Computational Biology 1 (FFR110), Examples sheet 4

Vitalii Iarko (930505-4711, iarko@student.chalmers.se, no ECTS)

January 5, 2015

Abstract

In this example sheet we consider two separate topics. The first one - we obtain stochastic dynamics in large but finite population of SIS model and contrast it to the corresponding deterministic dynamics. In addition to this we simulate the obtained dynamics and investigate infection extinction time and distribution in quasi-steady state and compare results to the theory. In the second part we, using Kuramoto model, investigate the syncrhonization and again compare results to the theory.

15 pages overall.
The report is available online: `http://nrg3.github.io/comp_bio_4.pdf`.

Contents

1 Stochastic dynamics in large but finite populations. **1**

1.1 a) Steady states and their linear stability analysis. 2
1.2 b) Stochastic model. 2
1.3 c) Simulations in quasi-steady state. 3
1.4 d) Extinction time. 4
1.5 e) Quasi-steady state distribution. 4

2 Synchronisation. **5**

2.1 a) Theory. 6
2.2 b) Simulations. 6

A File `1cd.nb`: **7**

B File `1e.nb`: **11**

C File `2.nb`: **13**

1 Stochastic dynamics in large but finite populations.

In this task we contrast the determinists disease dynamics to the corresponding disease dynamics in large but finite populations (stochastic). The analysis is made for the SIS model, which deterministic dynamics is given by

$$\frac{\mathrm{d}\,I}{\mathrm{d}\,t} = \frac{\alpha}{N}SI - \beta I \tag{1}$$

$$\frac{\mathrm{d}\,S}{\mathrm{d}\,t} = -\frac{\alpha}{N}SI + \beta I \tag{2}$$

Here, the parameters α and β are positive constants and $N = S + I$.

In order to compare the dynamics we conduct linear steady state analysis of deterministic model at first, then obtain corresponding stochastic model though the Master equation and define quasi-steady state. Finally, we conduct simulations of obtained model in order to illustrate differences from deterministic model and to investigate time to extinction and distribution in quasi-steady state.

1.1 a) Steady states and their linear stability analysis.

Note that $\frac{\mathrm{d}\,I}{\mathrm{d}\,t} + \frac{\mathrm{d}\,S}{\mathrm{d}\,t} = 0 = \frac{\mathrm{d}\,N}{\mathrm{d}\,t}$ (using (1) + (2)), i.e. population size N is constant and, thus, equations (1) and (2) are coupled as $S = N - I$, therefore, we may consider only the first one, rewritten as:

$$\frac{\mathrm{d}\,I}{\mathrm{d}\,t} = \alpha I\left(1 - \frac{I}{N}\right) - \beta I \equiv f(I) \tag{3}$$

The case when infection sustains ad infinitum corresponds to stable steady state with $I^* > 0$. The steady states are:

$$f(I^*) = 0 \Leftrightarrow I^* = 0 \vee I^* = \frac{(\alpha - \beta)N}{\alpha}$$

We are looking only for $I^* > 0$, therefore, consider $I^* = \frac{(\alpha-\beta)N}{\alpha}$. Using the slope criterion [1, p. 49]:

$$f'(I) = \alpha - \beta - \frac{2\alpha I}{N}$$

$$f'(I^*) = \beta - \alpha$$

The infection sustains ad infinitum $\Leftrightarrow I^* > 0 \wedge f'(I^*) < 0 \Leftrightarrow \beta < \alpha$ (using $\alpha, \beta > 0$).

1.2 b) Stochastic model.

Assuming that in a short time interval the number of infected individuals changes by $+1$ or -1 because of

$$\text{new infection: } n-1 \to n \text{ at rate } \lambda_{n-1} = \alpha\left(1 - \frac{n-1}{N}\right)(n-1),$$

$$\text{recovery: } n+1 \to n \text{ at rate } \mu_{n+1} = \beta(n+1),$$

we obtain the Master equation for the probability $\rho_n(t)$ to observe n infected individuals at time t in a finite population of N individuals:

$$\rho_n(t+\Delta t) = \rho_n(t)[1 - \lambda_n \Delta t - \mu_n \Delta t] + \rho_{n-1}(t)\lambda_{n-1}\Delta t + \rho_{n+1}(t)\mu_{n+1}\Delta t$$

Assuming that Δt is small we obtain:

$$\frac{\partial \rho_n}{\partial t} = \frac{\rho_n(t+\Delta t) - \rho_n(t)}{\Delta t} = \lambda_{n-1}\rho_{n-1} + \mu_{n+1}\rho_{n+1} - (\mu_n + \lambda_n)\rho_n = (\mathbb{E}^- - 1)\lambda_n\rho_n + (\mathbb{E}^+ - 1)\mu_n\rho_n,$$

where $\mathbb{E}^\pm g_n = g_{n\pm1}$.

In order to consider large values of N, introduce $I = \frac{n}{N}$, define $\lambda(I) = \frac{\lambda_n}{N}$ and $\mu(I) = \frac{\mu_n}{N}$.

In case of smooth function g one may represent $\mathbb{E}^\pm$ in terms of derivatives:

$$\mathbb{E}^\pm g(I) = g\left(I \pm \frac{1}{N}\right) = \sum_{k=0}^{\infty} \frac{\left(\pm\frac{1}{N}\right)^k}{k!}\frac{\partial^k g}{\partial I^k} \equiv \exp\left(\pm\frac{1}{N}\frac{\partial}{\partial I}\right) g(I).$$

Expanding the equation to lower order in $\frac{1}{N}$, we obtain

$$\frac{\partial \rho}{\partial t} \approx \frac{\partial}{\partial I}[\mu(I) - \lambda(I)]\rho(I),$$

that corresponds to

$$\frac{\mathrm{d}\, I}{\mathrm{d}\, t} = \lambda(I) - \mu(I) = \alpha I(1 - I) - \beta I.$$

This is the deterministic dynamics (3) up to rescaling of I with a factor of N.

Nevertheless, there is an important difference between stochastic and deterministic dynamics, namely, when parameter values are such that the infection under deterministic dynamics persists ad infinitum, in stochastic case it will eventually die out because of stochastic deviations. Such a state corresponds to a stable in deterministic case and is called quasi-steady state, since it might be mistakenly considered as stable - the infection stays for a long time in the vicinity of this state and then die out.

1.3 c) Simulations in quasi-steady state.

The simulations were held using the fact, that time to next infection or recovery is distributed exponentially with mean λ_n^{-1} or μ_n^{-1} correspondingly. At time t we draw two numbers from corresponding distributions and execute the event with smaller time delay, updating the state and adding the delay to the current time.

Average at time t was calculated as

$$\frac{\sum_{i=1}^{150} \#\text{ invectives in run } i \text{ at time } t}{\#\text{ runs}},$$

conditional average:

$$\frac{\sum_{i=1}^{150} \#\text{ invectives in run } i \text{ at time } t}{\max\{1; \#\text{ runs in which } \#\text{ invectives at time } t \text{ is non zero}}\}.$$

The results are shown on Figure 1. The averages were calculated at times 0, 0.5, ..., 49.5, 50, 55, ..., 1545, 1550 with linear approximation in between. As one can see at short times curves agree, but then they begin to differ. The explanation of this is that in some runs the infectives go extinct and this is counted into entire average, but not in conditional, which includes only deviations around quasi-stable steady state. In case of deterministic dynamics under the same conditions the infection sustains ad infinitum, therefore, two curves would be exactly the same.

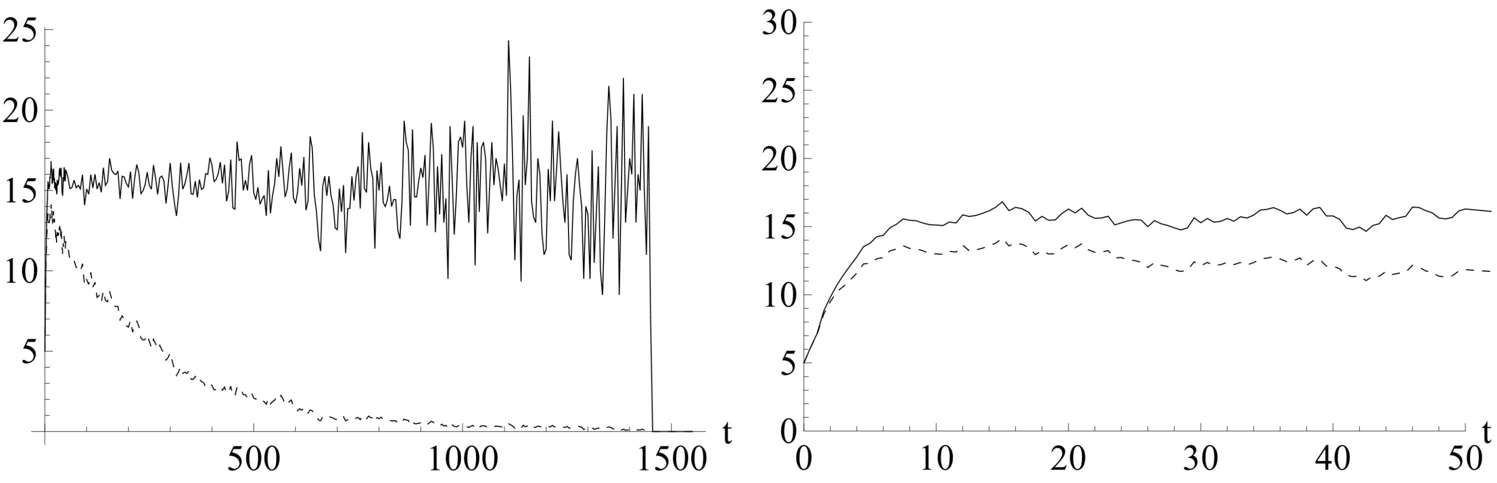


Figure 1: Average number of infected individuals (dashed - all runs, solid - conditional) vs. t. Entire time range is shown on the left, zoom in - right. 150 runs were held under $\alpha = 1.75$, $\beta = 1$, $N = 40$, $I(0) = 5$.

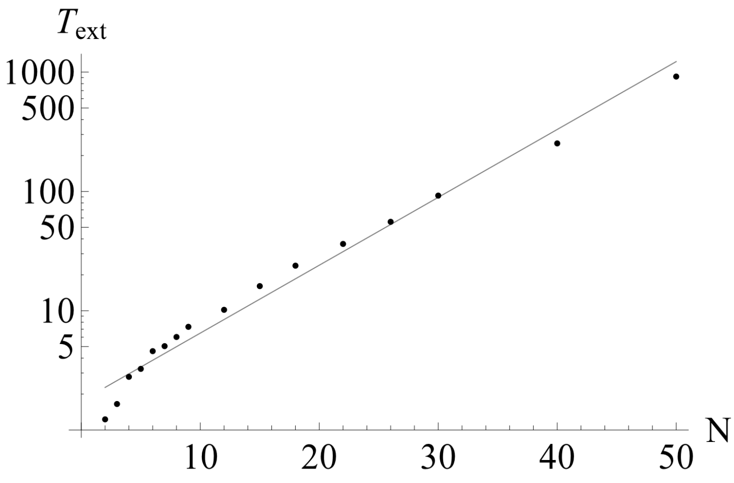
1.4 d) Extinction time.

According to the lecture notes [1, p. 307]

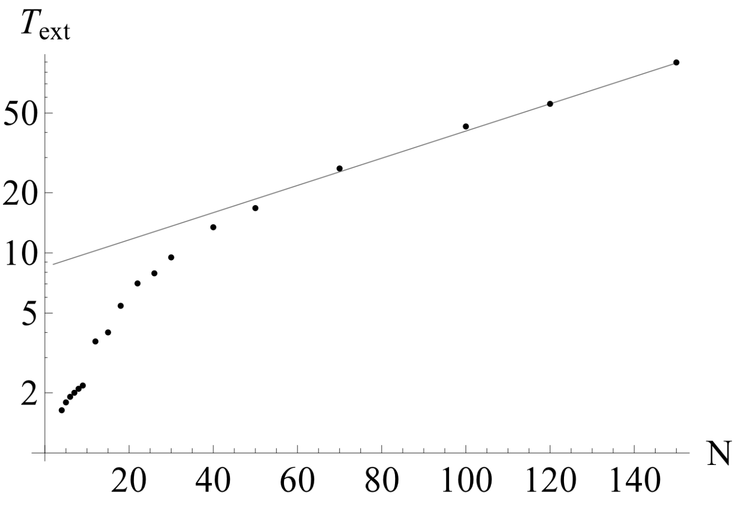
$$\log T_{ext} \sim N \left[\log \frac{\alpha}{\beta} - \left(1 - \frac{\beta}{\alpha}\right) \right] \tag{4}$$

i.e. $S(\alpha, \beta) \sim \log \frac{\alpha}{\beta} - \left(1 - \frac{\beta}{\alpha}\right)$.

Comparison of this expression and results of simulations are shown on Figure 2. Here we consider different values of α and β. As one can see in the first case ($\alpha = 1.75$) they agree well except region of $N < 4$, i.e. range of validity of Eq. (4) is $N \geq 4$, the second case ($\alpha = 1.25$) - $N \geq 70$.



(a) $\alpha = 1.75$, $\beta = 1$, $C = 1.75$, 300 runs per point.



(b) $\alpha = 1.25$, $\beta = 1$, $C = 8.5$, 1000 runs per point.

Figure 2: Time to extinction T_{ext} vs. population size N. Solid line - theory $C * \exp[N(\log \frac{\alpha}{\beta} - \left(1 - \frac{\beta}{\alpha}\right))]$, points - simulations.

1.5 e) Quasi-steady state distribution.

We compute the quasi-steady state distribution of the number of infected individuals as amount of time when there are exactly k infectives for $k \in \{1, ..., N\}$. In order to obtain more data we repeat the simulation with different random seeds and sum up obtained times. It was unclear whether including periods when the population goes extinct substantially influences the result. We checked this by comparing data obtained on all simulations and only on longer than 2000 time units (using only first parts of them with length 2000 time units). The results differ insignificantly. Another possible issue might be transient period. Since we start close to stable steady state in deterministic dynamics, one may expect no transient period, but we work with small value of N and the dynamics works like deterministic only for large N. In order to check this we compared data obtained on entire dataset and without initial part of length 100 time units in each run. This makes no difference, therefore, we work with entire dataset.

The comparison of results with theory is shown on Figure 3. In order to make the illustration clearer we plotted $\frac{-\log \text{distribution}}{N}$ instead of raw distribution. Numeric data from simulations was normalized (divided by maximal value) at first. As one can see theory predicts correctly deviations from a Gaussian of the right tail, but the left tail (corresponding to few infected individuals) is described poorly(nevertheless still better then a Gaussian).

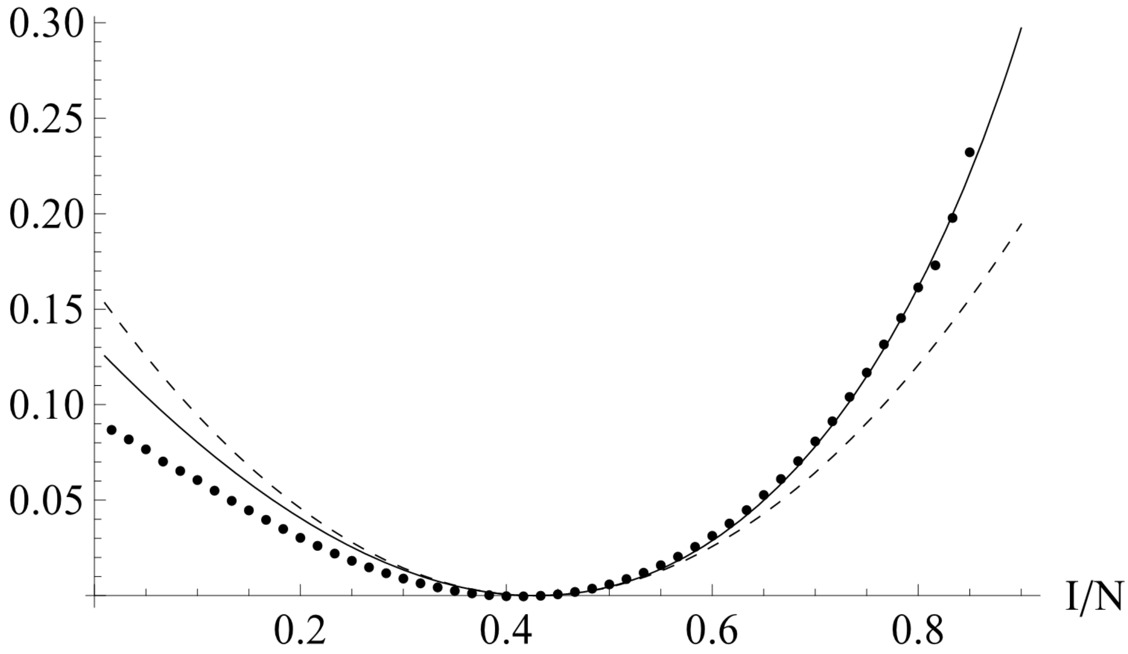


Figure 3: The quasy-steady state distribution of the number of infected individuals. Comparison of simulations and theory. $\frac{-\log \text{distribution}}{N}$ is plotted. Dots - simulations (150 runs), dashed line - a Gaussian $\frac{r_0}{2}[I-(1-\frac{1}{r_0})]^2$, solid line - prediction $(-\int_{1-r_0^{-1}}^{I} \mathrm{d}\,I' \log[r_0(1-I')] = -1 + r_0^{-1} + I - \log r_0^{-1} + \log[1-I] - I\log[r_0 - r_0 I])$, where $r_0 = \frac{\alpha}{\beta}$. Parameter values are $N = 60$, $\alpha = 1.75$, $\beta = 1$.

2 Synchronisation.

In this task we work with Kuramoto model, which describes the dynamics of the phases θ_i of N coupled oscillators

$$\frac{\mathrm{d}\,\theta_i}{\mathrm{d}\,t} = \omega_i + \frac{K}{N}\sum_{j=1} N \sin(\theta_j - \theta_i),$$

where frequencies ω_i are random, drawn from a symmetric distribution $g(\omega) = (\gamma/\pi)[\omega^2+\gamma^2]^{-1}$ (Lorentzian).

By means of simulations we investigate synchronization and compare results to the theory from the lecture notes.

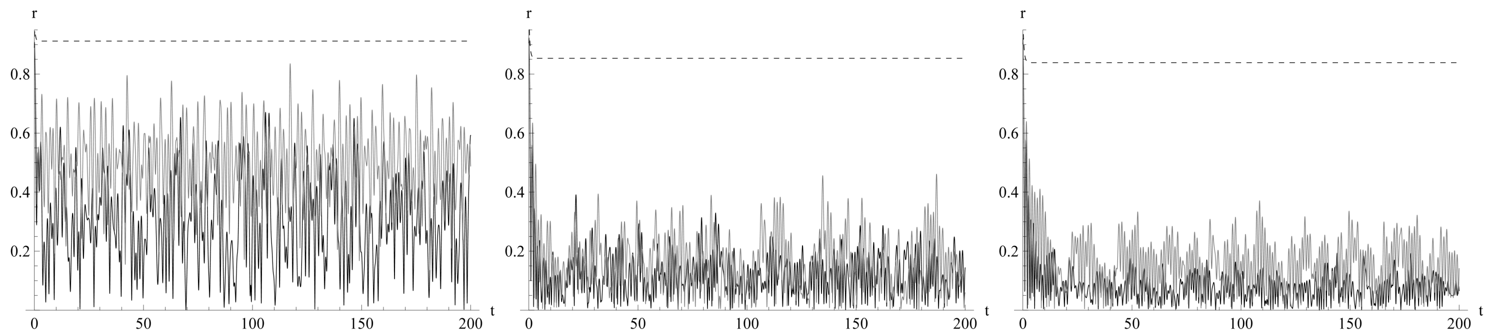
2.1 a) Theory.

According to the lecture notes [1, p. 324] the biffurcation value is $K_c = 2\gamma$ and in the vicinity of bifurcation the order parameter is

$$r = \sqrt{\frac{\mu}{1+\mu}} \Rightarrow C = (1+\mu)^{-\frac{1}{2}} \text{ with } \mu = \frac{K - K_c}{K_c}$$

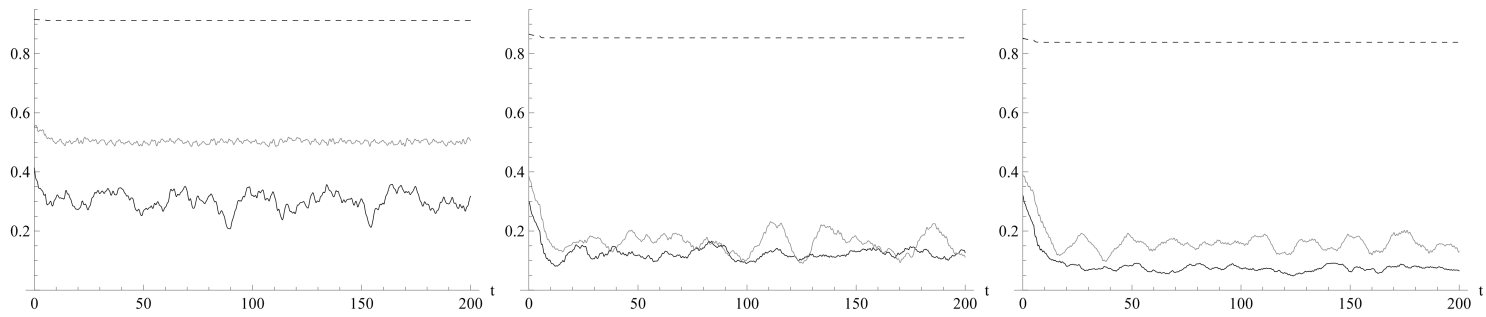
2.2 b) Simulations.

The raw results of simulation are shown on Figure 4, for better visualisation running averages are presented on Figure 5 . According to the mean-field theory from **2a** for values 5, 2.01, 1 of K order parameter should be close to ≈ 0.77; ≈ 0.07 and ≈ 0 correspondingly. As we can see on the Figure 5a for small values of N the theory works poorly, but with increasing of N to 100 and 300 (Figures 5b and 5c correspondingly) it begins to work much better (one may see slow convergence to predicted values with increasing of N). Therefore, our results confirms that the mean-field theory works better with higher number of oscillators N. In our case $N = 300$ is not sufficiently large and obtained values still differ from predicted ones. In addition to this one may see that fluctuations amplitude decreases with increasing of N as it was shown in the lecture notes.



(a) $N = 20$. (b) $N = 100$. (c) $N = 300$.

Figure 4: Order parameter r vs. time t under different K and N. Dashed curve corresponds to $K = 5$, gray - 2.01, black - 1. Other parameters are $\Delta t = 0.01$, $\gamma = 1$ ($K_c = 2$).



(a) $N = 20$. (b) $N = 100$. (c) $N = 300$.

Figure 5: Running average of order parameter r (window width 5 time units) vs. time t under different K and N. Dashed curve corresponds to $K = 5$, gray - 2.01, black - 1. Other parameters are $\Delta t = 0.01$, $\gamma = 1$ ($K_c = 2$).

References

[1] Bernhard Mehlig, *Computational Biology A: growth, morphogenesis, and ecological problems (lecture notes)*;

A File 1cd.nb:

```

codeDirectory = NotebookDirectory[];
CreateDirectory[codeDirectory <> "..\\report\\pics\\"]; // Quiet
SetDirectory[codeDirectory <> "..\\report\\pics\\"];
hack = Function[x, (*
   for static relative sizes of elements in plots while rasterizing *)


   First@ImportString[ExportString[x, "PDF"]]
   ];


alpha = 1.75; beta = 1. ; (*alpha > beta*)
lambda[n_, populationSize_] := alpha*(1 - n/populationSize)*n;
mu[n_, populationSize_] := beta*n;

(*mode = 0 - return curve *)
(*mode = 1 - return extinction time*)
(*mode = 2 - return distribution *)
GetCurve = Function[{populationSize, i0, seed, mode},
   Block[{i, time, result, used, l, m, timeToEvent, event, tl, tm},
    SeedRandom[seed];

    i = i0;
    time = 0;

    (*PrintTemporary[Dynamic[{i,time,used}]];*)

    If[mode == 0,
     result = Table[{0, i0}, {1000}];
     used = 2;
     ];

    If[mode == 2,
     result = Table[0, {populationSize}];
     ];

    While[i != 0,

     l = lambda[i, populationSize];
     m = mu[i, populationSize];
     If[l != 0,
      tl = RandomVariate[ExponentialDistribution[l]];
      ,
      tl = Infinity
      ];
     tm = RandomVariate[ExponentialDistribution[m]];

     If[tm < tl,
      timeToEvent = tm;
      event = -1;
      ,
      timeToEvent = tl;
      event = +1;
      ];

     (*
     l=lambda[i,populationSize];
```

```
      m=mu[i ,populationSize];
      timeToEvent = RandomVariate[ExponentialDistribution[l+m]];
      event=RandomChoice[{l ,m}\[Rule]{+1,-1}];
      *)

      If[mode == 2,
       result[[i]] += timeToEvent;
       ];

      time += timeToEvent;
      i += event;

      If[mode == 0,
       If[used == Length[result],
        result =
          Table[If[z < Length[result], result[[z]], {0, 0}], {z,
            2*Length[result]}];
        ];

       result[[used]][[1]] = time;
       result[[used]][[2]] = i;
      ++used;
       ];

      ];


     answer = {};

     If[mode == 0,
      answer = Drop[result, {used, Length[result]}];
      ];
     If[mode == 1,
      answer = time;
      ];

     If[mode == 2,
      answer = result;
      ];

     answer

     ]
    ];

(*1c*)
runs = 150;
done = 0;
SetSharedVariable[done];
PrintTemporary[Dynamic[done]];
curves = ParallelTable[++done;
   GetCurve[40, 5, seed, 0], {seed, 123, 123 + runs - 1}];

mid = 50;
times = Join[Table[i, {i, 0, mid, 0.5}],
   Table[i, {i, mid, max + 100, 5}]];
values = Table[0, {Length[times]}];
nonzeroQuantity = values;
zeroQuantity = values;
```

```
For[i = 1, i <= runs, ++i,
  index = 1;
  For[at = 1, at <= Length[times], ++at,
   atTime = times[[at]];
   While[index <= Length[curves[[i]]] &&
     curves[[i]][[index]][[1]] <= atTime, ++index];
   values[[at]] += curves[[i]][[index - 1]][[2]];
   If[curves[[i]][[index - 1]][[2]] == 0,
    ++zeroQuantity[[at]];
    ,
    ++nonzeroQuantity[[at]];
    ];
   ];
  ];

firstCurve =
  Table[{times[[i]],
    values[[i]]/(nonzeroQuantity[[i]] + zeroQuantity[[i]])}, {i,
    Length[times]}];
secondCurve =
  Table[{times[[i]],
    values[[i]]/(If[nonzeroQuantity[[i]] == 0, 1,
       nonzeroQuantity[[i]]])}, {i, Length[times]}];


pic = ListLinePlot[{firstCurve, secondCurve},
   PlotRange -> {{0, 50}, {0, 30}},
   PlotStyle -> {{Black, Dashed}, {Black}},
   AxesStyle -> Directive[18],
   AxesLabel -> {"t", ""}
   ];

Export["1c_a.png", hack[pic], ImageResolution -> 300];
pic = Show[pic, PlotRange -> All];
Export["1c_b.png", hack[pic], ImageResolution -> 300];


(*d*)
alpha = 1.75;
beta = 1.;

runs = 15;
Ns = {2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 18, 22, 26, 30, 40, 50};
CreateDirectory[codeDirectory <> "1d"];
For[startSeed = 123, startSeed < 123 + 300, startSeed += runs,
  times = Ns;
  For[i = 1, i <= Length[Ns], ++i,
   state = Round[(alpha - beta)*Ns[[i]]/alpha];
   times[[i]] =
    ParallelTable[
     GetCurve[Ns[[i]], state, startSeed + z - 1, 1], {z, 1, runs}];
   ];

  Export[codeDirectory <> "1d\\1d_" <> ToString[startSeed] <> ".txt",
   times, "Table"];
  ];
Itimes = Import[codeDirectory <> "1d\\1d_123.txt", "Table"];
For[ti = 123 + runs, ti < startSeed, ti += runs,
  t = Import[codeDirectory <> "1d\\1d_" <> ToString[ti] <> ".txt",
    "Table"];
```

```
  For[z = 1, z <= Length[Itimes], ++z,
   Itimes[[z]] = Join[Itimes[[z]], t[[z]]];
   ];
  ];

theory = Table[{n,
    1.75*Exp[n*(Log[alpha/beta] - (1 - beta/alpha))]}, {n, Min[Ns],
    Max[Ns], 0.01}];

pic = ListLogPlot[
  {Table[{Ns[[i]], Mean[Itimes[[i]]]}, {i, Length[Ns]}],
   theory
   },
  PlotStyle -> {Black, Gray},
  Joined -> {False, True},
  AxesStyle -> Directive[18],
  AxesLabel -> {"N", "\!\(\*SubscriptBox[\(T\), \(ext\)]\)"}
  ]
Export["1d.png", hack[pic], ImageResolution -> 300];


(*d*)
alpha = 1.25;
beta = 1.;

runs = 10;
Ns = {2, 3, 4, 5, 6, 7, 8, 9, 12, 15, 18, 22, 26, 30, 40, 50, 70, 100,
    120, 150};
CreateDirectory[codeDirectory <> "1d"];
For[startSeed = 123, startSeed < 123 + 1000, startSeed += runs,
  times = Ns;
  For[i = 1, i <= Length[Ns], ++i,
   state = Round[(alpha - beta)*Ns[[i]]/alpha];
   times[[i]] =
    ParallelTable[
     GetCurve[Ns[[i]], state, startSeed + z - 1, 1], {z, 1, runs}];
   ];

  Export[codeDirectory <> "1d\\1d_" <> ToString[startSeed] <> ".txt",
   times, "Table"];
  ];
Itimes = Import[codeDirectory <> "1d\\1d_123.txt", "Table"];
For[ti = 123 + runs, ti < startSeed, ti += runs,
  t = Import[codeDirectory <> "1d\\1d_" <> ToString[ti] <> ".txt",
    "Table"];
  For[z = 1, z <= Length[Itimes], ++z,
   Itimes[[z]] = Join[Itimes[[z]], t[[z]]];
   ];
  ];

theory = Table[{n,
    8.5*Exp[n*(Log[alpha/beta] - (1 - beta/alpha))]}, {n, Min[Ns],
    Max[Ns], 0.01}];

pic = ListLogPlot[
  {Table[{Ns[[i]], Mean[Itimes[[i]]]}, {i, Length[Ns]}],
   theory
   },
  PlotStyle -> {Black, Gray},
  Joined -> {False, True},
```

```
  AxesStyle -> Directive[18],
  AxesLabel -> {"N", "\!\(\*SubscriptBox[\(T\), \(ext\)]\)"}
  ]
Export["1d_b.png", hack[pic], ImageResolution -> 300];
```

B File `1e.nb`:

```
codeDirectory = NotebookDirectory[];
CreateDirectory[codeDirectory <> "..\\report\\pics\\"]; // Quiet
SetDirectory[codeDirectory <> "..\\report\\pics\\"];
hack = Function[x, (*
   for static relative sizes of elements in plots while rasterizing *)


   First@ImportString[ExportString[x, "PDF"]]
   ];

alpha = 1.75; beta = 1. ; (*alpha > beta*)
lambda[n_, populationSize_] := alpha*(1 - n/populationSize)*n;
mu[n_, populationSize_] := beta*n;

(*mode = 0 - return curve *)
(*mode = 1 - return extinction time*)
(*mode = 2 - return distribution *)
GetCurve = Function[{populationSize, i0, seed, mode, endTime},
   Block[{i, time, result, used, l, m, timeToEvent, event, tl, tm},
    SeedRandom[seed];

    i = i0;
    time = 0;

    (*PrintTemporary[Dynamic[{i,time,used}]];*)

    If[mode == 0,
     result = Table[{0, i0}, {1000}];
     used = 2;
     ];

    If[mode == 2,
     result = Table[0, {populationSize}];
     ];

    While[i != 0 && time < endTime,

     l = lambda[i, populationSize];
     m = mu[i, populationSize];
     If[l != 0,
      tl = RandomVariate[ExponentialDistribution[l]];
      ,
      tl = Infinity
      ];
     tm = RandomVariate[ExponentialDistribution[m]];

     If[tm < tl,
      timeToEvent = tm;
      event = -1;
      ,
      timeToEvent = tl;
      event = +1;
      ];
```

```

      If[i >= populationSize && event == +1,
       Print["err"];
       Print[{l, m, tl, tm}];
       Return[1];
       ];

      (*
      l=lambda[i,populationSize];
     m=mu[i,populationSize];
      timeToEvent = RandomVariate[ExponentialDistribution[l+m]];
      event=RandomChoice[{l,m}\[Rule]{+1,-1}];
      *)

      If[mode == 2,
       result[[i]] += timeToEvent;
       ];

      time += timeToEvent;
      i += event;

      If[mode == 0,
       If[used == Length[result],
        result =
          Table[If[z < Length[result], result[[z]], {0, 0}], {z,
            2*Length[result]}];
        ];

       result[[used]][[1]] = time;
       result[[used]][[2]] = i;
      ++used;
       ];

      ];


    answer = {};

    If[mode == 0,
     answer = Drop[result, {used, Length[result]}];
     ];
    If[mode == 1,
     answer = time;
     ];

    If[mode == 2,
     answer = result;
     ];

    answer

    ]
   ];

alpha = 1.75;
beta = 1.;
n = 60;
state = Round[(alpha - beta)*n/alpha];

runs = 150;
```

```
endTime = Infinity;

done = 0;
SetSharedVariable[done];

CreateDirectory[codeDirectory <> "1e"];
ParallelDo[
  curve = GetCurve[n, state, 123 + seed, 0, endTime];
  Export[codeDirectory <> "1e\\" <> ToString[seed] <> ".txt", curve,
   "Table"];
  ++done;
  , {seed, 0, runs - 1}];

alpha = 1.75;
beta = 1.;
r0 = alpha/beta;
n = 60;
state = Round[(alpha - beta)*n/alpha];

distr = Table[0, {n}];

For[seed = 0, seed < runs, ++seed,
  curve =
   Import[codeDirectory <> "1e\\" <> ToString[seed] <> ".txt",
    "Table"];
  For[i = 1, i + 1 <= Length[curve], ++i,
   val = curve[[i]][[2]];
   time = curve[[i + 1]][[1]] - curve[[i]][[1]];
   distr[[val]] += time;
   ];
  ];

alpha = 1.75;
beta = 1.;
r0 = alpha/beta;
n = 60;


t = Integrate[Log[r0*(1 - x)], x];
v = -FullSimplify[-(t /. {x -> 1 - r0^(-1)} ) + (t /. {x -> ti})];
d = distr;
s = Max[d];
d /= s;
d = Table[{i/n, -Log[d[[i]]]/n}, {i, 1, n}];
pic = Show[
   Plot[{v, r0/2*(ti - (1 - 1/r0))^2}, {ti, 0.01, 0.9},
    PlotStyle -> {{Black}, {Black, Dashed}},
    AxesLabel -> {"I/N", ""},
    AxesStyle -> Directive[14]
    ],
   ListPlot[d, PlotStyle -> Black, PlotMarkers -> {Automatic, 5}]
   ];
Export["1e.png", hack[pic], ImageResolution -> 300];
```

C File 2.nb:

```

codeDirectory = NotebookDirectory[];
CreateDirectory[codeDirectory <> "..\\report\\pics\\"]; // Quiet
SetDirectory[codeDirectory <> "..\\report\\pics\\"];
```

```
hack = Function[x, (*
   for static relative sizes of elements in plots while rasterizing *)


   First@ImportString[ExportString[x, "PDF"]]
   ];

Simulate = Function[{n, K, gamma, seed, dt, endTime},
   Block[{omega, phase, time, ksi, r, other, used, curve, x, y},

    SeedRandom[seed];

    omega = RandomVariate[CauchyDistribution[0, gamma], n]; (*unclear*)


    omega = Mod[omega, 2*Pi];(*unclear*)

    phase = Table[RandomReal[{-Pi/2; Pi/2}], {n}];

    curve = Table[{i, r}, {i, 0, endTime, dt}];
    used = 1;


    For[time = 0, time <= endTime, time += dt,


     x = Sum[Cos[phase[[i]]], {i, n}];
     y = Sum[Sin[phase[[i]]], {i, n}];
     r = Sqrt[x*x + y*y]/n;

     curve[[used]][[2]] = r;

     ++used;
     phase +=
      dt*(omega +
         K/n*Table[Sum[Sin[phase[[j]] - phase[[i]]], {j, n}], {i, n}]);

     ];

    curve
    ]
   ];

gamma = 1;
Kc = 2*gamma;


Ns = {20, 100, 300};
Ks = {1, 2.01, 2.1, 5};

gamma = 1;
seed = 123;
dt = 0.01;
endTime = 200;

CreateDirectory[codeDirectory <> "raw"];
For[runs = 1, runs <= 2, ++runs,
  ParallelDo[
    n = Ns[[iNs]];
    K = Ks[[iKs]];
```

```
    data = Simulate[n, K, gamma, seed + runs - 1, dt, endTime];
    Export[
     codeDirectory <> "raw//" <> ToString[runs] <> "_" <>
      ToString[iKs] <> "_" <> ToString[iNs] <> ".txt", data, "Table"];
    , {iKs, Length[Ks]}, {iNs, Length[Ns]}
    ];
  ];


runs = 2;
iKss = {1, 2, 3};
pics = Table[
  ListLinePlot[
   Table[
    t = Import[
      codeDirectory <> "raw//" <> ToString[runs] <> "_" <>
       ToString[iKss[[iKs]]] <> "_" <> ToString[iNs] <> ".txt",
      "Table"];
    td = Map[#[[2]] &, t];
    td = MeanFilter[td, 0];
    t = Table[{t[[i]][[1]], td[[i]]}, {i, Length[t]}];
    t
    , {iKs, 3}]
   , PlotStyle -> {{Black}, {Gray}, {Black, Dashed}}
   , AxesLabel -> {"t", "r"}
   , AxesStyle -> Directive[12]
   , PlotRange -> {0, 0.95}
   ]
  , {iNs, 3}]
For[i = 1, i <= Length[pics], ++i,
  Export["2b_part_1_" <> ToString[i] <> ".png", hack[pics[[i]]],
    ImageResolution -> 300];
  ];


iKss = {1, 2, 3};
pics = Table[
  ListLinePlot[
   Table[
    t = Import[
      codeDirectory <> "raw//" <> ToString[runs] <> "_" <>
       ToString[iKss[[iKs]]] <> "_" <> ToString[iNs] <> ".txt",
      "Table"];
    td = Map[#[[2]] &, t];
    td = MeanFilter[td, 500];
    t = Table[{t[[i]][[1]], td[[i]]}, {i, Length[t]}];
    t
    , {iKs, 3}]
   , PlotStyle -> {{Black}, {Gray}, {Black, Dashed}}
   , AxesLabel -> {"t", ""}
   , AxesStyle -> Directive[12]
   , PlotRange -> {0, 0.95}
   ]
  , {iNs, 3}]
For[i = 1, i <= Length[pics], ++i,
  Export["2b_part_1_" <> ToString[i] <> "_r.png", hack[pics[[i]]],
    ImageResolution -> 300];
  ];
```