Branching Process Simulator in R

A. Tchorbadjieff[1], L. Tomov[2(✉)], and P. Mayster[3]

[1] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Sofia, Bulgaria
atchorbadjieff@math.bas.bg
[2] Department of Informatics, New Bulgarian University, 1618 Sofia, Bulgaria
lptomov@nbu.bg
[3] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Sofia, Bulgaria
penka.mayster@math.bas.bg

Abstract. We developed a set of software functionalities in R for simulation of branching processes. Originally it was designed for simulating the branching mechanism of a cosmic ray atmosphere cascade, beginning with electron-photon cascade. Further this simulator was adapted for applications epidemiology with extended set of probability distributions such as Poisson, Negative Binomial, shifted Geometric and Polya-Aepply used either for modeling the initial conditions for linear birth-death processes or branching process mechanism following predefined probabilistic distribution. The simulator is applied mostly when analytical solutions give convergent infinite series. It uses the capability of R for parallel computation and applies the Object-Oriented Programming paradigm (a secure type encapsulation).

Keywords: Branching processes · R · parallel computing · epidemiology · physics

1 The simulator

1.1 Purpose

The cosmic rays are the biggest natural permanently occurring branching process. Partially, the electron-photon pair production induced cascades naturally imply connection to Yule-Furry process and age-dependent Markov branching processes. This is the reason that it is the first mathematically modelled particle showers during the down of nuclear era [1, 2]. However, the explicit solution of the backward Kolmogorov equation for more complicated and larger multi-type branching processes is extremely difficult, even analytically insolvable. This makes the need of numerical solutions and computer simulation important and indispensable part in research.

As a part of this effort a new simulator[1] is developed as a modeling tool for cosmic ray air shower using branching processes [3]. It works as a generator of multiple independent

[1] https://gitlab.com/Tchorbadjieff/covid-19.

T. Zlateva and G. Tuparov (Eds.): CSECS 2023, LNICST 514, pp. 73–86, 2023.
https://doi.org/10.1007/978-3-031-44668-9_6

trials, with their aggregated results serving as a good approximation to the real result. Not surprisingly, soon the tool began to evolve in different directions, not limiting only to cosmic rays. One of the initially developed functionalities is the implementation of linear birth-death process induced from different random initial conditions [4]. Later, the available functionalities have been extending with inclusion the option of different branching mechanism, basically following well-known probabilistic distributions.

The mathematical formalism from the very beginning is based on assumption of existence some probabilities $p, 0 < p < 1$, that either photon splits on electron and positron, or a new photon production due to charged particles deacceleration by breaking radiation. The simulator is developed in R and uses Object Oriented Programming paradigm. These processes are naturally simulated better with multiple parallel threads and the simulator is written for this purpose. We run it on the Avitohol supercomputer, belonging to Bulgarian Academy of Sciences (IICT-BAS[2]). Since the start of the COVID-19 pandemic, this simulator has been easily adapted also for modeling the disease spread in conjunction with changepoint analysis [5].

1.2 Structure and Functionality

Structure. The epidemiology version of simulator actually implements linear birth-death process with initial conditions following either Negative Binomial or Poisson distribution. It consists of two files with core functionalities – Cascade.R and Run_Cascade.R and two specialized files for epidemiology – Branching_MP.R and CPoints_Init3.R.

The Cascade.R defines the class, which contains the information about given particle (reinterpreted later as infected person) which has some properties relevant only to the electron-photon cascade and other sub-atomic processes that can be used in different context. The specific properties are *type, energy, angle of scattering*, while *age, time, number and depth* are more general. The angle of scattering and the energy are main properties of any natural particle cascades. However, despite of that in the current version they are not completely implemented yet, they provide preserved definition for further reinterpretation beyond particle physics constraints, to address conditions of infection in epidemiology modeling – like environmental and social factors. The last one, depth, gives the position of the particle on the cascade chain or the depth of the branching tree. We give here the class definition without the constructor and the validity check.

We apply S4 as a formal approach to OOP, which has specialized methods for creating classes, such as setClass() [6]. It provides multiple inheritance which we do not use in our simulator due to the issues with extensibility and maintainability it introduces [7]. S4 defines slots, named components of the object accessed with the subsetting operator @ (at). These are the class members. There is a setMethod() function that defines accessor functions for the members of class and assigns them. This is analogous to the properties in C# classes, a way to get and set the values without separating these two logically independent functionalities. In our code we do not use inheritance, but with the extension of the functionality we may have to – and if we do, S4 supplies us with the necessary functionality. The setClass() method can use the argument "contains" to specify the base

[2] https://www.iict.bas.bg/avitohol/.

class. Employing the idea of Alan Kay "Until real software engineering is developed, the next best practice is to develop with a dynamic system that has extreme late binding in all aspects." [8] In R the class definition and the object construction both occur at run time. This makes impossible to create invalid object, by redefining the class after an object has been constructed. Every class definition in R should have a prototype, for default values (analogous to the role of parameter less constructor in C#). Validation can be done with function SetValidity() or as in our case with user supplied function for simple objects. The SetValidity() however, in the general context is more appropriate, because we want to have meaningful and informative error messaging to improve maintainability and prevent dangerous errors [9].

```
ParticleInfo = setClass(
  "ParticleInfo",

  slots = c(
    type= "numeric" , #"gamma"==1, "e-"==2, "e+"==3
    #type = "character", # "gamma", "e-", "e+"
    E = "numeric", # Energy, in MeV
    t = "numeric", # Time
    theta = "numeric", # Total scattering angle
    age = "numeric", # Age
    number="numeric",
    depth="numeric"
  ),
prototype = list(type = 2, E = 1000, t = 0, theta = 0,
age = 0, depth=0),

  # Check for validity after construction
  validity = function(object) {
    return(!((object@type < 1 | object@type > 3) | ob-
ject@E <= 0 | object@t<=0 | (object@theta < -180 | ob-
ject@theta > 180) | object@age < 0 | object@number < 0 |
object@depth < 0))
  }
```

The ParticleInfo class is being used as part of array, in which each element holds the number of particles at a given discrete step of time t. For each given element of the structure, a random number is being generated (for instance with random binomial outcome as it is shown in example) [10].

```
trial=function(i,strct, p)
{
  return (rbinom(1,strct[[i]]@number, p))
}
```

Main Method for Cascade. The branching cascade is calculated iteratively. At each state the survival of the previous step is doubled, the depth of the tree increases with one, and the discrete time step also increases with one. The age of particles here is fixed every time. The current version of the simulator works with the symmetric process with equal lifespans for all particles and the survived particles from previous step are doubled (each particle is replaced by two new particles), so the half of them is "reincarnated"[3], exactly in line with the famous myth of Hydra – "cut one head and two new shall arise). Every particle has a certain probability of dying, so survived particles at a given time step are a subset of the generated particles at the previous subset, combined with the newly generated particles at this step t. This process is inside the for loop in the code, shown below.

The most important part in this process is the definition of probabilities. The software relies on values in the classical probabilistic range between 0 and 1. However, usually these values rely on conversion from other measurement units. In high energy particle physics this measure is cross section. For the case of Covid-19 simulation, the probabilities are obtained from the rate of daily infection changes.

```
spawnYule_basic=function(part.atDepth, depth, p.dead)
{
  photons.which = getIndexByType(part.atDepth,1)
  photons= unlist(sapply(photons.which , FUN=iter,
strct=part.atDepth))
  p=1/(p.dead+1)
  n.dead = unlist(sapply(1:length(photons), FUN=trial,
strct=photons, p=1-p))
  surv=unlist(sapply(1:length(photons), FUN=function(i,
strct){ return (max(0,strct[[i]]@number-n.dead[i]))},
strct=photons))

  for (i in 1:length(photons))
  {
    photons[[i]]@number = 2*surv[i]
    photons[[i]]@depth = photons[[i]]@depth +1
    photons[[i]]@age = 1
    photons[[i]]@t = photons[[i]]@t +1
  }

    ret= c(photons)

  return (ret)
}
```

[3] https://www.rand.org/content/dam/rand/pubs/reports/2009/R381.pdf.

Cascade Shower. It is in the same file Cascade.R, with default probability of dying set to ½ and with asymmetric process set to false. Asymmetric processes with different lifespans for particles are to be implemented in the future with asynchronous programming model.

```
shower.Extention = function(part, depth, p.dead=1/2,
isAssym=FALSE)
{
  idx=getIndexByDepth(part, depth)
  pool=part[idx]
  ph.index=getIndexByType(pool,1)

  result=NULL
  if(!is.null(ph.index))
  {
    tmp = spawnYule_basic(pool[ph.index],depth, p.dead )
    if(length(tmp)>0)
      result= tmp
  }

  return (result)
}
```

Running the Cascade

Important properties of any branching process are initial conditions and the phenomenology of the process development. For the formal, the importance of initial conditions for linear birth-death process is considered in [2–4]. Later, this assumption is empirically observed as a pure migration of infected population due to first days of Covid-19 pandemic in [5]. In the current available software implementation as options for initial conditions are considered Poisson, Binomial, Negative Binomial and related Geometric distribution, Polia Aeply distributions and Polia urn generator.

Initially, the process of branching is assumed only as binary outcome – death or multiple births. However, with functionalities extension of the code the realization of trial function is extending by inclusion of another probability branching mechanisms – Negative Binomial and Poisson distributions.

The effectiveness and precision of the results from the simulator depend on the number of computed trajectories. This is a direct result of the stochastic nature of every trajectory due to random trials. This implies the requirement of very high number of repeated computations and their statistical aggregation. These receptions are implemented through multiple independent instances of cascade trajectories. They are executed by function main_iter() that runs the cascade iteratively, calling the *shower.Extention()* from Cascade.R in a **for** loop with extending the depth at each step and with a probability of dying, calculated at every step.

Main_iter() is called in *main()* function iteratively, bringing the complexity to $O(n^2)$. This is the place where trajectory repetitions are executed. To extend computational effectiveness of contemporary computer multi-processing hardware, the independent calculations of different trajectories are implemented on parallel computation with doParallel library in R [11]. In the following code block we show how in R we can combine resources for shared memory between nodes and processes, by loading packages and sources and calling linear_predict() with *lapply*() for linux systems. Separate applications of a function to members of list are prerequisite for parallel processing.

```
no_cores <- detectCores() - 1
cl <- makeCluster(no_cores)
registerDoParallel(cl)
system.time(foreach(i = 1:n_sel, .combine = list, .ex-
port= c('distr.pois','data.by.country'),.pack-
ages=c('readxl', 'xts', 'changepoint', 'MASS', 'grDevic-
es')) %dopar%{
  source("Run_Cascade.R")
  library(kSamples)
  library("dgof")
  set.seed(1)
  output=linearPredict(i, bridge[i])
  lapply(1:length(output), function(j) {is.append=TRUE;
if(j==1){is.ap-
pend=FALSE;};write.table(output[[j]],file=paste("Results/
/predicted_",selection[i],".csv", sep=""), append =
is.append,col.names = !is.append, row.names = FALSE,
sep=",")})})
```

Verification.
The simulator is designated to implement stochastic process, resulting to stochastic outcome. This lack of deterministic results implies the requirement for different software verification process. At first, the outcome is designed to consist of large number repeated runs generated by random generator, creating multiple trajectories, as much as possible. The expected results are obtained from averaging over all generated directories. The variability is also represented by results variation in generated directories.

Having different outcome in every run, the classical debugging procedure becomes non-effective and obsolete. Thus, the process of software verification and debug fixing is designed as a statistical learning process, starting from the comparison between already available analytical results and their computer modelling by software. This process is permanently repeated after every new development. For instance, the influence of initial conditions on linear birth-death process was tested by comparison between analytical results and model outcome in [4]. The same comparison is done for geometric distribution, using the analytical results in [12].

The trial mechanism is verified separately during development by directly analytical computing branching processes. For instance, let us consider the following branching cascade of 3 particles with its generating function:

$$\begin{aligned} h1(s_1, s_2, s_3) &= (1 - 2p) + ps_1 + ps_1s_2 \\ h2(s_1, s_2, s_3) &= (1 - 2p) + ps_2 + ps_1s_2 \\ h3(s_1, s_2, s_3) &= (1 - 2p) + ps_2 + 2ps_1s_3 \end{aligned} \tag{1}$$

The probability of reproduction is $0 < \mathrm{p} < 1/2$, having that the process is without particle deaths when $\mathrm{p} = 1/2$. The matrixes of expectations $\mathbf{M_{ij}}$ and characteristic determinant $\mathbf{\Lambda}$ are computed as follows:

$$\mathrm{M_{ij}} = \begin{bmatrix} p & p & p \\ p & 2p & 0 \\ p & 0 & 2p \end{bmatrix} \tag{2}$$

$$\Lambda = \begin{bmatrix} p - \lambda & p & p \\ p & 2p - \lambda & 0 \\ p & 0 & 2p - \lambda \end{bmatrix} \tag{3}$$

As straightforward computations show, the process is critical when $\mathbf{p} = 1/3 < 1/2$. The process is subcritical and supercritical, respectively, when $\mathbf{p}$ is less or above 1/3. The simulated results are in complete agreement with the predicted results after only 10000 runs (Fig. 1). Note, that critical process acquires quickly stable asymmetry proportion between electrons and positrons.

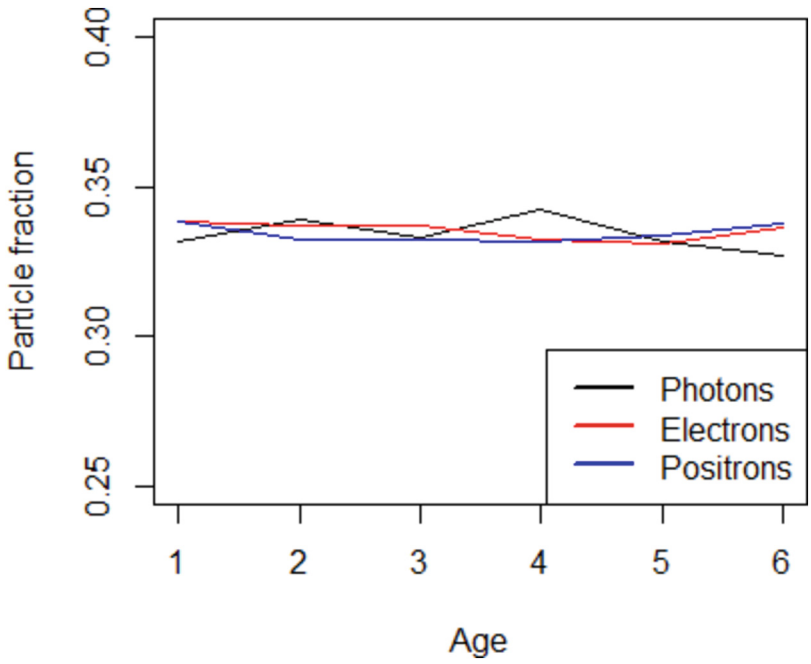

Fig. 1. Software verification with 1000 runs of the electron-photon branching cascade

2 Applications

Application to Covid – When the Covid-19 outburst occurred the need for large number re-computations and predictions of newly infected persons grew. To keep track with it, we automated completely the process[4] by development of the dedicated method linear_predict(). The process, which is fully described in our paper [5], combines linear birth-death process with Poisson and negative Binomial initial conditions with change-point analysis to detect any regime changes of daily infection rates and to recalculate due probability distributions. This works due to the Markov property of the branching process of linear birth-death. The changepoint operations are implemented in CPoins Init3.R[5], separately from the core functionality in order for it to remain reusable and extendable. It allows continuous adaptive and relatively precise short-term prediction of daily cases for an ongoing pandemic with rapid viral evolution, multiple variants and wide spectrum of measures and adaptive reactions from the public to the pandemic. It can be relatively easily adapted to predict cases by age group, bed occupancy in hospitals, and deaths for the initial waves of pandemics (there is recurrent relation between lethality in a given wave and in previous waves). We show some results of the work of Branching_MP.R on Fig. 2 and Fig. 3

[4] https://gitlab.com/Tchorbadjieff/covid-19/-/blob/main/Branching_MP.R.
[5] https://gitlab.com/Tchorbadjieff/covid-19/-/blob/main/CPoints_Init3.R.

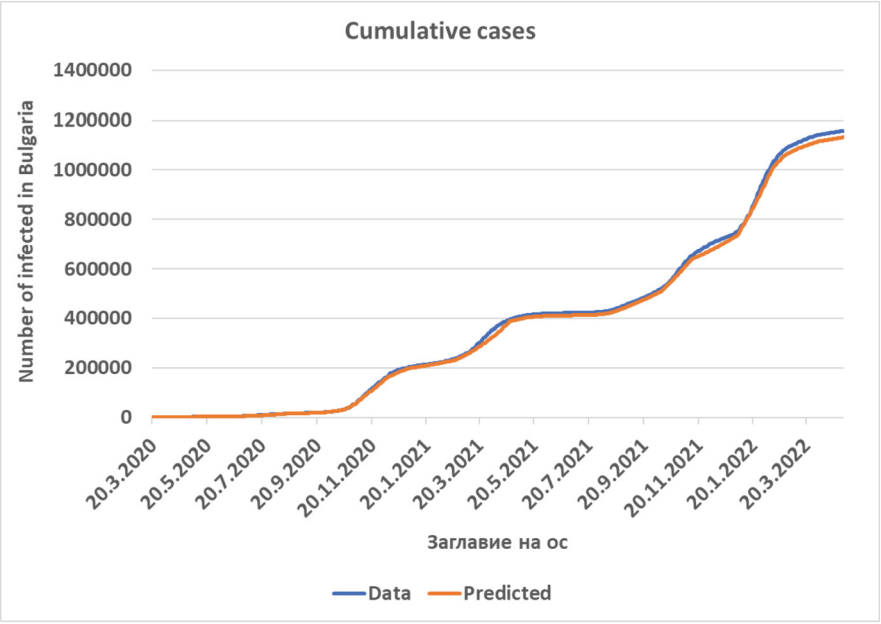

Fig. 2. Predicted cumulative number of infected individuals in Bulgaria for 20.3.2020–30.4.2022

Applications for Branching Inference: An important application of this simulator is for study of industrial scale processes driven by branching mechanism following different distributions and occurring in changing environments when the analytical solutions are not feasible. An example of analytically solvable process is the one driven by geometric branching mechanism [12].

However, the difficulty of solution grows enormously when other distributions are in consideration for branching modelling. For instance, when the branching process consists of multiple number of reactions **r, r > 0** until the experiment stops. In this case, it follows Negative Binomial distribution NB(r,p) with probability of **p**; $0 < p < 1.$. Then, the multiplicity depends on mean (energy) E(N) and Variations (dissipation)D(N):

$$\mathrm{E(N)} = \frac{(1-\mathrm{p})\mathrm{r}}{\mathrm{p}} \tag{4}$$

$$\mathrm{D(N)} = \frac{(1-\mathrm{p})\mathrm{r}}{\mathrm{p2}} \tag{5}$$

Another, possible case is when the branching concerns equally distributed molecules in volume **V** of a continuous region **T**, $V = \int_T d\xi$. The expected mean value of random variable X in this case is [13], see:

$$E(X) = V^{-n} \int_T d\xi_1 \int_T d\xi_2 ... \int_T d\xi_n X(\xi_1, \xi_2, ..., \xi_n)$$

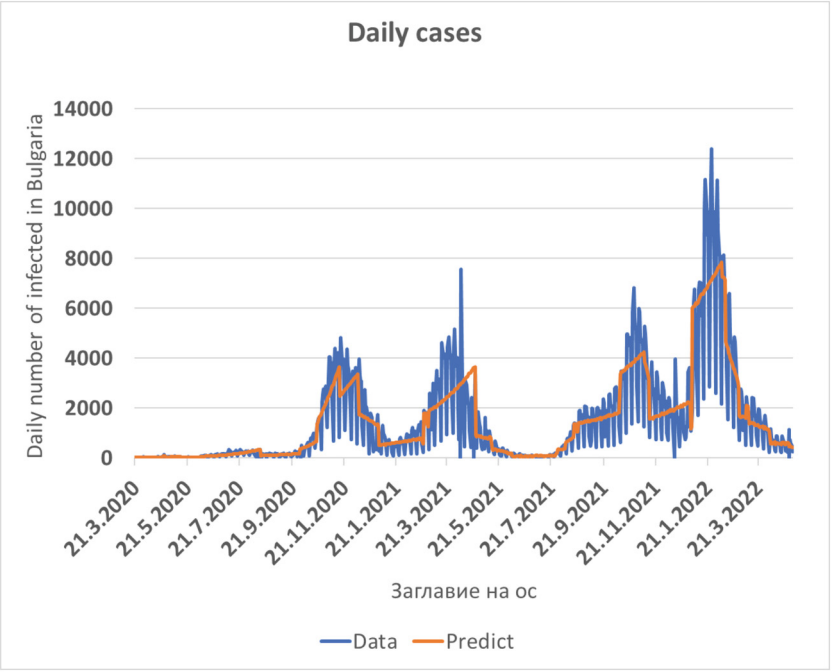

Fig. 3. Predicted daily number of infected individuals in Bulgaria for 20.3.2020–30.4.2022

Then, due to the binomial trial for $T_1 \subset T$ and using probability $p = V_1/V$ and densities $\rho = n/V$ we obtain Poisson distribution (Po(n)):

$$P(R = r) = \frac{e^{-\rho V_1}(\rho V_1)^r}{r!} \tag{7}$$

The variety of possible reasons to use these three distributions, Ge, NB and Po, could be extended to larger class of branching processes. For their probability generating function (p.g.f.) in critical case, $h(s) = p(0) + p(1)s + p(2)s^2 + \ldots$, there is a vertical asymptote for Ge and NB distributions, but not for Poisson (Fig. 4). Moreover, the Po distribution yields the best fit to the tangent $x = y$ in the neighbourhood of $s = 1$ (Fig. 5).

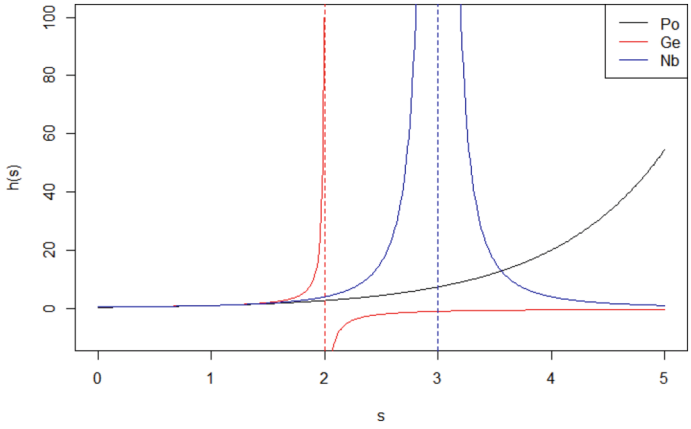

Fig. 4. Probability generating function (p.g.f.) h(s) in critical case for Po, NB and Ge distributions.

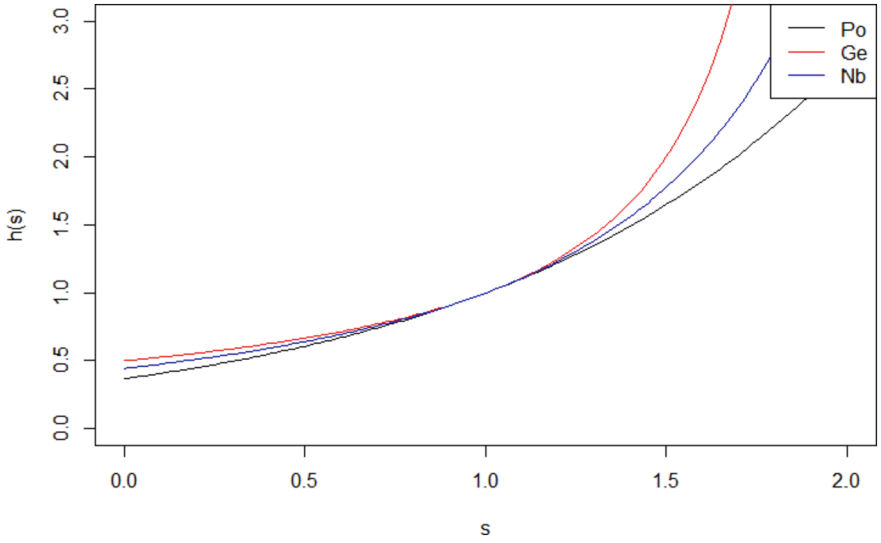

Fig. 5. Probability generating function (p.g.f.) h(s) in the neighbourhood of s = 1 for critical case for Po, NB and Ge distributions.

However, the solutions either for sub-critical of super-critical processes with these three distributions or for any other more complicated distributed processes is not so straight forward. This is not only due to computational complications, but also because the correct identification of exact branching nature of the process, mainly in cases of small data sizes. In this case, the simulator could be used to generate multiple results to test different hypotheses about fit to the real data. The process can be easily implemented by redefining the trial() function[6] with the opted definition and generate enough large

[6] https://gitlab.com/Tchorbadjieff/branching-simulator.

number of trajectories. This procedure is also used to calibrate and verify the simulator work to already available results in [12]. Some random trajectories are computed and demonstrated in Fig. 6a–6d.

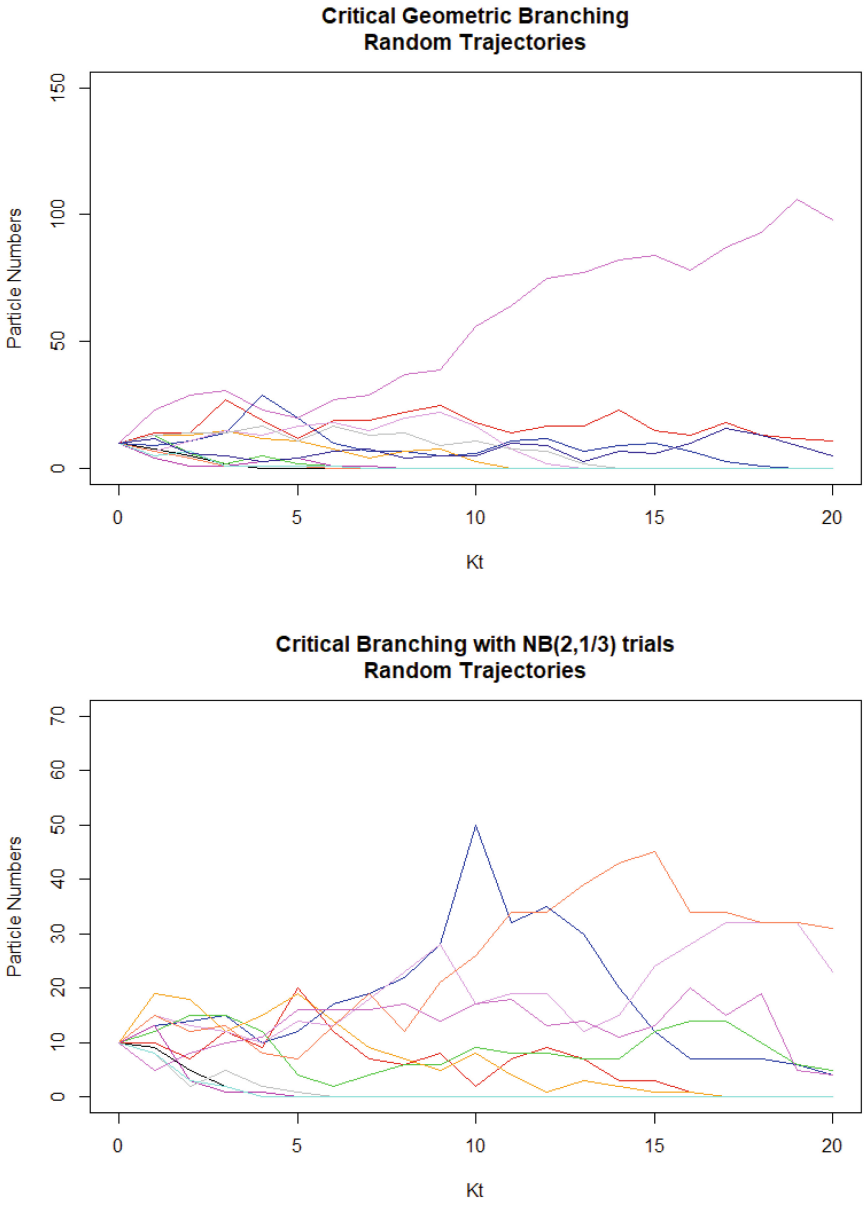

Fig. 6. **a.** Critical branching process with shifted geometric distribution. **b.** Critical branching process with negative binomial distribution with $r = 2$ and $p = 1/3$. **c.** Critical branching process with negative binomial distribution with $r = 3$ and $p = 1/4$. **d.** Branching process with Poisson distribution.

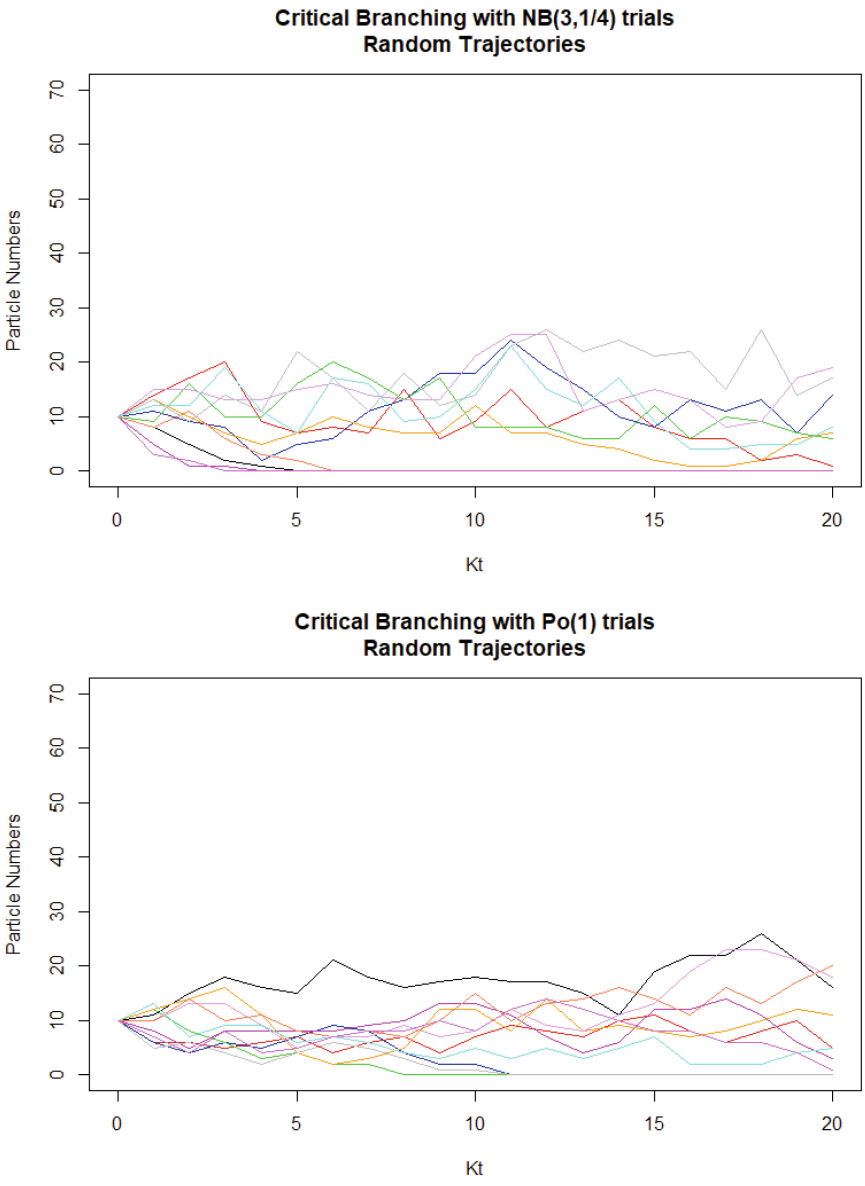

Fig. 6. (*continued*)

3 Conclusions and Future Work

The presented here branching simulator already produced some unexpected practical results – simulated very well development of Covid-19 outbursts, proving that even the most complex ongoing pandemic could be predicted without certain knowledge

either of the characteristics of the viral variants, the measures that the governments take or the ability to predict the evolution of variants. The tool is open source and open for functionalities upgrade. These upgrades could include not only particle interaction processes and changing initial conditions, but other methods of statistical learnings and automatization. Another direction for improvements is the inclusion of abundant graphical features. These upgrades could be implemented easily, as inclusion change point tools in Covid-19 application. The list of ideas how this simulator can be extended and improved, may not be limited only to epidemiology and cosmic rays' physics, but for general application in applied mathematical modeling, training machine learning tools or educational purposes. A logical development in the medium-term future is to create a library package and to include it in CRAN.

Assen Tchorbadjieff acknowledges the support by the Bulgarian National Science Fund, grant No KP-06-H22/3.

We acknowledge the access to the e-infrastructure provided by the Grant No. D01–168/28.07.2022 "National Centre for High Performance and Distributed Computing" of the Ministry of Education and Science of Bulgaria.

Latchezar Tomov is grantee of the European Union-NextGenerationEU, through the National Recovery and Resilience Plan of the Republic of Bulgaria, project № BG-RRP-2.004–0008-C01.

References

1. Harris, T.E.: The Theory of Branching Processes. Springer-Verlag, Berlin (1963)
2. Sevastyanov, B.A.: Branching Processes. Nauka, Moscow (1971). (in Russian)
3. Tchorbadjieff, A.: Using branching processes to simulate cosmic rays cascades. Pliska Studia Math. Bulgarica **27**, 103–114 (2017)
4. Tchorbadjieff, A., Mayster, P.: Models induced from critical birth–death process with random initial conditions. J. Appl. Stat. **47**(13–15), 2862–2878 (2020). https://doi.org/10.1080/02664763.2020.1732309
5. Tchorbadjieff, A., Tomov, L.P., Velev, V., Dezhov, G., Manev, V., Mayster, P.: On regime changes of COVID-19 outbreak. J. Appl. Stat. (2023). https://doi.org/10.1080/02664763.2023.2177625
6. Wickham, H.: Advanced R. CRC Press (2019)
7. Wang, Y, [王焱林]: Revisiting multiple inheritance for modularity and reuse. (Thesis). University of Hong Kong, Pokfulam, Hong Kong SAR (2019)
8. https://wiki.c2.com/?AlanKaysDefinitionOfObjectOriented. Accessed 10 July 2023
9. Leveson, N.G., Turner, C.S.: An investigation of the Therac-25 accidents. Computer **26**(7), 18–41 (1993). https://doi.org/10.1109/MC.1993.274940
10. Kachitvichyanukul, V., Schmeiser, B.W.: Binomial random variate generation. Commun. ACM. ACM **31**, 216–222 (1988)
11. Microsoft Corporation, Weston, S., doParallel: Foreach Parallel Adaptor for the 'parallel' Package (2019). https://CRAN.R-project.org/package=doParallel
12. Tchorbadjieff, A., Mayster, P.: Geometric branching reproduction Markov processes. Mod. Stochast.: Theory Appl. **7**(4), 357–378 (2020). https://doi.org/10.15559/20-VMSTA163
13. Whittle, P.: Probability. Penguin Books (1976). ISBN: 0140800859, 9780140800852