Оглавление

Обзор литературы

1. Constable GW, McKane AJ. Models of genetic drift as limiting forms of the Lotka-Volterra competition model. Phys Rev Lett. 2015 Jan 23;114(3):038101. doi: 10.1103/PhysRevLett.114.038101. Epub 2015 Jan 22.PMID: 25659024

В данной статье описывается связь стохастической модели Лотки-Вольтерры с конвенциональными моделями генетического дрейфа, в частности с моделью Морана. Помимо всего прочего, приведены основные элементы модели Лотки-Вольтерры, которые будут в дальнейшем использованы для моделирования задач.

Мы начинаем с популяции, в которой находятся n_1 гаплоидных носителей первой аллели A_1 и n_2 носителей второй A_2. Они рождаются с интенсивностями b_1 и b_2 и умирают с интенсивностями d_1 и d_2 соответственно.

Предположение о том, что размер популяции N является фиксированным (fixed) имеет за собой как биологические, так и математические причины, и поэтому едва бывает когда-либо поставлен под вопрос.

В отличие от популярных моделей Райта-Фишера и Морана, постулирование постоянности размера популяции не используется, поскольку такой подход способствует тому, что один параметр отвечает как за рождение, так и за смерть, что не позволяет разделить влияния каждого из этих по своему существу не совсем диаметрально противоположных процессов.

В некоторых других моделях вводится понятие эффективного размера популяции для избегания изменений в размере популяции (например, в среднем) для поддержания размера популяции N.

В модели Лотки-Вольтерры регулятором постоянства общего количества аллелей служит понятие конкуренции среди представителей как разных, так и одной и той же аллели, математически вводимое коэффициентом c_{ij}, который

показывает, с какой интенсивностью носитель аллели A_i погибает в процессе конкуренции с носителем аллели A_j.

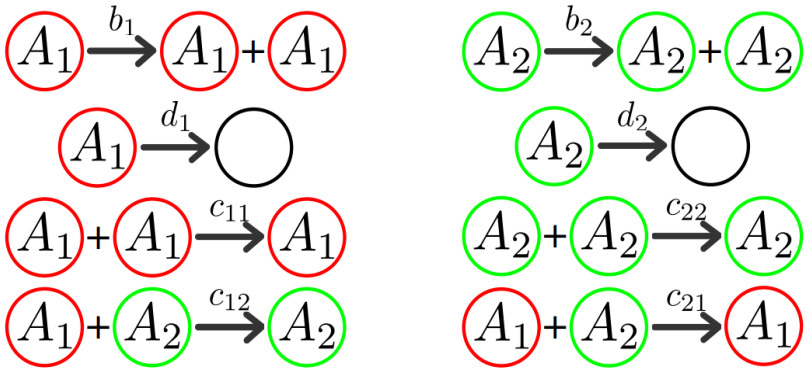

Рисунок 0. Рождение, смерть и конкуренция носителей аллели.

Состояние системы однозначно определяется парой (n_1, n_2) – количествами носителей каждой аллели. Следующим состоянием может быть одно из четырех возможных: $(n_1 + 1, n_2)$, $(n_1, n_2 + 1)$, $(n_1 - 1, n_2)$, $(n_1, n_2 - 1)$. Первые два могут быть достигнуты только рождением новой особи, а последние два – гибелью или следствием конкуренции, которые приведены на Рисунке 0.

Для перехода из текущего состояния в следующее вводятся интенсивности:

$$T_1(n_1 + 1, n_2 | n_1, n_2) = b_1 \frac{n_1}{V}$$

$$T_2(n_1, n_2 + 1 | n_1, n_2) = b_2 \frac{n_2}{V}$$

$$T_3(n_1 - 1, n_2 | n_1, n_2) = d_1 \frac{n_1}{V} + c_{11} \frac{n_1}{V} \frac{n_1}{V} + c_{12} \frac{n_2}{V} \frac{n_1}{V}$$

$$T_4(n_1, n_2 - 1 | n_1, n_2) = d_2 \frac{n_2}{V} + c_{22} \frac{n_2}{V} \frac{n_2}{V} + c_{21} \frac{n_1}{V} \frac{n_2}{V}$$

Параметр V – это не общее население популяции. Это некий размер системы, в котором расположена популяция. Обычно это площадь или объем территории, на которой обитают представители популяции.

2. Blythe, R. A.; McKane, A. J. Stochastic models of evolution in genetics, ecology and linguistics. J. Stat. Mech. No 07, P07018, 2007.

В данной статье конкретно модель Лотки-Вольтерры отсутствует, но дается общее представление о стохастических моделях эволюции и их применение в генетике, экологии и лингвистике. В многих случаях эти модели разрешимы и дают представление об особенностях процессов эволюции в системе.

В статье приведены биологические термины, некоторые из которых в дальнейшем будут использоваться в данной работе:

Ген – переменная единица, которая несет информацию о некоторой черте особи.

Аллель – вариант гена.

Гаплоидный организм – это организм, который несет в себе только один вариант каждого гена.

Диплоидный организм – это организм, который несет в себе два варианта каждого гена.

Нейтральная модель – это модель без естественного отбора.

Идеальная популяция – это популяция, в которой особь, заменяемая в процессе рождения нового организма, выбирается с помощью равномерного распределения

Генетический дрейф - случайные изменения частот аллелей, происходящие в популяции при смене поколения.

В частности, в статье речь идет о нейтральных моделях, которыми могут, кстати говоря, являться некоторые частные случаи моделей Лотки-Вольтерры.

Сначала рассматривается модель Райта-Фишера как марковский процесс с дискретным временем, в котором общее количество аллелей считается постоянным каждое поколение, а каждый ген в текущем поколении является точной копией какого-то гена из прошлого. Все организмы являются гаплоидными и размножаются бесполым путем. Новое поколение $t+1$ возникает за счет бесконечного количества гамет, случайно образованных организмами поколения t перед смертью. Вводятся понятия о вероятностях перехода от одного состояния системы к другому, вектора и матрицы переходных вероятностей.

Вектор $\underline{P}(t) = \begin{pmatrix} P(1,t) \\ P(2,t) \\ P(3,t) \\ \vdots \end{pmatrix}$ содержит компоненты $P(i,t)$, которые равны вероятности, что в поколении t будет i аллелей A

Коэффициенты $p_{n\,n'}$ матрицы перехода $P = \begin{pmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{pmatrix}$ показывают шанс того, что при текущем количестве $n'A$ аллелей в следующем поколении будет n A аллелей. В дальнейшем матрица используется для нахождения собственных значений.

Сумма компонент вектора и суммы коэффициентов каждого столбца матрицы должны равняться единице. Такие матрицы называют стохастическими.

Матрица и вектор связаны следующим образом:

$$\underline{P}(t) = P\underline{P}(t-1) = PP\underline{P}(t-2) = P^t\underline{P}(0)$$

Затем идет обзор модели Морана, которая, как далее выяснится, связана с моделью Лотки-Вольтерры. В отличие от Райта-Фишера, после N выборок ал-

лелей, где N – размер популяции, не обязательно каждый ген обязан быть заменен. Поколением считается период времени, за который происходит одна выборка аллели. Поэтому за одну смену поколение количество аллелей может либо уменьшится на 1, либо не измениться, либо увеличиться на 1. Следовательно, матрица перехода получается трехдиагональная, что позволяет найти точные значения собственных значений и соответствующие собственные вектора.

Далее авторы отходят от концепции дискретного времени и переходят к непрерывному. Для этого вводятся понятия интенсивности (скорости) перехода за единицу времени, которые связаны следующим образом:

$$P_{n\,n'} = T(n|n')\Delta t + O(\Delta t)^2$$

Далее в статье рассматриваются мутации, острова, миграции, уравнение Фоккера-Планка, коалесцентная теория, неидеальные популяции, эффективный размер популяции, естественного отбора, которые, однако, не представляют ценности для данной научно-исследовательской работы.

3. Карлин С. Основы теории случайных процессов. М.: "Мир", 1971, — 537 с.

Данная книга содержит как основные элементы теории вероятностей, так и теории случайных процессов, причем присутствует целая глава, посвященная случайным (а также детерминированным) генетическим и экологическим процессам.

Пропуская определения функций распределения, плотности, математического ожидания, производящих функций и прочих базовых элементов теории, стоит, однако, обратить внимание на известную, и крайне важную теорему, которая является строгим математическим аргументом для справедливости анализа математической модели, речь о которой пойдет в данной работе. Речь идет о *Законе больших чисел*:

(*Слабая формулировка*) Пусть $X_1, X_2, \ldots$ - независимые одинаково распределенные случайные величины со средним m и конечной дисперсией. Тогда для любого $\varepsilon > 0$

$$\lim_{n\to\infty} P\left\{\left|\frac{X_1 + \cdots + X_n}{n} - m\right| \le \varepsilon\right\} = 1$$

(*Усиленная формулировка*) Пусть $X_1, X_2, \ldots$ - те же случайные величины. Тогда для любых $\varepsilon, \delta > 0$ существует целое число $N(\varepsilon, \delta)$, такое, что

$$\lim_{n\to\infty} P\left\{\left|\frac{X_1 + \cdots + X_n}{n} - m\right| \ge \varepsilon \text{ для всех } n \ge N(\varepsilon, \delta)\right\} < \delta$$

Далее в книге приводится определения марковского и стационарного процесса, кои будут иметь место в модели.

Пусть задана случайная величина X_t, которая принимает некое значение в момент времени t. Марковским процессом называется такой процесс, что является верным следующее:

$$P\left(a < X_t \le b \middle| X_{t_1} = x_1, X_{t_2} = x_2, \ldots, X_{t_n} = x_n\right) = P\left(a < X_t \le b \middle| X_{t_n} = x_n\right)$$

при $t_1 < t_2 < \cdots < t_n < t$.

Стационарным процессом в узком процессе называется такой процесс, что совместные распределения семейств случайных величин

$$(X_{t_1+h}, X_{t_2+h}, \ldots, X_{t_n+h}) \text{ и } (X_{t_1}, X_{t_2}, \ldots, X_{t_n})$$

одинаковы при всех $h > 0$ и всех $t_1, t_2, \ldots, t_n$

Среди примеров марковских цепей, приводимых в дальнейшем в книге, особенно примечательны так называемые “случайные блуждания”.

Одномерный случай строится следующим образом: если текущее состояние объекта i, то за один шаг он может перейти в одно из соседних состояний ($i-1$ или $i+1$), либо остаться в i. Если пространство состояний – множество неотрицательных целых чисел, то матрица переходных вероятностей случайного блуждания имеет вид:

$$P=\begin{pmatrix} r_0 & p_0 & 0 & \cdots & \ldots & & & \\ q_1 & r_1 & p_1 & 0 & \ldots & & & \\ 0 & q_2 & r_2 & p_2 & \ldots & & & \\ \vdots & . & & \ddots & & & & \\ & & . & 0 & q_i & r_i & p_i & 0 \\ & & & & . & . & . & . \end{pmatrix}$$

где $q_i > 0, r_i > 0, p_i \geq 0$, q_i+r_i+p_i=1, $i = 1,2, \ldots, r_0 \geq 0, p_0 \geq 0, r_0 + p_0 = 1$

$P(X_{n+1} = i+1 | X_n = i) = p_i;\ P(X_{n+1} = i-1 | X_n = i) = q_i;$

$P(X_{n+1} = i | X_n = i) = r_i$

Для n −мерного случая случайного блуждания соседними состояниями является целочисленная решетка в E^n, точки которой это наборы целочисленных чисел. В классическом симметричном варианте вероятности определяются следующим образом:

$$P_{kl}=\begin{cases} \dfrac{1}{2n}, \text{если } \displaystyle\sum_{i=1}^{n} |l_i - k_i| = 1 \\ 0 \text{ в противном случае} \end{cases}$$

Несколько глав далее начинается описание марковский цепей с непрерывным временем. Вводится понятие пуассоновского процесса:

Пуассоновский процесс – марковский процесс, принимающий неотрицательные целочисленные значения и обладающий следующими свойствами:

1. $P(X(t+h) - X(t) = 1 | X(t) = x) = \lambda h + o(h)$ при $h \to 0$ $(x = 0, 1, 2..)$
2. $P(X(t+h) - X(t) = 0 | X(t) = x) = 1 - \lambda h + o(h)$ при $h \to 0$
3. $X(0) = 0$

Затем рассматриваются примеры таких процессов, одним из которых является процесс рождения и гибели, который представляет собой марковский процесс с состояниями 0, 1, 2, … и стационарными вероятностями перехода, т.е.

$$P_{ij}(h) = P(X(t+h) = j | X(t) = i)$$

$P_{ij}(t)$ удовлетворяет следующим постулатам:

1. $P_{i,i+1}(h) = \lambda_i h + o(h)$ при $h \to 0, i \geq 0$ – вероятность рождения
2. $P_{i,i-1}(h) = \mu_i h + o(h)$ при $h \to 0, i \geq 1$ – вероятность гибели
3. $P_{i,i}(h) = 1 - (\lambda_i + \mu_i)h + o(h)$ при $h \to 0, i \geq 0$
4. $P_{i,j}(o) = \delta_{ij}$
5. $\mu_0 = 0, \lambda_0 > 0,\ \lambda_i, \mu_i > 0$

Матрица переходных вероятностей имеет следующий вид:

$$P = \begin{pmatrix} -\lambda_0 & \lambda_0 & 0 & \cdots & \ldots & & & \\ \mu_1 & -(\mu_1 + \lambda_1) & \lambda_1 & 0 & \ldots & & & \\ 0 & \mu_2 & -(\mu_2 + \lambda_2) & \lambda_2 & \ldots & & & \\ \vdots & . & & \ddots & & & & \\ & & . & 0 & \mu_i & -(\mu_i + \lambda_i) & \lambda_i & 0 \\ & & & & . & . & . & . \end{pmatrix}$$

Коэффициенты удовлетворяют уравнению Чепмена-Колмогорова:

$$P_{ij}(t+s) = \sum_{k \in E} P_{ik}(t) P_{kj}(s) \quad \forall\, i, j = 1, 2, \ldots; t, s \geq 0$$

При такой математической модели возникает крайне важный факт, который строго доказывается в книге – длительность пребывания процесса $X(t)$ в состоянии i имеет показательное распределение с параметром $\lambda = \lambda_i + \mu_i$.

4. Hening, A., Nguyen, D.H. Stochastic Lotka–Volterra food chains. J. Math. Biol. 77, 135–163 (2018).

В данной статье рассматривается модель Лотки-Вольтерра из $n-1$ хищника и 1 жертвы. $j-1$-ый хищники поедают j-ые особи и пожираются $j+1$-ыми хищниками.

Модель описывается следующей системой дифференциальных уравнений:

$$\begin{aligned}
dx_1(t) &= x_1(t)(a_{10} - a_{11}x_1(t) - a_{12}x_2(t))\,dt \\
dx_2(t) &= x_2(t)(-a_{20} + a_{21}x_1(t) - a_{23}x_3(t))\,dt \\
&\vdots \\
dx_{n-1}(t) &= x_{n-1}(t)(-a_{n-1,0} + a_{n-1,n-2}x_{n-2}(t) - a_{n-1,n}x_n)\,dt \\
dx_n(t) &= x_n(t)(-a_{n0} + a_{n,n-1}x_{n-1}(t))\,dt.
\end{aligned}$$

где $x_i(t)$ – плотность i-ых особей.

Будем называть систему постоянной, если любое решение $x(t) = (x_1(t), \dots, x_n(t))$ с $x(0) \in R_+^n$ удовлетворяет:

$$\lim_{t\to\infty} \sup x_i(t) > 0, i = 1, \dots, n$$

В противном случае, если для особей i верно:

$$\lim_{t\to\infty} x_i(t) = 0$$

то говорим, что особи i вымерли.

Было показано, что постоянность или вымирание может быть определена одним единственным параметром, который зависит от коэффициентов a_{ij}. Он определяется следующим образом:

$$\kappa(n) = \kappa = a_{10} - \frac{a_{11}}{a_{21}}\left[a_{20} + \sum_{j=2}^{k}\left(\prod_{i=2}^{j}\frac{a_{2i-2,2i-1}}{a_{2i,2i-1}}\right)a_{2j,0}\right] - \sum_{j=1}^{l}\left(\prod_{i=1}^{j}\frac{a_{2i-1,2i}}{a_{2i+1,2i}}\right)a_{2j+1,0}$$

где $k = [\frac{n}{2}]$, $l = \begin{cases} \frac{n}{2} - 1, \text{если } n - \text{четное} \\ \frac{(n-1)}{2}, \text{если } n - \text{нечетное} \end{cases}$

Теорема 4.1 Пищевая цепь, определенная выше, является постоянной, если к$(n) > 0$ и не является (т.е. какие-то особи вымирают), если к$(n) < 0$.

5. Mimmo Iannelli, Andrea Pugliese An Introduction to Mathematical Popula-tion Dynamics: Along the trail of Volterra and Lotka Unitext 79 - La Ma-tematica per il 3+2 Springer International Publishing 2014.

Высокий интерес представляет глава о стохастических моделях роста популяции. В ней написано, что в стохастических моделях будущие состояния популяции абсолютно являются результатом выборки из некого случайного распределения, и что точно предсказать будущее нельзя, даже если знать настоящее, за исключением некоторых тривиальных случаев (например, носители одной или всех аллелей вымерли).

В главе рассматривается демографическая стохастичность (demographic stochasticity), в которых биологические популяции конечны и дискретны – они меняются только если один или более индивидов рождается, умирает, иммигрирует или эмигрирует.

Рассматривается процесс рождения и смерти со следующими предположениями:

$$P(N(t+h) = i+1 | N(t) = i) = \lambda_i h + o(h)$$
$$P(N(t+h) = i-1 | N(t) = i) = \mu_i h + o(h)$$
$$\sum_{j:|j-i|>1} P(N(t+h) = j | N(t) = i) = o(h)$$

где $N(t)$ – случайная величина, означающая количество индивидов в момент времени t.

$N(t)$ не может быть отрицательной, поэтому необходимо предположить, что $\mu_0 = 0$.

В частности, рассматриваются две модели роста популяции:

1. Мальтузианская, в которой интенсивности рождения и смерти линейно зависят от текущего количества индивидов в популяции:

$$\lambda_i = \lambda i, \mu_i = \mu i$$

где λ и μ положительные константы

2. Логистическая, в которой интенсивности имеют следующий вид:

$$\lambda_i = \lambda i, \mu_i = \mu i + vi^2$$

где константы λ, μ и v положительные.

Вторая модель представляет особый интерес, поскольку рассматриваемая в этой работе модель Лотки-Вольтерры является ее представителем.

Следующий параграф главы посвящен вопросу о вероятности вымирания популяции, что уже гораздо более релевантно. Предполагается:

$$\lambda_0 = 0,\ \lambda_i + \mu_i > 0 \text{ при } i > 0$$

Первое условие означает, что в состоянии 0, т.е. при гибели каждого индивида, популяции рождение невозможно, второе – то, что состояние 0 – единственное, которое обладает таким свойством.

Приводятся формулы для вероятности рождения и гибели:

$$p_i = \frac{\lambda_i}{\lambda_i + \mu_i}, q_i = \frac{\mu_i}{\lambda_i + \mu_i}$$

После нескольких страниц математических выкладок выходит то, что популяция, подчиняющаяся логистической модели, всегда обречена на вымирание вне зависимости от начального состояния.

Для Мальтузианской модели условие $\lambda \leq \mu$ гарантирует вымирание популяции.

Для этой модели также рассматривается вопрос о времени вымирания популяции.

$$\tau = \inf\{t: N(t) = 0\}$$

Итого:

$$M[\tau|N(0) = i] = +\infty \text{ при } \lambda = \mu$$

$$M[\tau|N(0) = 1] = \frac{1}{\lambda}\log\left(\frac{\mu}{\mu - \lambda}\right) \text{ при } \lambda < \mu$$

Затем ставится такой же вопрос при ограничении размера популяции следующим образом:

$$\lambda_i = \begin{cases} \lambda \text{i}, & \text{если i} < K \\ 0, & \text{если иначе} \end{cases}, \mu_i = \mu i$$

что позволяет рассмотреть задачу при $\lambda > \mu$

Получаем:

$$M[\tau|N(0) = 1] \approx \frac{1}{\left(1 - \frac{\mu}{\lambda}\right)K}\left(\frac{\lambda}{\mu}\right)^{K-1} \text{ при } \lambda > \mu$$

$$M[\tau|N(0) = 1] \approx \log(K) + \gamma \text{ при } \lambda = \mu$$

$$M[\tau|N(0) = 1] = \frac{1}{\lambda}\log\left(\frac{\mu}{\mu - \lambda}\right) \text{ при } \lambda < \mu$$

Описание рассматриваемых моделей

Имеется неограниченная популяция, в которой n_1 носителей первой аллели A_1 и n_2 носителей второй аллели A_2, причем $n_1 \geq 1, n_2 \geq 1$. Они могут размножаться с интенсивностями b_1 и b_2, погибать под влиянием интенсивностей смерти d_1 и d_2 и конкуренции $c_{11}, c_{12}, c_{21}, c_{22}$. c_{ij} показывает, с какой интенсивностью носитель аллели A_i погибает в процессе конкуренции с носителем аллели A_j.

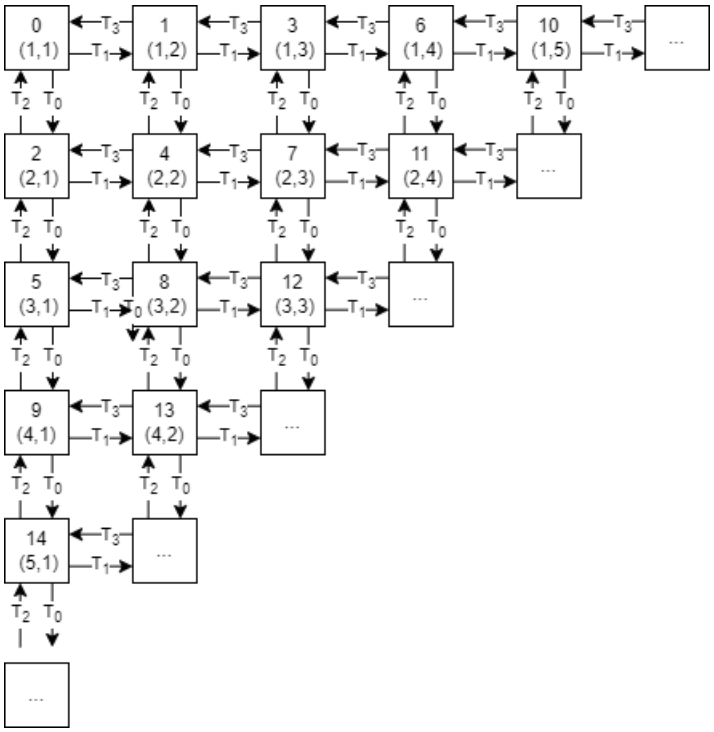

Рисунок 1. Состояния системы и переходы между ними.

Состояние системы может обозначаться:

- как двумерная строка (n_1, n_2), компоненты которой это количество носителей каждой из аллелей.
- неотрицательным целым числом; нумерация осуществляется вдоль диагоналей (см. Рисунок 1)

Для перехода из двумерного представления в одномерное используется следующая формула:

$$(x, y) \to \frac{y(y-1)}{2} + (x-1)\left(\frac{x}{2} + y\right)$$

Алгоритм перехода из одномерного представления в двумерное:

```
def from1dto2d(k):
 i = 1
 while (int(i*(i+1)*0.5) - 1 < k):       #находим номер диагонали
  i+=1
 return (k - int(i*(i-1)*0.5) + 1, int(i*(i+1)*0.5) - k)
```

Начальным состоянием является (1,1) в двумерной и 0 в одномерной нотации. Соседними (или смежными) состояниями, в которые можно перейти из (n_1, n_2) могут являться следующие четыре возможных: $(n_1, n_2 + 1)$; $(n_1 + 1, n_2)$; $(n_1 - 1, n_2)$, если $n_1 > 0$; $(n_1, n_2 - 1)$, если $n_2 > 0$.

Интенсивности переходов имеют следующий вид:

$$T_0(n_1 + 1, n_2 | n_1, n_2) = b_1 \frac{n_1}{V}$$

$$T_1(n_1, n_2 + 1 | n_1, n_2) = b_2 \frac{n_2}{V}$$

$$T_2(n_1 - 1, n_2 | n_1, n_2) = d_1 \frac{n_1}{V} + c_{11} \frac{n_1}{V} \frac{n_1}{V} + c_{12} \frac{n_2}{V} \frac{n_1}{V}$$

$$T_3(n_1, n_2 - 1 | n_1, n_2) = d_2 \frac{n_2}{V} + c_{22} \frac{n_2}{V} \frac{n_2}{V} + c_{21} \frac{n_1}{V} \frac{n_2}{V}$$

Для подсчета вероятностей перехода используется следующая формула:

$$P_{\vec{n},\overrightarrow{n'}} = \frac{T_i(\vec{n}, \overrightarrow{n'})}{\sum_j T_j(\vec{n})}$$

где $T_i(\vec{n}, \overrightarrow{n'})$ это интенсивность перехода из состояния $\vec{n}$ в $\overrightarrow{n'}$, а $\sum_j T_j(\vec{n}, \overrightarrow{n'})$ – сумма интенсивностей перехода в смежные к $\vec{n}$ состояния.

Для выбора следующего состояния осуществляется выборка случайного числа α из равномерно распределенного отрезка [0,1]. Затем из α пооче-

редно вычитаются упорядоченные вероятности перехода. Состояние, при вычитании вероятности перехода в которое α становится неположительным значением, является следующим.

Для подсчета времени пребывания в состоянии $\vec{n}$ осуществляется выборка случайного числа из показательного распределения с параметром $\lambda = \sum_j T_j(\vec{n}, \overrightarrow{n'})$.

Введем следующее обозначение:

$$T_i(n_1, n_2) = T_i(n_1 \pm 1, n_2 \pm 1 | n_1, n_2)$$

Тогда СЛАУ, решением к-ого является стационарное распределение имеет вид:

$$\begin{cases} T_3(1,2)r_1 + T_2(2,1)r_2 = (T_1(1,1) + T_0(1,1))r_0 \\ T_1(1,1)r_0 + T_3(1,3)r_3 + T_2(2,2)r_4 = (T_0(1,2) + T_1(1,2) + T_3(1,2))r_1 \\ T_0(1,1)r_0 + T_3(2,2)r_4 + T_2(3,1)r_5 = (T_0(2,1) + T_1(2,1) + T_2(2,1))r_2 \\ T_1(1,2)r_1 + T_3(1,4)r_6 + T_2(2,3)r_7 = (T_0(1,3) + T_1(1,3) + T_3(1,3))r_3 \\ T_0(1,2)r_1 + T_1(2,1)r_2 + T_2(3,2)r_8 + T_3(2,3)r_7 = (T_0(2,2) + T_1(2,2) + T_2(2,2) + T_3(2, \\ T_0(2,1)r_2 + T_3(3,2)r_8 + T_2(4,1)r_9 = (T_0(3,1) + T_1(3,1) + T_2(3,1))r_5 \\ \dots \\ \dots \\ \sum r_i = 1 \end{cases}$$

Среднее количество особей $que_i, i = 1,2$, будем считать по формулам:

$$que_1 = \sum_{k=1}^{\infty} k \sum_{l=1}^{\infty} \nu_{(k,l)}$$

$$que_2 = \sum_{l=1}^{\infty} l \sum_{k=1}^{\infty} \nu_{(k,l)}$$

где $\nu_{(k,l)}$ – относительная частота посещения состояния (k, l).

Результаты расчетов

1. Нахождение стационарного распределения методом Монте-Карло

Смоделируем первые 1000 событий в системе и посчитаем относительные частоты посещения состояний. Запустим данное моделирование по $N = 100$ траекториям. Осуществим такую процедуру 10 раз.

Таблица 1 - Вектора частот при $b = [1,1], d = [2,2], c = [[1,2][2,1]]$, 10^3 шагов, $N = 10^2$ траекторий; $\nu_i -$ относительная частота состояния i.

№	ν_0	ν_1	ν_2	ν_3	ν_4	ν_5	ν_6	ν_7	ν_8	ν_9
1	0.18266	0.15875	0.15978	0.07579	0.06724	0.07636	0.0364	0.02773	0.02838	0.0374
2	0.18533	0.16297	0.15797	0.07585	0.06711	0.07455	0.0351	0.0281	0.02728	0.0362
3	0.18324	0.15951	0.1576	0.07495	0.06852	0.07389	0.03734	0.02887	0.02858	0.0358
4	0.18177	0.15827	0.15718	0.07502	0.06641	0.07435	0.03648	0.02845	0.02742	0.03626
5	0.18337	0.15961	0.15953	0.07654	0.0663	0.07502	0.03802	0.0273	0.02663	0.03591
6	0.18349	0.1594	0.1609	0.07465	0.06869	0.07617	0.03582	0.02766	0.02857	0.03634
7	0.18293	0.15677	0.1593	0.0759	0.06673	0.07411	0.03724	0.02983	0.02755	0.03607
8	0.18262	0.15874	0.15911	0.0738	0.06703	0.07526	0.0361	0.02689	0.02766	0.03556
9	0.18281	0.15888	0.16043	0.07506	0.06892	0.07543	0.03595	0.02868	0.02818	0.03627
10	0.18353	0.15871	0.15858	0.07509	0.06688	0.07482	0.03656	0.02747	0.0282	0.03703

Таблица 2 - Вектора частот при $b = [2,2], d = [1,1], c = [[1,2][2,1]]$, 10^3 шагов, $N = 10^2$ генераций

№	ν_0	ν_1	ν_2	ν_3	ν_4	ν_5	ν_6	ν_7	ν_8	ν_9
1	0.00194	0.00164	0.00137	0.00118	0.00101	0.00093	0.00083	0.00063	0.00073	0.00081
2	0.00187	0.00153	0.00131	0.00098	0.00111	0.00084	0.00065	0.00072	0.00073	0.0007
3	0.00212	0.0017	0.00171	0.00105	0.00119	0.0013	0.00089	0.00074	0.00092	0.00095
4	0.00203	0.00162	0.00156	0.00102	0.00098	0.00118	0.0008	0.00068	0.00087	0.00081
5	0.00213	0.00151	0.00168	0.0011	0.00092	0.00118	0.00088	0.00063	0.00086	0.00092
6	0.00199	0.00145	0.00156	0.00107	0.00089	0.00118	0.0009	0.00066	0.00071	0.001
7	0.00205	0.00125	0.0016	0.00086	0.00084	0.00109	0.00088	0.00063	0.00061	0.00083
8	0.002	0.00161	0.00158	0.00117	0.001	0.0011	0.00097	0.00071	0.00056	0.00073
9	0.00191	0.00167	0.00161	0.00135	0.00124	0.00107	0.00141	0.00073	0.00082	0.00073
10	0.00213	0.00169	0.0018	0.00097	0.00135	0.00135	0.00072	0.00077	0.00094	0.00112

Таблица 3 - Вектора частот при $b = d = [1,1], c = [[1,2][2,1]]$, 10^3 шагов, $N = 10^2$ генераций

№	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
1	0.01199	0.01409	0.01291	0.01089	0.00986	0.01039	0.00897	0.00653	0.00719	0.00895
2	0.01368	0.01456	0.0149	0.01167	0.01043	0.01224	0.01017	0.0073	0.0076	0.01071
3	0.01022	0.01145	0.01213	0.01015	0.00931	0.0098	0.00947	0.00723	0.00672	0.00865
4	0.01278	0.01386	0.01417	0.01185	0.00975	0.01246	0.01019	0.00825	0.00769	0.01074
5	0.01125	0.01209	0.01187	0.01045	0.00822	0.00912	0.00987	0.00652	0.00562	0.00776
6	0.01243	0.01387	0.0129	0.01184	0.00971	0.01007	0.01023	0.00719	0.00702	0.00951
7	0.01136	0.01216	0.0127	0.00956	0.00875	0.01054	0.00882	0.00665	0.00705	0.0089
8	0.01172	0.0126	0.01266	0.01089	0.00933	0.01059	0.00969	0.00778	0.00733	0.01019
9	0.01248	0.01448	0.01392	0.01204	0.01025	0.01179	0.01034	0.00719	0.00812	0.0109
10	0.01274	0.01353	0.01482	0.01135	0.01039	0.01199	0.00963	0.00776	0.00798	0.01068

2. Последовательный вывод характеристик при генерации.

Моделируем поведение популяции за первые 10^4 шагов. С периодом $T = 1000$ шагов высчитываем que_1 и que_2. Затем высчитываем приращения характеристик $\Delta que_{ij} = que_{i,j} - que_{i,j-1}$.

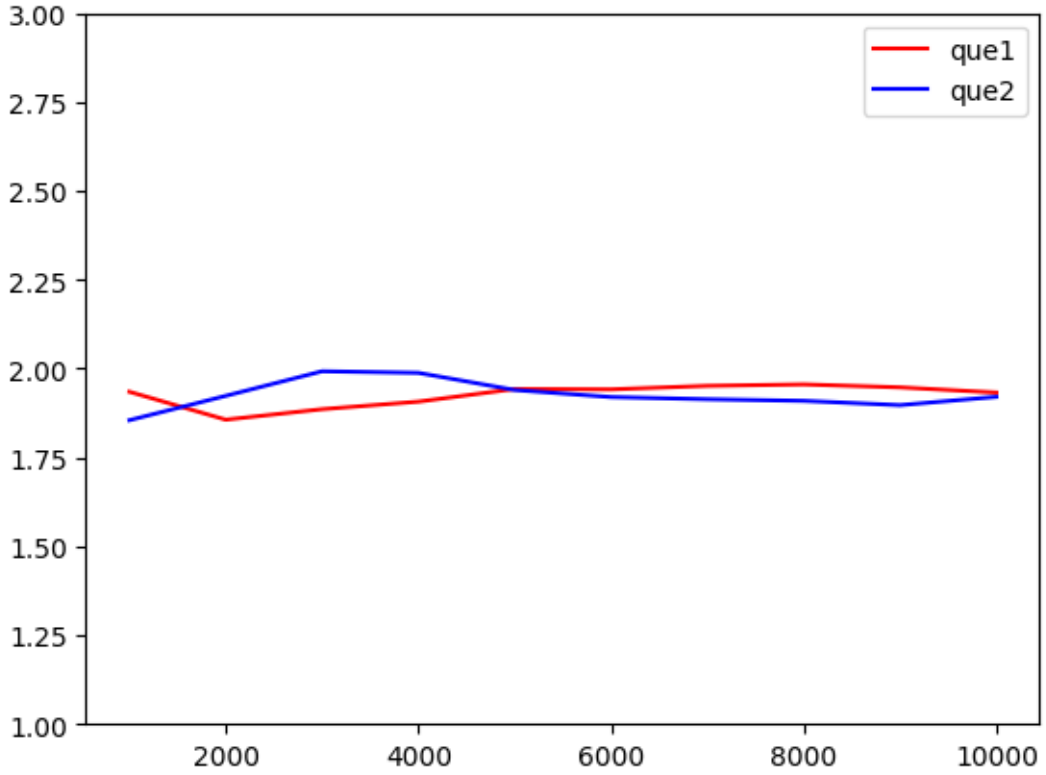

Рисунок 2. Зависимость среднего количества особей от количества шагов при $b = [1,1], d = [2,2], c = [[1,2][2,1]]$, $N = 10^4$, $T = 10^3$.

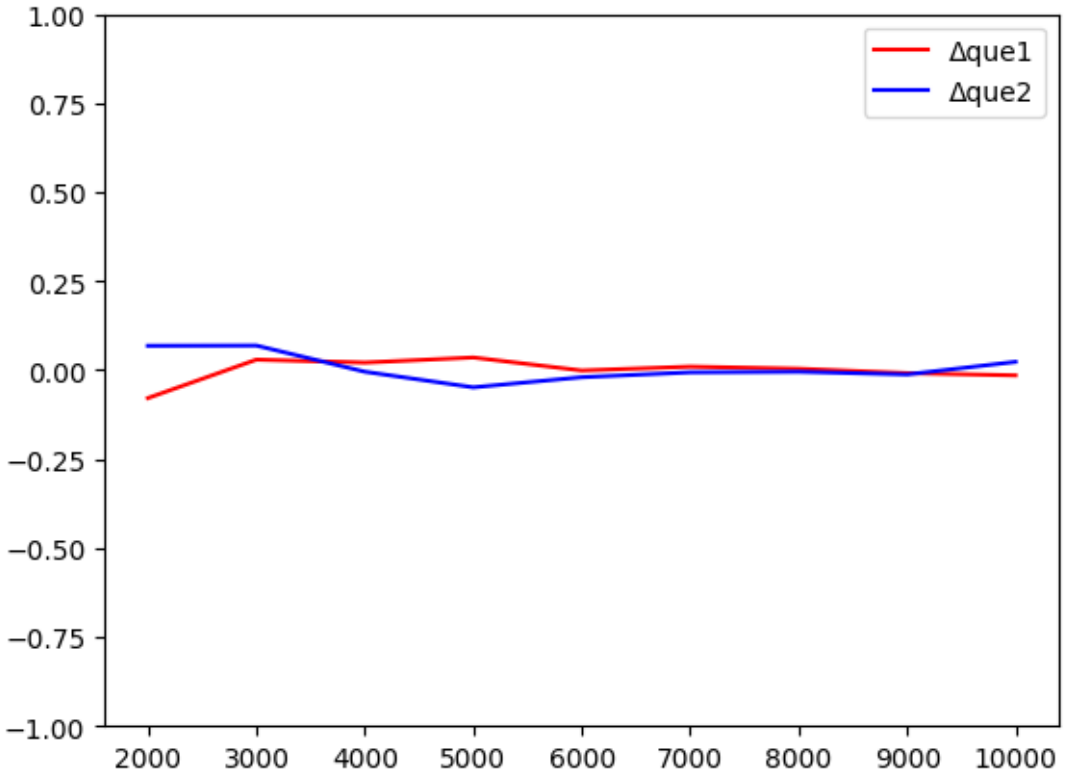

Рисунок 3. Зависимость приращения среднего количества особей от количества шагов при $b = [1,1], d = [2,2], c = [[1,2][2,1]]$, $N = 10^4$, $T = 10^3$.

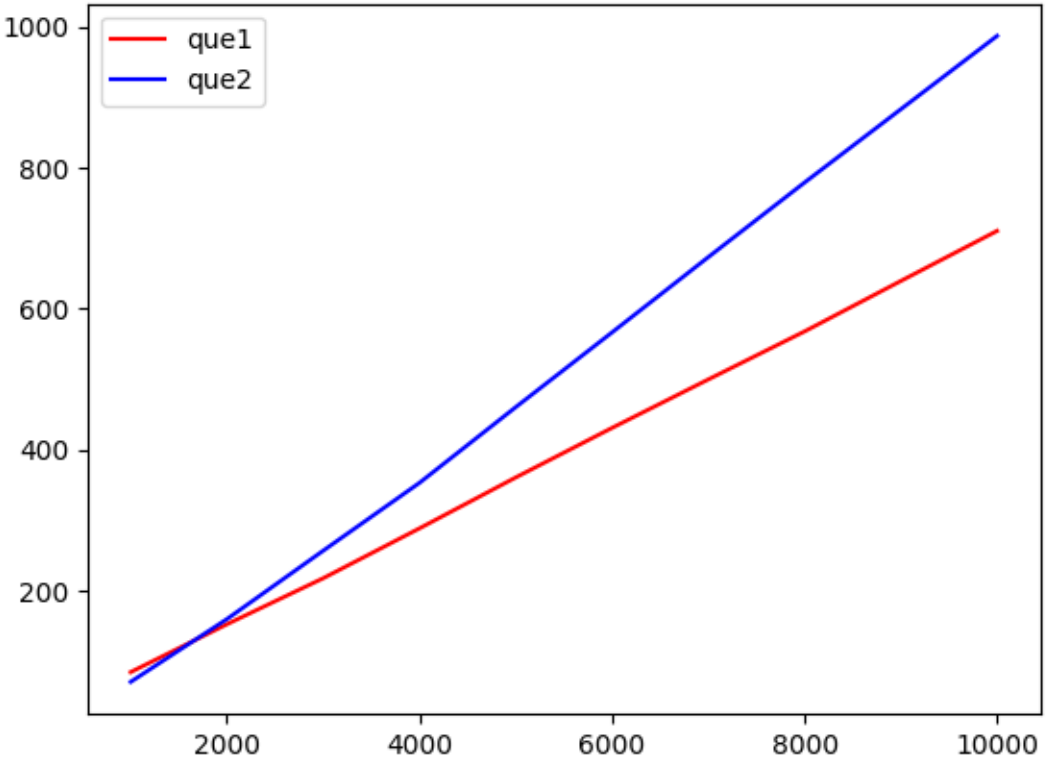

Рисунок 4. Зависимость среднего количества особей от количества шагов при $b = [2,2], d = [1,1], c = [[1,2][2,1]], N = 10^4, T = 10^3$.

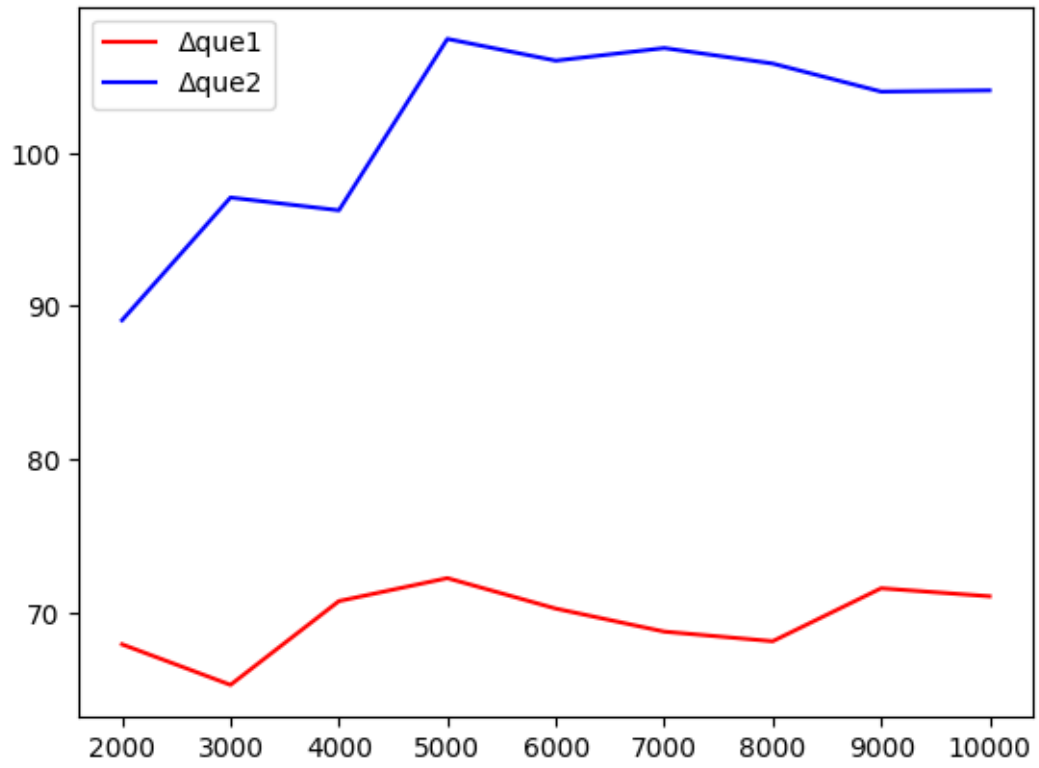

Рисунок 5. Зависимость приращения среднего количества особей от количества шагов при $b = [2,2], d = [1,1], c = [[1,2][2,1]], N = 10^4, T = 10^3$.

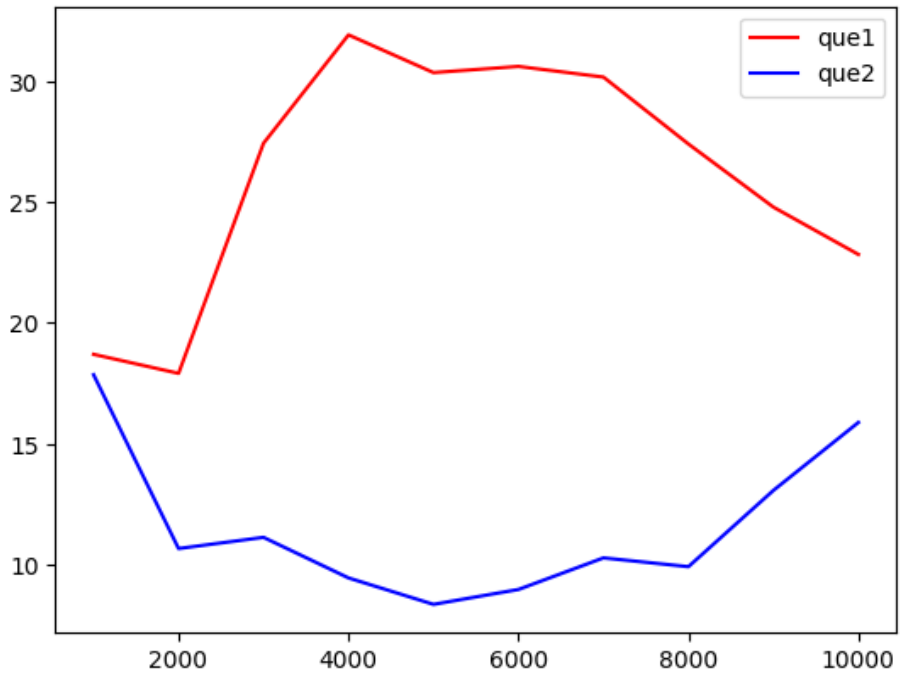

Рисунок 6. Зависимость среднего количества особей от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 10^4$, $T = 10^3$.

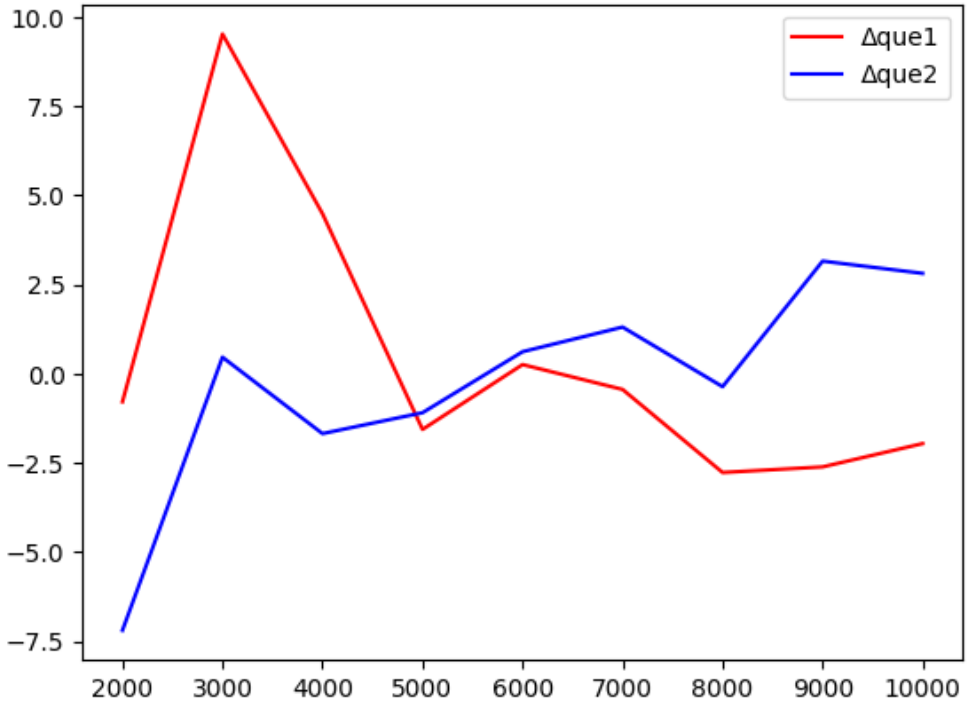

Рисунок 7. Зависимость приращения среднего количества особей от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 10^4$, $T = 10^3$.

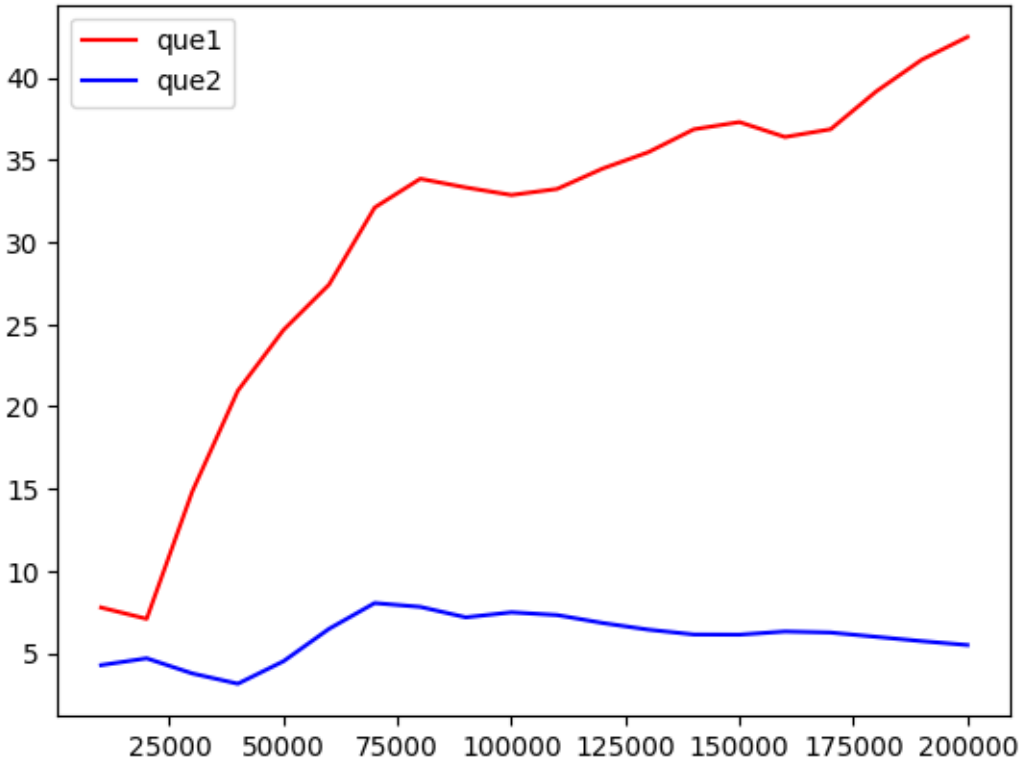

Рисунок 8. Зависимость среднего количества особей от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 2 \cdot 10^5$, $T = 10^4$.

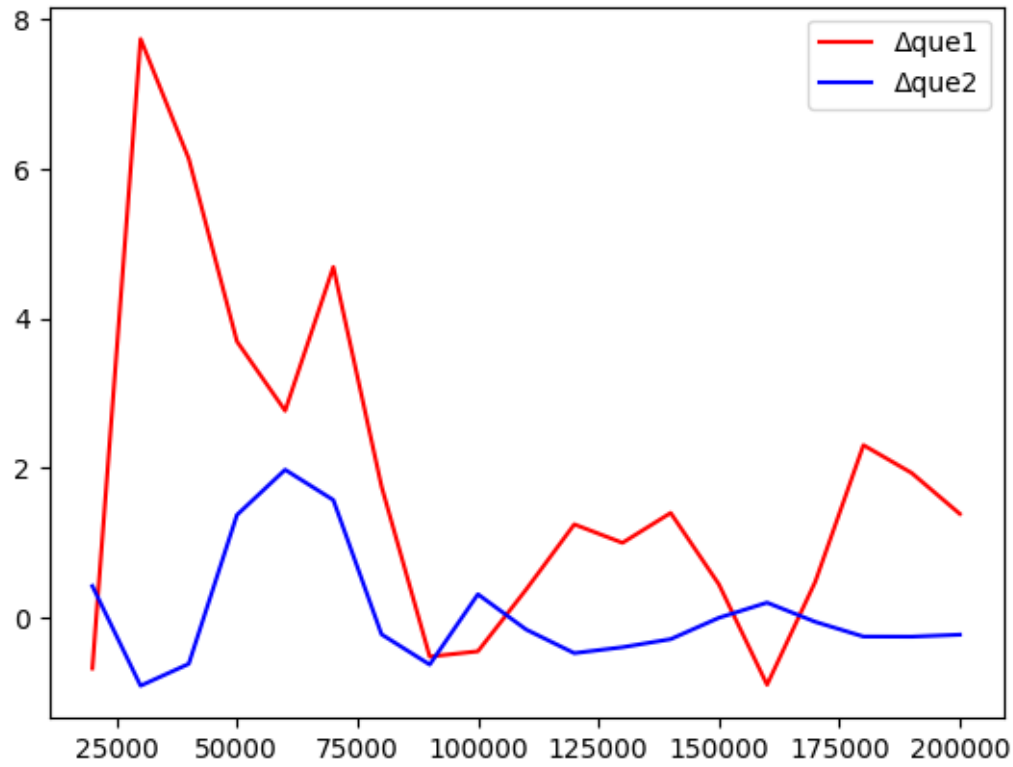

Рисунок 9. Зависимость приращения среднего количества особей от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 2 \cdot 10^5$, $T = 10^4$.

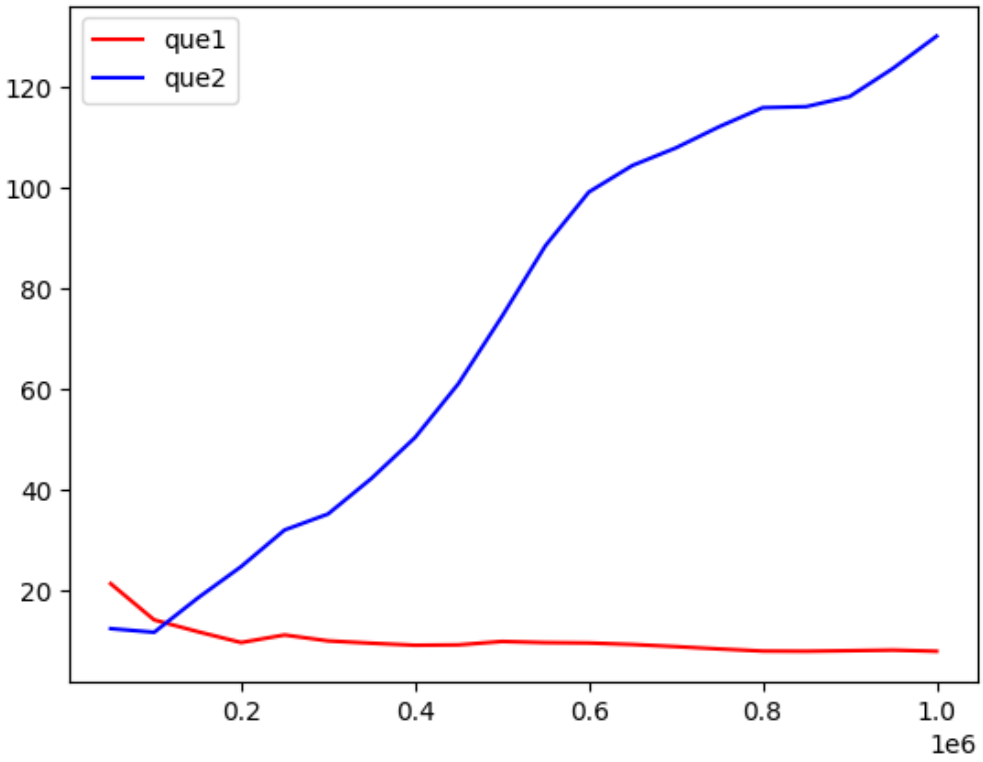

Рисунок 10. Зависимость среднего количества особей от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 10^6$, $T = 5 \cdot 10^4$.

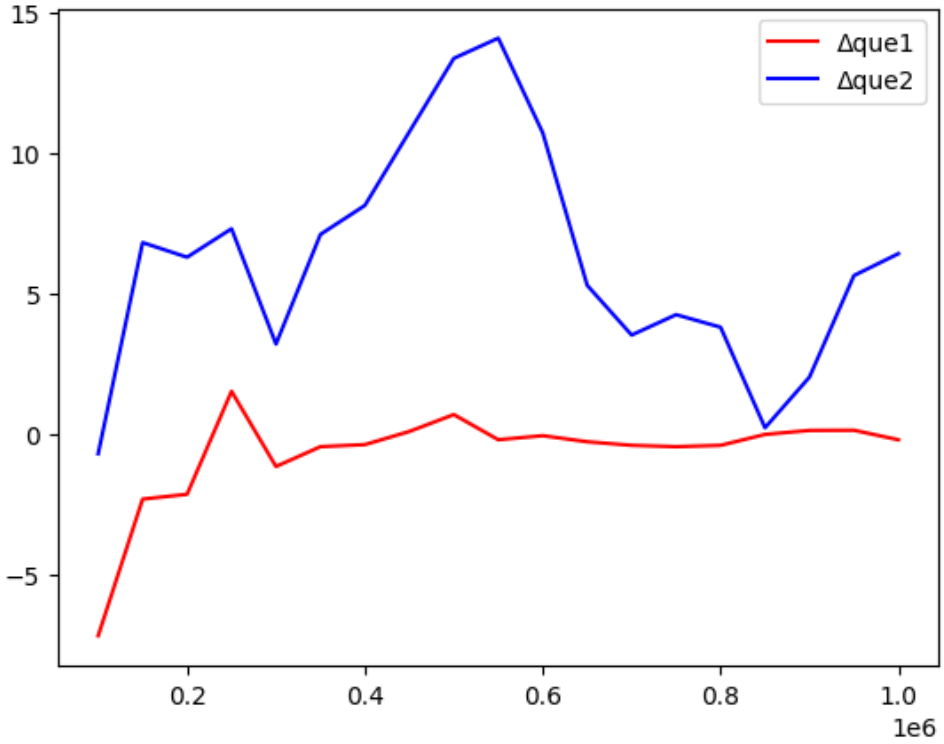

Рисунок 11. Зависимость приращения среднего количества особей от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 10^6$, $T = 5 \cdot 10^4$.

3. Получение значения характеристик в стационарном режиме с заданной точностью.

Генерируем популяцию и каждый раз при осуществлении заданного фиксированного числа шагов T высчитываем que_i. Если разность между какими-то последними двумя que_i меньше ε, то останавливаем цикл и выводим получившиеся значения характеристик.

При генерации n шагов есть (как бы мала она не была) вероятность того, что всеми событиями будут рождения одних и тех же особей, поэтому для двумерного массива относительных частот разумно задать размер $n \times n$. Однако, в виртуальной среде, которой осуществляется генерация, существует ограничение на создание больших массивов; супремум находится где-то около 30000^2 элементов.

Эмпирическим путем выяснено, что для случая $b = [1,1], d = [2,2], c = [[1,2][2,1]]$ количество особей для каждой аллели составляет не более пары десятков. Поэтому принято решение ввести следующую эвристику: массив частот будет иметь размерность 50×50.

Таблица 4 - Значения que_i при $b = [1,1], d = [2,2], c = [[1,2][2,1]]$, интервал шагов $T = 10^5$, $\varepsilon = 10^{-4}$

№ генерации	que_1	que_2
1	1.96918	1.97478
2	1.9757	1.97034
3	1.97468	1.9686
4	1.96704	1.96783
5	1.97503	1.96865
6	1.9709	1.96824
7	1.97363	1.96601
8	1.96627	1.98074
9	1.97118	1.97357
10	1.97743	1.96115

4. Последовательный вывод характеристик при g траекториях

Моделируем поведение популяции за первые N шагов. С периодом T шагов высчитываем que_1 и que_2. Проделываем это g раз

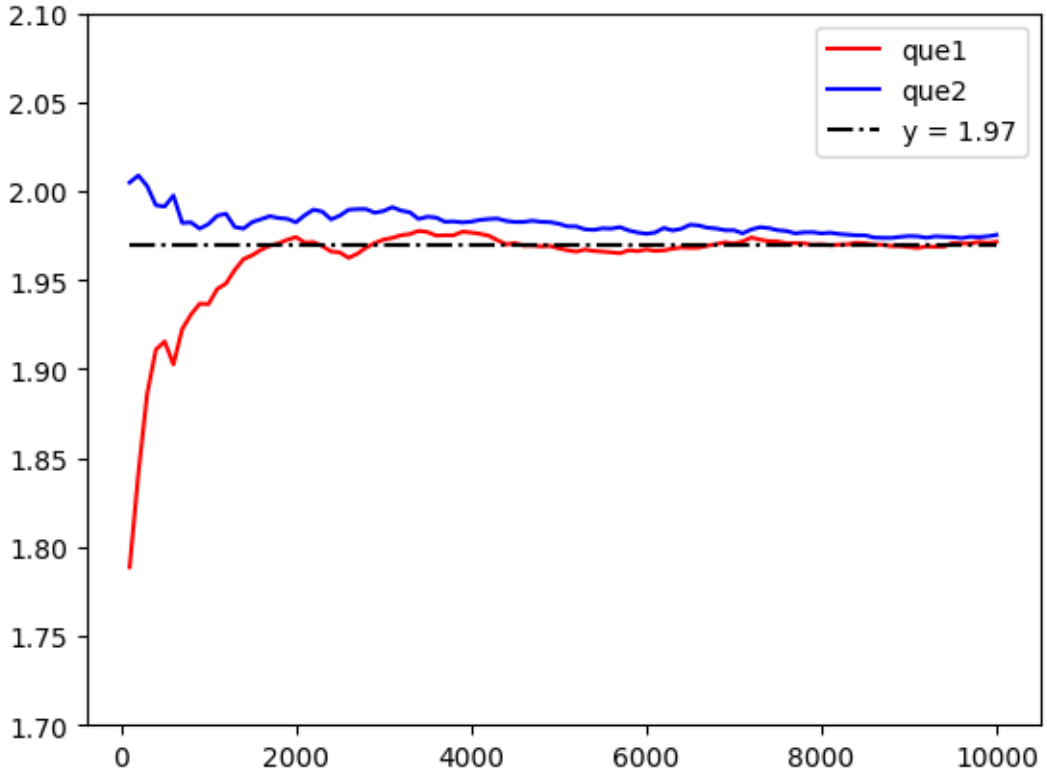

Рисунок 12. Зависимость среднего количества особей от количества шагов при $b = [1,1], d = [2,2], c = [[1,2][2,1]]$, $N = 10^4$, интервал шагов $T = 100$, $g = 100$ и прямая $y = 1.97$.

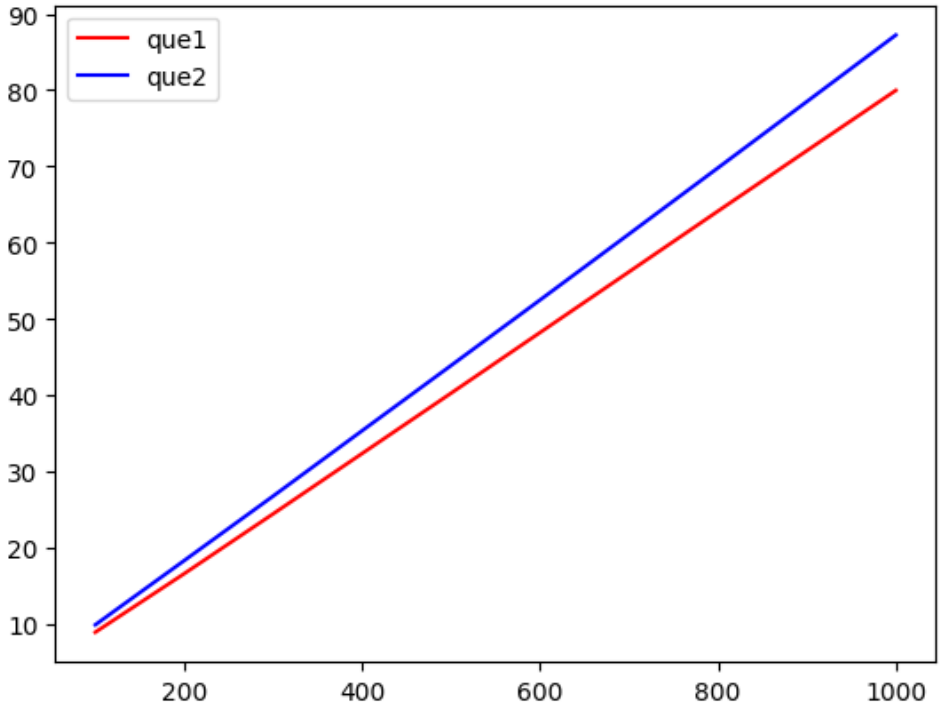

Рисунок 13. Зависимость среднего количества особей от количества шагов при $b = [2,2], d = [1,1], c = [[1,2][2,1]]$, $N = 1000$, интервал шагов $T = 100$, $g = 1000$.

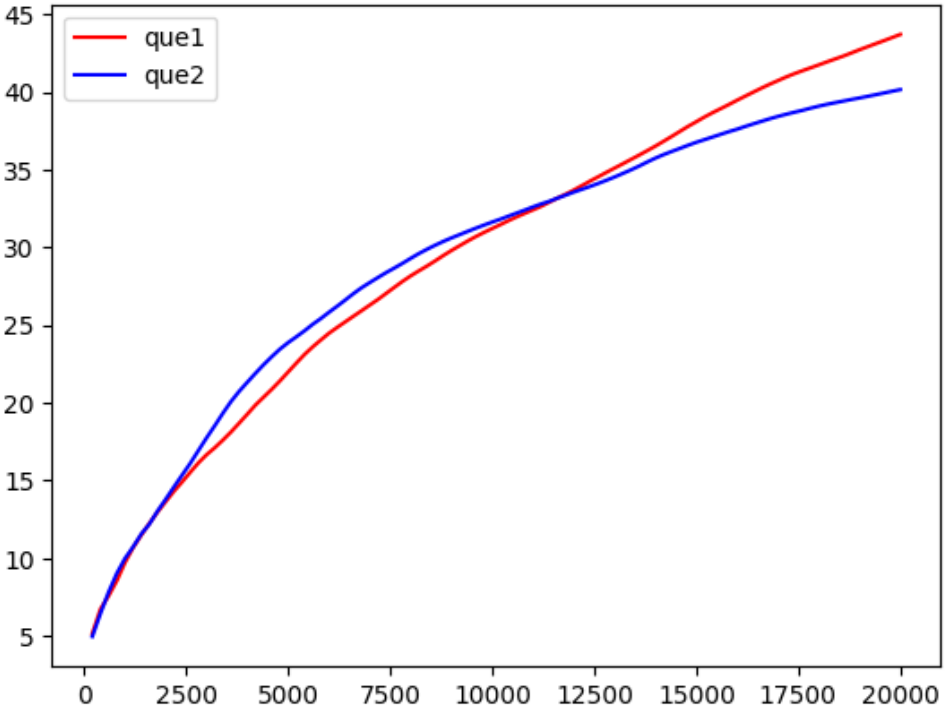

Рисунок 14. Зависимость среднего количества особей от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 20000$, интервал шагов $T = 200$, $g = 100$.

Для параметров $b = [2,2], d = [1,1], c = [[1,2][2,1]]$ зависимость напоминает линейную. Проверим эту гипотезу, увеличив количество траекторий и применив метод наименьших квадратов

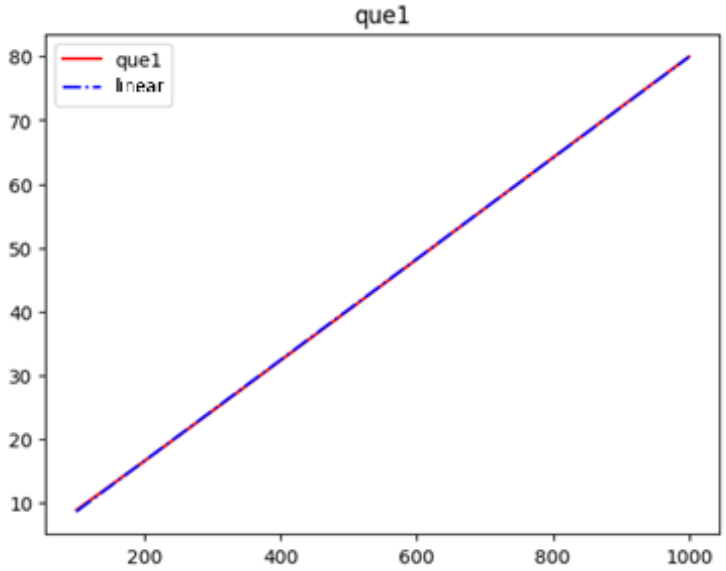

Рисунок 15. Зависимость среднего количества особей первого вида от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 1000$, интервал шагов $T = 100$, $g = 1000$ и нахождение методом наименьших квадратов наилучшей по аппроксимации кривой $y = ax + b$.

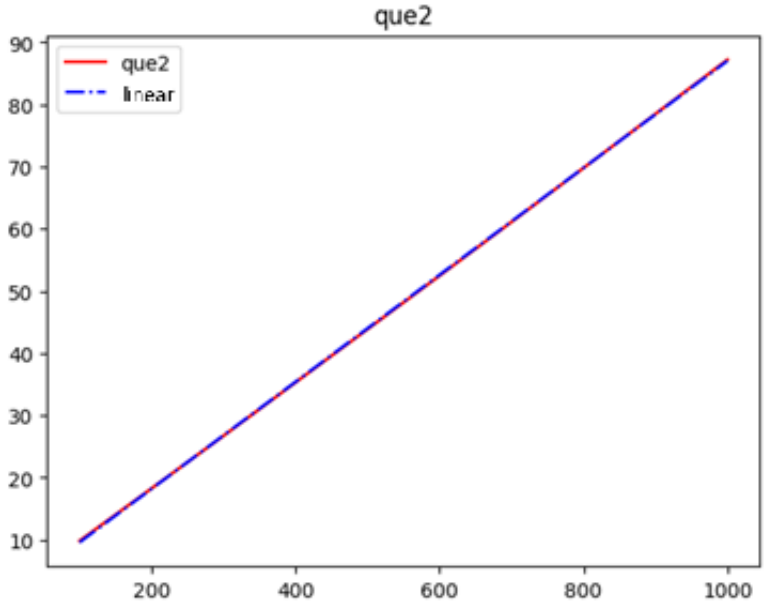

Рисунок 16. Зависимость среднего количества особей второго вида от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 1000$, интервал шагов $T = 100$, $g = 1000$ и нахождение методом наименьших квадратов наилучшей по аппроксимации кривой $y = ax + b$.

Для параметров $b = d = [1,1], c = [[1,2][2,1]]$ зависимость весьма похожа на $y = a\sqrt{x} + b$. Проверим эту гипотезу, уменьшив количество шагов N для ускоренного получения результата, увеличив количество траекторий и применив метод наименьших квадратов

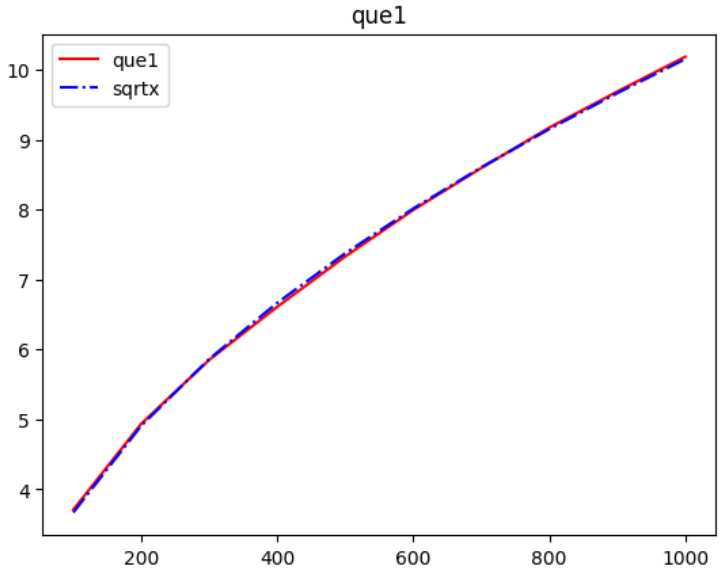

Рисунок 17. Зависимость среднего количества особей первого вида от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 1000$, интервал шагов $T = 100$, $g = 1000$ и нахождение методом наименьших квадратов наилучшей по аппроксимации кривой $y = a\sqrt{x} + b$.

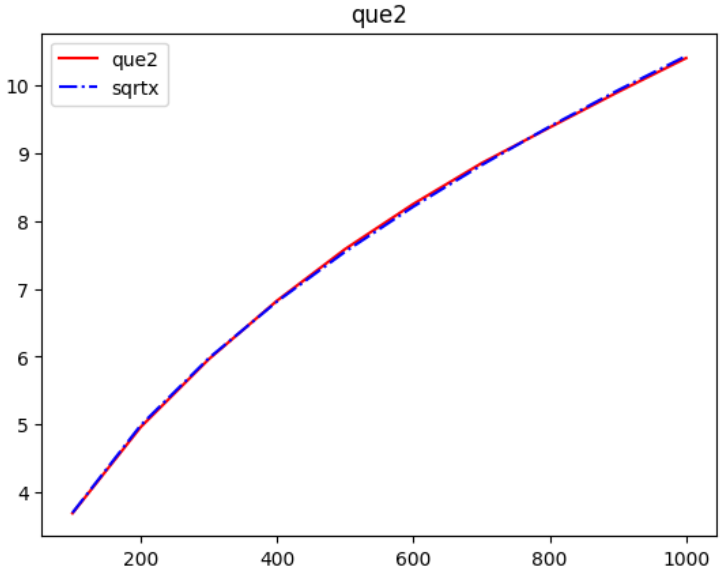

Рисунок 18. Зависимость среднего количества особей второго вида от количества шагов при $b = d = [1,1], c = [[1,2][2,1]]$, $N = 1000$, интервал шагов $T = 100$, $g = 1000$ и нахождение методом наименьших квадратов наилучшей по аппроксимации кривой $y = a\sqrt{x} + b$.

Анализ результатов и выводы

Система при параметрах $b = [1,1], d = [2,2], c = [[1,2][2,1]]$

Из таблицы 1 видно, что около 85% событий сосредоточены в первых 10 состояниях. Это значит, что большинство времени система имеет общее количество особей $n_1 + n_2 \leq 5$ (см. Рис 1). Из рисунков 2, 3, 12 ясно, что среднее количество особей фиксируется на величине близкой к 2.

Посчитаем среднее арифметическое средних количеств особей из таблицы 4:

$$\overline{que_1} = 1.9721, \overline{que_2} = 1.9699.$$

Итого, судя по всему, значение que_i в стационарном режиме при данных параметрах приблизительно равняется 1.97.

Система при параметрах $b = [2,2], d = [1,1], c = [[1,2][2,1]]$

Из таблицы 2 видно, что количество особей весьма редко меньше 5, исходя из того, что сумма относительных частот в первых 10 состояниях обычно составляет около 0.01. Судя по рисунку 4, система остается там только некоторое время после начала наблюдения, затем она навечно покидает эти состояния.

Рисунки 4, 13, 15, 16 также показывают, что среднее количество особей монотонно увеличивается с течением времени, причем зависимость имеет заметный линейный характер. Из рисунка 5 видно, что приращения не увеличиваются, скорее даже колеблются около некого значения. Примечателен тот факт, что количество носителей одной аллели явно преобладает перед количеством носителей другой, и это поведение сохраняется на протяжении всего наблюдаемого времени.

Система при параметрах $b = d = [1,1], c = [[1,2][2,1]]$

Если брать данные таблицы 3, то получается, что около 10% событий происходят в состояниях с $n_1 + n_2 \leq 5$. Как и следовало ожидать, это гораздо больше, чем в случае, когда $b = 2d$, и значительно меньше, чем при $d = 2b$.

Примечательно, что в рисунке 6 наблюдается некоторая антисимметричность поведения системы: как только количество особей одного вида растет, количество других падает, и наоборот. Рисунок 7 показывает, что хоть и в середине наблюдения количества особей менялись слабо, под конец они вдруг снова начали меняться.

Вообще, рисунки 6, 8, 10 демонстрируют, что при таких параметрах каждый раз количество носителей одной аллели преобладает перед количеством носителей другой. Возможно, наблюдается некий эффект “снежного кома” – из-за того, что рождаются представители одного вида, интенсивность их рождения повышается, а поскольку параметры $b_1 = b_2$, то интенсивность размножения этого вида становится больше интенсивности другого, следовательно, чаще рождаются особи этого вида, и т.д.

Рисунки 7, 9, 11 показывают, что рост популяции имеет колебательный характер, причем в некоторых местах (особенно это видно на рисунке 9) следующего рода: как только приращения количества особей одного вида уменьшаются, начинают увеличиваться приращения количества особей другого вида, и наоборот; это та самая динамика, которая задается дифференциальными уравнениями Лотка-Вольтерра.

Рисунки 14, 17, 18 демонстрируют, что при таких параметрах зависимость количества особей от количества шагов характерен функции $a\sqrt{x} + b$.

Список литературы

1. Blythe, R. A.; McKane, A. J. Stochastic models of evolution in genetics, ecology and linguistics. J. Stat. Mech. No 07, P07018, 2007.
2. Constable GW, McKane AJ. Models of genetic drift as limiting forms of the Lotka-Volterra competition model. Phys Rev Lett. 2015 Jan 23;114(3):038101. doi: 10.1103/PhysRevLett.114.038101. Epub 2015 Jan 22.PMID: 25659024.
3. Карлин С. Основы теории случайных процессов. М.: "Мир", 1971, — 537 с.
4. Hening, A., Nguyen, D.H. Stochastic Lotka–Volterra food chains. J. Math. Biol. 77, 135–163 (2018).
5. Mimmo Iannelli, Andrea Pugliese An Introduction to Mathematical Population Dynamics: Along the trail of Volterra and Lotka Unitext 79 - La Matematica per il 3+2 Springer International Publishing 2014.
6. Dayar T., Mikeev L., Verena W. On the numerical analysis of stochastic Lotka-Volterra models./ Proceedings of the International Multiconference on Computer Science and Information Technology, 2010, Vol. 5.
7. Young, G., Belmonte, A. Fixation in the stochastic Lotka-Volterra model with small fitness trade-offs.- Journal of Mathematical Biology, 2022, 85.
8. Ермаков С.М., Михайлов Г.А. Статистическое моделирование М.: ФИЗМАТЛИТ, 1982. - 296 с. 2-е изд., дополн.

Приложения

Код программы расчета

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import plotly.express as px
import scipy.optimize


"Начальные значения"


birth : float = np.array([2,1])
death : float = np.array([1,2])
comp : float = np.array([[1,2],[2,1]])


"Переход из двумерного в одномерное представления состояния"


def from2dto1d(v):
  return int(v[1]*(v[1]-1)*0.5 + (v[0]-1)*(v[0]*0.5+v[1]))

def from1dto2d(k):
  i = 1
  while (int(i*(i+1)*0.5) - 1 < k): #находим номер диагонали
    i+=1
  return (k - int(i*(i-1)*0.5) + 1, int(i*(i+1)*0.5) - k)


"Расчет среднего количества особей"


def que(v, n):
  sum_1 = np.zeros(n)
  sum_2 = np.zeros(n)
  for i in range(n):
    sum_1[from1dto2d(i)[0]] += v[i]
    sum_2[from1dto2d(i)[1]] += v[i]
  que1 = 0
  que2 = 0
  for i in range(n):
    que1 += sum_1[i]*i
    que2 += sum_2[i]*i
  return que1, que2

def que2d(v, n):
  sum_1 = np.zeros(n)
  sum_2 = np.zeros(n)
  for i in range(n):
    for j in range(n):
```

```
      sum_1[i] += v[i][j]
      sum_2[j] += v[i][j]
  que1 = 0
  que2 = 0
  for i in range(n):
    que1 += sum_1[i]*i
    que2 += sum_2[i]*i
  return que1, que2

def que_dif(que1, que2):
  n = len(que1) - 1
  que1_diff = np.zeros(n)
  que2_diff = np.zeros(n)
  for i in range(n):
    que1_diff[i] = que1[i+1]-que1[i]
    que2_diff[i] = que2[i+1]-que2[i]
  return que1_diff, que2_diff

"Вычисление максимальной абсолютной разницы между частотами текущей некого
предыдущего состояния"

def v_error(v, v_prev, eps):
  sz = len(v)
  if_close = np.zeros((sz, sz))
  sum_part_v = 0
  for i in range(sz):
    for j in range(sz):
      if v[i][j] == 0:
        if_close[i][j] = -1
      elif abs(v[i][j] - v_prev[i][j]) < eps:
        if_close[i][j] = 1
        sum_part_v += v[i][j]
      else:
        if_close[i][j] = 0
  return sum_part_v, if_close

"Вычисление характеристик относительно 'хороших' состояний" #NOT FINISHED;
WHAT IS 'хорошее' состояние?#

def que_part(v, if_state):
  sum_1 = np.zeros(len(v))
  sum_2 = np.zeros(len(v))
  for i in range(len(v)):
    for j in range(len(v)):
      if (if_state[i][j] == 1):
        sum_1[i] += v[i][j]
        sum_2[j] += v[i][j]
  que1 = 0
  que2 = 0
  for i in range(len(v)):
    que1 += sum_1[i]*i
    que2 += sum_2[i]*i
  return que1, que2
```

```
"Переход в следующее состояние системы"


"""
dictionary for states
0 - down(b0) 1 - right(b1) 2 - up(d0 c00 c01) 3 - left(d1 c10 c11)
"""
direction_to_next_state = {
    0 : [1,0],
    1 : [0,1],
    2 : [-1,0],
    3 : [0,-1]
}

def next_chosen_state(state,b,c,d, V_inv):
  """
  evaluating the probabilities
  """
  lambda_sum : float = 0
  next_state_prob : float = np.zeros(4)
  next_state_prob[0] = b[0]*state[0] * V_inv
  next_state_prob[1] = b[1]*state[1] * V_inv
  lambda_sum += next_state_prob[0] + next_state_prob[1]
  if (state[0] > 1):
    next_state_prob[2] = d[0]*state[0] * V_inv + (c[0][0]*pow(state[0],2) +
c[0][1]*state[0]*state[1]) * V_inv * V_inv
    lambda_sum += next_state_prob[2]
  if (state[1] > 1):
    next_state_prob[3] = d[1]*state[1] * V_inv + (c[1][1]*pow(state[1],2) +
c[1][0]*state[1]*state[0]) * V_inv * V_inv
    lambda_sum += next_state_prob[3]
  n_s_p = [x / lambda_sum for x in next_state_prob]
  """
  choosing the direction (left up right down); next state
  """
  x = np.random.uniform(0,1)
  i : int = -1
  while (x > 0):
    i += 1
    x -= n_s_p[i]

  state[0] += direction_to_next_state[i][0]
  state[1] += direction_to_next_state[i][1]

  """
  evaluating time of transition into next state
  """
  t = np.random.exponential(1/sum(next_state_prob))

  return t, state

def next_state(state, TDeg, SDeg, N1, N2, b, c, d, V_inv):
  t, state = next_chosen_state(state, b, c, d, V_inv)
  TDeg += t
  SDeg += 1
```

```
  N1 += state[0]
  N2 += state[1]
  return state, TDeg, SDeg, N1, N2

"Вариация параметров"

def generation_param(code, fst_spc, char_type, i, N = 100):
  match code:
    case 'b1':
      return generation([birth[0]+0.05*i, birth[1]], comp, death, fst_spc,
N)[char_type]
    case 'b2':
      return generation([birth[0], birth[1]+0.05*i], comp, death, fst_spc,
N)[char_type]
    case 'c11':
      return generation(birth, [[comp[0][0]+0.05*i,comp[0][1]], [comp[1][0],
comp[1][1]]], death, fst_spc, N)[char_type]
    case 'c12':
      return generation(birth, [[comp[0][0],comp[0][1]+0.05*i], [comp[1][0],
comp[1][1]]], death, fst_spc, N)[char_type]
    case 'c21':
      return generation(birth, [[comp[0][0],comp[0][1]], [comp[1][0]+0.05*i,
comp[1][1]]], death, fst_spc, N)[char_type]
    case 'c22':
      return generation(birth, [[comp[0][0],comp[0][1]], [comp[1][0],
comp[1][1]+0.05*i]], death, fst_spc, N)[char_type]
    case 'd1':
      return generation(birth, comp, [death[0]+0.05*i, death[1]], fst_spc,
N)[char_type]
    case 'd2':
      return generation(birth, comp, [death[0], death[1]+0.05*i], fst_spc,
N)[char_type]
    case _:
      raise Exception("Wrong parameter code")
```

Генерации

```
"Классический вариант симуляции"

def generation(b,c,d,fst_spc = 1, scd_spc = 1, N = 100, V = 5):
  tdeg : float = 0
  sdeg : float = 0
  mn1 : float = 0
  mn2 : float = 0
  N_inv : float = 1 / N
  V_inv : float = 1 / V

  for i in range(N):
    i_state : float = np.array([fst_spc, scd_spc])
    i_time : float = 0
    i_s : float = 0
    i_n1 : float = fst_spc
    i_n2 : float = scd_spc
    while(i_state[0] + i_state[1] != 0):
      i_state, i_time, i_s, i_n1, i_n2 = next_state(i_state, i_time, i_s,
```

```
i_n1, i_n2, b, c, d, V_inv)
      "print(i_s, i_state, i_time)"
    i_n1 /= i_s
    i_n2 /= i_s
    tdeg += i_time
    sdeg += i_s
    mn1 += i_n1
    mn2 += i_n2
  tdeg *= N_inv
  sdeg *= N_inv
  mn1 *= N_inv
  mn2 *= N_inv

  return tdeg, sdeg, mn1, mn2

def single_generation(b,c,d,fst_spc = 1, scd_spc = 1, V = 5):
  return generation(b, c, d, fst_spc, scd_spc, 1, V)

"Генерации с вариацией параметра"

def generation_variation(param_code, fst_spc, char_type, N):
  x = np.linspace(start=2, stop=5, num=61, endpoint=True)
  y = np.zeros(61)
  match char_type:
    case 'tdeg':
      char : int = 0
    case 'sdeg':
      char : int = 1
    case 'mn1':
      char : int = 2
    case 'mn2':
      char : int = 3
    case _:
      raise Exception("Wrong parameter code")
  for i in range(61):
    y[i] : float = generation_param(param_code, fst_spc, char, i, N)
  return x, y

"Генерация с ограниченным количеством шагов"

def finite_generation(b,c,d,number_of_steps = 1000, fst_spc = 1, scd_spc = 1,
N = 100, bound = 5500):
  tdeg : float = 0
  sdeg : float = 0
  mn1 : float = 0
  mn2 : float = 0
  n1 : int = 0
  n2 : int = 0
  maxn1 : int = 1
  maxn2 : int = 1
  N_inv : float = 1 / N
  sz = min(bound, int(number_of_steps * (number_of_steps + 1) / 2))
  Vf = np.zeros(sz)

  for i in range(N):
```

```
    i_state : int = np.array([fst_spc, scd_spc])
    i_time : float = 0
    i_s : float = 0
    i_n1 : float = fst_spc
    i_n2 : float = scd_spc
    j = 0
    while(i_state[0] + i_state[1] != 0 and j < number_of_steps):
      j += 1
      Vf[from2dto1d(i_state)] += 1
      i_state, i_time, i_s, i_n1, i_n2 = next_state(i_state, i_time, i_s,
i_n1, i_n2, b, c, d, 1.0 / (i_n1 + i_n2))
      if i_state[0] > maxn1:
        maxn1 = i_state[0]
      if i_state[1] > maxn2:
        maxn2 = i_state[1]
      "print(i_s, i_state, i_time)"
    i_n1 /= i_s
    i_n2 /= i_s
    tdeg += i_time
    sdeg += i_s
    mn1 += i_n1
    mn2 += i_n2
    n1 = i_state[0]
    n2 = i_state[1]
  tdeg *= N_inv
  sdeg *= N_inv
  mn1 *= N_inv
  mn2 *= N_inv
  n1 *= N_inv
  n2 *= N_inv
  Vf *= N_inv / number_of_steps

  return n1, n2, tdeg, sdeg, mn1, mn2, Vf, maxn1, maxn2

"Генерация пока не достигнуто квазистационарное состояние системы" #NOT
FINISHED#

def generation_until_error(b,c,d, interval = 1000, fst_spc = 1, scd_spc = 1,
num_of_intervals = 100, bound = 100, eps = 0.001, sum_stab_v = 0.9):
  i_state : int = np.array([fst_spc, scd_spc])
  i_time : float = 0
  i_s : float = 0
  i_n1 : float = fst_spc
  i_n2 : float = scd_spc
  sz = min(bound, interval*num_of_intervals)
  Vf = np.zeros((sz, sz))
  v = np.zeros((sz, sz))
  v_prev = np.zeros((sz, sz))
  if_close = np.zeros((sz, sz))
  sum_part_v = 0

  k = 0
  while (sum_part_v <= sum_stab_v and k < num_of_intervals):
    v_prev = v
    for i in range(interval*k, interval*(k+1)):
```

```
      Vf[i_state[0]][i_state[1]] += 1
      i_state, i_time, i_s, i_n1, i_n2 = next_state(i_state, i_time, i_s,
i_n1, i_n2, b, c, d, 1.0 / (i_n1 + i_n2))
    v = Vf / (interval*(k+1))
    sum_part_v, if_close = v_error(v, v_prev, eps)
    print(sum_part_v)
    k += 1

  return sum_part_v, k, que_part(v, if_close), v, v_prev, if_close

"Генерация с ограниченным количеством шагов и последовательным выведением
характеристик"

#DEPRECATED#
def finite_generation_interval_ques(b,c,d,number_of_steps = 1000,
char_intervals = 10, fst_spc = 1, scd_spc = 1):
  i_state : int = np.array([fst_spc, scd_spc])
  i_time : float = 0
  i_s : float = 0
  i_n1 : float = fst_spc
  i_n2 : float = scd_spc
  max_state : int = int(number_of_steps * (number_of_steps + 1) / 2)
  interval = int(number_of_steps / char_intervals)
  Vf = np.zeros(max_state)

  que1 = np.zeros(char_intervals)
  que2 = np.zeros(char_intervals)

  for k in range(char_intervals):
    for i in range(interval*k, interval*(k+1)):
      Vf[from2dto1d(i_state)] += 1
      i_state, i_time, i_s, i_n1, i_n2 = next_state(i_state, i_time, i_s,
i_n1, i_n2, b, c, d, 1.0 / (i_n1 + i_n2))
    v = Vf / (interval*(k+1))
    que1[k], que2[k] = que(v, int(interval*(k+1) * (interval*(k+1) + 1) *
0.5))

  return que1, que2

#USE THIS INSTEAD#
def finite_generation_ques(b,c,d,number_of_steps = 1000, char_intervals = 10,
fst_spc = 1, scd_spc = 1, bound = 100):
  i_state : int = np.array([fst_spc, scd_spc])
  i_time : float = 0
  i_s : float = 0
  i_n1 : float = fst_spc
  i_n2 : float = scd_spc
  interval = int(number_of_steps / char_intervals)
  sz = min(bound, number_of_steps*char_intervals)
  Vf = np.zeros((sz, sz))

  que1 = np.zeros(char_intervals)
  que2 = np.zeros(char_intervals)
```

```
  for k in range(char_intervals):
    for i in range(interval*k, interval*(k+1)):
      Vf[i_state[0]][i_state[1]] += 1
      i_state, i_time, i_s, i_n1, i_n2 = next_state(i_state, i_time, i_s,
i_n1, i_n2, b, c, d, 1.0 / (i_n1 + i_n2))
    v = Vf / (interval*(k+1))
    que1[k], que2[k] = que2d(v, sz)
    #print("que1_k {} que1_k-1 {} que2_k {} que2_k-1
{}".format(round(que1[k],5),round(que1[k-1],5),round(que2[k],5),round(que2[k-
1],5)))

  return que1, que2

"Генерация с ограниченным количеством шагов и последовательным выведением
характеристик с N траекториями"

def finite_generation_ques_N_generations(b,c,d,number_of_steps = 1000,
char_intervals = 10, fst_spc = 1, scd_spc = 1, bound = 100, N = 10):
  N_que1, N_que2 = np.zeros(char_intervals), np.zeros(char_intervals)
  for i in range(N):
    print("iteration={}".format(i))
    tmp_q1, tmp_q2 =
finite_generation_ques(b,c,d,number_of_steps,char_intervals,fst_spc,scd_spc,b
ound)
    for j in range(char_intervals):
      N_que1[j] += tmp_q1[j]
      N_que2[j] += tmp_q2[j]
  return N_que1 / N, N_que2 / N

"Генерация пока que_i имеют разность больше eps"

def generation_ques_until_error(b,c,d, interval = 1000, fst_spc = 1, scd_spc
= 1, num_of_intervals = 100, bound = 100, eps=0.0001):
  i_state : int = np.array([fst_spc, scd_spc])
  i_time : float = 0
  i_s : float = 0
  i_n1 : float = fst_spc
  i_n2 : float = scd_spc
  Vf = np.zeros((bound, bound))

  que1 = np.zeros(num_of_intervals)
  que2 = np.zeros(num_of_intervals)

  k = 0
  for i in range(interval):
      Vf[i_state[0]][i_state[1]] += 1
      i_state, i_time, i_s, i_n1, i_n2 = next_state(i_state, i_time, i_s,
i_n1, i_n2, b, c, d, 1.0 / (i_n1 + i_n2))
  v = Vf / interval
  que1[0], que2[0] = que2d(v, bound)
  for i in range(interval):
      Vf[i_state[0]][i_state[1]] += 1
      i_state, i_time, i_s, i_n1, i_n2 = next_state(i_state, i_time, i_s,
i_n1, i_n2, b, c, d, 1.0 / (i_n1 + i_n2))
```

```
  v = Vf / (interval*2)
  que1[1], que2[1] = que2d(v, bound)
  k = 2
  while (abs(que1[k-1] - que1[k-2]) > eps and abs(que2[k-1] - que2[k-2]) >
eps and k < num_of_intervals):
    for i in range(interval*k, interval*(k+1)):
      Vf[i_state[0]][i_state[1]] += 1
      i_state, i_time, i_s, i_n1, i_n2 = next_state(i_state, i_time, i_s,
i_n1, i_n2, b, c, d, 1.0 / (i_n1 + i_n2))
    v = Vf / (interval*(k+1))
    que1[k], que2[k] = que2d(v, bound)
    print("que1_k {} que1_k-1 {} que2_k {} que2_k-1
{}".format(round(que1[k],5),round(que1[k-1],5),round(que2[k],5),round(que2[k-
1],5)))
    k += 1

  return k, que1, que2
```

Построение графиков

```
"Построение зависимости характеристик от вариации параметра" #6 sem

def plot_dependency(param_code, fst_spc, char_type = 'tdeg', plot_color =
'r', N = 100):
  x, y = generation_variation(param_code, fst_spc, char_type, N)
  fig = plt.figure(param_code + ' (' + str(fst_spc) + ',' + str(5-fst_spc) +
'); ' + char_type)
  plt.plot(x, y, linewidth=1, color=plot_color)
  plt.title('Varying of parameter ' + param_code + ' starting from (' +
str(fst_spc) + ',' + str(5-fst_spc) + ') with ' + str(N) + ' trajectories')
  plt.ylabel(char_type)

"Построение характеристик que_i"

def plot_ques(number_of_steps, char_intervals, que1, que2, y_lim=[-1,-1]):
  if y_lim ==[-1,-1]:
    y_lim = [0, max(max(que1),max(que2))]
  h = int(number_of_steps/char_intervals)
  x = np.arange(h, number_of_steps + h, h)
  plt.plot(x, que1, label = "que1", color = 'r')
  plt.plot(x, que2, label = "que2", color = 'b')
  plt.ylim(y_lim)
  plt.legend()

"Построение характеристик que_i и наложение функции методом МНК"

def plot_ques_and_fit(f, number_of_steps, char_intervals, que1, que2,
plot_label):
  h = int(number_of_steps/char_intervals)
  x = np.arange(h, number_of_steps + h, h)

  fig = plt.figure("que1")
  plt.plot(x, que1, label = "que1", color = 'r')
  plt.title("que1")
  a, b = scipy.optimize.curve_fit(f, xdata = x, ydata = que1)[0]
```

```
  plt.plot(x,f(x,a,b), label = plot_label, color = 'g', linestyle='-.')
  plt.legend()

  fig = plt.figure("que2")
  plt.plot(x, que2, label = "que2", color = 'b')
  plt.title("que2")
  a, b = scipy.optimize.curve_fit(f, xdata = x, ydata = que2)[0]
  plt.plot(x,f(x,a,b), label = plot_label, color = 'g', linestyle='-.')
  plt.legend()

"Сохранить que_i как таблицу"

def save_ques_as_table(que1, que2):
  df = pd.DataFrame(data={"que1" : que1, "que2" : que2})
  df.to_excel("ques.xlsx")
```