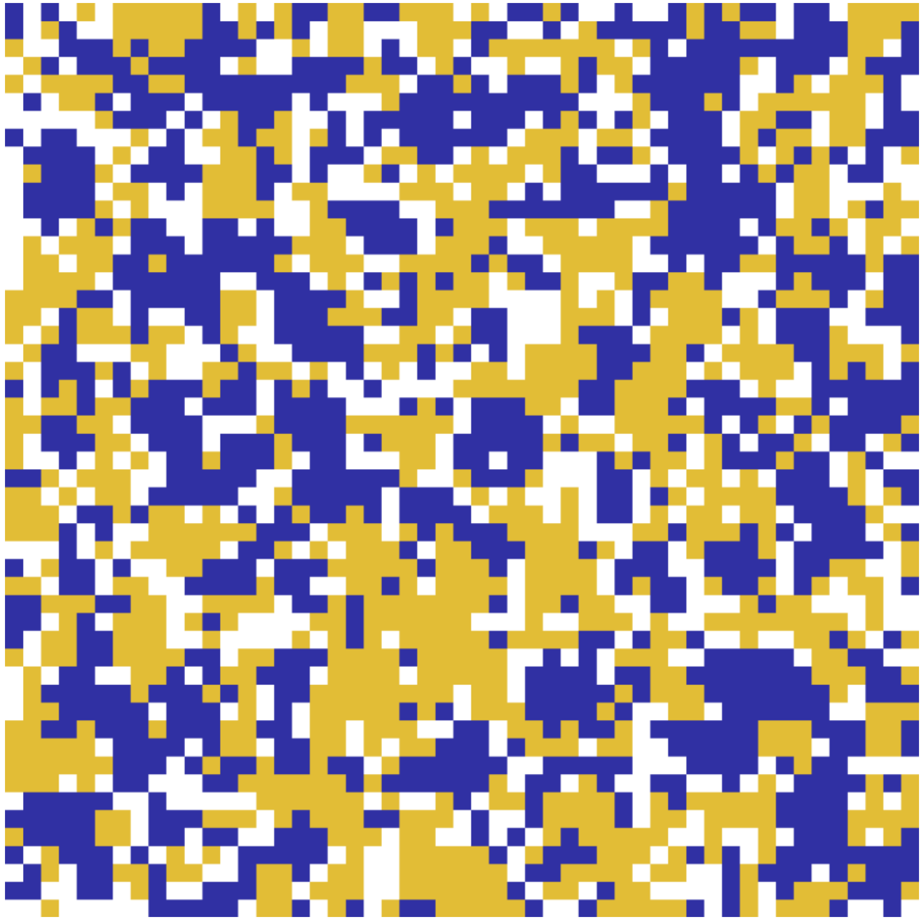
Schelling segregation model

- Environment: two-dimensional grid
- Agents belong to one of two groups
- Agents want to live close to others like themselves
    - Agents have preference ($\boldsymbol{B_a}$) for the proportion of neighbours like them ($\boldsymbol{B}$)
- If B < B_a, then move to random free location on grid
- Else stay

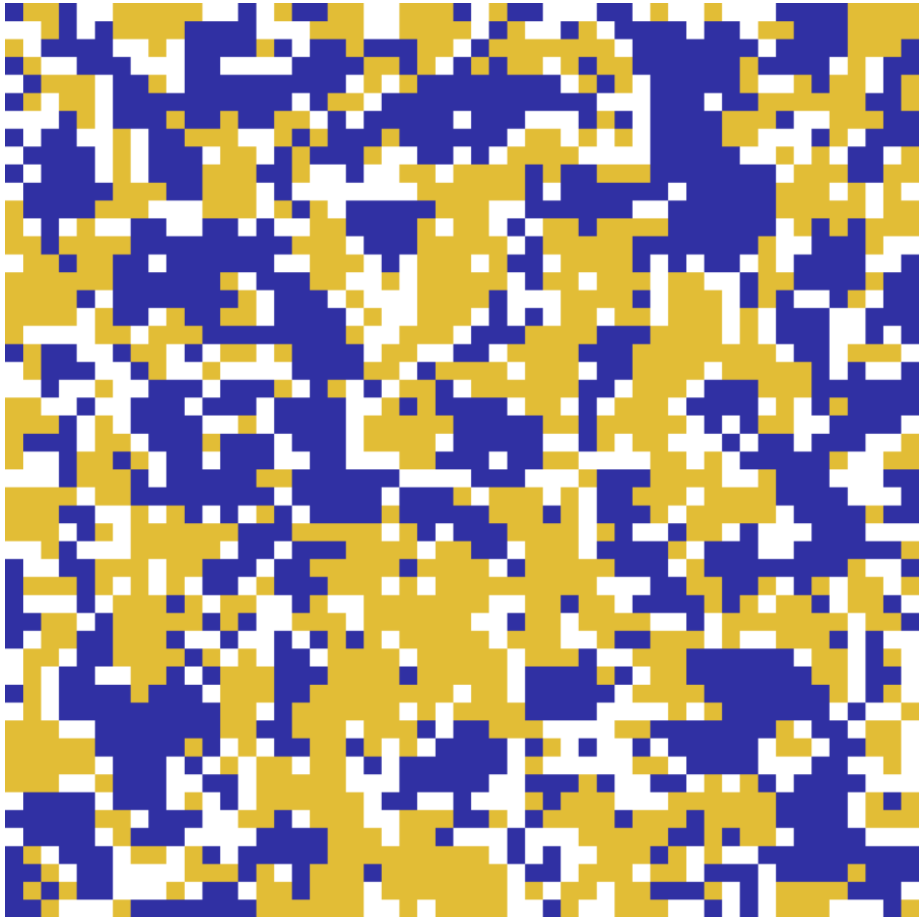
Schelling segregation model

```r
# in-group preference
Ba ← 0.5

# initialize population
pop ← init_population(c(0.5, 0.5))

# occupation matrix
M ← matrix(data = pop, nrow = N)

# happiness matrix
H ← matrix(data = FALSE, nrow = N, ncol = N)

# run for 50 iterations
for (i in 1:50) {
  H ← compute_happiness(M, Ba)
  M ← move_agents(M, H)
}
```

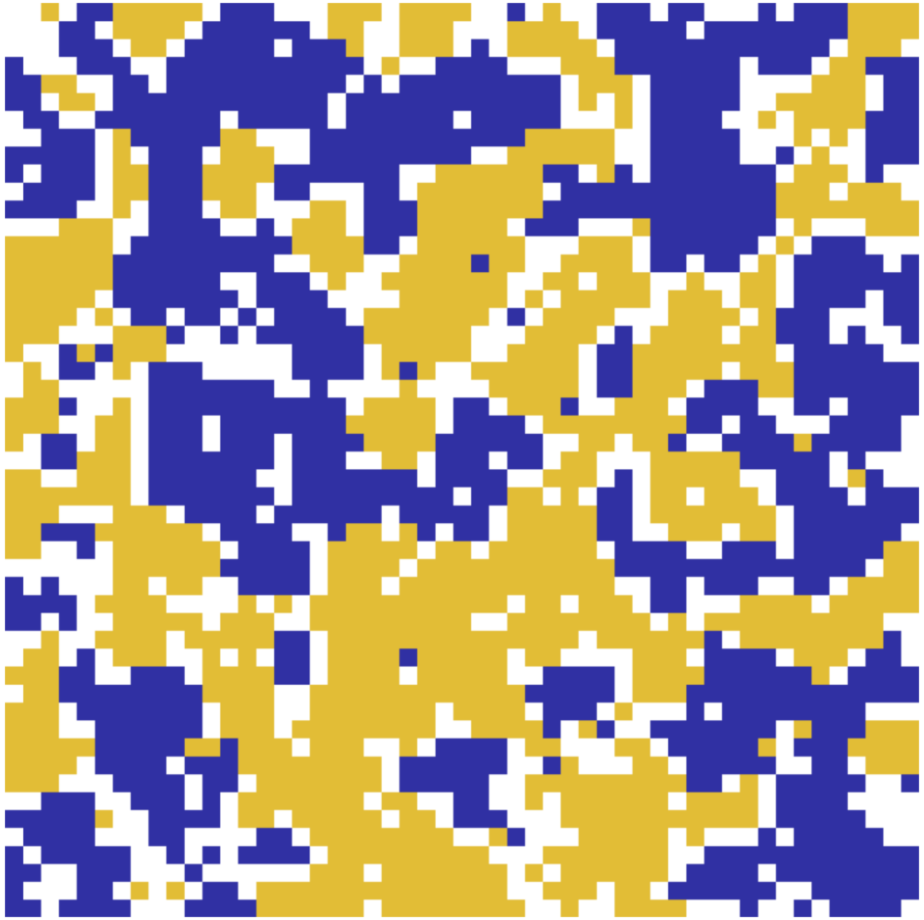
Schelling segregation model



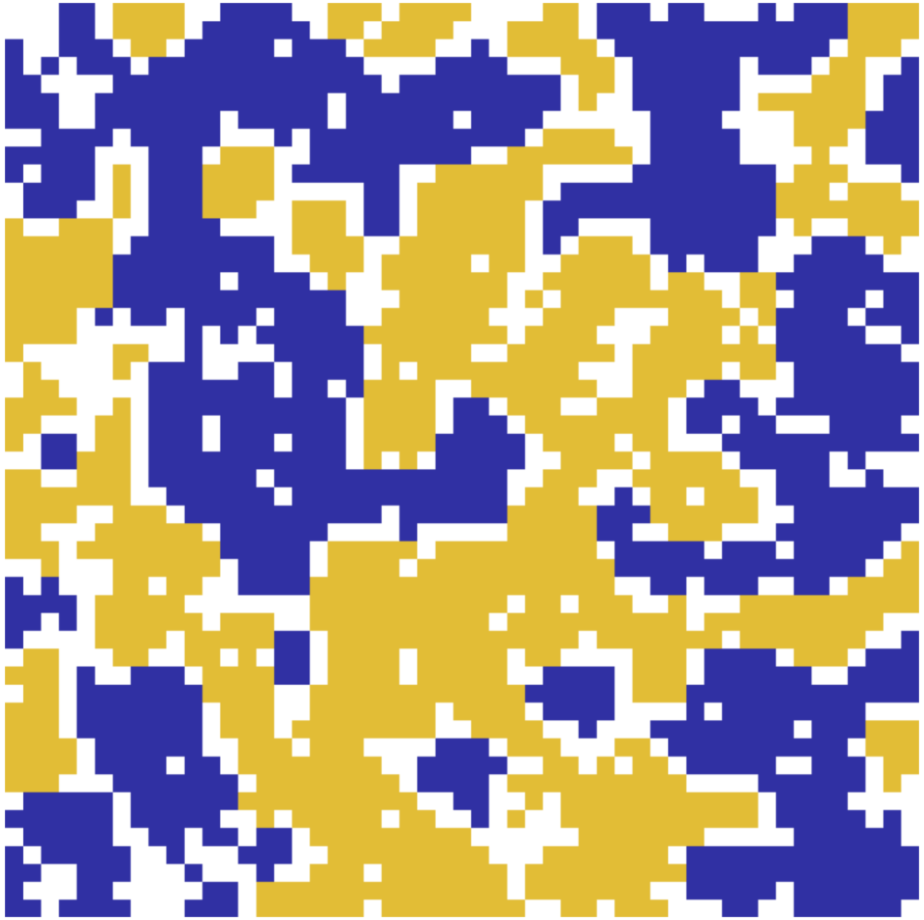
Schelling segregation model



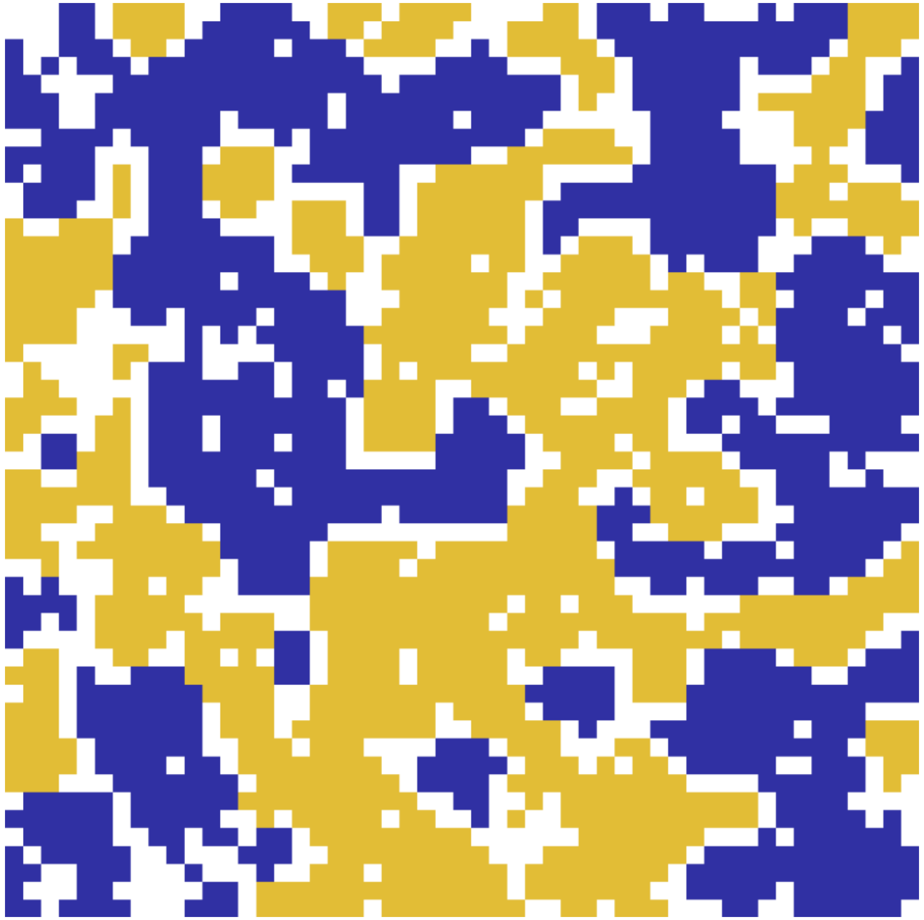
Schelling segregation model



Schelling segregation model



Schelling segregation model



Schelling segregation model

- Micro behaviour: happy or unhappy with current location -> move or stay

- Macro phenomenon: how does the distribution of agents over the grid look?

Schelling segregation model

- Schelling's finding: for groups of equal size with $B_a \gtrsim 0.33$, the system is likely to end in a **segregated** state
- Below that, the system will stay in a **mixed** / random state

Conclusion

Even if there is only a mild in-group preference, the world might still end up very segregated!

(keep assumptions in mind ☺)

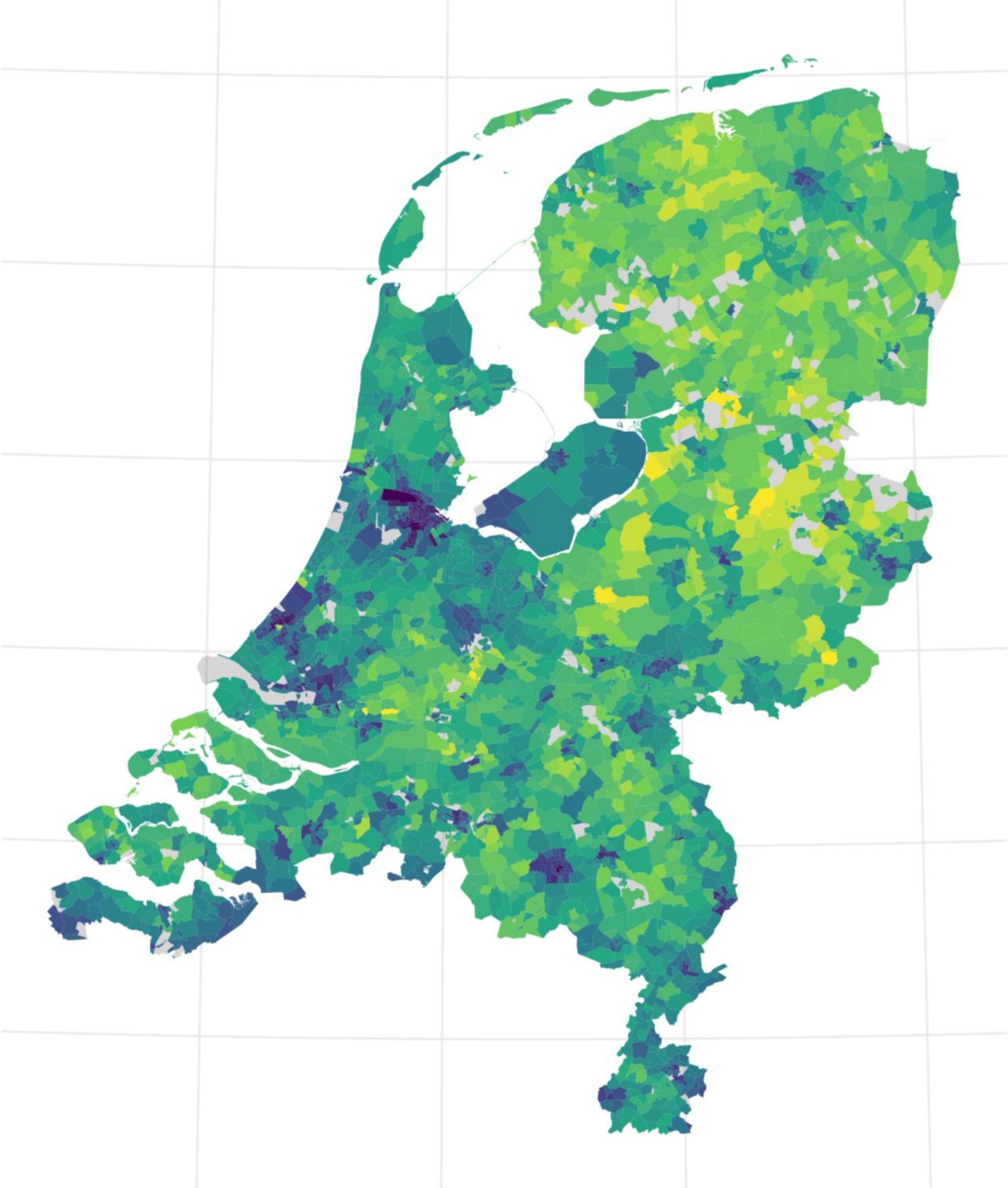
Schelling segregation model

- Note: **randomness** in initialization & in movement to different locations
- At which B_a will the system segregate?
- Need to run this model many times for different B_a and compute expectation (average over the iterations)
- Monte carlo **simulation**

Schelling segregation model

Some more parameters you may want to vary:

- Number of distinct populations
- Relative population sizes
- Number of free spots in the grid
- Neighbour preference
- Radius for looking at neighbours
- Other extensions...

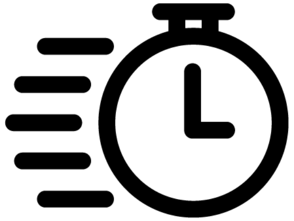


This will take a long time

Speeding up the ABM

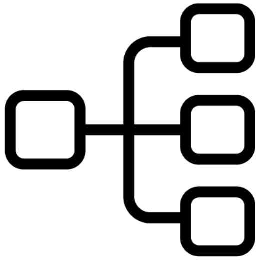
Two options

faster



Write faster, optimized code

parallel



Run multiple ABMs at the same time

Two options

f l

Write ltiple
opt t the
code same time

faster

Speeding up slow code

- There isn't one solution for all types of code
- Speeding up slow code takes time
- Investigate smarter algorithms for your problem!
- If you are rewriting your R code, use vectorized & matrix operations where possible (faster than loops!)
- Use benchmarking to check the speed & memory usage of your functions (I like `bench::mark()`)

Speeding up slow code

- Another step: rewrite in C++
- Depending on problem, this may yield great speedup

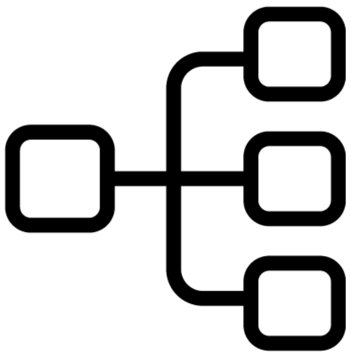
- In R: **Rcpp** package helps with this

```
sourceCpp("my_cpp_function.cpp")
```

An example of Rcpp speedup is in the hands-on later

```
# A tibble: 2 x 13
  expression      min  median `itr/sec` mem_alloc `gc/sec` n_itr  n_gc total_time result
  <bch:expr> <bch:tm> <bch:t>     <dbl> <bch:byt>    <dbl> <int> <dbl>   <bch:tm> <list>
1 r           724.2ms 724.2ms      1.38    4.76MB     5.52     1     4      724ms <NULL>
2 cpp          65.6ms  71.2ms     13.8    41.42MB    11.9      7     6      506ms <NULL>
# ... with 3 more variables: memory <list>, time <list>, gc <list>
```

parallel



Parallel programming

- Many problems are of the “embarrassingly parallel” type
  Little to no effort required to separate problem into number of parallel tasks

- ABM itself is **not** embarrassingly parallel: time step 3 requires results from time step 2!
- Running the whole ABM several times to average over uncertainty **is** embarrassingly parallel

Parallel programming

- Computers nowadays can do more than one task at a time: threads
  - Often: 4 or 8 threads
  - Bigger computers have 12, 16 or 32 threads
  - Depending on computer, potential speedup of 32 times! (remember that our Rcpp effort gave ~10 times)

- Parallel programming is built into R (package **parallel**)

```r
library(parallel)

# create a function to run in parallel
my_func <- function(i) sprintf("this is iteration %i", i)

# instantiate 8 workers
clus <- makeCluster(8)

# run the function for iteration 1:100
parSapply(clus, 1:100, my_func)

#>  [1] "this is iteration 1"   "this is iteration 2"   "this is iteration 3"
#>  [4] "this is iteration 4"   "this is iteration 5"   "this is iteration 6"
#>  [7] "this is iteration 7"   "this is iteration 8"   "this is iteration 9"
#> [10] "this is iteration 10"  "this is iteration 11"  "this is iteration 12"
#> [13] "this is iteration 13"  "this is iteration 14"  "this is iteration 15"
#> [16] "this is iteration 16"  "this is iteration 17"  "this is iteration 18"
#> [19] "this is iteration 19"  "this is iteration 20"  "this is iteration 21"
#> ...

# stop the workers (we're done with them now)
stopCluster(clus)
```

An example of parallel programming is in the hands-on later

Interim conclusion

1. Today we are working with the Schelling agent-based model
2. Running the abm with different settings takes a long time
3. We can program the abm itself more efficiently
4. We can perform the abm in parallel

Let’s try it out!

Hands-on session 1