Санкт–Петербургский государственный университет

НЕЧИПОРУК Александра Алексеевна

Выпускная квалификационная работа

Исследование моделей динамики мнений в социальных сетях

Уровень образования: магистратура
Направление 01.04.02 «Прикладная математика и информатика»
Основная образовательная программа ВМ.5504.2021
«Исследование операций и системный анализ»

Научный руководитель:
профессор, кафедра математической
теории игр и статистических решений,
д.ф. - м.н. доцент Парилина Елена Михайловна

Рецензент:
Директор департамента экономики
и планирования, АО "ГК Эталон
к.ф. - м.н. Марковкин Михаил Викторович

Санкт-Петербург
2023 г.

Содержание

Введение

Социальные сети уже многие годы притягивают к себе внимание. С появлением в Веб-пространстве первых социальных сетей стало легче получать информацию о круге общения, и это привлекло исследователей. Столкновение разных мнений и со временем достигнутый консенсус является важным предметом исследований, ведь знание принципов достижения консенсуса позволяет влиять на результат, например, вложением ресурсов в рекламу.

Обзор литературы

Первая работа о динамике мнений исследовала вопрос о достижимости консенсуса. В работе Де Грота [1] представлена модель динамики, где каждый участник меняет свое мнение, взвешивая мнения каждого связанного с ним агента сети и свое собственное. Позднее эта модель усовершенствовалась. Стоит выделить следующие модернизации:

- Модель Фриедкина-Джонсена [2], где каждому агенту был добавлен параметр восприимчивости к чужому мнению.
- Модель Хегсельманна-Крауза [3], в которой появляется порог доверия, участники учитывают только тех, чьи мнения отличаются от собственного не более чем на этот порог.
- Модель с двумя центрами влияния [4, 5].

Эти модели описывают само взаимодействие в сети, но не исследуют возможность влияния на нее. За последние несколько лет было представлено множество работ с исследованием динамики мнений как задач управления [6, 7, 8, 9, 10], с одним центром, способным воздействовать на некоторых агентов сети, с целью достижения определенного среднего мнения в сети. Так же исследуются теоретико игровые модели [6, 11] и теоретико игровые кооперационные модели [12]. Исследуется и вопрос разбиения узлов, для упрощения исследования больших сетей [13, 14].

Постановка задачи

Основной задачей настоящей работы является создание алгоритма численного решения задачи управления мнениями агентов, представленной в [7] с использованием подхода преобразования сетевой структуры такой модели, который был представлен в статьях [15, 16] и в дипломной работе [17]. Для реализации этой цели был предложен алгоритм, и решена задача экспериментальной проверки этапов работы этого алгоритма для определенных видов графов.

Глава 1. Модель

1.1 Описание модели

За основу взята модель, предложенная в [7]. Исследуется динамика мнений общества с дискретным бесконечным временем. Общество рассматривается как пара (N, g), где N – конечное множество агентов общества, и g – граф, который отражает социальные взаимосвязи. Граф $g = (N, E)$ определяется как N – множество вершин, E – множество ребер. Существует независимый член общества, влияющий на мнение некоторых агентов, которого называют игроком. Динамику мнений определяют уравнения:

$$x_i(t+1) = x_i(t) + a_i \left(\frac{x_i(t) + \sum_{j \in S_i} x_j(t)}{|S_i| + 1} - x_i(t) \right) + u_i(t), i \in N, \qquad (1)$$

где $x_i(t) \in \mathbb{R}^1$ – мнение агента i в момент времени t, $a_i \in \mathbb{R}_+$ – коэффициент восприимчивости к чужому мнению каждого агента общества, $S_i = \{j : (i,j) \in E\}$ – множество соседей агента i в графе g, $u_i \in U \subset (-\infty, \infty)$ – управление игрока на агента i.

Задача игрока состоит в поддержании мнения агентов общества на заданном уровне $\overline{x}$, затрачивая при этом минимум ресурсов на контроль мнением. Для этой задачи функционал определяется следующим образом

$$J(u) = \sum_{t=0}^{\infty} \delta^t \sum_{i=1}^{n} \left[(x_i(t) - \overline{x})^2 + \gamma u_i^2(t) \right],$$

где $\delta \in (0; 1)$ – коэффициент дисконтирования, $\gamma > 0$ – цена единицы контроля.

В векторном виде эти формулы представимы следующим образом:

$$x(t+1) = Ax(t) + Bu(t), \qquad (2)$$

где $x(t) = (x_1(t), x_2(t), \ldots, x_n(t))' \in \mathbb{R}^n$, и

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix}, \; B = \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{pmatrix},$$

где $b_i \in \{0, 1\} \; \forall i \in \{1, 2, \ldots, n\}$ определяет агентов, подверженных влиянию игрока и

$$a_{ij} = \begin{cases} 1 - \dfrac{|S_i| a_i}{|S_i| + 1} & i = j, \\ \dfrac{a_i}{|S_i| + 1} & j \in S_i, \\ 0 & j \notin \{S_i \cup i\}. \end{cases}$$

Функционал в векторном виде представим следующим образом

$$J(u) = \sum_{t=0}^{\infty} \delta^t \left[x'(t) Q x(t) + q x(t) + n \overline{x}^2 + \gamma u^2(t) \right], \tag{3}$$

где $Q = \mathbb{I}_n$ — единичная матрица размера n и $q = (-2\overline{x}, -2\overline{x}, \ldots, -2\overline{x})$.

1.2 Модернизация модели

В статьях [15, 16] и дипломной работе [17] мною была предложена модернизация представленной выше модели. В ней рассмотрим динамику мнений, определяемую уравнениями 1, но у каждого агента в функционале будет учтена "ценность" следующим образом

$$J(u) = \sum_{t=0}^{\infty} \delta^t \sum_{i=1}^{n} \left[q_i (x_i(t) - \overline{x})^2 + \gamma u_i^2(t) \right],$$

где $q_i \in \mathbb{R}_+$ – ценность агента i. И в векторном виде для уравнений 2

$$J(u) = \sum_{t=0}^{\infty} \delta^t \left[x'(t) Q x(t) + q x(t) + \sum_{i=1}^{n} q_i \overline{x}^2 + \gamma u^2(t) \right], \tag{4}$$

где $q = (-2q_1\overline{x}, -2q_2\overline{x}, \ldots, -2q_n\overline{x})$ и

$$Q = \begin{pmatrix} q_1 & 0 & \ldots & 0 \\ 0 & q_2 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & q_n \end{pmatrix}.$$

Глава 2. Теоретические аспекты исследования

В этой главе описана используемая теория. Сначала будут описаны утверждения, которые используются в работе и были получены из источников, а затем дополнительные утверждения, которые были сформулированы в рамках работы над проектом.

2.1 Утверждения, полученные из источников

В этом разделе описаны необходимые для этой работы утверждения, которые были получены мной и другими авторами ранее.

2.1.1 Решение задачи управления для модели динамики мнений без ценности

В статье [7] её авторами было предложено решение задачи для базовой модели с управлением в классе позиционных стратегий.

Утверждение 2.1. *Оптимальное управление в задаче минимизации 3 с динамикой состояния 2 определяется управлением вида*

$$u(t, x(t)) = cx(t) + c_0,$$

$$c_0 = -\frac{1}{2}\delta k B(\gamma + \delta B'KB)^{-1},$$

$$c = -(\gamma + \delta B'KB)^{-1}\delta B'KA,$$

и решением системы уравнений

$$K = Q + \gamma c'c + \delta(A + Bc)'K(A + Bc), \tag{5}$$

$$k = q + 2\gamma c c_0 + \delta k(A + Bc) + 2\delta c_0 B' K(A + Bc), \tag{6}$$

$$k_0 = n\overline{x}^2 + \gamma c_0^2 + \delta c_0 kB + \delta c_0^2 (B)' KB + \delta k_0. \tag{7}$$

В доказательстве этого утверждения применялся метод динамического программирования. Были составлены уравнения Беллмана, в предположении определенного вида управления.

2.1.2 Эквивалентные преобразования для модели динамики мнений с ценностью

В статьях [15, 16] и в дипломной работе [17], мною уже были сформулированы некоторые утверждения, которые позволяют уменьшать размерность задачи управления для некоторых видов графов. В этом разделе я опишу полученные ранее результаты.

Для этой работы я рассмотрю один вид преобразования, описанный мной ранее. Для его обозначения требуется определение моделей соответствующих двум видам графов: граф-звезда и граф "2 агента".

Граф-звезда характеризуется тем, что все связи в нем исходят из одного агента, будем называть его центр. Управление производится только на центр. Изолированных агентов нет. Пример такого графа представлен на рис. 1. На этом графе динамика мнений будет определяться уравнениями:

$$\widetilde{x}_i(t+1) = \begin{cases} \widetilde{x}_1(t) + \widetilde{a}_1 \left(\dfrac{\sum_{j=1}^{n} \widetilde{x}_j(t)}{n} - \widetilde{x}_1(t) \right) + u(t), & i = 1, \\ \widetilde{x}_i(t) + \widetilde{a}_i \left(\dfrac{\widetilde{x}_i(t) + \widetilde{x}_1(t)}{2} - \widetilde{x}_i(t) \right), & i \neq 1. \end{cases} \tag{8}$$

Или в векторном виде

$$\widetilde{x}(t+1) = \widetilde{A}\widetilde{x}(t) + \widetilde{B}u(t),$$

где

$$\widetilde{A} = \begin{pmatrix} 1 - \frac{\widetilde{a}_1(n-1)}{n} & \frac{\widetilde{a}_1}{n} & \frac{\widetilde{a}_1}{n} & \dots & \frac{\widetilde{a}_1}{n} \\ \frac{\widetilde{a}_2}{2} & 1 - \frac{\widetilde{a}_2}{2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \frac{\widetilde{a}_n}{2} & 0 & 0 & \dots & 1 - \frac{\widetilde{a}_n}{2} \end{pmatrix}, \; B = \begin{pmatrix} 1 \\ 0 \\ \dots \\ 0 \end{pmatrix}.$$

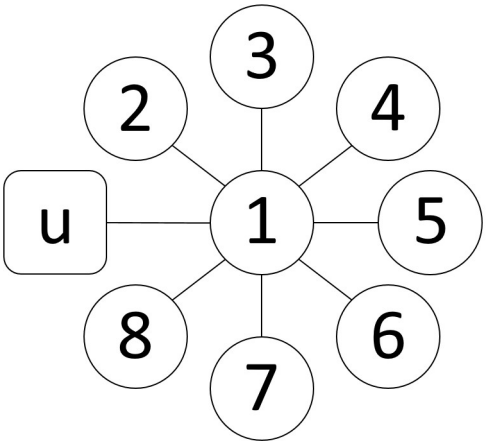

Рис. 1: Изображение графа-звезды при $n = 8$

Граф "2 агента" является двумя связанными вершинами. Управление производится на одну из вершин, как изображено на рис. 2. Динамика мнений на графе определяется уравнениями:

$$\widehat{x}_i(t+1) = \begin{cases} \widehat{x}_1(t) + \widehat{a}_1 \left(\dfrac{\widehat{x}_1(t) + \widehat{x}_2(t)}{2} - \widehat{x}_1(t) \right) + u(t), & i = 1, \\ \widehat{x}_2(t) + \widehat{a}_2 \left(\dfrac{\widehat{x}_1(t) + \widehat{x}_2(t)}{2} - \widehat{x}_2(t) \right), & i = 2. \end{cases} \tag{9}$$

Или в векторном виде

$$\widehat{x}(t+1) = \widehat{A}\widehat{x}(t) + \widehat{B}u(t),$$

где

$$\widetilde{A} = \begin{pmatrix} 1 - \frac{\widehat{a}_1}{2} & \frac{\widehat{a}_1}{2} \\ \frac{\widehat{a}_2}{2} & 1 - \frac{\widehat{a}_2}{2} \end{pmatrix}, \; B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

В работах [15, 17] мной были сформулированы и доказаны следующие

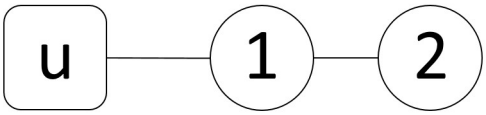

Рис. 2: Изображение графа "2 агента"

утверждения.

Утверждение 2.2. *Для модели динамики, определяемой уравнениями 8, при* $\widetilde{a}_2 = \widetilde{a}_3 = \ldots = \widetilde{a}_n$ *и* $\widetilde{x}_2(0) = \widetilde{x}_3(0) = \ldots = \widetilde{x}_n(0)$ *и модели, задаваемой уравнениями 9, при одинаковом управлении и при соотношении констант* $\widehat{a}_2 = \widetilde{a}_2$, $\widehat{a}_1 = \widetilde{a}_1 \frac{2(n-1)}{n}$ *и* $\widehat{x}_1(0) = \widetilde{x}_1(0)$, $\widehat{x}_2(0) = \widetilde{x}_2(0)$ *будут выполнятся равенства:*

$$\widehat{x}_1(t) = \widetilde{x}_1(t), \qquad \widehat{x}_2(t) = \widetilde{x}_2(t) = \widetilde{x}_3(t) = \cdots = \widetilde{x}_n(t) \qquad \forall t.$$

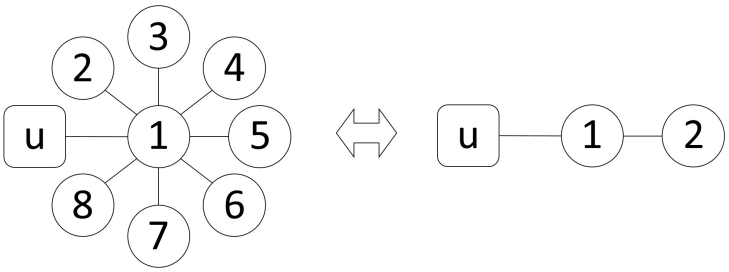

Рис. 3: Иллюстрация преобразования из утверждения 2.2.

Утверждение 2.3. *При условиях, описанных в Утверждении 2.2, задачи, определяемые функционалами*

$$\widetilde{J}(u) = \sum_{t=0}^{\infty} \delta^t \sum_{i=1}^{n} \left[\widetilde{q}_i (\widetilde{x}_i(t) - \overline{x})^2 + \gamma u_i^2(t) \right],$$

$$\widehat{J}(u) = \sum_{t=0}^{\infty} \delta^t \sum_{i=1}^{2} \left[\widehat{q}_i (\widehat{x}_i(t) - \overline{x})^2 + \gamma u_i^2(t) \right],$$

для моделей, определяемых графом-звездой и графом "2 агента" соответственно, являются эквивалентными при $\widehat{q}_1 = \widetilde{q}_1$, $\widehat{q}_2 = \sum_{i=2}^{n} \widetilde{q}_i$.

2.2 Утверждения, полученные для исследования

В этом разделе описаны новые утверждения, которые были получены специально для этой работы.

2.2.1 Решение задачи управления для модели динамики мнений с ценностью

В этом разделе по аналогии с доказательством из [7] будет получена система уравнений, которую требуется решить для нахождения решения в классе позиционных стратегий задачи модернизированной модели, определяемой системой 2 и функционалом 4.

Используется метод динамического программирования. Уравнение Беллмана для задачи минимизации имеет вид:

$$V(t,x) = \min_{u(t)\in U}[x'(t)Qx(t) + qx(t) + q_s\overline{x}^2 + \gamma u^2(t) + \delta V(t+1, x(t+1))] \tag{10}$$

где $q_s = \sum_{i=1}^{n} q_i$. Предполагается, что функция Беллмана, связанная с задачей минимизации 4, определяется как

$$V(t,x(t)) = x'(t)Kx(t) + kx(t) + k_0, \tag{11}$$

где

$$K = \begin{pmatrix} k_{11}, & k_{12}, & \dots & k_{1n} \\ k_{21}, & k_{22}, & \dots & k_{2n} \\ \dots & \dots & \dots & \dots \\ k_{n1}, & k_{n2}, & \dots & k_{nn} \end{pmatrix}, \; k = (k_1, k_2, \dots, k_n),$$

при этом $k_{ij} = k_{ji}$, $i \neq j$. Также предполагается, что используется позици-

онное управление вида

$$u(t, x(t)) = cx(t) + c_0, \tag{12}$$

где $c = (c_1, c_2, \ldots, c_n) \in \mathbb{R}^n, c_0 \in \mathbb{R}^1$.

Подставим сначала 11 в 10, а затем подставим 2, получим

$$\begin{aligned} V(t,x) = \min_{u(t)\in U}[x'(t)Qx(t) + qx(t) + q_s\overline{x}^2 + \gamma u^2(t) + \\ +\delta x'(t+1)Kx(t+1) + \delta kx(t+1) + \delta k_0] = \\ = \min_{u(t)\in U}[x'(t)Qx(t) + qx(t) + q_s\overline{x}^2 + \gamma u^2(t) + \\ +\delta(Ax(t) + Bu(t))'K(Ax(t) + Bu(t)) + \delta k(Ax(t) + Bu(t)) + \delta k_0]. \end{aligned} \tag{13}$$

Найдем минимум в правой части, получим уравнение

$$2\gamma u(t) + 2\delta B'KBu(t) + \delta kB + \delta B'KAx(t) + \delta(Ax(t))'KBu(t) = 0.$$

Так как $K' = K$, то $B'KAx(t) = ((Ax(t))'KB)'$. И так как $B'KAx(t)$ это число, то $B'KAx(t) = (Ax(t))'KB$. А значит

$$u^*(t) = -(\gamma + \delta B'KB)^{-1}\delta\left[\frac{1}{2}kB + B'KAx(t)\right].$$

Соответственно

$$c_0 = -\frac{1}{2}\delta kB(\gamma + \delta B'KB)^{-1}, \tag{14}$$

$$c = -(\gamma + \delta B'KB)^{-1}\delta B'KA. \tag{15}$$

Чтобы найти значения K и k подставим $u^*(t)$ в 13, получим

$$\begin{aligned} x'(t)Kx(t) + kx(t) + k_0 = x'(t)Qx(t) + qx(t) + q_s\overline{x}^2 + \gamma(cx(t) + c_0)^2 + \\ +\delta[(A + Bc)x(t) + Bc_0]'K[(A + Bc)x(t) + Bc_0] + \\ +\delta k[(A + Bc)x(t) + Bc_0] + \delta k_0. \end{aligned}$$

Тогда значения K, k и k_0 могут быть найдены как решение следующей

системы

$$K = Q + \gamma c'c + \delta(A + Bc)'K(A + Bc), \tag{16}$$

$$k = q + 2\gamma cc_0 + \delta k(A + Bc) + 2\delta c_0 B'K(A + Bc), \tag{17}$$

$$k_0 = q_s\overline{x}^2 + \gamma c_0^2 + \delta c_0 kB + \delta c_0^2 (B)'KB + \delta k_0. \tag{18}$$

Утверждение 2.4. *Оптимальное управление в задаче минимизации 4 с динамикой состояния 2 определяется системой 12, 14, 15 и решением системы уравнений 16, 17, 18.*

2.2.2 Эквивалентные преобразования для различных начальных состояний

В этом разделе уточняется утверждение 2.3 для определенного вида графов, и с видом управления, описанного в утверждении 2.4. Следствие является аналогом утверждения 2.3, с возможностью использования при различных начальных состояниях.

Следствие 2.1. *Рассмотрим модели динамики мнений, определяемой уравнениями 8, при* $\widetilde{a}_2 = \widetilde{a}_3 = \ldots = \widetilde{a}_n$ *и модель, задаваемую уравнениями 9. При одинаковом управлении и при соотношении констант* $\widehat{a}_2 = \widetilde{a}_2$, $\widehat{a}_1 = \widetilde{a}_1 \frac{2(n-1)}{n}$ *решение задач, определяемых функционалами*

$$\widetilde{J}(u) = \sum_{t=0}^{\infty} \delta^t \sum_{i=1}^{n} \left[(\widetilde{x}_i(t) - \overline{x})^2 + \gamma u_i^2(t) \right],$$

$$\widehat{J}(u) = \sum_{t=0}^{\infty} \delta^t \sum_{i=1}^{2} \left[\widehat{q}_i (\widehat{x}_i(t) - \overline{x})^2 + \gamma u_i^2(t) \right],$$

для моделей, определяемых графом-звездой и графом "2 агента" соответственно, эквивалентны, при соотношении $\widehat{q}_1 = 1$, $\widehat{q}_2 = n - 1$, *в предположении, что управление имеет вид*

$$u(t, x(t)) = cx(t) + c_0. \tag{19}$$

При этом управления равны, когда значения c и c_0соотносятся следующим образом $\widetilde{c}_0 = \widehat{c}_0$, $\widetilde{c}_1 = \widehat{c}_1$, $\widehat{c}_2 = \sum_{i=2}^{n} \widetilde{c}_i$

Доказательство. Нахождение значений c_0, c в предположении вида управления 19 не зависит от $x(t)$. Следовательно модель, соответствующую графу-звезде с разными начальными состояниями, для поиска значений c_0, c можно свести к модели, соответствующей графу-звезде с одинаковыми начальными состояниями, без изменений этих значений. К модели такого вида применимы утверждения 2.2, 2.3, что позволяет говорить о равенстве состояний для нецентральных агентов $\widehat{x}_2(t) = \widetilde{x}_i(t)\ \forall i \in 2, 3, \ldots, n\ \forall t$. При равенстве этих состояний $\widehat{c}x(t) + \widehat{c}_0 = \widetilde{c}x(t) + \widetilde{c}_0$ только если $\widetilde{c}_0 = \widehat{c}_0$, $\widetilde{c}_1 = \widehat{c}_1$, $\widehat{c}_2 = \sum_{i=2}^{n} \widetilde{c}_i$ □

2.2.3 Соотношение решений K, k, k_0, c, c_0

Для дальнейшей работы с преобразованием моделей требуется не только соотношение значений c и c_0, которое было показано в предыдущем разделе, но и соотношения соответствующих значений K, k, k_0 между моделями.

Утверждение 2.5. *Если для модели, соответствующей графу "2-агента" с единичными весами, существует решение*

$$K = \begin{pmatrix} k_{11} & k_{12} \\ k_{12} & k_{22} \end{pmatrix}, k = \begin{pmatrix} k_1 \\ k_2 \end{pmatrix}, c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix},\ k_0,\ c_0$$

то для модели, соответствующей граф-звезде, при соотношении пара-

метров, описанном в утверждениях 2.2, 2.3, значения

$$\widetilde{K} = \begin{pmatrix} k_{11} & k_{12}\frac{1}{n-1} & k_{12}\frac{1}{n-1} & \dots & k_{12}\frac{1}{n-1} \\ k_{12}\frac{1}{n-1} & k_{22}\frac{1}{(n-1)^2} & k_{22}\frac{1}{(n-1)^2} & \dots & k_{22}\frac{1}{(n-1)^2} \\ k_{12}\frac{1}{n-1} & k_{22}\frac{1}{(n-1)^2} & k_{22}\frac{1}{(n-1)^2} & \dots & k_{22}\frac{1}{(n-1)^2} \\ \dots & \dots & \dots & \dots & \dots \\ k_{12}\frac{1}{n-1} & k_{22}\frac{1}{(n-1)^2} & k_{22}\frac{1}{(n-1)^2} & \dots & k_{22}\frac{1}{(n-1)^2} \end{pmatrix}, \ \widetilde{k} = \begin{pmatrix} k_1 \\ k_2\frac{1}{n-1} \\ \dots \\ k_2\frac{1}{n-1} \end{pmatrix},$$

$$\widetilde{c} = \begin{pmatrix} c_1 \\ c_2\frac{1}{n-1} \\ \dots \\ c_2\frac{1}{n-1} \end{pmatrix}, \quad \widetilde{k}_0 = k_0, \ \widetilde{c}_0 = c_0$$

также будут являться решениями системы, за исключением уравнений определяющих k_{ii} $\forall i \in 2, 3, \dots, n$.

Доказательство. Соотношение значений c и c_0 для разных моделей было уже доказано выше. Далее будет показано, что если K, k, k_0 является решением, то и $\widetilde{K}$, $\widetilde{k}$, $\widetilde{k}_0$ является решением соответствующей задачи, за искдючением уравнений, определяющих k_{ii} $\forall i \in 2, 3, \dots, n$.

Распишем систему для модели, соответствующей графу-звезде.

$$\widetilde{K} = \widetilde{Q} + \gamma\widetilde{c}'\widetilde{c} + \delta(\widetilde{A} + \widetilde{B}\widetilde{c})'K(\widetilde{A} + \widetilde{B}\widetilde{c}), \tag{20}$$

$$\widetilde{k} = \widetilde{q} + 2\gamma\widetilde{c}\widetilde{c}_0 + \delta\widetilde{k}(\widetilde{A} + \widetilde{B}\widetilde{c}) + 2\delta\widetilde{c}_0\widetilde{B}'\widetilde{K}(\widetilde{A} + \widetilde{B}\widetilde{c}), \tag{21}$$

$$\widetilde{k}_0 = n\overline{x}^2 + \gamma\widetilde{c}_0^2 + \delta\widetilde{c}_0\widetilde{k}\widetilde{B} + \delta\widetilde{c}_0^2\widetilde{B}'\widetilde{K}\widetilde{B} + \delta\widetilde{k}_0. \tag{22}$$

Сначала проверим, что при таких соотношениях K, k, k_0 будут выполняться соотношения c, c_0. Сначала распишем c_0.

$$\widetilde{c}_0 = -\frac{\delta\widetilde{k}_1}{2(\gamma + \delta\widetilde{k}_{11})} = -\frac{\delta k_1}{2(\gamma + \delta k_{11})} = c_0$$

Распишем подробнее c. Начнем с c_1.

$$\widetilde{c}_1 = -\frac{\delta\left(\widetilde{a}_{11}\widetilde{k}_{11} + \sum_{j=2}^{n}\widetilde{a}_{j1}\widetilde{k}_{1j}\right)}{\gamma + \delta\widetilde{k}_{11}} = -\frac{\delta\left(\widehat{a}_{11}k_{11} + \sum_{j=2}^{n}\widehat{a}_{21}\frac{k_{12}}{n-1}\right)}{\gamma + \delta k_{11}} = \\ = -\frac{\delta\left(\widehat{a}_{11}k_{11} + \widehat{a}_{21}k_{12}\right)}{\gamma + \delta k_{11}} = c_1$$

Затем c_i $i \neq 1$

$$\widetilde{c}_i = -\frac{\delta\left(\widetilde{a}_{1i}\widetilde{k}_{11} + \widetilde{a}_{ii}\widetilde{k}_{1i}\right)}{\gamma + \delta\widetilde{k}_{11}} = -\frac{\delta\left(k_{11}\frac{\widehat{a}_{12}}{n-1} + \widehat{a}_{22}\frac{k_{12}}{n-1}\right)}{\gamma + \delta k_{11}} = \frac{c_i}{n-1}$$

Соотношение подтверждено. Распишем подробнее 22.

$$\widetilde{k}_0 = n\overline{x}^2 + \gamma\widetilde{c}_0^2 + \delta\widetilde{c}_0\widetilde{k}_1 + \delta\widetilde{c}_0^2\widetilde{k}_{11} + \delta\widetilde{k}_0 = \\ = 2\overline{x}^2 + \gamma c_0^2 + \delta c_0 k_1 + \delta c_0^2 k_{11} + \delta k_0 = k_0.$$

Распишем 21 сначала для первого элемента вектора

$$\widetilde{k}_1 = -2\overline{x} + 2\gamma\widetilde{c}_1\widetilde{c}_0 + \delta(\widetilde{k}_1(\widetilde{a}_{11} + \widetilde{c}_1) + \widetilde{k}_2\widetilde{a}_{21} + \cdots + \widetilde{k}_n\widetilde{a}_{n1}) + \\ + 2\delta\widetilde{c}_0(\widetilde{k}_{11}(\widetilde{a}_{11} + \widetilde{c}_1) + \widetilde{k}_{12}\widetilde{a}_{21} + \cdots + \widetilde{k}_{1n}\widetilde{a}_{n1}) = \\ = -2\overline{x} + 2\gamma c_1 c_0 + \delta(k_1(1 - \frac{(n-1)\widetilde{a}_1}{n} + c_1) + \sum_{i=2}^{n} k_2\frac{\widetilde{a}_i}{(n-1)2}) + \\ + 2\delta c_0(k_{11}(1 - \frac{(n-1)\widetilde{a}_1}{n} + c_1) + \sum_{i=2}^{n} k_{12}\frac{\widetilde{a}_i}{(n-1)2}) = \\ = -2\overline{x} + 2\gamma c_1 c_0 + \delta(k_1(1 - \frac{\widehat{a}_1}{2} + c_1) + k_2\frac{\widehat{a}_2}{2}\sum_{i=2}^{n}\frac{1}{(n-1)} + \\ + 2\delta c_0(k_{11}(1 - \frac{\widehat{a}_1}{2} + c_1) + k_{12}\frac{\widehat{a}_2}{2}\sum_{i=2}^{n}\frac{1}{(n-1)}) = \\ = -2\overline{x} + 2\gamma c_1 c_0 + \delta(k_1(\widehat{a}_{11} + c_1) + k_2\widehat{a}_{21}) + \\ + 2\delta c_0(k_{11}(\widehat{a}_{11} + c_1) + k_{12}\widehat{a}_{21}) = k_1.$$

Затем для i-го элемента $\forall i \in \{2, 3, \ldots, n\}$

$$\widetilde{k}_i = -2\overline{x} + 2\gamma\widetilde{c}_i\widetilde{c}_0 + \delta(\widetilde{k}_1(\widetilde{a}_{1i} + \widetilde{c}_i) + \widetilde{k}_2\widetilde{a}_{2i} + \cdots + \widetilde{k}_n\widetilde{a}_{ni}) +$$
$$+2\delta\widetilde{c}_0(\widetilde{k}_{11}(\widetilde{a}_{1i} + \widetilde{c}_i) + \widetilde{k}_{12}\widetilde{a}_{2i} + \cdots + \widetilde{k}_{1n}\widetilde{a}_{ni}) =$$
$$= -2\overline{x}(n-1)\frac{1}{n-1} + 2\gamma\frac{1}{n-1}c_2c_0 + \delta(k_1(\frac{\widetilde{a}_1}{n} + \frac{1}{n-1}c_2) +$$
$$+k_2\frac{1}{n-1}(1 - \frac{\widetilde{a}_i}{2})) + 2\delta c_0(k_{11}(\frac{\widetilde{a}_1}{n} + \frac{1}{n-1}c_2) + k_{12}\frac{1}{n-1}(1 - \frac{\widetilde{a}_i}{2})) =$$
$$= -2\overline{x}(n-1)\frac{1}{n-1} + 2\gamma\frac{1}{n-1}c_2c_0 + \delta(k_1(\frac{\widehat{a}_1}{2}\frac{1}{n-1} + \frac{1}{n-1}c_2) +$$
$$+k_2\frac{1}{n-1}(1 - \frac{\widehat{a}_i}{2})) + 2\delta c_0(k_{11}(\frac{\widetilde{a}_1}{n} + \frac{1}{n-1}c_2) + k_{12}\frac{1}{n-1}(1 - \frac{\widehat{a}_i}{2})) = \frac{1}{n-1}k_i.$$

И подробнее рассмотрим 20. будет рассмотрено 3 вида выражений. Сначала элемент k_{11}

$$\widetilde{k}_{11} = 1 + \gamma\widetilde{c}_1^2 + \delta(\widetilde{a}_{11} + \widetilde{c}_1)\left((\widetilde{a}_{11} + \widetilde{c}_1)\widetilde{k}_{11} + \sum_{j=2}^{n}\widetilde{a}_{j1}\widetilde{k}_{1j}\right) +$$
$$+\delta\sum_{i=2}^{n}\left(\widetilde{a}_{11}(\widetilde{a}_{11} + \widetilde{c}_1)\widetilde{k}_{1i} + \sum_{j=2}^{n}\widetilde{a}_{j1}\widetilde{k}_{ij}\right) =$$
$$= 1 + \gamma c_1^2 + \delta(1 - \frac{(n-1)\widetilde{a}_1}{n} + \widetilde{c}_1)\left((1 - \frac{(n-1)\widetilde{a}_1}{n} + c_1)k_{11} + \frac{k_{12}}{n-1}\sum_{j=2}^{n}\frac{\widetilde{a}_j}{2}\right) +$$
$$+\delta\sum_{i=2}^{n}\left((1 - \frac{(n-1)\widetilde{a}_1}{n})(1 - \frac{(n-1)\widetilde{a}_1}{n} + \widetilde{c}_1)\frac{k_{12}}{n-1} + \sum_{j=2}^{n}\frac{\widetilde{a}_j}{2}\widetilde{k}_{ij}\right) =$$
$$= 1 + \gamma c_1^2 + \delta(1 - \frac{\widehat{a}_1}{2} + \widetilde{c}_1)\left((1 - \frac{\widehat{a}_1}{2} + c_1)k_{11} + \frac{k_{12}}{n-1}\widehat{a}_2\sum_{j=2}^{n}\frac{1}{2} +\right)$$
$$+\delta\sum_{i=2}^{n}\left((1 - \frac{\widehat{a}_1}{2})(1 - \frac{\widehat{a}_1}{2} + \widetilde{c}_1)\frac{k_{12}}{n-1} + \frac{\widehat{a}_2}{2}\sum_{j=2}^{n}\widetilde{k}_{ij}\right) =$$
$$= 1 + \gamma c_1^2 + \delta(\widehat{a}_{11} + c_1)\left((\widehat{a}_{11} + c_1)k_{11} + \widehat{a}_{21}k_{12}\right) +$$
$$+\delta\left(\widehat{a}_{11}(\widehat{a}_{11} + c_1)k_{12} + \widehat{a}_{21}k_{22}\right) = k_{11}.$$

Потом элементы k_{1i}, k_{i1} $\forall i \in \{2, 3, \ldots, n\}$. Обратим внимание, что все матрицы в выражении, включая саму матрицу K, являются симметричными,

поэтому не умаляя общности будем рассматривать только k_{1i}

$$\widetilde{k}_{1i} = 0 + \gamma\widetilde{c}_1\widetilde{c}_i + \delta(\widetilde{c}_i + \widetilde{a}_{1i})\left((\widetilde{a}_{11} + \widetilde{c}_1)k_{11} + \sum_{j=2}^{n}\widetilde{a}_{j1}\widetilde{k}_{j1}\right) +$$
$$+\delta\widetilde{a}_{11}\left((\widetilde{a}_{11} + \widetilde{c}_1)k_{1i} + \sum_{j=2}^{n}\widetilde{a}_{ji}\widetilde{k}_{ji}\right) =$$
$$= \gamma\frac{c_1c_2}{n-1} + \delta\left(\frac{c_2}{n-1} + \frac{\widetilde{a}_1}{n}\right)\left(k_{11}(c_1 + 1 - \frac{\widetilde{a}_1(n-1)}{n}) + \frac{\widehat{a}_2}{2(n-1)}\sum_{j=2}^{n}k_{12}\right) +$$
$$+\delta(1 - \frac{\widehat{a}_2}{2})\left(\frac{k_{12}}{n-1}(c_1 + 1 - \frac{\widetilde{a}_1(n-1)}{n}) + k_{12}\frac{\widehat{a}_2}{2(n-1)}\right) =$$
$$= \gamma\frac{c_1c_2}{n-1} + \delta\left(\frac{c_2}{n-1} + \frac{\widehat{a}_1}{2(n-1)}\right)\left(k_{11}(c_1 + 1 - \frac{\widehat{a}_1}{2}) + \frac{\widehat{a}_2}{2}k_{12}\right) +$$
$$+\delta(1 - \frac{\widehat{a}_2}{2})\left(\frac{k_{12}}{n-1}(c_1 + 1 - \frac{\widehat{a}_1}{2}) + k_{12}\frac{\widehat{a}_2}{2(n-1)}\right) =$$
$$= \frac{1}{n-1}\big(\gamma c_1c_2 + \delta(c_i + \widehat{a}_{12})\left((\widehat{a}_{11} + c_1)k_{11} + \widehat{a}_{21}k_{21}\right) +$$
$$+\delta\widehat{a}_{11}\left((\widehat{a}_{11} + c_1)k_{12} + \widetilde{a}_{22}\widetilde{k}_{22}\right)\big) = \frac{k_{12}}{n-1}.$$

Потом элементы k_{ij} $i \neq j$, которые не подходят под предыдущие случаи.

$$k_{ij} = 0 + \gamma\widetilde{c}_i\widetilde{c}_j + \delta(\widetilde{a}_{1j} + \widetilde{c}_j)((\widetilde{a}_{1i} + \widetilde{c}_i)\widetilde{k}_{11} + \widetilde{a}_{ii}\widetilde{k}_{i1}) +$$
$$+\delta\widetilde{a}_{jj}((\widetilde{a}_{1i} + \widetilde{c}_i)\widetilde{k}_{1j} + \widetilde{a}_{ii}\widetilde{k}_{ij}) =$$
$$= \gamma\frac{c_2c_2}{(n-1)^2} +$$
$$+\delta\left(\frac{\widetilde{a}_1}{n} + \frac{c_2}{n-1}\right)\left((\frac{\widetilde{a}_1}{n} + \frac{c_2}{n-1})k_{11} + ((1 - \frac{\widehat{a}_2}{2})\frac{k_{12}}{n-1})\right) +$$
$$+\delta(1 - \frac{\widehat{a}}{2})\left((\frac{\widetilde{a}_1}{n} + \frac{c_2}{n-1})\frac{k_{12}}{n-1} + ((1 - \frac{\widehat{a}_2}{2})\frac{k_{22}}{(n-1)^2})\right) =$$

$$= \gamma \frac{c_2^2}{(n-1)^2} +$$
$$+\delta \left(\frac{\widehat{a}_1}{2(n-1)} + \frac{c_2}{n-1} \right) \left((\frac{\widehat{a}_1}{2(n-1)} + \frac{c_2}{n-1}) k_{11} + ((1 - \frac{\widehat{a}_2}{2}) \frac{k_{12}}{n-1}) \right) +$$
$$+\delta (1 - \frac{\widehat{a}_2}{2}) \left((\frac{\widehat{a}_1}{2(n-1)} + \frac{c_2}{n-1}) \frac{k_{12}}{n-1} + ((1 - \frac{\widehat{a}_2}{2}) \frac{k_{22}}{(n-1)^2}) \right) =$$
$$= \frac{1}{(n-1)^2} \big(\gamma c_2^2 + \delta (\widehat{a}_{12} + c_2)((\widehat{a}_{12} + c_2) k_{11} + \widehat{a}_{22} k_{12}) +$$
$$+\delta \widehat{a}_{22} ((\widehat{a}_{12} + c_2) k_{12} + \widehat{a}_{22} k_{22}) \big) = \frac{k_{22}}{(n-1)^2}$$

Доказано, что все соотношения удовлетворяют заявленным уравнениям. □

Глава 3. Алгоритм преобразования сетевой структуры

1. Преобразовать модель к виду, к которому можно применить метод уменьшения размерности графа.

2. Применить эквивалентное преобразование к графу меньшей размерности, следуя утверждениям 2.2, 2.3.

3. На получившемся графе находить решение системы, описанной в утверждении 2.4 символьными методами (Mathematica). Среди корней выбрать тот, где матрица K положительно определенная.

4. Взяв для начальной точки решение, полученное в пункте 3, провести численное решение той же системы (Python).

5. Численно получить решение эквивалентной задачи большей размерности, используя утверждение 2.5 и следствие 2.1 (Python).

6. Взяв для начальной точки решение, полученное в 5 пункте алгоритма, провести численное решение системы, описанной в утверждении 2.1, соответствующей графу, до преобразования в 1 пункте алгоритма.

Дальнейшее исследование направлено на поиск правила, по которому можно выполнять 1 пункт алгоритма.

Глава 4. Экспериментальная часть исследования

4.1 Эксперимент №1 Соотношение решений K, k, k_0

В этом эксперименте исследуется возможность преобразования решения, полученного решением задачи, соответствующей графу "2 агента" к решению задачи, соответствующей граф-звезде.

4.1.1 Цель эксперимента

Показать, что, если для модели, соответствующей графу "2-агента", существует решение

$$K = \begin{pmatrix} k_{11} & k_{12} \\ k_{12} & k_{22} \end{pmatrix}, k = \begin{pmatrix} k_1 \\ k_2 \end{pmatrix}, \ k_0$$

такое, что матрица K положительно определенная, то для модели, соответствующей граф-звезде, при соотношении параметров, описанном в утверждениях 2.2 2.3 значения

$$\widetilde{K} = \begin{pmatrix} k_{11} & k_{12}\frac{1}{n-1} & k_{12}\frac{1}{n-1} & \dots & k_{12}\frac{1}{n-1} \\ k_{12}\frac{1}{n-1} & k_{22}\frac{1}{(n-1)^2} & k_{22}\frac{1}{(n-1)^2} & \dots & k_{22}\frac{1}{(n-1)^2} \\ k_{12}\frac{1}{n-1} & k_{22}\frac{1}{(n-1)^2} & k_{22}\frac{1}{(n-1)^2} & \dots & k_{22}\frac{1}{(n-1)^2} \\ \dots & \dots & \dots & \dots & \dots \\ k_{12}\frac{1}{n-1} & k_{22}\frac{1}{(n-1)^2} & k_{22}\frac{1}{(n-1)^2} & \dots & k_{22}\frac{1}{(n-1)^2} \end{pmatrix}, \ \widetilde{k} = \begin{pmatrix} k_1 \\ k_2\frac{1}{n-1} \\ \dots \\ k_2\frac{1}{n-1} \end{pmatrix}, \ \widetilde{k}_0 = k_0, \tag{23}$$

могут являться начальными для использования численного метода Ньютона. Метод будет сходится и полученная матрица $\widetilde{K}$ будет положительно определенной. Полученные управления в этих двух задачах при этом будут равны.

4.1.2 Теоретические обоснования

В утверждении 2.5 показано, что при таком соотношении значений значительная часть уравнений выполняется, что является предпосылкой к использованию именно такого соотношения в качестве начальной точки алгоритма.

В следствии 2.1 показано, что для равенства управлений, которое следует из преобразования по утверждениям 2.2, 2.3 требуется определенное соотношение значений c и c_0, которое и будет использовано в качестве проверки результата эксперимента, поскольку этот шаг алгоритма предполагает, что полученное решение большей задачи будет соотносится с решением меньшей задачи.

4.1.3 Описание эксперимента

Для проведения эксперимента генерировались различные наборы параметров для задачи. Брались все значения параметров в заданных границах с определенным малым шагом. Для каждого набора параметров выполнялись следующие шаги.

- Решить задачу соответствующей графу "2 агента".
- Сопоставить параметры задачи для граф-звезды в соотношении указанном в утверждении 2.2.
- Применить метод Ньютона для решения системы уравнений, описанных в утверждении 2.4, соответствующей граф-звезде, выбрав начальную точку по формуле 23.
- Обработать полученные решения. Проверить положительную определенность матриц K. Проверить корректность соотношения значений c_0 и c.

4.1.4 Реализация эксперимента

Для получения решения задачи, соответствующей графу "2 агента" были написаны программы на Wolfram language Приложение 1 и на Python

Приложение 2. Программа на Wolfram language решает систему уравнений, описанную в утверждении 2.4 символьными методами, и среди полученных решений выбирает решение с положительно определенной матрицей K, полученные решения записываются в файлы. Программа на Python решает численным методом Ньютона ту же систему, начиная с полученного первой программой решения, для достижения большей точности, и записывает полученный результат.

Для решения задачи, соответствующей граф-звезде, методом Ньютона написан код на Python Приложение 3, в нем за начальную точку для алгоритма берется решение из предыдущей программы, преобразованное по формуле 23.

Чтобы обработать полученные результаты был написан код на Python Приложение 4, в котором проверялись положительная определенность матрицы K, и корректность соотношений значений c_0 и c. Отслеживались нарушения этих условий и подсчитывалось количество нарушений.

4.1.5 Результаты эксперимента

В ходе реализации эксперимента было сгенерировано более 400000 комбинаций параметров для размерностей графа-звезды от 3 вершин до 9. Все результаты эксперимента показали возможность использования предложенного подхода, поскольку во всех случаях выполнялись отслеживаемые условия. Значения в решаемой системе уравнений при подстановке полученных решений близки к равным и полученная матрица K положительно определенная.

4.2 Эксперимент №2 изменение параметров на графе малого размера

В этом эксперименте исследуется возможность преобразования решения задачи, соответствующей граф-звезде с одинаковыми параметрами a в нецентральных агентах, к задаче, соответствующей граф-звезде с различными параметрами a в случае трех агентов в сети.

4.2.1 Цель эксперимента

Исследовать при каких значениях параметров a для модели, соответствующей граф-звезде с тремя агентами с не совпадающими параметрами a в нецентральных агентах, допустимо использование решения K, k, k_0 системы уравнений 5, 6, 7, полученного при решении задачи соответствующей граф-звезде с тремя агентами с совпадающими параметрами a в нецентральных агентах.

4.2.2 Описание эксперимента

Для проведения эксперимента были взяты результаты 1 эксперимента. Эти результаты являются набором решений задачи, соответствующей граф-звезде с одинаковыми параметрами в нецентральных элементах, для различных наборов параметров. Были сгенерированы наборы параметров a для нецентральных элементов, являющиеся перебором всех наборов значений от 0 до 1 с определенным малым шагом. Каждое решение использовалось как начальное значение для алгоритма Ньютона для задачи, в которой были изменены только параметры нецентральных элементов. Подсчитывались успешные запуски алгоритма.

4.2.3 Реализация эксперимента

Для реализации эксперимента был написан код на Python Приложение 5. Программа обрабатывает полученные в 1 эксперименте решения для графов с 3-мя вершинами. Для двух нецентральных элементов генерируются различные наборы значений параметров с заданным малым шагом. Остальные параметры сохраняются. Каждое решение используется как начальное значение для алгоритма Ньютона для измененной задачи. Успешные запуски подсчитывались по следующим критериям: в полученном решении матрица K положительно определенная, при подстановке в уравнения полученное решение, значения близки к равным.

4.2.4 Результаты эксперимента

Эксперимент был проведен для более 60000 наборов решений и 50 наборов параметров a. В результате эксперимента все наборы решений показали, что результаты подходят в качестве начального значения для метода Ньютона для любых наборов параметров a при условии, что $a_i \in (0, 1)$ $\forall i \in 2, 3, \ldots, n$. Все решения показали, что значения в решаемой системе уравнений при подстановке полученных решений близки к равным, и полученная матрица K положительно определенная. Таким образом в качестве первого действия алгоритма допустимо преобразовать параметры в нецентральных элементах к любым параметрам в рамках ограничения.

4.3 Эксперимент №3 изменение параметров на графе большего размера

Поскольку случай с двумя нецентральными агентами ограничен тем, что в нем невозможна ситуация, что большинство агентов значительно отличается от меньшинства, поэтому этот эксперимент повторяет эксперимент 2, но исследуется граф с тремя нецентральными агентами.

4.3.1 Цель эксперимента

Исследовать при каких значениях параметров a для модели, соответствующей граф-звезде с четырьмя агентами с не совпадающими параметрами a в нецентральных агентах, допустимо использование решения K, k, k_0 системы уравнений 5, 6, 7, полученного при решении задачи соответствующей граф-звезде с четырьмя агентами с совпадающими параметрами a в нецентральных агентах.

4.3.2 Описание эксперимента

4.3.3 Реализация эксперимента

Для реализации эксперимента был написан код на Python Приложение 6, который аналогичен коду из 2 эксперимента, но генерирует значения уже для

4.3.4 Результаты эксперимента

Эксперимент был проведен для более 60000 наборов решений и 150 наборов параметров a. В результате эксперимента все наборы решений показали, что результаты подходят в качестве начального значения для метода Ньютона для любых наборов параметров a при условии, что $a_i \in (0, 1)$ $\forall i \in 2, 3, \ldots, n$. Все решения показали, что значения в решаемой системе уравнений при подстановке полученных решений близки к равным, и полученная матрица K положительно определенная. Результат аналогичен результату 2 эксперимента. Таким образом, в качестве первого действия алгоритма, допустимо преобразовать параметры в нецентральных элементах к любым параметрам в рамках ограничения.

Заключение

Предложен алгоритм численного решения модели динамики мнений с управлением с использованием подхода преобразования сетевой структуры. Экспериментально проверен алгоритм для моделей соответствующих граф-звездам.

Список литературы

[1] DeGroot M. H. Reaching a consensus //Journal of the American Statistical Association. – 1974. – Т. 69. – №. 345. – С. 118-121.

[2] Friedkin N. E., Johnsen E. C. Social influence and opinions //Journal of Mathematical Sociology. – 1990. – Т. 15. – №. 3-4. – С. 193-206.

[3] Hegselmann R., Krause U. Opinion dynamics driven by various ways of averaging //Computational Economics. – 2005. – Т. 25. – №. 4. – С. 381-405.

[4] Bure V. M., Parilina E. M., Sedakov A. A. Consensus in a social network with two principals //Automation and Remote Control. – 2017. – Т. 78. – №. 8. – С. 1489-1499.

[5] Mazalov V., Parilina E. Game of competition for opinion with two centers of influence //International Conference on Mathematical Optimization Theory and Operations Research. – Springer, Cham, 2019. – С. 673-684.

[6] Mazalov V., Parilina E. The Euler-Equation Approach in Average-Oriented Opinion Dynamics //Mathematics. – 2020. – Т. 8. – №. 3. – С. 1–6.

[7] Mazalov V. V., Dorofeeva Y. A., Parilina E. M. Opinion control in a team with complete and incomplete communication // Contributions to Game Theory and Management. - 2020. - Т. 13. С. 324–334.

[8] Дорофеева Ю. А. Влияние управления на динамику мнений участников коллектива //Труды Карельского научного центра Российской академии наук. – 2020. – №. 7. - С. 28-33.

[9] Chkhartishvili A. G., Gubanov D. A., Novikov D. A. Social Networks: Models of information influence, control and confrontation. – Springer, 2018. – Т. 189.

[10] Гао, Ц. Динамика мнений в мультиагентных системах с оптимальным выбором моментов проверки мнений / Ц. Гао, Е. М. Парилина // Ма-

тематическая теория игр и ее приложения. – 2022. – Т. 14, № 4. – С. 3-23.

[11] Sedakov A. A., Zhen M. Opinion dynamics game in a social network with two influence nodes // Вестник Санкт-Петербургского университета. Прикладная математика. Информатика. Процессы управления. – 2019. – №. 1. - С. 118–125.

[12] Рогов М. А., Седаков А. А. Согласованное влияние на мнения участников социальной сети // Математическая теория игр и её приложения. – 2018. – Т. 10. – №. 4. – С. 30-58.

[13] Avrachenkov K. E., Kondratev A. Y., Mazalov V. V. Cooperative game theory approaches for network partitioning //International Computing and Combinatorics Conference. – Springer, Cham, 2017. – С. 591-602.

[14] Mazalov V. V. Comparing game-theoretic and maximum likelihood approaches for network partitioning //Transactions on Computational Collective Intelligence XXXI. – Springer, Berlin, Heidelberg, 2018. – С. 37-46.

[15] Нечипорук, А. А. Эквивалентность моделей динамики мнений в социальных сетях // Процессы управления и устойчивость. – 2021. – Т. 8, № 1. – С. 435-439.

[16] Нечипорук, А. А. Эквивалентность моделей динамики мнений в социальных сетях с повторяющейся структурой // Процессы управления и устойчивость. – 2022. – Т. 9, № 1. – С. 427-431.

[17] Нечипорук, А. А. Анализ параметров моделей динамики мнений в социальных сетях // URL: http://hdl.handle.net/11701/32217 (Дата обращения 17.05.2023)

ПРИЛОЖЕНИЕ №1

```
ClearAll[Evaluate[Context[] <> "*"]];
tries = Import["..\\Python\\2_agents_param\\List_of_Tries.txt", "Table"];
Triesn = tries[[1]][[1]];
Triesdelta = tries[[2]][[1]];
Triesgamma = tries[[3]][[1]];
Triesxhat = tries[[4]][[1]];
Triesa = tries[[5]][[1]];
ag = 2;
B:=PadRight[{1}, ag];
A:=ToExpression[Import["A_Mathematica\\A_2_agents.txt", "Text"]];

Q:=DiagonalMatrix[q0];
q:= -2 * q0 * xHat;
q0:={1, n - 1};
k:=Table[ToExpression[k <> ToString[i]], {i, 1, ag}];
K:=Table[If[i<=j, ToExpression[StringJoin[k, ToString[i], ToString[j]]],
ToExpression[StringJoin[k, ToString[j], ToString[i]]]], {i, ag}, {j, ag}];
k0:=k0;
c0:= -(0.5 * d * k.B)/(gamma + d * B.K.B);
c:= -d * B.K.A/(gamma + d * B.K.B);
eq1:=k0 - Total[q0] * xHat^2 - gamma * c0^2 - d * k.B * c0 - d * (B * c0).K.B * c0
eq2:=K - Q - gamma * c.c - d * Transpose[A + B.c].K.(A + B.c)==0;
eq3:=k - q - 2 * gamma * c * c0 - d * k.(A + B.c) - 2 * d * c0 * Transpose[(A + B.c)
```

```
a = {0, 0};
For[n = 3, n < Triesn, n++,
For[xHat = -Triesxhat/10, xHat<=0, xHat+=1/10,
For[d = 1/Triesdelta, d < 1, d+=1/Triesdelta,

For[gamma = 1/10, gamma < Triesgamma/10, gamma+=1/10,
For[a1 = 1/10, a1 < Triesa/10, a1+=1/10,
a[[1]] = a1 * 2 * (n - 1)/n;
For[a[[2]] = 1/10, a[[2]] < Triesa/10, a[[2]]+=1/10,
sols = Solve[Join[eq1, eq2, eq3], Flatten[{k0, Flatten[k], Flatten[K]}]];
For[i = 1, i < Length[sols], i++,
If[PositiveDefiniteMatrixQ[K/.sols[[i]]], Kopt:=Evaluate[K/.sols[[i]]];
kopt:=Evaluate[k/.sols[[i]]];
k0opt:=Evaluate[k0/.sols[[i]]];

Break[];

];
];
filename = ToString[StringForm[" 1\\2ag_res1_2_3_4_5_6.txt",
ToString[n],
ToString@NumberForm[N[d], {Infinity, 1}],
ToString@NumberForm[N[gamma], {Infinity, 1}],
ToString@NumberForm[N[xHat], {Infinity, 1}],
ToString@NumberForm[N[a1], {Infinity, 1}],
```

```
ToString@NumberForm[N[a[[2]]], {Infinity, 1}]];
Put[Kopt, kopt, k0opt, filename];


]
]
]
]
]
]
```

ПРИЛОЖЕНИЕ №2

```
import numpy as np
import multiprocessing
from scipy.optimize import fsolve


def f_solve(N, Q, q, delta, gamma, x_hat, A, B, K_Math,
    k_Math, k_0_Math, filename):
    def f_c_0(k, K):
        return -0.5 * delta * np.matmul(k, B) * (gamma +
            delta * np.matmul(np.matmul(np.transpose(B), K
            ), B)) ** -1

    def f_c(K):
        return -delta * np.matmul(np.matmul(np.transpose(
            B), K), A) / (
                gamma + delta * np.matmul(np.matmul(np.
                    transpose(B), K), B))

    def f_K(c, K):
        return Q + gamma * np.matmul(np.transpose(c), c)
            + delta * np.matmul(
              np.matmul(np.transpose(A + np.matmul(B, c)),
                K), (A + np.matmul(B, c)))

    def f_k(c, c_0, k, K):
        return q + 2 * gamma * c * c_0 + delta * np.
            matmul(k, (A + np.matmul(B, c))) + 2 * delta *
             c_0 * np.matmul(
              np.matmul(np.transpose(B), K), (A + np.matmul
                (B, c)))
```

```
def f_k_0(c_0, k, K, k_0):
    return N * x_hat ** 2 + gamma * c_0 ** 2 + delta
        * c_0 * np.matmul(k, B) + delta * c_0 ** 2 *
        np.matmul(
          np.matmul(np.transpose(B), K),
          B) + delta * k_0

def F(all_in):
    v_K = all_in[:2 * 2].reshape(2, 2)
    v_k = all_in[2 * 2:2 * 2 + 2]
    v_k_0 = all_in[2 * 2 + 2]

    v_c_0 = f_c_0(v_k, v_K)
    v_c = f_c(v_K)

    diff_K = f_K(v_c, v_K) - v_K
    diff_k = f_k(v_c, v_c_0, v_k, v_K) - v_k
    diff_k_0 = f_k_0(v_c_0, v_k, v_K, v_k_0) - v_k_0
    all_in = np.concatenate((diff_K.flatten(), diff_k
        .flatten(), diff_k_0.flatten()))

    return all_in

KKK = np.concatenate((K_Math.flatten(), k_Math, [
    k_0_Math]))
sol = fsolve(F, KKK)
res_K = sol[:2 * 2].reshape(2, 2)
res_k = sol[2 * 2:2 * 2 + 2]
res_k_0 = sol[2 * 2 + 2]

res_c = f_c(res_K)
res_c_0 = f_c_0(res_k, res_K)
```

```
    with open("2_agents_sols\\" + filename, 'w') as filew
        :
            print(res_K, file=filew)
            print(is_positive_definite(res_K), file=filew)
            print(res_c, file=filew)
            print(res_c_0, file=filew)
            print(res_k, file=filew)
            print(res_k_0, file=filew)
    return sol


def is_positive_definite(matrix):
    eigenvalues, _ = np.linalg.eig(matrix)
    return np.all(eigenvalues > 0)


def float_range(start, stop, step):
    while start < stop:
        yield start
        start += step


def reading_Kkk(file_name):
    with open("..\\files_Mathematica\\Exp_1\\" +
        file_name) as file:
            data_Math = file.readlines()
    data_K = ''
    for i in data_Math[:-2]:
        data_K += i
    data_K = data_K.replace('{', '[')
    data_K = data_K.replace('}', ']')
    K = np.array(eval(data_K))
    k = np.array([float(num) for num in data_Math
```

```
        [-2][1:-2].split(", ")])
    k_0 = float(data_Math[-1])
    return K, k, k_0


if __name__ == '__main__':
    with open("2_agents_param\\List_of_Tries.txt", 'r')
        as file:
        Tries_n = int(file.readline())
        Tries_delta = int(file.readline())
        Tries_gamma = int(file.readline())
        Tries_x_hat = int(file.readline())
        Tries_a = int(file.readline())
    a = [0, 0]
    with open('2_agents_param\\A_2_agents.txt', 'r') as
        file:
        array_string = file.readline()
    B = np.zeros((2, 1))
    B[0, 0] = 1
    dataset = []
    args_set = []
    args_set2 = []
    file_count = 0
    last_file = ''
    for n in range(3, Tries_n, 1):
        print(f"{n} start make dataset")
        q0 = np.array([1.0, n - 1.0])
        Q = np.diag(q0)
        for x_hat in float_range(- Tries_x_hat / 10, 0 +
            1 / 10, 1 / 10):
            q = -2 * q0 * x_hat
            for delta in float_range(0 + 1 / Tries_delta,
                1, 1 / Tries_delta):
```

```
                    for gamma in float_range(0 + 1 / 10,
                        Tries_gamma / 10, 1 / 10):
                        for a0 in float_range(0 + 1 / 10,
                            Tries_a / 10, 1 / 10):
                            a[0] = a0 * 2 * (n - 1) / n
                            for a[1] in float_range(0 + 1 /
                                10, Tries_a / 10, 1 / 10):
                                dataset.append([n, delta,
                                    gamma, x_hat, a])
                                A = np.array(
                                    [[1 - 1 * a[0] / 2, a[0]
                                        / 2],
                                     [a[1] / 2, 1 - a[1] /
                                        2]])  # np.array(eval
                                        (array_string))
                                filename = f"2ag_res{n}_{
                                    round(delta, 1)}_{round(
                                    gamma, 1)}_{round(x_hat, 
                                    1)}_{round(a0, 1)}_{round(
                                    a[1], 1)}.txt"
                                try:
                                    K_Math, k_Math, k_0_Math
                                        = reading_Kkk(filename
                                        )
                                except:
                                    continue
                                last_file = filename
                                file_count += 1
                                args_set.append((n, Q, q,
                                    delta, gamma, x_hat, A, B,
                                     K_Math, k_Math, k_0_Math,
                                     filename))
print("datasets ready")
```

```
print(file_count, last_file)
num_processes = multiprocessing.cpu_count()
pool = multiprocessing.Pool(processes=num_processes)
results = pool.starmap(f_solve, args_set)
pool.close()
pool.join()
```

ПРИЛОЖЕНИЕ №3

```
import numpy as np
import multiprocessing
from scipy.optimize import fsolve


def f_solve(N, Q, q, delta, gamma, x_hat, A, B, K_Math,
    k_Math, k_0_Math, filename):
    def f_c_0(k, K):
        return -0.5 * delta * np.matmul(k, B) * (gamma +
            delta * np.matmul(np.matmul(np.transpose(B), K
            ), B)) ** -1

    def f_c(K):
        return -delta * np.matmul(np.matmul(np.transpose(
            B), K), A) / (
                gamma + delta * np.matmul(np.matmul(np.
                    transpose(B), K), B))

    def f_K(c, K):
        return Q + gamma * np.matmul(np.transpose(c), c)
            + delta * np.matmul(
              np.matmul(np.transpose(A + np.matmul(B, c)),
                K), (A + np.matmul(B, c)))

    def f_k(c, c_0, k, K):
        return q + 2 * gamma * c * c_0 + delta * np.
            matmul(k, (A + np.matmul(B, c))) + 2 * delta *
             c_0 * np.matmul(
              np.matmul(np.transpose(B), K), (A + np.matmul
                (B, c)))
```

```
def f_k_0(c_0, k, K, k_0):
    return N * x_hat ** 2 + gamma * c_0 ** 2 + delta
        * c_0 * np.matmul(k, B) + delta * c_0 ** 2 *
        np.matmul(
          np.matmul(np.transpose(B), K),
          B) + delta * k_0

def F(all_in):
    v_K = all_in[:N * N].reshape(N, N)
    v_k = all_in[N * N:N * N + N]
    v_k_0 = all_in[N * N + N]

    v_c_0 = f_c_0(v_k, v_K)
    v_c = f_c(v_K)

    diff_K = f_K(v_c, v_K) - v_K
    diff_k = f_k(v_c, v_c_0, v_k, v_K) - v_k
    diff_k_0 = f_k_0(v_c_0, v_k, v_K, v_k_0) - v_k_0
    all_in = np.concatenate((diff_K.flatten(), diff_k
        .flatten(), diff_k_0.flatten()))

    return all_in

KKK = np.concatenate((K_Math.flatten(), k_Math, [
    k_0_Math]))
sol = fsolve(F, KKK)
res_K = sol[:N * N].reshape(N, N)
res_k = sol[N * N:N * N + N]
res_k_0 = sol[N * N + N]

res_c = f_c(res_K)
res_c_0 = f_c_0(res_k, res_K)
```

```
    with open("n-star_sols\\" + filename, 'w') as filew:
        print(res_K, file=filew)
        print(is_positive_definite(res_K), file=filew)
        print(res_c, file=filew)
        print(res_c_0, file=filew)
        print(res_k, file=filew)
        print(res_k_0, file=filew)
    return sol


def is_positive_definite(matrix):
    eigenvalues, _ = np.linalg.eig(matrix)
    return np.all(eigenvalues > 0)


def float_range(start, stop, step):
    while start < stop:
        yield start
        start += step


def reading_Kkk(file_name):
    with open("2_agents_sols\\" + file_name) as file:
        data_Math = file.readlines()
    data_K = []
    data_K.append([float(num) for num in data_Math
       [0][2:-2].split()])
    data_K.append([float(num) for num in data_Math
       [1][2:-3].split()])
    K = np.array(data_K)
    k = np.array([float(num) for num in data_Math
       [-2][1:-2].split()])
    k_0 = float(data_Math[-1])
```

```
    return K, k, k_0


if __name__ == '__main__':
    with open("2_agents_param\\List_of_Tries.txt", 'r')
        as file:
        Tries_n = int(file.readline())
        Tries_delta = int(file.readline())
        Tries_gamma = int(file.readline())
        Tries_x_hat = int(file.readline())
        Tries_a = int(file.readline())

    with open('2_agents_param\\A_2_agents.txt', 'r') as
        file:
        array_string = file.readline()

    dataset = []

    args_set2 = []
    file_count = 0
    last_file = ''
    for n in range(3, Tries_n, 1):
        with open(f'n-star_param\\A_{n}_star.txt', 'r')
            as file:
            array_string = file.readline()
        args_set = []
        B = np.zeros((n, 1))
        B[0, 0] = 1
        q0 = np.ones(n)
        Q = np.diag(q0)
        a = np.zeros(n)
        for x_hat in float_range(- Tries_x_hat / 10, 0 +
            1 / 10, 1 / 10):
```

```
q = -2 * q0 * x_hat
for delta in float_range(0 + 1 / Tries_delta,
    1, 1 / Tries_delta):
     for gamma in float_range(0 + 1 / 10,
        Tries_gamma / 10, 1 / 10):
          for a[0] in float_range(0 + 1 / 10,
             Tries_a / 10, 1 / 10):
               for a1 in float_range(0 + 1 / 10,
                   Tries_a / 10, 1 / 10):
                    a[1:] = a1

                    A = np.zeros((n, n))
                    np.fill_diagonal(A, 1 - a1 /
                       2)
                    A[0, 0] = 1 - (n - 1) * a[0]
                       / n
                    A[0, 1:] = a[0] / n
                    A[1:, 0] = a1 / 2

                    filename = f"2ag_res{n}_{
                       round(delta, 1)}_{round(
                       gamma, 1)}_{round(x_hat, 
                       1)}_{round(a[0], 1)}_{
                       round(a[1], 1)}.txt"
                    try:
                         K_2ag_sol, k_2ag_sol,
                            k_0_2ag_sol =
                            reading_Kkk(filename)
                    except:
                         continue
                    last_file = filename
                    file_count += 1
```

```
                K_nstar_start = np.zeros((n,
                    n))
                K_nstar_start[0, 0] =
                    K_2ag_sol[0, 0]
                K_nstar_start[1:, 0] =
                    K_2ag_sol[1, 0] / (n - 1)
                K_nstar_start[0, 1:] =
                    K_2ag_sol[0, 1] / (n - 1)
                K_nstar_start[1:, 1:] =
                    K_2ag_sol[1, 1] / (n - 1)
                    / (n - 1)

                k_nstar_start = np.zeros(n)
                k_nstar_start[0] = k_2ag_sol
                    [0]
                k_nstar_start[1:] = k_2ag_sol
                    [1] / n

                k_0_nstar_start = k_0_2ag_sol

                args_set.append(
                    (n, Q, q, delta, gamma,
                        x_hat, A, B,
                        K_nstar_start,
                        k_nstar_start,
                        k_0_nstar_start,
                       filename))
print(f"datasets for {n} ready")
print(file_count, last_file)
num_processes = multiprocessing.cpu_count()
pool = multiprocessing.Pool(processes=
    num_processes)
results = pool.starmap(f_solve, args_set)
```

```
pool.close()
pool.join()
print(f"solving {n} ready")
```

ПРИЛОЖЕНИЕ №4

```
import numpy as np
import os
import re
from tqdm import tqdm


def reading_Kb_kk(file_name):
    with open(file_name) as file:
        data_Math = file.read()
    data_Math = re.sub(r'\s+]', ']', data_Math)
    data_Math = re.sub(r'\[\s+', '[', data_Math)
    start_K = data_Math.find('[[') + 2
    fin_K = data_Math.find(']]')
    start_c = data_Math.find('[[', fin_K) + 2
    fin_c = data_Math.find(']]', start_c)
    start_c_0 = data_Math.find('[[', fin_c) + 2
    fin_c_0 = data_Math.find(']]', start_c_0)
    start_k = data_Math.find('[', fin_c_0) + 1
    fin_k = data_Math.find(']', start_k)
    start_k_0 = fin_k + 2
    start_b = fin_K + 3
    fin_b = start_c

    rows = data_Math[start_K:fin_K].split(']\n [')

    data_K = []
    for i in rows:
        data_K.append([float(num) for num in re.split(r'
            [\s\n]+', i)])
    K = np.array(data_K)
```

```
    k = np.array([float(num) for num in re.split(r'[\s\n
        ]+', data_Math[start_k:fin_k])])
    c = np.array([float(num) for num in re.split(r'[\s\n
        ]+', data_Math[start_c:fin_c])])

    k_0 = float(data_Math[start_k_0:])
    c_0 = float(data_Math[start_c_0:fin_c_0])

    b = "True" in data_Math[start_b:fin_b]

    return K, b, c, c_0, k, k_0


count_correct = 0
count_1m = 0
count_2m = 0
count_3m = 0
count_4m = 0
count_5m = 0

count_1b = 0
count_2b = 0
count_3b = 0
count_4b = 0
count_5b = 0
for i in tqdm(os.listdir("n-star_sols\\")):
    if i == '2ag_res3_0.1_0.1_-0.1_0.1_0.5.txt':
        print('')
    try:
        K_star, b_star, c_star, c_0_star, k_star,
            k_0_star = reading_Kb_kk("n-star_sols\\" +
            i)
        K_2ag, b_2ag, c_2ag, c_0_2ag, k_2ag, k_0_2ag =
```

```
            reading_Kb  kk("2_agents_sols\\" + i)
        except:
            continue
        b1 = np.isclose(c_star[0] - c_2ag[0], 0, atol=1e-07)
        b2 = np.isclose(c_0_star - c_0_2ag, 0, atol=1e-07)
        b3 = np.isclose(sum(c_star[1:]) - c_2ag[1], 0, atol=1
           e-07)
        b4 = b_star
        b5 = b_2ag
        bs = int(b1) + b2 + b3 + b4 + b5
        count_1b += not b1
        count_2b += not b2
        if not b2:
            print(i)
        count_3b += not b3
        count_4b += not b4
        count_5b += not b5
        if bs == 5:
            count_correct += 1
        elif bs == 4:
            count_1m += 1
        elif bs == 3:
            count_2m += 1
        elif bs == 2:
            count_3m += 1
        elif bs == 1:
            count_4m += 1
        else:
            count_5m += 1
print("Without mistakes:", count_correct)
print("1 mistake:", count_1m)
print("2 mistakes:", count_2m)
print("3 mistakes:", count_3m)
```

```
print("4 mistakes:", count_4m)
print("5 mistakes:", count_5m)
print("-----------------------------------")
print("Types of errors that were found")
print("c_0 do not match:", count_2b)
print("c_1 do not match:", count_1b)
print("c do not correlate:", count_3b)
print("matrix K for 2ag is not positive definite:",
    count_4b)
print("matrix K for n-star is not positive definite:",
    count_5b)
```

ПРИЛОЖЕНИЕ №5

```
import numpy as np
import glob
import re
import itertools
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
from tqdm import tqdm


def f_solve(N, Q, q, delta, gamma, x_hat, A, B, K_Math,
    k_Math, k_0_Math, filename):
    def f_c_0(k, K):
        return -0.5 * delta * np.matmul(k, B) * (gamma +
            delta * np.matmul(np.matmul(np.transpose(B), K
            ), B)) ** -1

    def f_c(K):
        return -delta * np.matmul(np.matmul(np.transpose(
            B), K), A) / (
                gamma + delta * np.matmul(np.matmul(np.
                    transpose(B), K), B))

    def f_K(c, K):
        return Q + gamma * np.matmul(np.transpose(c), c)
            + delta * np.matmul(
              np.matmul(np.transpose(A + np.matmul(B, c)),
                K), (A + np.matmul(B, c)))

    def f_k(c, c_0, k, K):
        return q + 2 * gamma * c * c_0 + delta * np.
            matmul(k, (A + np.matmul(B, c))) + 2 * delta *
```

```
            c_0 * np.matmul(
             np.matmul(np.transpose(B), K), (A + np.matmul
                (B, c)))

    def f_k_0(c_0, k, K, k_0):
        return N * x_hat ** 2 + gamma * c_0 ** 2 + delta
            * c_0 * np.matmul(k, B) + delta * c_0 ** 2 *
            np.matmul(
              np.matmul(np.transpose(B), K),
              B) + delta * k_0

    def F(all_in):
        v_K = all_in[:N * N].reshape(N, N)
        v_k = all_in[N * N:N * N + N]
        v_k_0 = all_in[N * N + N]

        v_c_0 = f_c_0(v_k, v_K)
        v_c = f_c(v_K)

        diff_K = f_K(v_c, v_K) - v_K
        diff_k = f_k(v_c, v_c_0, v_k, v_K) - v_k
        diff_k_0 = f_k_0(v_c_0, v_k, v_K, v_k_0) - v_k_0
        all_in = np.concatenate((diff_K.flatten(), diff_k
            .flatten(), diff_k_0.flatten()))

        return all_in

    KKK = np.concatenate((K_Math.flatten(), k_Math, [
        k_0_Math]))
    sol = fsolve(F, KKK)
    return sol
```

```
def f_c_0_check(k, K):
    return -0.5 * delta * np.matmul(k, B) * (gamma +
        delta * np.matmul(np.matmul(np.transpose(B), K), B
        )) ** -1


def f_c_check(K):
    return -delta * np.matmul(np.matmul(np.transpose(B),
        K), A) / (
            gamma + delta * np.matmul(np.matmul(np.
                transpose(B), K), B))


def f_K_check(c, K):
    return Q + gamma * np.matmul(np.transpose(c), c) +
        delta * np.matmul(
          np.matmul(np.transpose(A + np.matmul(B, c)), K),
              (A + np.matmul(B, c)))


def f_k_check(c, c_0, k, K):
    return q + 2 * gamma * c * c_0 + delta * np.matmul(k,
         (A + np.matmul(B, c))) + 2 * delta * c_0 * np.
        matmul(
          np.matmul(np.transpose(B), K), (A + np.matmul(B,
              c)))


def f_k_0_check(c_0, k, K, k_0):
    return n * x_hat ** 2 + gamma * c_0 ** 2 + delta *
        c_0 * np.matmul(k, B) + delta * c_0 ** 2 * np.
        matmul(
          np.matmul(np.transpose(B), K),
```

```
        B) + delta * k_0


def F_check(all_in):
    v_K = all_in[:n * n].reshape(n, n)
    v_k = all_in[n * n:n * n + n]
    v_k_0 = all_in[n * n + n]

    v_c_0 = f_c_0_check(v_k, v_K)
    v_c = f_c_check(v_K)

    diff_K = f_K_check(v_c, v_K) - v_K
    diff_k = f_k_check(v_c, v_c_0, v_k, v_K) - v_k
    diff_k_0 = f_k_0_check(v_c_0, v_k, v_K, v_k_0) -
       v_k_0
    all_in = np.concatenate((diff_K.flatten(), diff_k.
       flatten(), diff_k_0.flatten()))
    return all_in


def is_positive_definite(matrix):
    eigenvalues, _ = np.linalg.eig(matrix)
    return np.all(eigenvalues > 0)


def float_range(start, stop, step):
    while start < stop:
        yield start
        start += step


def reading_Kb_kk(file_name):
    with open(file_name) as file:
```

```
    data_Math = file.read()
data_Math = re.sub(r'\s+]', ']', data_Math)
data_Math = re.sub(r'\[\s+', '[', data_Math)
start_K = data_Math.find('[[') + 2
fin_K = data_Math.find(']]')
start_c = data_Math.find('[[', fin_K) + 2
fin_c = data_Math.find(']]', start_c)
start_c_0 = data_Math.find('[[', fin_c) + 2
fin_c_0 = data_Math.find(']]', start_c_0)
start_k = data_Math.find('[', fin_c_0) + 1
fin_k = data_Math.find(']', start_k)
start_k_0 = fin_k + 2
start_b = fin_K + 3
fin_b = start_c

rows = data_Math[start_K:fin_K].split(']\n [')

data_K = []
for i in rows:
    data_K.append([float(num) for num in re.split(r'
        [\s\n]+', i)])
K = np.array(data_K)

k = np.array([float(num) for num in re.split(r'[\s\n
    ]+', data_Math[start_k:fin_k])])
c = np.array([float(num) for num in re.split(r'[\s\n
    ]+', data_Math[start_c:fin_c])])

k_0 = float(data_Math[start_k_0:])
c_0 = float(data_Math[start_c_0:fin_c_0])

b = bool(data_Math[start_b:fin_b])
```

```
    return K, b, c, c_0, k, k_0


def read_params(filename):
    if filename[-16] != '-':
        delta = float(filename[-23:-20])
        gamma = float(filename[-19:-16])
        x_hat = float(filename[-15:-12])
        a0 = float(filename[-11:-8])
        a1 = float(filename[-7:-4])
    else:
        delta = float(filename[-24:-21])
        gamma = float(filename[-20:-17])
        x_hat = float(filename[-16:-12])
        a0 = float(filename[-11:-8])
        a1 = float(filename[-7:-4])
    return delta, gamma, x_hat, a0, a1


count_progs = 0

t = 10
Cou = np.zeros((t, t))
n = 3
for filename in tqdm(glob.glob('./n-star_sols/2ag_res3*')
    ):
    count_progs+=1
    a = np.zeros(n)
    delta, gamma, x_hat, a[0], a[1:] = read_params(
        filename)
    K_simple, _, _, _, k_simple, k_0_simple =
        reading_Kb_kk(filename)
    B = np.zeros((n, 1))
```

```
        B[0, 0] = 1
        q0 = np.ones(n)
        Q = np.diag(q0)
        q = -2 * q0 * x_hat
        A = np.zeros((n, n))
        A[0, 0] = 1 - (n - 1) * a[0] / n
        A[0, 1:] = a[0] / n

        for a1, a2 in itertools.product(range(1, t+1), range
            (1, t+1)):
            A[1, 1] = 1 - a1 / 2 / 10
            A[1, 0] = a1 / 2 / 10
            A[2, 2] = 1 - a2 / 2 / 10
            A[2, 0] = a2 / 2 / 10
            sol = f_solve(n, Q, q, delta, gamma, x_hat, A, B,
                K_simple, k_simple, k_0_simple,
                            filename)
            res_K = sol[:n * n].reshape(n, n)
            Cou[a1 - 1, a2 - 1] += int(is_positive_definite(
                res_K) and all(
                np.isclose(F_check(sol), np.zeros(n * (n + 1)
                    + 1))))
print(count_progs)
plt.imshow(Cou, cmap='gray')
plt.grid(color='red', linewidth=1)
x_labels = np.arange(0, 11, 1)
y_labels = np.arange(0, 11, 1)
plt.xticks(np.arange(-0.5, 10, 1), x_labels)
plt.yticks(np.arange(-0.5, 10, 1), y_labels)
plt.colorbar()
plt.show()
```

ПРИЛОЖЕНИЕ №6

```
import numpy as np
import multiprocessing
import glob
import re
import itertools
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
from tqdm import tqdm


def f_solve(N, Q, q, delta, gamma, x_hat, A, B, K_Math,
  k_Math, k_0_Math, filename):
    def f_c_0(k, K):
        return -0.5 * delta * np.matmul(k, B) * (gamma +
          delta * np.matmul(np.matmul(np.transpose(B), K
          ), B)) ** -1

    def f_c(K):
        return -delta * np.matmul(np.matmul(np.transpose(
          B), K), A) / (
                gamma + delta * np.matmul(np.matmul(np.
                  transpose(B), K), B))

    def f_K(c, K):
        return Q + gamma * np.matmul(np.transpose(c), c)
          + delta * np.matmul(
            np.matmul(np.transpose(A + np.matmul(B, c)),
              K), (A + np.matmul(B, c)))

    def f_k(c, c_0, k, K):
        return q + 2 * gamma * c * c_0 + delta * np.
```

```
        matmul(k, (A + np.matmul(B, c))) + 2 * delta *
         c_0 * np.matmul(
          np.matmul(np.transpose(B), K), (A + np.matmul
             (B, c)))

    def f_k_0(c_0, k, K, k_0):
        return N * x_hat ** 2 + gamma * c_0 ** 2 + delta
           * c_0 * np.matmul(k, B) + delta * c_0 ** 2 *
           np.matmul(
             np.matmul(np.transpose(B), K),
             B) + delta * k_0

    def F(all_in):
        v_K = all_in[:N * N].reshape(N, N)
        v_k = all_in[N * N:N * N + N]
        v_k_0 = all_in[N * N + N]

        v_c_0 = f_c_0(v_k, v_K)
        v_c = f_c(v_K)

        diff_K = f_K(v_c, v_K) - v_K
        diff_k = f_k(v_c, v_c_0, v_k, v_K) - v_k
        diff_k_0 = f_k_0(v_c_0, v_k, v_K, v_k_0) - v_k_0
        all_in = np.concatenate((diff_K.flatten(), diff_k
           .flatten(), diff_k_0.flatten()))

        return all_in

    KKK = np.concatenate((K_Math.flatten(), k_Math, [
       k_0_Math]))
    sol = fsolve(F, KKK)
    return sol
```

```
def f_c_0_check(k, K):
    return -0.5 * delta * np.matmul(k, B) * (gamma +
        delta * np.matmul(np.matmul(np.transpose(B), K), B
        )) ** -1


def f_c_check(K):
    return -delta * np.matmul(np.matmul(np.transpose(B),
        K), A) / (
            gamma + delta * np.matmul(np.matmul(np.
                transpose(B), K), B))


def f_K_check(c, K):
    return Q + gamma * np.matmul(np.transpose(c), c) +
        delta * np.matmul(
          np.matmul(np.transpose(A + np.matmul(B, c)), K),
              (A + np.matmul(B, c)))


def f_k_check(c, c_0, k, K):
    return q + 2 * gamma * c * c_0 + delta * np.matmul(k,
         (A + np.matmul(B, c))) + 2 * delta * c_0 * np.
        matmul(
          np.matmul(np.transpose(B), K), (A + np.matmul(B,
              c)))


def f_k_0_check(c_0, k, K, k_0):
    return n * x_hat ** 2 + gamma * c_0 ** 2 + delta *
        c_0 * np.matmul(k, B) + delta * c_0 ** 2 * np.
        matmul(
```

```
            np.matmul(np.transpose(B), K),
            B) + delta * k_0


def F_check(all_in):
    v_K = all_in[:n * n].reshape(n, n)
    v_k = all_in[n * n:n * n + n]
    v_k_0 = all_in[n * n + n]

    v_c_0 = f_c_0_check(v_k, v_K)
    v_c = f_c_check(v_K)

    diff_K = f_K_check(v_c, v_K) - v_K
    diff_k = f_k_check(v_c, v_c_0, v_k, v_K) - v_k
    diff_k_0 = f_k_0_check(v_c_0, v_k, v_K, v_k_0) -
       v_k_0
    all_in = np.concatenate((diff_K.flatten(), diff_k.
       flatten(), diff_k_0.flatten()))
    return all_in


def is_positive_definite(matrix):
    eigenvalues, _ = np.linalg.eig(matrix)
    return np.all(eigenvalues > 0)


def float_range(start, stop, step):
    while start < stop:
        yield start
        start += step


def reading_Kb_kk(file_name):
```

```
with open(file_name) as file:
    data_Math = file.read()
data_Math = re.sub(r'\s+]', ']', data_Math)
data_Math = re.sub(r'\[\s+', '[', data_Math)
start_K = data_Math.find('[[') + 2
fin_K = data_Math.find(']]')
start_c = data_Math.find('[[', fin_K) + 2
fin_c = data_Math.find(']]', start_c)
start_c_0 = data_Math.find('[[', fin_c) + 2
fin_c_0 = data_Math.find(']]', start_c_0)
start_k = data_Math.find('[', fin_c_0) + 1
fin_k = data_Math.find(']', start_k)
start_k_0 = fin_k + 2
start_b = fin_K + 3
fin_b = start_c

rows = data_Math[start_K:fin_K].split(']\n [')

data_K = []
for i in rows:
    data_K.append([float(num) for num in re.split(r'
        [\s\n]+', i)])
K = np.array(data_K)

k = np.array([float(num) for num in re.split(r'[\s\n
    ]+', data_Math[start_k:fin_k])])
c = np.array([float(num) for num in re.split(r'[\s\n
    ]+', data_Math[start_c:fin_c])])

k_0 = float(data_Math[start_k_0:])
c_0 = float(data_Math[start_c_0:fin_c_0])

b = bool(data_Math[start_b:fin_b])
```

```
    return K, b, c, c_0, k, k_0


def read_params(filename):
    if filename[-16] != '-':
        delta = float(filename[-23:-20])
        gamma = float(filename[-19:-16])
        x_hat = float(filename[-15:-12])
        a0 = float(filename[-11:-8])
        a1 = float(filename[-7:-4])
    else:
        delta = float(filename[-24:-21])
        gamma = float(filename[-20:-17])
        x_hat = float(filename[-16:-12])
        a0 = float(filename[-11:-8])
        a1 = float(filename[-7:-4])
    return delta, gamma, x_hat, a0, a1


count_progs = 0
t = 10
Cou = np.zeros((t, t, t))
n = 4
for filename in tqdm(glob.glob('./n-star_sols/2ag_res4*')
    ):
    if count_progs == 5:
        break
    count_progs += 1
    a = np.zeros(n)
    delta, gamma, x_hat, a[0], a[1:] = read_params(
        filename)
    K_simple, _, _, _, k_simple, k_0_simple =
```

```
        reading_Kb_kk(filename)
    B = np.zeros((n, 1))
    B[0, 0] = 1
    q0 = np.ones(n)
    Q = np.diag(q0)
    q = -2 * q0 * x_hat
    A = np.zeros((n, n))
    A[0, 0] = 1 - (n - 1) * a[0] / n
    A[0, 1:] = a[0] / n

    for a1, a2, a3 in itertools.product(range(1, t + 1),
        range(1, t + 1), range(1, t + 1)):
        A[1, 1] = 1 - a1 / 2 / 10
        A[1, 0] = a1 / 2 / 10
        A[2, 2] = 1 - a2 / 2 / 10
        A[2, 0] = a2 / 2 / 10
        A[3, 3] = 1 - a3 / 2 / 10
        A[3, 0] = a3 / 2 / 10
        sol = f_solve(n, Q, q, delta, gamma, x_hat, A, B,
            K_simple, k_simple, k_0_simple,
                        filename)
        res_K = sol[:n * n].reshape(n, n)
        Cou[a1 - 1, a2 - 1, a3 - 1] += int(
            is_positive_definite(res_K) and all(
                np.isclose(F_check(sol), np.zeros(n * (n + 1)
                    + 1))))

print(count_progs)
fig, axes = plt.subplots(10, 1, figsize=(6, 20))
for i in range(10):
    axes[i].imshow(Cou[:, :, i], cmap='gray')
plt.show()
```