



**МИНОБРНАУКИ РОССИИ**  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**«МИРЭА - Российский технологический университет»**  
**РТУ МИРЭА**

---

**Институт Информационных Технологий**  
**Кафедра Вычислительной Техники**

**ПРАКТИЧЕСКАЯ РАБОТА №2**  
  
**по дисциплине**  
**«Разработка обеспечивающих подсистем систем поддержки**  
**принятия решений»**

Студент группы: ИКБО-04-22

Кликушин В.И.  
(Ф. И.О. студента)

Преподаватель

Гуличева А.А.  
(Ф.И.О. преподавателя)

Москва 2025

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1 ПОСТАНОВКА ЗАДАЧИ .....	5
2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ .....	6
2.1 Классификация марковских процессов .....	7
2.2 Марковские случайные процессы с дискретными состояниями и дискретным временем.....	7
2.3 Непрерывные цепи Маркова .....	9
3 ДОКУМЕНТАЦИЯ К ДАННЫМ.....	11
3.1 Описание предметной области .....	11
3.2 Анализ данных.....	11
3.3 Предобработка данных .....	15
4 ПРАКТИЧЕСКАЯ ЧАСТЬ .....	16
4.1 Функциональные возможности .....	16
4.2 Дискретная Марковская цепь .....	20
4.3 Непрерывная Марковская цепь .....	26
ЗАКЛЮЧЕНИЕ .....	31
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ .....	32
ПРИЛОЖЕНИЯ.....	33

# ВВЕДЕНИЕ

Современные стриминговые платформы и онлайн-кинотеатры ежедневно сталкиваются с проблемой эффективного управления вниманием пользователей в условиях практически неограниченного выбора контента. С расширением каталогов до десятков тысяч фильмов и сериалов традиционные методы ручного поиска становятся недостаточными для удовлетворения индивидуальных предпочтений пользователей. Проблема информационной перегрузки особенно остро стоит в сфере развлекательного контента, где пользователи тратят значительное время на поиск релевантных фильмов вместо непосредственного просмотра.

Актуальность разработки интеллектуальных рекомендательных систем подтверждается статистикой ведущих платформ: согласно исследованиям, более 80% просмотров на Netflix и 35% покупок на Amazon генерируются алгоритмами рекомендаций. Эффективные системы персонализации не только улучшают пользовательский опыт, но и напрямую влияют на ключевые бизнес-показатели, увеличивая удержание пользователей и монетизацию сервисов.

В настоящее время доминирующими подходами в рекомендательных системах являются методы коллаборативной фильтрации, основанные на поиске схожих пользователей (user-based) или объектов (item-based). Однако эти методы имеют фундаментальное ограничение — они не учитывают временную динамику предпочтений пользователей, рассматривая историю оценок как статичный набор данных. В реальности вкусы пользователей эволюционируют: переход от одних жанров к другим, сезонные предпочтения и изменение интересов со временем остаются за рамками традиционных подходов.

В данной работе предлагается использование марковских процессов для моделирования временной динамики пользовательских предпочтений. Марковские цепи позволяют учитывать не только что пользователь смотрел, но и в какой последовательности, а также временные интервалы между просмотрами. Такой подход особенно актуален для сервисов с большим объемом

временных меток, где можно выявить паттерны переходов между жанрами и темами.

Для экспериментальной проверки эффективности подхода выбран датасет MovieLens, содержащий реальные оценки пользователей с временными метками. В работе реализованы и сравниваются два типа марковских моделей: дискретные цепи Маркова для моделирования переходов между жанрами и непрерывные цепи Маркова для учета временных интервалов между оценками.

Структура работы включает: анализ предметной области и данных, теоретическое обоснование марковских процессов, практическую реализацию рекомендательной системы с поддержкой различных стратегий рекомендаций, а также сравнительную оценку эффективности предложенного подхода. Особое внимание уделено визуализации марковских цепей и анализу матриц переходов, что обеспечивает прозрачность и интерпретируемость работы алгоритмов.

Практическая значимость работы заключается в разработке прототипа системы, способной адаптироваться к изменяющимся предпочтениям пользователей и учитывать временные паттерны их поведения, что может быть использовано для улучшения рекомендаций в реальных стриминговых платформах.

# 1 ПОСТАНОВКА ЗАДАЧИ

Цель работы: приобрести навыки реализации модельных коллаборативных методов рекомендательных систем, основанных на поиске скрытых факторов методом Марковских цепей.

Задачи: создать программную реализацию модельного метода рекомендательной системы (РС), основанной на дискретной и непрерывной Марковских цепях, включающую актуальную предметную область для применения РС (вроде маркетплейса, медиа ресурсов, соц. сетей, экономической сферы и т.д.) и набор начальных данных для неё (опрос покупателей, статистика за временной период, готовые вероятностные данные и т.д.), матрицу переходных вероятностей дискретной и матрицу плотностей вероятностей непрерывной Марковской цепи, строящуюся автоматически после ввода начальных данных (либо вводимую в качестве начальных данных), вектор начальных состояний, вывод состояния системы на шаге  $n$  для дискретной цепи или в случае с непрерывной цепью в момент времени  $t$ , свойств системы на основе матрицы переходных вероятностей (переходы, вероятности переходов и стационарности), графическую модель цепи (draw.io, yEd, Paint) с определёнными и подписанными элементами графа состояний системы (с обозначением состояний и их переходов в соответствии с тем, как они называются и располагаются в программной реализации).

## 2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Марковские случайные процессы названы по имени выдающегося русского математика А. А. Маркова (1856–1922), впервые начавшего изучение вероятностной связи случайных величин и создавшего теорию, которую можно назвать «динамикой вероятностей». В дальнейшем основы этой теории явились исходной базой общей теории случайных процессов, а также таких важных прикладных наук, как теория диффузионных процессов, теория надежности, теория массового обслуживания и, соответственно, в рекомендательных системах. Для математического описания многих операций, развивающихся в форме случайного процесса, может быть с успехом применен математический аппарат, разработанный в теории вероятностей для Марковских случайных процессов.

Функция  $X(t)$  называется случайной, если ее значение при любом аргументе  $t$  является случайной величиной. Случайная функция  $X(t)$ , аргументом которой является время, называется случайным процессом.

Марковские процессы являются частным видом случайных процессов. Особое место марковских процессов среди других классов случайных процессов обусловлено следующими обстоятельствами: для марковских процессов хорошо разработан математический аппарат, позволяющий решать многие практические задачи; с помощью марковских процессов можно описать (точно или приближенно) поведение достаточно сложных систем.

Случайный процесс, протекающий в какой-либо системе  $S$ , называется марковским (или процессом без последствия), если он обладает следующим свойством: для любого момента времени  $t_0$  вероятность любого состояния системы в будущем (при  $t > t_0$ ) зависит только от ее состояния в настоящем (при  $t = t_0$ ) и не зависит от того, когда и каким образом система  $S$  пришла в это состояние. То есть в марковском случайном процессе будущее развитие процесса не зависит от его предыстории.

## 2.1 Классификация марковских процессов

Классификация марковских случайных процессов производится в зависимости от непрерывности или дискретности множества значений функции  $X(t)$  и параметра  $t$ . Различают следующие основные виды марковских случайных процессов.

- с дискретными состояниями и дискретным временем (цепь Маркова);
- с непрерывными состояниями и дискретным временем (марковские последовательности);
- с дискретными состояниями и непрерывным временем (непрерывная цепь Маркова);
- с непрерывным состоянием и непрерывным временем.

## 2.2 Марковские случайные процессы с дискретными состояниями и дискретным временем

Если множество состояний, в которых может находиться процесс счётное, то есть все возможные состояния могут быть пронумерованы, то соответствующий процесс называется случайным процессом с дискретными состояниями или просто дискретным случайным процессом.

Марковские процессы с дискретными состояниями удобно иллюстрировать с помощью так называемого графа состояний, где кружками обозначены состояния  $S_1, S_2, \dots, S_n$  системы  $S$ , а стрелками – возможные переходы из состояния в состояние.

На графе отмечают только непосредственные переходы, а не переходы через другие состояния. Возможные задержки в прежнем состоянии изображают «петлей», т. е. стрелкой, направленной из данного состояния в него же. Число состояний системы может быть, как конечным, так и бесконечным (но счетным).

Марковский случайный процесс с дискретными состояниями и дискретным временем называют марковской цепью. Для такого процесса

моменты  $t_1, t_2, \dots$ , когда система  $S$  может менять свое состояние, рассматривают как последовательные шаги процесса, а в качестве аргумента, от которого зависит процесс, выступает не время  $t$ , а номер шага  $1, 2, \dots, k, \dots$ . Случайный процесс в этом случае характеризуется последовательностью состояний  $S(0), S(1), S(2), \dots, S(k), \dots$ , где  $S(0)$  – начальное состояние системы (перед первым шагом);  $S(1)$  – состояние системы после первого шага;  $S(k)$  – состояние системы после  $k$ -го шага.

Вероятностями состояний цепи Маркова называются вероятности  $P_i(k)$  того, что после  $k$ -го шага (и до  $(k+1)$ -го) система  $S$  будет находиться в состоянии  $S_i (i = 1, 2, \dots, n)$ . Очевидно, для любого  $k$  выполняется условие, представленное в Формуле 2.1.

$$\sum_{i=1}^n P_i(k) = 1 \quad (2.1)$$

Начальным распределением вероятностей Марковской цепи называется распределение вероятностей состояний в начале процесса (Формула 2.2).

$$P_1(0), P_2(0), \dots, P_n(0) \quad (2.2)$$

В частном случае, если начальное состояние системы  $S$  в точности известно  $S(0) = S_i$ , то начальная вероятность  $P_i(0) = 1$ , а все остальные равны нулю.

Вероятностью перехода (переходной вероятностью) на  $k$ -м шаге из состояния  $S_i$  в состояние  $S_j$  называется условная вероятность того, что система  $S$  после  $k$ -го шага окажется в состоянии  $S_j$  при условии, что непосредственно перед этим (после  $k-1$  шага) она находилась в состоянии  $S_i$ .

Поскольку система может пребывать в одном из  $n$  состояний, то для каждого момента времени  $t$  необходимо задать  $n^2$  вероятностей перехода  $P_{ij}$ , которые удобно представить в виде матрицы (Формула 2.3).



$$\|P_{ij}\| = \begin{pmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & \dots & P_{2n} \\ \dots & \dots & \dots & \dots \\ P_{n1} & P_{n2} & \dots & P_{nn} \end{pmatrix} \quad (2.3)$$

где  $P_{ij}$  – вероятность перехода за один шаг из состояния  $S_i$  в состояние  $S_j$ ;

$P_{ii}$  – вероятность задержки системы в состоянии  $S_i$ .

Если переходные вероятности не зависят от номера шага (от времени), а зависят только от того, из какого состояния в какое осуществляется переход, то соответствующая цепь Маркова называется однородной.

## 2.3 Непрерывные цепи Маркова

Марковский случайный процесс с дискретными состояниями и непрерывным временем называется непрерывной цепью Маркова при условии, что переход системы из состояния в состояние происходит не в фиксированные, а в случайные моменты времени.

Пусть система характеризуется  $n$  состояниями  $S_1, S_2, \dots, S_n$ , а переход из состояния в состояние может осуществляться в любой момент времени. Обозначим через  $P_i(t)$  вероятность того, что в момент времени  $t$  система  $S$  будет находиться в состоянии  $S_i (i = 1, 2, \dots, n)$ . Требуется определить для любого  $t$  вероятности состояний  $P_1(t), P_2(t), \dots, P_n(t)$ . Очевидно, что имеет место нормировочное условие (Формула 2.4).

$$\sum_{i=1}^n P_i = 1 \quad (2.4)$$

Для процесса с непрерывным временем вместо переходных вероятностей  $P_{ij}$  рассматриваются плотности вероятностей перехода  $\lambda_{ij}$ , представляющие собой предел отношения вероятности перехода системы за время  $\Delta t$  из состояния  $S_i$  в состояние  $S_j$  к длине промежутка  $\Delta t$  (Формула 2.5).

$$\lambda_{ij} = \lim_{\Delta t \rightarrow 0} \frac{P_{ij}(t; \Delta t)}{\Delta t} \quad (2.5)$$

где  $P_{ij}(t; \Delta t)$  – вероятность того, что система, пребывавшая в момент  $t$  в состоянии  $S_i$ , за время  $\Delta t$  перейдет из него в состояние  $S_j$ .

Из определения плотностей вероятности перехода  $\lambda_{ij}$  видно, что они в общем случае зависят от времени  $t$ , неотрицательны и в отличие от вероятностей могут быть больше 1.

Если при любых  $i \neq j$  плотности вероятностей переходов не зависят от времени  $t$ , и тогда вместо  $\lambda_{ij}(t)$  будем писать просто  $\lambda_{ij}$ , то Марковский процесс с непрерывным временем называется однородным. Если же хотя бы при одной паре значений  $i \neq j$  плотность вероятности перехода  $\lambda_{ij}$  изменяется с течением времени  $t$ , процесс называется неоднородным. Таким образом, если  $\lambda_{ij} = \text{const}$ , то процесс называется однородным, если плотность вероятности зависит от времени  $\lambda_{ij} = \lambda_{ij}(t)$ , то процесс – неоднородный.

Вероятности состояний  $P_i(t); i = 1, \dots, n$  (неизвестные вероятностные функции) являются решением системы дифференциальных уравнений (Формула 2.6).

$$\frac{dP_i(t)}{dt} = -P_i(t) * \sum_{j=1}^n \lambda_{ij} + \sum_{j=1}^n P_{ij}(t) * \lambda_{ij} \quad (2.6)$$

Система представляет собой систему  $n$  обыкновенных линейных однородных дифференциальных уравнений первого порядка с постоянными коэффициентами. Эта система называется системой дифференциальных уравнений Колмогорова. Величина  $P_{ij}(t) * \lambda_{ij}$  называется потоком вероятности перехода из состояния  $S_i$  в  $S_j$ , причем интенсивность потоков  $\lambda_{ij}$  может зависеть от времени или быть постоянной.

## 3 ДОКУМЕНТАЦИЯ К ДАННЫМ

### 3.1 Описание предметной области

Данные взяты из открытого сервиса MovieLens, предоставляемого исследовательской группой GroupLens Университета Миннесоты. Датасет ml-latest-small содержит информацию о рейтингах и тегах пользователей для фильмов.

Целью использования этих данных является построение рекомендательной системы фильмов, которая может предсказывать оценки для фильмов, не просмотренных пользователем, и рекомендовать новые фильмы на основе исторических оценок и тегов.

Датасет состоит из следующих сущностей:

- пользователи (User): анонимизированные идентификаторы пользователей, которые выставляли оценки фильмов и добавляли теги;
- фильмы (Movie): идентификаторы, название, жанры, год выпуска и ссылки на внешние источники (IMDb, TMDb);
- рейтинги (Rating): пользователь, фильм, оценка от 0.5 до 5 и временная метка;
- теги (Tag): пользователь, фильм, текст тега и временная метка.

### 3.2 Анализ данных

Датасет ml-latest-small представляет собой компактную выборку данных MovieLens, содержащую оценки пользователей и информацию о фильмах. Он включает чуть более ста тысяч рейтингов, чуть меньше десяти тысяч фильмов и около шести сотен пользователей, что делает его удобным для экспериментов и обучения рекомендательных систем без необходимости использования мощных вычислительных ресурсов.

Каждая запись о рейтинге состоит из идентификатора пользователя, идентификатора фильма, выставленной оценки и временной метки. Основная информация о данных в файле ratings.csv представлена на Рисунке 3.2.1.

```
><class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   userId      100836 non-null  int64
1   movieId     100836 non-null  int64
2   rating      100836 non-null  float64
3   timestamp   100836 non-null  int64
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

Рисунок 3.2.1 - Основная информация о данных в файле ratings.csv

Распределение выставленных оценок представлено на Рисунке 3.2.2.

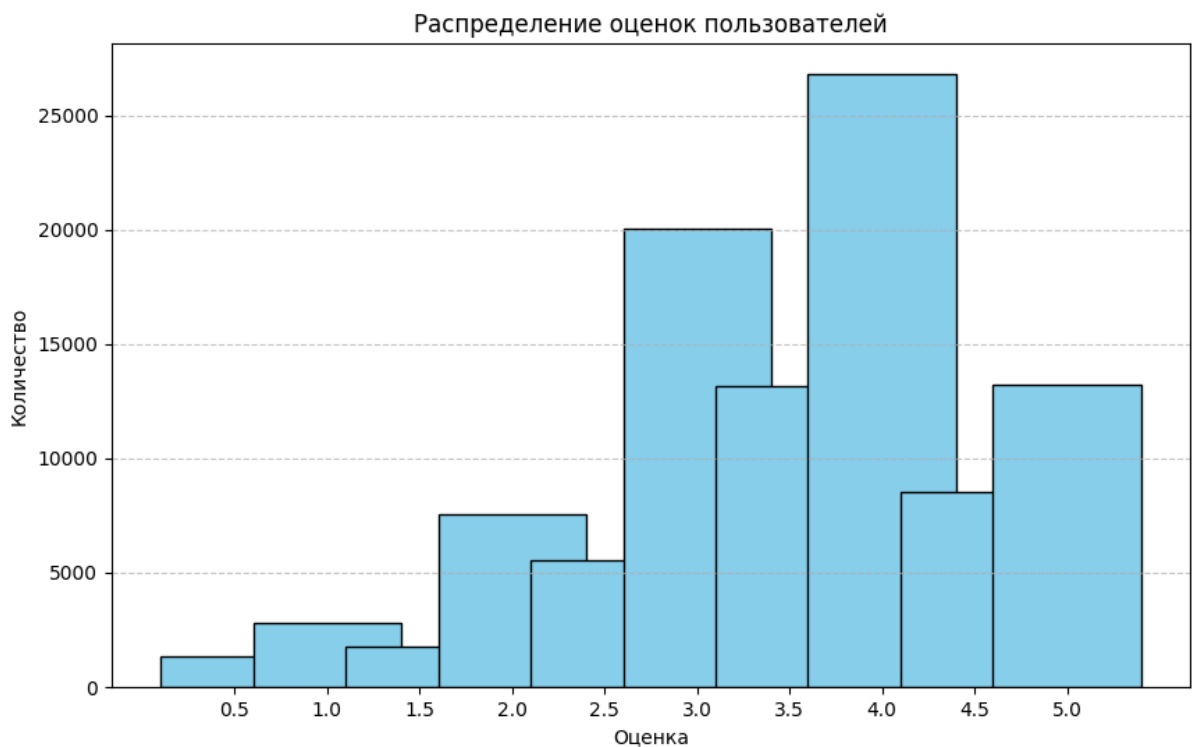


Рисунок 3.2.2 – Распределение оценок

Оценки распределены по шкале от 0.5 до 5 с шагом 0.5, что позволяет достаточно точно моделировать предпочтения пользователей, одновременно упрощая обработку данных. Анализ распределения оценок показывает, что наиболее популярные значения находятся в диапазоне от 3 до 4 баллов, что отражает склонность пользователей давать средние и положительные оценки.

Более экстремальные оценки, например 0.5 или 5, встречаются реже, но именно они могут быть особенно информативны при построении рекомендательных моделей, так как ясно отражают сильные предпочтения или отторжение.

Фильмы в датасете сопровождаются информацией о жанрах. Основная информация о данных в файле movies.csv отображена на Рисунке 3.2.3.

```
>>> movies
Out[1]: <class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
---  --
 0   movieId     9742 non-null   int64  
 1   title       9742 non-null   object  
 2   genres      9742 non-null   object  
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
```

Рисунок 3.2.3 - Основная информация о данных в файле movies.csv

Датафрейм не содержит нулевых значений или пропусков. Первые 5 строк датафрейма представлены на Рисунке 3.2.4.

	movieId	title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy

(.venv) PS C:\python\_projects\MIREA\Разработка обеспечивающих подсистем систем поддержки принят

Рисунок 3.2.4 – Первые пять строк датафрейма movies

Каждый фильм может относиться к нескольким жанрам, которые объединены в одну строку через символ |. Анализ жанров показывает, что большинство фильмов имеют два-три жанра, что отражает реальную практику киноиндустрии, где произведения редко ограничиваются одним жанром. Также есть небольшое количество фильмов без указанных жанров, что требует дополнительной предобработки для корректного включения таких данных в рекомендательную систему.

В названиях фильмов часто содержится год выпуска, заключённый в скобки. Это позволяет извлечь дополнительную информацию о времени выхода фильма и анализировать данные по временным периодам.

Теги, оставленные пользователями, представляют собой текстовые описания, которые дают дополнительные сведения о предпочтениях. Информация о данных в файле tags.csv представлена на Рисунке 3.2.5.

```
><class 'pandas.core.frame.DataFrame'>
RangeIndex: 3683 entries, 0 to 3682
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   userId      3683 non-null   int64
1   movieId     3683 non-null   int64
2   tag         3683 non-null   object
3   timestamp   3683 non-null   int64
dtypes: int64(3), object(1)
memory usage: 115.2+ KB
```

Рисунок 3.2.5 - Основная информация о данных в файле tags.csv

Анализ тегов показывает, что пользователи используют их довольно разнообразно: одни ограничиваются жанровыми метками, другие оставляют более субъективные описания вроде «funny» или «thriller». Теги помогают расширить рекомендации, особенно в гибридных системах, где учитываются как оценки, так и семантические характеристики фильмов.

Временные метки, присутствующие как в рейтингах, так и в тегах, позволяют анализировать динамику активности пользователей. С их помощью можно определить, когда пользователи наиболее активно выставляли оценки, выявить сезонные колебания интереса к фильмам или построить модели, учитывающие эволюцию предпочтений во времени.

Общий анализ показывает, что датасет ml-latest-small хорошо сбалансирован для экспериментов с рекомендательными системами. Он содержит достаточно данных для выявления закономерностей в поведении пользователей и предпочтениях фильмов, но при этом не требует огромных вычислительных ресурсов, что позволяет быстро проводить предобработку и обучение моделей.

### 3.3 Предобработка данных

В рамках предобработки удален год выпуска фильма из названия и добавлен отдельный столбец `year` в соответствующий датафрейм.

Временные метки приведены к типу данных `datetime` для более удобной обработки. Жанры преобразованы в список, обработаны фильмы с отсутствующими жанрами.

## 4 ПРАКТИЧЕСКАЯ ЧАСТЬ

### 4.1 Функциональные возможности

В практической части реализована интерактивная консольная система для работы с данными MovieLens. Основная цель заключалась в создании среды, позволяющей пользователю просматривать фильмы, искать их по названию и жанрам, оценивать и получать рекомендации на основе собственных оценок и оценок других пользователей. Полный код реализации кинотеатра с рекомендательной системой представлен в Приложении А.

Главное меню кинотеатра представлено на Рисунке 4.1.1.

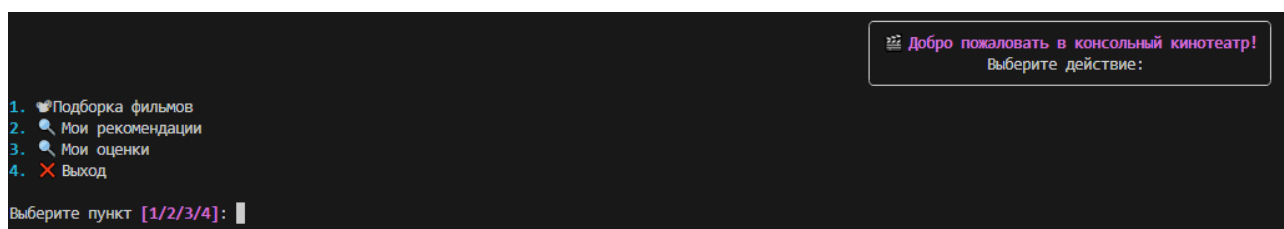


Рисунок 4.1.1 – Главное меню

Пользователю доступен просмотр каталога фильмов, ленты рекомендаций, ранее поставленных оценок, также реализована возможность выхода из приложения.

Каталог фильмов представлен на Рисунке 4.1.2.

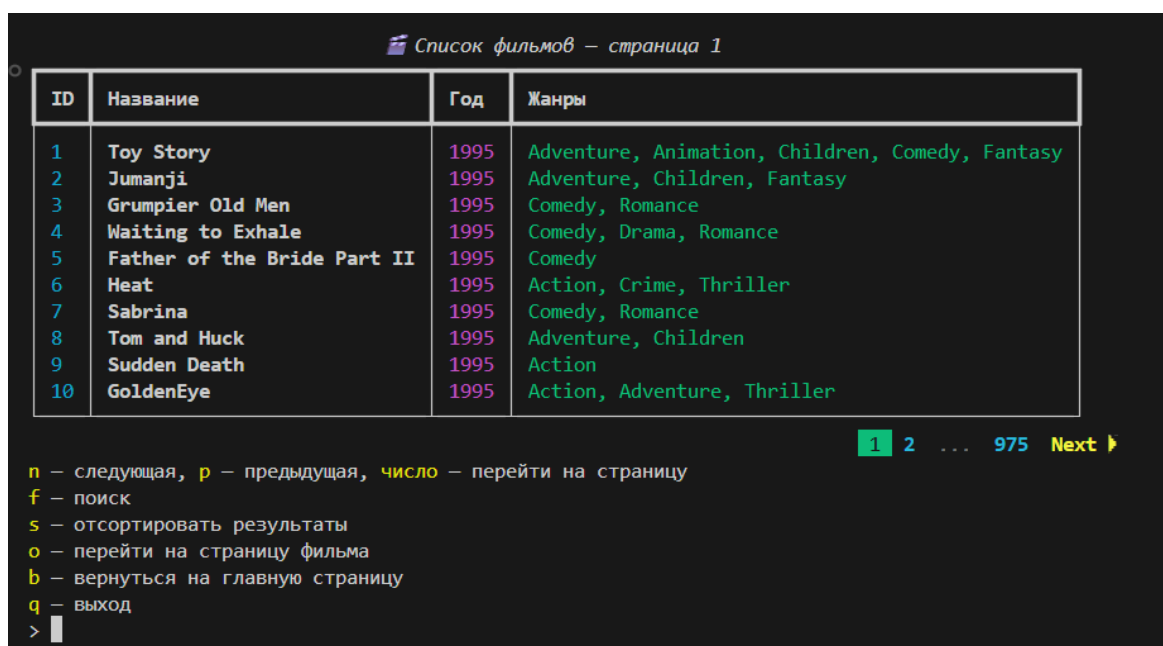


Рисунок 4.1.2 – Каталог фильмов



Сортировка каталога возможна по году выпуска фильм, по жанрам, по идентификатору. Пример сортировки каталога представлен на Рисунке 4.1.3.

Сортировка по Год (desc) – страница 1

ID	Название	Год	Жанры
183635	Maze Runner: The Death Cure	2018	Action, Mystery, Sci-Fi, Thriller
183959	Tom Segura: Disgraceful	2018	Comedy
185033	I Kill Giants	2018	Drama, Fantasy, Thriller
188797	Tag	2018	Comedy
185473	Blockers	2018	Comedy
185435	Game Over, Man!	2018	Action, Comedy
183611	Game Night	2018	Action, Comedy, Crime, Horror
185585	Pacific Rim: Uprising	2018	Action, Fantasy, Sci-Fi
184471	Tomb Raider	2018	Action, Adventure, Fantasy
183295	Insidious: The Last Key	2018	Horror, Mystery, Thriller

1 2 ... 975 Next ▶

n – следующая, p – предыдущая, число – перейти на страницу  
 o – открыть страницу фильма, b – назад  
 > █

Рисунок 4.1.3 – Сортировка каталога по году по убыванию

Добавлен механизм поиска по каталогу с фильтрацией по году выпуска, жанрам. Поиск осуществляется по названию путем проверки заданной подстроки. Интерфейс поиска представлен на Рисунке 4.1.4.

Введите название фильма (Enter – пропустить): alien

Выберите жанры

№	Жанр
1	Action
2	Adventure
3	Animation
4	Children's
5	Comedy
6	Crime
7	Documentary
8	Drama
9	Fantasy
10	Film-Noir
11	Horror
12	Musical
13	Mystery
14	Romance
15	Sci-Fi
16	Thriller
17	War
18	Western

Введите номера жанров через пробел (Enter – пропустить): █

Рисунок 4.1.4 – Интерфейс поиска

Результат поиска на запрос alien представлены на Рисунке 4.1.5.

Результаты поиска: 20

ID	Название	Год	Жанры
1200	Aliens	1986	Action, Adventure, Horror, Sci-Fi
1214	Alien	1979	Horror, Sci-Fi
1320	Alien³ (a.k.a. Alien 3)	1992	Action, Horror, Sci-Fi, Thriller
1690	Alien: Resurrection	1997	Action, Horror, Sci-Fi
3701	Alien Nation	1988	Crime, Drama, Sci-Fi, Thriller
4526	My Stepmother Is an Alien	1988	Comedy, Romance, Sci-Fi
5051	Italian for Beginners (Italiensk for begyndere)	2000	Comedy, Drama, Romance
5704	Without Warning (a.k.a. Alien Warning) (a.k.a. It Came Without Warning)	1980	Horror, Sci-Fi
6835	Alien Contamination	1980	Action, Horror, Sci-Fi
6899	Alien from L.A.	1988	Sci-Fi

1 2 Next ▶




Фильтры: Название: alien, Жанры: —, Год: —  
 n — следующая, p — предыдущая, число — перейти на страницу  
 s — отсортировать результаты  
 o — открыть страницу фильма по ID  
 r — сброс фильтров  
 b — назад  
 > |

Рисунок 4.1.5 – Результаты поиска на запрос alien

Для составлений рекомендаций пользователю необходимо поставить оценки фильмам. Для этого реализована возможность открытия «страницы» фильма, где доступен просмотр тэгов фильма и присутствует возможность оценить фильм.

Интерфейс страницы фильма представлен на Рисунке 4.1.6.

ID 1200

 Название: Aliens
  Год: 1986
  Жанры: ['Action', 'Adventure', 'Horror', 'Sci-Fi']

Теги

action, aliens, horror, sci-fi, space, space craft, SPACE TRAVEL, suspense, space

Доступные действия:

r — оценить фильм  
 b — назад  
 q — выйти из приложения  
 > |

Рисунок 4.1.6 – Интерфейс страницы фильма

Пример выставления оценки показан на Рисунке 4.1.7.

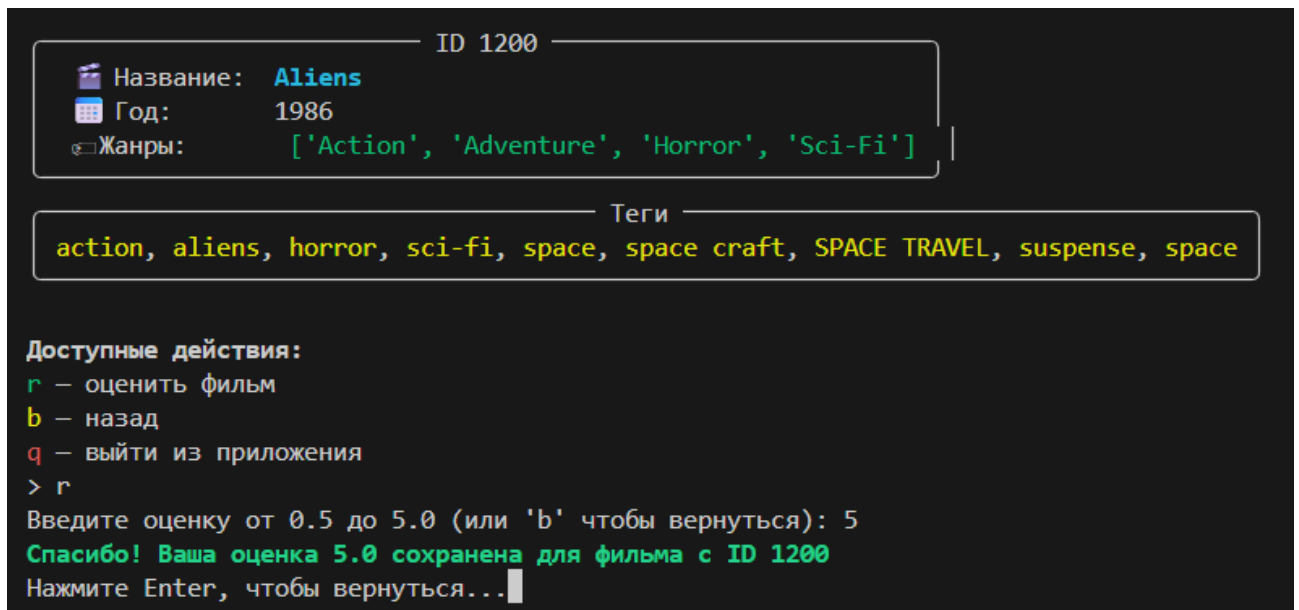


Рисунок 4.1.7 – Выставление оценки для фильма

Проставленные оценки можно посмотреть на отдельной странице в главном меню (Рисунок 4.1.8).

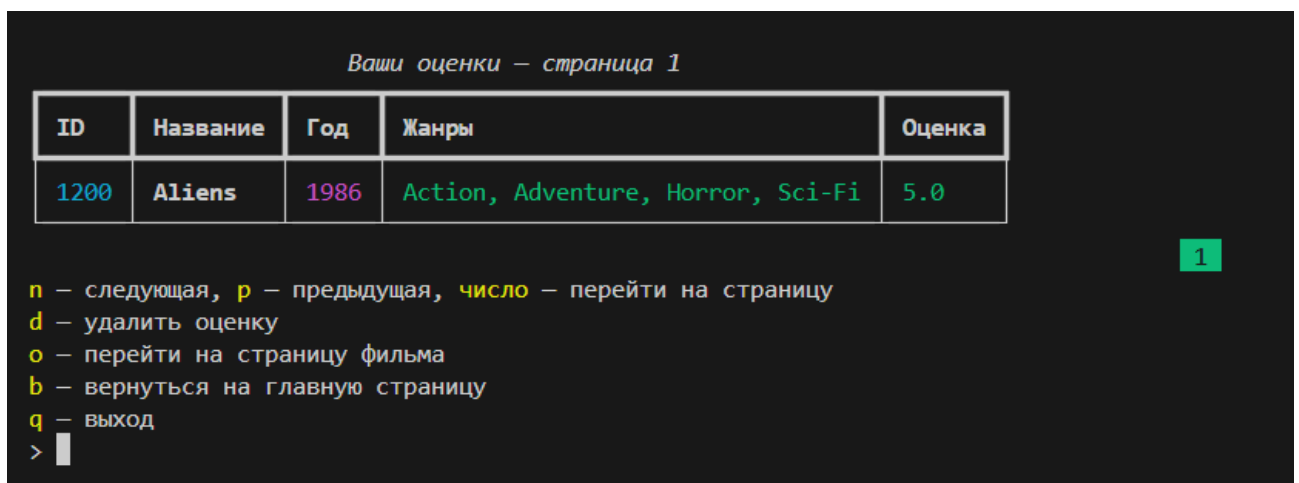


Рисунок 4.1.8 – Проставленные оценки

Добавлена возможность удалить оценку по идентификатору фильма (Рисунок 4.1.9).

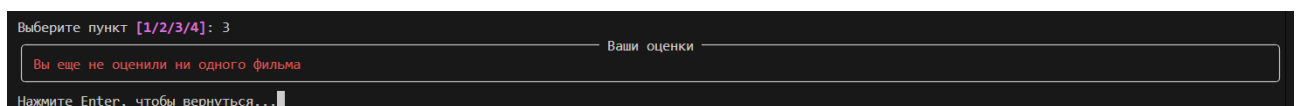


Рисунок 4.1.9 – Удаление оценки

Оценки на фильм с идентификатором 1200 успешно удалена, текущий пользователь не оценил ни одного фильма.

## 4.2 Дискретная Марковская цепь

В рамках практической части реализована рекомендательная система, основанная на дискретной цепи Маркова. Данный подход позволяет моделировать переходы пользователя между различными жанрами фильмов на основе истории его оценок и предсказывать наиболее вероятные жанры для будущих просмотров.

Дискретная цепь Маркова реализована в отдельном классе `MarkovChainRecommender`, представленном в файле `MarkovChain.py`. Содержание файла представлено в Приложении Б.

В контексте рекомендательной системы состояния цепи соответствуют жанрам фильмов. Каждый фильм может принадлежать к нескольким жанрам, и при формировании последовательности оценок пользователя формируется цепочка переходов между жанрами.

После выставления пользователем хотя бы двух оценок система строит матрицу переходов, где элемент  $P_{ij}$  отражает вероятность перехода из жанра  $i$  в жанр  $j$ . Общий алгоритм построения матрицы переходов:

1. Извлекаются все фильмы, оцененные пользователем, в хронологическом порядке.
2. Для каждой последовательной пары фильмов ( $i$ -й и  $i + 1$ -й) анализируются их жанры.
3. Если текущий фильм имеет несколько жанров, а следующий также имеет несколько жанров, вес перехода распределяется равномерно между всеми возможными переходами.

Переходы нормализуются так, чтобы сумма вероятностей по каждой строке была равна 1. Если для какого-то жанра нет исходящих переходов (например, пользователь оценил только один фильм этого жанра), для него устанавливается самопереход с вероятностью 1, что гарантирует корректность стохастической модели.

Рассмотрим процесс построения матрицы переходов на конкретном примере.

Предположим, что пользователь выставил оценки фильмам, представленным на Рисунке 4.2.1.

ID	Название	Год	Жанры	Оценка
5	Father of the Bride Part II	1995	Comedy	3.0
10	GoldenEye	1995	Action, Adventure, Thriller	4.0

n – следующая, p – предыдущая, число – перейти на страницу  
 d – удалить оценку  
 o – перейти на страницу фильма  
 b – вернуться на главную страницу  
 q – выход  
 >

Рисунок 4.2.1 – Оценки пользователя

Сформировано множество всех жанров, встречающихся в оценённых фильмах (Формула 4.1).

$$S = G_1 \cup G_2 = \{Comedy, Action, Adventure, Thriller\} \quad (4.1)$$

Количество состояний после оценки двух фильмов:  $|S| = 4$ .

Матрица переходов инициализирована нулями в начальный момент времени (Формула 4.2).

$$\|P_{ij}\| = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (4.2)$$

Считаем, что индексация жанров такова, что они упорядочены по алфавиту (*Action, Adventure, Comedy, Thriller*).

Каждый жанр из первого фильма порождает переход ко всем жанрам второго фильма с равной вероятностью, описанной Формулой 4.3.

$$P(next\_genre | current\_genre) = \frac{1}{|G_2|} \quad (4.3)$$

Второй фильм содержит три жанра одновременно, значит вес одного перехода равен  $1/3$ . Получены следующие вероятности переходов:

$$\begin{aligned} P(Comedy \rightarrow Action) &= \frac{1}{3} \\ P(Comedy \rightarrow Adventure) &= \frac{1}{3} \\ P(Comedy \rightarrow Thriller) &= \frac{1}{3} \end{aligned}$$

Все остальные элементы строки *Comedy* равны нулю, так как других переходов не было.

Далее выполняется нормализация каждой строки, чтобы сумма вероятностей по строке равнялась 1.

Для всех строк, где нет переходов (*Action, Adventure, Thriller*), устанавливается самопереход (Формула 4.4).

$$P_{ii} = 1 \quad (4.4)$$

Итоговая матрица переходов задана Формулой 4.5, где строки и столбцы соответствуют жанрам в алфавитном порядке.

$$\|P_{ij}\| = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.5)$$

Программная реализация позволяет посмотреть на матрицу переходов после каждого шага  $k$  (выставление новой оценки, удаление старых оценок). Построенная матрица переходов представлена на Рисунке 4.2.2.

*Матрица переходов между жанрами*

From/To	Action	Adventure	Comedy	Thriller
Action	1.000	0	0	0
Adventure	0	1.000	0	0
Comedy	0.333	0.333	0	0.333
Thriller	0	0	0	1.000

Нажмите Enter, чтобы продолжить...

Рисунок 4.2.2 – Построенная матрица переходов

Программная реализация подтвердила корректность ручного расчета для матрицы переходов.

При генерации рекомендаций дополнительно выводится информация о последнем оцененном фильме, а также процентные соотношения для переходов в различные состояния (Рисунок 4.2.3).

*Вероятности жанров (через 1 шаг(a))*

Жанр	Вероятность	Процент
Action	0.333	33.3%
Adventure	0.333	33.3%
Thriller	0.333	33.3%

Последний оцененный фильм: GoldenEye  
 Жанры: Action, Adventure, Thriller  
 Нажмите Enter, чтобы продолжить...

Рисунок 4.2.3 – Вероятности жанров

На рисунке 4.2.3 показаны вероятности переходов из жанров последнего оцененного фильма. Процентные значения рассчитываются на основе умножения начального вектора на матрицу переходов в степени  $k$  (количество шагов), что позволяет прогнозировать предпочтения пользователя на несколько шагов вперед.

Выбор последнего оцененного фильма в качестве начального вектора обусловлен фундаментальным свойством марковских процессов - отсутствием последствия. Согласно определению цепи Маркова (Раздел 2.2), будущее

состояние системы зависит только от текущего состояния и не зависит от предыстории. В контексте рекомендательной системы это означает, что следующий фильм, который захочет посмотреть пользователь, наиболее сильно зависит от его последних предпочтений, а не от всей истории просмотров. Таким образом, вектор начальных вероятностей  $p^0$  сосредотачивается на жанрах последнего оцененного фильма, что соответствует марковскому свойству и обеспечивает релевантность рекомендаций.

Рекомендованные фильмы представлены на Рисунке 4.2.4.

Рекомендации (Марковская цепь, 1 шаг(a)) – страница 1

ID	Название	Год	Жанры
480	Jurassic Park	1993	Action, Adventure, Sci-Fi, Thriller
3623	Mission: Impossible II	2000	Action, Adventure, Thriller
1370	Die Hard 2	1990	Action, Adventure, Thriller
2993	Thunderball	1965	Action, Adventure, Thriller
1544	Lost World: Jurassic Park, The	1997	Action, Adventure, Sci-Fi, Thriller
8972	National Treasure	2004	Action, Adventure, Drama, Mystery, Thriller
2414	Young Sherlock Holmes	1985	Action, Adventure, Children, Fantasy, Mystery, Thriller
8644	I, Robot	2004	Action, Adventure, Sci-Fi, Thriller
1374	Star Trek II: The Wrath of Khan	1982	Action, Adventure, Sci-Fi, Thriller
648	Mission: Impossible	1996	Action, Adventure, Mystery, Thriller

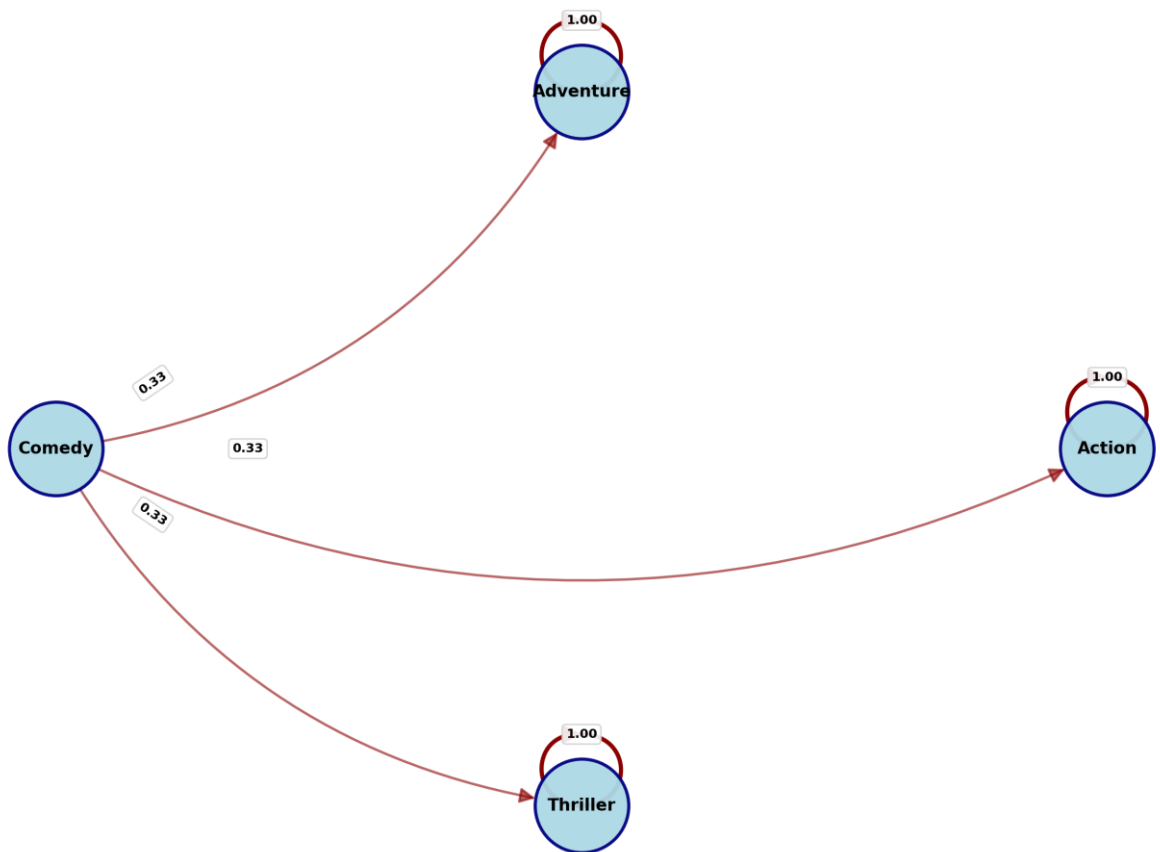
n – следующая, p – предыдущая, число – перейти на страницу  
o – открыть страницу фильма, b – назад  
> 1 2 Next ▶

Рисунок 4.2.4 – Рекомендованные фильмы

На первых страницах рекомендаций выводятся фильмы, содержащие все три целевых жанра (Action, Adventure, Thriller), что соответствует максимальной суммарной вероятности переходов из последнего оцененного фильма, а на последних страницах – фильмы, содержащие лишь один из жанров, что соответствует одному состоянию, в которое пользователь может перейти.

На рисунке 4.2.5 представлена графическая модель дискретной цепи Маркова, автоматически сгенерированная системой.





**Рисунок 4.2.5 - Графическая модель состояний цепи**

Граф визуализирует следующие элементы:

- узлы (круги): представляют состояния системы - жанры фильмов;
- ребра (стрелки): показывают возможные переходы между состояниями;
- веса переходов: числовые значения на стрелках отражают вероятности переходов между жанрами;
- петли: стрелки, направленные в тот же узел, обозначают вероятность остаться в текущем жанре.

В представленном графе четко видны рассчитанные ранее вероятности: из Comedy равновероятные переходы в Action, Adventure и Thriller (по 0.33), а для остальных жанров - самопереходы с вероятностью 1.0.

Практическая ценность данного подхода заключается в его способности улавливать динамику пользовательских предпочтений. В отличие от статических

методов, цепь Маркова учитывает временную последовательность оценок, позволяя системе адаптироваться к изменяющимся вкусам пользователя. Например, если пользователь начинает переходить от комедий к триллерам, система своевременно скорректирует рекомендации в соответствии с этой тенденцией.

### 4.3 Непрерывная Марковская цепь

Дополнительно реализована рекомендательная система, основанная на непрерывной цепи Маркова. Данный подход позволяет учитывать временные интервалы между оценками пользователя и моделировать эволюцию его предпочтений в непрерывном времени, что обеспечивает более точные и адаптивные рекомендации.

Непрерывная цепь Маркова реализована в отдельном классе `ContinuousMarkovRecommender`, представленном в файле `ContinuousMarkovChain.py` (Приложение В).

Алгоритм построения матрицы интенсивностей в реализации включает следующие шаги:

1. Формирование хронологии оценок: оценки пользователя упорядочиваются по времени, при этом для оценок без временных меток генерируются реалистичные случайные интервалы.
2. Определение множества состояний: как и в дискретном случае, состояния соответствуют жанрам фильмов.
3. Расчет времени пребывания в состояниях: для каждого жанра вычисляется суммарное время, которое пользователь провел в этом состоянии.
4. Подсчет переходов между состояниями: анализируются последовательные пары оцененных фильмов и подсчитываются переходы между их жанрами.
5. Вычисление интенсивностей переходов: интенсивность перехода из

состояния  $S_i$  в состояние  $S_j$  вычисляется по Формуле 4.6.

$$\lambda_{ij} = \frac{N_{ij}}{T_i} \quad (4.6)$$

где  $N_{ij}$  – количество переходов из  $S_i$  в состояние  $S_j$ ;

$T_i$  – общее время пребывания в состоянии  $S_i$ .

Рекомендации формируются на основе вычисленных вероятностей жанров с дополнительным учетом временного фактора. Алгоритм включает:

1. Вычисление вероятностей жанров для заданного времени  $t$ .
2. Оценка релевантности каждого неоцененного фильма на основе совпадения жанров.
3. Учет временного фактора: для больших  $t$  увеличивается вес разнообразия жанров.
4. Ранжирование фильмов по итоговой оценке релевантности.

На Рисунке 4.3.1 представлены начальные оценки пользователя, на основе которых строится матрица интенсивностей.

Ваши оценки – страница 1

ID	Название	Год	Жанры	Оценка
5	Father of the Bride Part II	1995	Comedy	3.0
10	GoldenEye	1995	Action, Adventure, Thriller	4.0
1200	Aliens	1986	Action, Adventure, Horror, Sci-Fi	5.0

n – следующая, p – предыдущая, число – перейти на страницу  
 d – удалить оценку  
 o – перейти на страницу фильма  
 b – вернуться на главную страницу  
 q – выход  
 >

Рисунок 4.3.1 – Начальные оценки пользователя

Каждая оценка сопровождается временной меткой, что позволяет учитывать временные интервалы между просмотрами.

Реализована возможность задания времени для прогнозирования (в днях) и количества выдаваемых рекомендаций, как показано на Рисунке 4.3.2.

```
Выберите стратегию рекомендаций [user-based/item-based/markov-discrete/markov-continuous] (user-based): markov-continuous
Введите время для прогноза (в днях) (1.0): 0.5
Количество рекомендаций (20): 200

Хронология ваших оценок:
1. Father of the Bride Part II - оценка 3.0 - 2025-10-24 13:51
2. GoldenEye - оценка 4.0 - 2025-10-24 19:36
3. Aliens - оценка 5.0 - 2025-10-27 08:54
Показать матрицу интенсивностей? [y/n]:
```

Рисунок 4.3.2 – Интерфейс настройки рекомендаций непрерывной цепью Маркова

Пользователь может задать параметр времени  $t$  (в днях) для прогнозирования и количество рекомендаций. Система также отображает хронологию оценок пользователя со сгенерированными временными метками.

Построенная матрица интенсивностей переходов представлена на Рисунке 4.3.3.



Рисунок 4.3.3 – Матрица интенсивности переходов

Значения на диагонали (отрицательные) представляют общую интенсивность выхода из состояния, а недиагональные элементы показывают интенсивности переходов между различными жанрами. Более высокие значения соответствуют более вероятным переходам.

Вероятности жанров для времени  $t=0.5$  дней отображены на Рисунке 4.3.4.

Вероятности жанров (непрерывная цепь,  $t=0.5$ )

Жанр	Вероятность	Процент
Action	0.246	24.6%
Adventure	0.246	24.6%
Horror	0.200	20.0%
Sci-Fi	0.200	20.0%
Comedy	0.055	5.5%
Thriller	0.051	5.1%

Последний оцененный фильм: Aliens  
 Жанры: Action, Adventure, Horror, Sci-Fi  
 Нажмите Enter, чтобы продолжить...

Рисунок 4.3.4 – Вероятности жанров

Таблица показывает распределение вероятностей по жанрам через 0.5 дней после последней оценки. Жанры с наибольшими вероятностями будут преобладать в рекомендациях.

Первая страница рекомендаций для параметра  $t=0.5$  представлена на Рисунке 4.3.5.

Рекомендации (Непрерывная цепь Маркова,  $t=0.5$  дней) – страница 1

ID	Название	Год	Жанры
60471	Rogue	2007	Action, Adventure, Horror, Sci-Fi, Thriller
36509	Cave, The	2005	Action, Adventure, Horror, Mystery, Sci-Fi, Thriller
161918	Sharknado 4: The 4th Awakens	2016	Action, Adventure, Horror, Sci-Fi
610	Heavy Metal	1981	Action, Adventure, Animation, Horror, Sci-Fi
27032	Who Am I? (Wo shi shei)	1998	Action, Adventure, Comedy, Sci-Fi, Thriller
164226	Maximum Ride	2016	Action, Adventure, Comedy, Fantasy, Sci-Fi, Thriller
72165	Cirque du Freak: The Vampire's Assistant	2009	Action, Adventure, Comedy, Fantasy, Horror, Thriller
2617	Mummy, The	1999	Action, Adventure, Comedy, Fantasy, Horror, Thriller
3827	Space Cowboys	2000	Action, Adventure, Comedy, Sci-Fi
8633	Last Starfighter, The	1984	Action, Adventure, Comedy, Sci-Fi

n – следующая, p – предыдущая, число – перейти на страницу  
 o – открыть страницу фильма, b – назад  
 >

Рисунок 4.3.5 – Первая страница рекомендаций с параметром  $t=0.5$

Рекомендации сформированы на основе вероятностей жанров для короткого временного горизонта. Фильмы содержат жанры с наибольшими вероятностями, что обеспечивает релевантность рекомендаций текущим предпочтениям пользователя.

При увеличении временного горизонта до  $t=10$  дней распределение вероятностей жанров изменяется, как показано на Рисунке 4.3.6.

Вероятности жанров (непрерывная цепь,  $t=10.0$ )

Жанр	Вероятность	Процент
Comedy	0.403	40.3%
Thriller	0.198	19.8%
Adventure	0.153	15.3%
Action	0.153	15.3%
Horror	0.046	4.6%
Sci-Fi	0.046	4.6%

Последний оцененный фильм: Aliens  
 Жанры: Action, Adventure, Horror, Sci-Fi  
 Нажмите Enter, чтобы продолжить...

Рисунок 4.3.6 - Вероятности появления жанров для  $t=10$  дней

Первая страница рекомендаций для параметра  $t=10$  представлена на Рисунке 4.3.7.

Рекомендации (Непрерывная цепь Маркова,  $t=10.0$  дней) – страница 1

ID	Название	Год	Жанры
188797	Tag	2018	Comedy
185031	Alpha	2018	Adventure, Thriller
185435	Game Over, Man!	2018	Action, Comedy
185473	Blockers	2018	Comedy
190183	The Darkest Minds	2018	Sci-Fi, Thriller
187593	Deadpool 2	2018	Action, Comedy, Sci-Fi
90890	Jack and Jill	2011	Comedy
189333	Mission: Impossible - Fallout	2018	Action, Adventure, Thriller
189547	Iron Soldier	2010	Action, Sci-Fi
91485	Expendables 2, The	2012	Action, Adventure

n – следующая, p – предыдущая, число – перейти на страницу  
 o – открыть страницу фильма, b – назад

Рисунок 4.3.7 - Первая страница рекомендаций с параметром  $t=10$

Рекомендации для большего временного горизонта включают более разнообразные жанры, что соответствует увеличенной неопределенности в предпочтениях пользователя в отдаленной перспективе.

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной практической работы успешно достигнута поставленная цель: разработана и реализована рекомендательная система, основанная на методах марковских цепей, позволяющая моделировать динамику пользовательских предпочтений во времени. Все поставленные задачи выполнены в полном объеме.

Разработана функциональная консольная система на языке Python, интегрирующая интерактивный кинотеатр на основе датасета MovieLens и две ключевые стратегии рекомендаций на основе марковских процессов: дискретные цепи Маркова для моделирования переходов между жанрами и непрерывные цепи Маркова для учета временных интервалов между оценками. Система обеспечивает автоматическое построение матриц переходов и интенсивностей на основе пользовательских оценок, а также визуализацию графов состояний.

Проведенное исследование позволило выявить следующие ключевые особенности реализованных подходов.

Дискретные цепи Маркова показали высокую эффективность для моделирования непосредственных переходов между жанрами, обеспечивая релевантные рекомендации на основе последних оценок пользователя. Автоматически построенная матрица переходов точно отражает вероятности смены жанровых предпочтений, что подтверждается визуализацией графа состояний.

Непрерывные цепи Маркова продемонстрировали преимущество в задачах долгосрочного прогнозирования предпочтений, учитывая временные интервалы между оценками. Возможность задания временного горизонта прогнозирования позволяет системе адаптироваться к изменяющимся темпам потребления контента пользователем.

## СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин, А. Б. Безусловная оптимизация. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк — М. РТУ МИРЭА , 2020.
2. Сорокин, А. Б. Введение в генетические алгоритмы: теория, расчеты и приложения. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин — М. МИРЭА , 2018.
3. Рекомендательные системы: user-based и item-based [Электронный ресурс]: Habr. URL: <https://habr.com/ru/companies/surfingbird/articles/139518/> (Дата обращения: 24.10.2025).
4. Рекомендательные системы [Электронный ресурс]: URL: [https://neerc.ifmo.ru/wiki/index.php?title=Рекомендательные\\_системы](https://neerc.ifmo.ru/wiki/index.php?title=Рекомендательные_системы) (Дата обращения: 22.10.2025).
5. MovieLens [Электронный ресурс]: URL: <https://grouplens.org/datasets/movielens/> (Дата обращения: 13.10.2025).
6. Краткое введение в цепи Маркова [Электронный ресурс]: URL: <https://habr.com/ru/articles/455762/> (Дата обращения: 24.10.2025).
7. Марковская цепь [Электронный ресурс]: URL: [https://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%80%D0%BA%D0%BE%D0%B2%D1%81%D0%BA%D0%B0%D1%8F\\_%D1%86%D0%B5%D0%BF%D1%8C#.D0.A0.D0.B0.D1.81.D0.BF.D1.80.D0.B5.D0.B4.D0.B5.D0.BB.D0.B5.D0.BD.D0.B8.D0.B5\\_.D0.B2.D0.B5.D1.80.D0.BE.D1.8F.D1.82.D0.BD.D0.BE.D1.81.D1.82.D0.B5.D0.B9](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%80%D0%BA%D0%BE%D0%B2%D1%81%D0%BA%D0%B0%D1%8F_%D1%86%D0%B5%D0%BF%D1%8C#.D0.A0.D0.B0.D1.81.D0.BF.D1.80.D0.B5.D0.B4.D0.B5.D0.BB.D0.B5.D0.BD.D0.B8.D0.B5_.D0.B2.D0.B5.D1.80.D0.BE.D1.8F.D1.82.D0.BD.D0.BE.D1.81.D1.82.D0.B5.D0.B9) (Дата обращения: 25.10.2025).



## ПРИЛОЖЕНИЯ

Приложение А — Реализация кинотеатра с встроенной рекомендательной системой, основанной на поиске скрытых факторов методом Марковских цепей.

Приложение Б — Код файла `MarkovChain.py`.

Приложение В — Код файла `ContinuousMarkovChain.py`.

## Приложение А

### Реализация кинотеатра с встроенной рекомендательной системой, основанной на поиске скрытых факторов методом Марковских цепей

*Листинг А – Реализация кинотеатра с встроенной рекомендательной системой, основанной на поиске скрытых факторов методом Марковских цепей*

```
import os
import re
import pandas as pd
import numpy as np
from rich.console import Console
from rich.table import Table
from rich.prompt import Confirm, Prompt
from rich.panel import Panel
from rich.align import Align
from rich.text import Text
from enum import Enum
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

from MarkovChain import MarkovChainRecommender
from ContinuousMarkovChain import ContinuousMarkovRecommender

def extract_year(title):
    title = title.strip()
    match = re.search(r"\((\d{4})\)\\s*$", title)
    if match:
        return int(match.group(1))
    return np.nan

class Genres(Enum):
    Action = "Action"
    Adventure = "Adventure"
    Animation = "Animation"
    Children = "Children's"
    Comedy = "Comedy"
    Crime = "Crime"
    Documentary = "Documentary"
    Drama = "Drama"
    Fantasy = "Fantasy"
    FilmNoir = "Film-Noir"
    Horror = "Horror"
    Musical = "Musical"
    Mystery = "Mystery"
    Romance = "Romance"
    SciFi = "Sci-Fi"
    Thriller = "Thriller"
    War = "War"
    Western = "Western"

class MovieLensCinema:
    def __init__(self, path, per_page=10, enable_markov_visualization=True):
        self.path = path
        self.links = None
        self.movies = None
        self.ratings = None
        self.tags = None
```

```
        self.console = Console()
        self.rates = pd.DataFrame(columns=["movieId",
"rating"]).set_index("movieId")
        self.load_data()

        self.per_page = per_page
        self.markov_recommender = MarkovChainRecommender(
            self, auto_visualize=enable_markov_visualization
        )
        self.continuous_markov_recommender = ContinuousMarkovRecommender(
            self, auto_visualize=enable_markov_visualization
        )

    def load_data(self):
        links_path = os.path.join(self.path, "links.csv")
        movies_path = os.path.join(self.path, "movies.csv")
        ratings_path = os.path.join(self.path, "ratings.csv")
        tags_path = os.path.join(self.path, "tags.csv")

        if os.path.exists(links_path):
            self.links = pd.read_csv(
                links_path,
                encoding="utf-8",
                index_col="movieId",
                dtype={"imdbId": "int64", "tmdbId": "Int64"},
            )
        else:
            raise FileNotFoundError("Не найдено файла links.csv")

        if os.path.exists(movies_path):
            self.movies = pd.read_csv(
                movies_path, encoding="utf-8", index_col="movieId",
quotechar='"'
            )
            self.movies["title"] = self.movies["title"].str.strip()
            self.movies["genres"] = self.movies["genres"].apply(
                lambda x: [] if x == "(no genres listed)" else x.split("|")
            )
            self.movies["year"] = self.movies["title"].apply(extract_year)
            self.movies["year"] = self.movies["year"].astype("Int64")
            self.movies["title"] = self.movies["title"].apply(
                lambda t: re.sub(r"\s*(\d{4})$", "", t)
            )
        else:
            raise FileNotFoundError("Не найдено файла movies.csv")

        if os.path.exists(ratings_path):
            self.ratings = pd.read_csv(
                ratings_path,
                encoding="utf-8",
                dtype={
                    "userId": "int32",
                    "movieId": "int32",
                    "rating": "float32",
                    "timestamp": "int64",
                },
            )
            self.ratings["datetime"] = pd.to_datetime(
                self.ratings["timestamp"], unit="s"
            )
            self.ratings = self.ratings.drop(columns=["timestamp"])
```

```

        self.ratings.set_index(["userId", "movieId"], inplace=True)

    else:
        raise FileNotFoundError("Не найдено файла ratings.csv")

    if os.path.exists(tags_path):
        self.tags = pd.read_csv(
            tags_path,
            dtype={
                "userId": "int32",
                "movieId": "int32",
                "tag": "string",
                "timestamp": "int64",
            },
            quotechar='"',
        )
        self.tags["datetime"] = pd.to_datetime(self.tags["timestamp"],
unit="s")
        self.tags.set_index(["userId", "movieId"], inplace=True)
    else:
        raise FileNotFoundError("Не найдено файла tags.csv")

    def get_rating_matrix(self):
        df = self.ratings.reset_index()

        rating_matrix = df.pivot_table(
            index="userId", columns="movieId", values="rating"
        )

        return rating_matrix

    def show_movies(self, page=1):
        start = (page - 1) * self.per_page
        end = start + self.per_page
        subset = self.movies.iloc[start:end]

        table = Table(title=f"🎬 Список фильмов — страница {page}")

        table.add_column("ID", style="cyan", no_wrap=True)
        table.add_column("Название", style="bold")
        table.add_column("Год", justify="center", style="magenta")
        table.add_column("Жанры", style="green")

        for movie_id, row in subset.iterrows():
            genres_str = ", ".join(row["genres"]) if row["genres"] else "-"
            year_str = str(row["year"]) if not pd.isna(row["year"]) else "-"
            table.add_row(str(movie_id), row["title"], year_str, genres_str)

        self.console.print(table)

    def run(self):
        page = 1
        total_pages = (len(self.movies) - 1) // self.per_page + 1

        while True:
            os.system("cls" if os.name == "nt" else "clear")
            self.show_movies(page=page)

            self.show_paginator(page, total_pages)

            self.console.print(

```

### Продолжение Листинга А

```
        "[yellow]n[/yellow] - следующая, [yellow]p[/yellow] -
предыдущая, "
        "[yellow]число[/yellow] - перейти на страницу\n"
        "[yellow]f[/yellow] - поиск\n"
        "[yellow]s[/yellow] - отсортировать результаты\n"
        "[yellow]o[/yellow] - перейти на страницу фильма\n"
        "[yellow]b[/yellow] - вернуться на главную страницу\n"
        "[yellow]q[/yellow] - выход"
    )

    choice = input("> ").strip().lower()
    if choice == "n" and page < total_pages:
        page += 1
    elif choice == "p" and page > 1:
        page -= 1
    elif choice.isdigit():
        num = int(choice)
        if 1 <= num <= total_pages:
            page = num
    elif choice == "f":
        self.search_movies()
    elif choice == "s":
        self.sort_movies(self.movies)
    elif choice == "o":
        movie_id_str = Prompt.ask("Введите ID фильма (или 'b' для
отмены)")
        if movie_id_str.lower() == "b":
            continue
        if movie_id_str.isdigit():
            self.show_movie_page(int(movie_id_str))
        else:
            self.console.print("[red]Некорректный ID[/red]")
            input("Нажмите Enter, чтобы вернуться...")
    elif choice == "b":
        break
    elif choice == "q":
        if Confirm.ask("Are you sure?"):
            exit(0)

    def menu(self):
        while True:
            os.system("cls" if os.name == "nt" else "clear")

            self.console.print(
                Panel.fit(
                    "[bold magenta]🎬 Добро пожаловать в консольный
кинотеатр![/bold magenta]\nВыберите действие:"
                ),
                justify="center",
            )

            self.console.print("[cyan]1.[/cyan] 🎞 Подборка фильмов")
            self.console.print("[cyan]2.[/cyan] 🔍 Мои рекомендации")
            self.console.print("[cyan]3.[/cyan] 🔍 Мои оценки")
            self.console.print("[cyan]4.[/cyan] ❌ Выход")

            choice = Prompt.ask("\nВыберите пункт", choices=["1", "2", "3",
"4"])

            if choice == "1":
```

```

        self.run()
    elif choice == "2":
        self.show_recommendations()
    elif choice == "3":
        self.show_my_ratings()
    elif choice == "4":
        if Confirm.ask("[bold red]Вы действительно хотите выйти?[/bold
red]"):
            break

def show_movie_page(self, movie_id: int):
    """Страница фильма с информацией, тегами и возможностью оценки"""
    if movie_id not in self.movies.index:
        self.console.print(f"[red]Фильм с ID {movie_id} не найден.[/red]")
        input("Нажмите Enter, чтобы вернуться...")
        return

    while True:
        os.system("cls" if os.name == "nt" else "clear")

        movie = self.movies.loc[movie_id]

        info_table = Table(show_header=False, box=None)
        info_table.add_row(
            "🎬 Название:", f"[bold cyan]{movie['title']}[/bold cyan]"
        )
        info_table.add_row(
            "📅 Год:", str(movie["year"]) if movie["year"] == movie["year"]
else "--"
        )
        info_table.add_row("🎭 Жанры:", f"[green]{movie['genres']}[/green]")
        self.console.print(Panel(info_table, title=f"ID {movie_id}",
expand=False))

        if self.tags is not None:
            try:
                df = self.tags.xs(movie_id, level="movieId")
                movie_tags = df["tag"].tolist()
            except KeyError:
                movie_tags = []

            if movie_tags:
                tags_str = ", ".join(f"[yellow]{t}[/yellow]" for t in
movie_tags)
                self.console.print(Panel(tags_str, title="Теги",
expand=False))
            else:
                self.console.print(
                    Panel(
                        "[italic grey]Для этого фильма нет тегов[/italic
grey]",
                        title="Теги",
                        expand=False,
                    )
                )

        self.console.print(
            "\n[bold]Доступные действия:[/bold]\n"
            "[green]r[/green] — оценить фильм\n"
            "[yellow]b[/yellow] — назад\n"
            "[red]q[/red] — выйти из приложения"

```

```

    )

    choice = input("> ").strip().lower()
    if choice == "r":
        self.rate_movie(movie_id)
    elif choice == "b":
        break
    elif choice == "q":
        if Confirm.ask("Are you sure?"):
            exit(0)

def rate_movie(self, movie_id: int):
    """Простейшая система выставления оценки"""
    while True:
        rating = Prompt.ask(
            "Введите оценку от 0.5 до 5.0 (или 'b' чтобы вернуться)"
        )
        if rating.lower() == "b":
            break
        try:
            rating = float(rating)
            if 0.5 <= rating <= 5.0:
                self.rates.loc[movie_id, "rating"] = rating
                self.markov_recommender.reset_matrix()
                self.continuous_markov_recommender.reset_matrix()
                self.markov_recommender.build_transition_matrix()
                self.continuous_markov_recommender.build_intensity_matrix()
                self.console.print(
                    f"[bold green]Спасибо! Ваша оценка {rating} сохранена
для фильма с ID {movie_id}[/bold green]"
                )
                input("Нажмите Enter, чтобы вернуться...")
                break
            else:
                self.console.print(
                    "[red]Оценка должна быть в диапазоне 0.5-5.0[/red]"
                )
                input("Нажмите Enter, чтобы вернуться...")
                break
        except ValueError:
            self.console.print("[red]Введите корректное число или
'b'[/red]")
            input("Нажмите Enter, чтобы вернуться...")
            break

def search_movies(self):
    """Поиск фильмов по названию с фильтрацией по году и жанрам"""
    os.system("cls" if os.name == "nt" else "clear")
    search_query = (
        Prompt.ask("Введите название фильма (Enter –
пропустить)").strip().lower()
    )
    selected_genres = self.choose_genres()
    year_op, year_val = self.choose_year_filter()

    def apply_filters():
        df = self.movies
        if search_query:
            df = df[df["title"].str.lower().str.contains(search_query,
na=False)]
        if selected_genres:

```

```

        df = df[
            df["genres"].apply(
                lambda g: all(gen in g for gen in selected_genres)
            )
        ]
    if year_op is not None and year_val is not None:
        if year_op == "=":
            df = df[df["year"] == year_val]
        elif year_op == ">":
            df = df[df["year"] > year_val]
        elif year_op == "<":
            df = df[df["year"] < year_val]
    return df

filtered_df = apply_filters()
if filtered_df.empty:
    self.console.print(
        Panel("[red]❌ Фильмы не найдены[/red]", title="Результат")
    )
    input("Нажмите Enter, чтобы вернуться...")
    return

page = 1
per_page = self.per_page

while True:
    os.system("cls" if os.name == "nt" else "clear")

    total_pages = max(1, (len(filtered_df) - 1) // per_page + 1)
    page = min(page, total_pages)

    start = (page - 1) * per_page
    end = start + per_page
    page_data = filtered_df.iloc[start:end]

    table = Table(
        title=f"Результаты поиска: {len(filtered_df)}",
        show_header=True,
        header_style="bold magenta",
    )
    table.add_column("ID", style="cyan", width=6)
    table.add_column("Название", style="white")
    table.add_column("Год", style="yellow", width=8)
    table.add_column("Жанры", style="green")

    for idx, row in page_data.iterrows():
        year = str(row["year"]) if not pd.isna(row["year"]) else "-"
        genres = ", ".join(row["genres"]) if row["genres"] else "-"
        table.add_row(str(idx), row["title"], year, genres)

    self.console.print(table)

    self.show_paginator(page, total_pages)

    status = f"[bold]Фильтры:[/bold] Название: [cyan]{search_query or '-'}[/cyan], Жанры: [cyan]{'', '.join(selected_genres) if selected_genres else '-'}[/cyan], Год: [cyan]{year_op + str(year_val) if year_op and year_val else '-'}[/cyan]"

    self.console.print(status)

    self.console.print(

```



### Продолжение Листинга А

```
        "[yellow]n[/yellow] – следующая, [yellow]p[/yellow] –  
предыдущая, [yellow]число[/yellow] – перейти на страницу\n"  
        "[yellow]s[/yellow] – отсортировать результаты\n"  
        "[yellow]o[/yellow] – открыть страницу фильма по ID\n"  
        "[yellow]r[/yellow] – сброс фильтров\n"  
        "[yellow]b[/yellow] – назад"  
    )  
  
    choice = input("> ").strip().lower()  
    if choice == "n" and page < total_pages:  
        page += 1  
    elif choice == "p" and page > 1:  
        page -= 1  
    elif choice == "s":  
        self.sort_movies(filtered_df)  
    elif choice.isdigit():  
        num = int(choice)  
        if 1 <= num <= total_pages:  
            page = num  
    elif choice == "o":  
        movie_id_str = Prompt.ask("Введите ID фильма (или 'b' чтобы  
вернуться)")  
        if movie_id_str.lower() == "b":  
            continue  
        try:  
            movie_id = int(movie_id_str)  
            if movie_id in self.movies.index:  
                self.show_movie_page(movie_id)  
            else:  
                self.console.print("[red]Фильм с таким ID не  
найден[/red]")  
                input("Нажмите Enter, чтобы продолжить...")  
        except ValueError:  
            self.console.print("[red]Введите корректный ID[/red]")  
            input("Нажмите Enter, чтобы продолжить...")  
    elif choice == "r":  
        search_query = ""  
        selected_genres = []  
        year_op, year_val = None, None  
        filtered_df = self.movies  
        page = 1  
    elif choice == "b":  
        break  
  
    def choose_genres(self):  
        """Выбор одного или нескольких жанров через консоль с таблицей"""  
        genre_list = list(Genres)  
        table = Table(  
            title="Выберите жанры", show_header=True, header_style="bold  
magenta"  
        )  
        table.add_column("№", justify="center", style="cyan", width=4)  
        table.add_column("Жанр", justify="left", style="green")  
  
        for i, genre in enumerate(genre_list, 1):  
            table.add_row(str(i), genre.value)  
  
        self.console.print(table)  
        choice = Prompt.ask(  
            "Введите номера жанров через пробел (Enter – пропустить)"  
        ).strip()
```

### Продолжение Листинга А

```
        if not choice:
            return []

        selected_genres = []
        for num in choice.split():
            num = num.strip()
            if num.isdigit():
                idx = int(num) - 1
                if 0 <= idx < len(genre_list):
                    selected_genres.append(genre_list[idx].value)
        return selected_genres

    def choose_year_filter(self):
        """Выбор фильтра по году с оператором"""
        op = Prompt.ask(
            "Выберите оператор для фильтра по году",
            choices=[">", "<", "="],
            default="=",
        )
        year_str = Prompt.ask("Введите год").strip()
        if not year_str.isdigit():
            self.console.print("[red]Некорректный год, фильтр не будет  
применён[/red]")
            return None, None
        return op, int(year_str)

    def show_paginator(self, page, total_pages):
        """Красивый центрированный пагинатор"""
        paginator_text = Text()
        last_was_ellipsis = False

        if page > 1:
            paginator_text.append("◀ Prev ", style="bold yellow")
        else:
            paginator_text.append(" ")

        for p in range(1, total_pages + 1):
            if p == 1 or p == total_pages or abs(p - page) <= 1:
                if p == page:
                    paginator_text.append(f" {p} ", style="reverse green")
                else:
                    paginator_text.append(f" {p} ", style="bold cyan")
                last_was_ellipsis = False
            else:
                if not last_was_ellipsis:
                    paginator_text.append(" ... ", style="dim")
                    last_was_ellipsis = True

        if page < total_pages:
            paginator_text.append(" Next ►", style="bold yellow")

        self.console.print(Align.center(paginator_text))

    def show_my_ratings(self):
        if self.rates.empty:
            self.console.print(
                Panel(
                    "[red]Вы еще не оценили ни одного фильма[/red]", title="Ваши  
оценки"
                )
            )
```

### Продолжение Листинга А

```
        input("Нажмите Enter, чтобы вернуться...")
        return

    page = 1
    df = self.rates.merge(self.movies, on="movieId")
    total_pages = (len(df) - 1) // self.per_page + 1

    while len(df):
        os.system("cls" if os.name == "nt" else "clear")

        start = (page - 1) * self.per_page
        end = start + self.per_page
        subset = df.iloc[start:end]

        rates = Table(show_header=True, title=f"Ваши оценки – страница
{page}")

        rates.add_column("ID", style="cyan", no_wrap=True)
        rates.add_column("Название", style="bold")
        rates.add_column("Год", justify="center", style="magenta")
        rates.add_column("Жанры", style="green")
        rates.add_column("Оценка", style="green")

        for movie_id, row in subset.iterrows():
            genres_str = ", ".join(row["genres"]) if row["genres"] else "-"
            year_str = str(row["year"]) if not pd.isna(row["year"]) else "-"
            rates.add_row(
                str(movie_id),
                row["title"],
                year_str,
                genres_str,
                str(row["rating"]),
            )

        self.console.print(rates)

        self.show_paginator(page, total_pages)

        self.console.print(
            "[yellow]n[/yellow] – следующая, [yellow]p[/yellow] –
предыдущая, "
            "[yellow]число[/yellow] – перейти на страницу\n"
            "[yellow]d[/yellow] – удалить оценку\n"
            "[yellow]o[/yellow] – перейти на страницу фильма\n"
            "[yellow]b[/yellow] – вернуться на главную страницу\n"
            "[yellow]q[/yellow] – выход"
        )

        choice = input("> ").strip().lower()
        if choice == "n" and page < total_pages:
            page += 1
        elif choice == "p" and page > 1:
            page -= 1
        elif choice.isdigit():
            num = int(choice)
            if 1 <= num <= total_pages:
                page = num
        elif choice == "d":
            movie_id_str = Prompt.ask(
                "Введите ID фильма, у которого хотите удалить оценку (или
'b' для отмены)"
            )
```

```

        if movie_id_str.lower() == "b":
            continue
        if movie_id_str.isdigit():
            id = int(movie_id_str)
            if id in self.rates.index:
                self.rates = self.rates.drop(index=id)
                df = df.drop(index=id)
                self.markov_recommender.reset_matrix()
                self.markov_recommender.build_transition_matrix()
            else:
                self.console.print(
                    "[red]Не найдено оценки для фильма с заданным
ID[/red]"
                )
                input("Нажмите Enter, чтобы вернуться...")
        else:
            self.console.print("[red]Некорректный ID[/red]")
            input("Нажмите Enter, чтобы вернуться...")
    elif choice == "o":
        movie_id_str = Prompt.ask("Введите ID фильма (или 'b' для
отмены)")

        if movie_id_str.lower() == "b":
            continue
        if movie_id_str.isdigit():
            self.show_movie_page(int(movie_id_str))
        else:
            self.console.print("[red]Некорректный ID[/red]")
            input("Нажмите Enter, чтобы вернуться...")
    elif choice == "b":
        break
    elif choice == "q":
        if Confirm.ask("Are you sure?"):
            exit(0)

def sort_movies(self, frame):
    """Сортировка фильмов по различным полям"""
    sort_fields = [
        ("ID", "movieId"),
        ("Название", "title"),
        ("Год", "year"),
        ("Жанры", "genres"),
    ]

    table = Table(
        title="Выберите поле для сортировки",
        show_header=True,
        header_style="bold magenta",
    )
    table.add_column("№", justify="center", style="cyan", width=4)
    table.add_column("Поле", style="green")

    for i, (label, _) in enumerate(sort_fields, 1):
        table.add_row(str(i), label)

    self.console.print(table)

    choice = Prompt.ask("Введите номер поля (или Enter для отмены)").strip()

    if not choice.isdigit():
        return

```

### Продолжение Листинга А

```
choice_num = int(choice)
if not 1 <= choice_num <= len(sort_fields):
    self.console.print("[red]Некорректный выбор[/red]")
    input("Нажмите Enter, чтобы вернуться...")
    return

field_label, field_name = sort_fields[choice_num - 1]

direction = Prompt.ask(
    "Выберите направление сортировки", choices=["asc", "desc"],
    default="asc"
)

ascending = direction == "asc"

if field_name == "movieId":
    sorted_df = frame.sort_index(ascending=ascending)
elif field_name == "genres":
    sorted_df = frame.copy()
    sorted_df["__sort_genre"] = sorted_df["genres"].apply(
        lambda g: g[0] if g else ""
    )
    sorted_df = sorted_df.sort_values("__sort_genre",
    ascending=ascending).drop(
        columns="__sort_genre"
    )
else:
    sorted_df = frame.sort_values(field_name, ascending=ascending)

self.paginated_view(
    sorted_df, title=f"Сортировка по {field_label} ({direction})"
)

def paginated_view(self, df, title="Список фильмов"):
    """Универсальный постраничный просмотр DataFrame фильмов"""
    page = 1
    total_pages = max(1, (len(df) - 1) // self.per_page + 1)

    while True:
        os.system("cls" if os.name == "nt" else "clear")
        start = (page - 1) * self.per_page
        end = start + self.per_page
        subset = df.iloc[start:end]

        table = Table(title=f"{title} - страница {page}", show_header=True)
        table.add_column("ID", style="cyan", no_wrap=True)
        table.add_column("Название", style="bold")
        table.add_column("Год", justify="center", style="magenta")
        table.add_column("Жанры", style="green")

        for movie_id, row in subset.iterrows():
            genres_str = ", ".join(row["genres"]) if row["genres"] else "-"
            year_str = str(row["year"]) if not pd.isna(row["year"]) else "-"
            table.add_row(str(movie_id), row["title"], year_str, genres_str)

        self.console.print(table)
        self.show_paginator(page, total_pages)
        self.console.print(
            "[yellow]n[/yellow] - следующая, [yellow]p[/yellow] -
предыдущая, "
            "[yellow]число[/yellow] - перейти на страницу\n"

```

### Продолжение Листинга А

```
        "[yellow]o[/yellow] - открыть страницу фильма,  
[yellow]b[/yellow] - назад"  
    )  
  
    choice = input("> ").strip().lower()  
    if choice == "n" and page < total_pages:  
        page += 1  
    elif choice == "p" and page > 1:  
        page -= 1  
    elif choice.isdigit():  
        num = int(choice)  
        if 1 <= num <= total_pages:  
            page = num  
    elif choice == "o":  
        movie_id_str = Prompt.ask("Введите ID фильма (или 'b' для  
отмены) ")  
        if movie_id_str.lower() == "b":  
            continue  
        if movie_id_str.isdigit():  
            self.show_movie_page(int(movie_id_str))  
    elif choice == "b":  
        break  
    os.system("cls" if os.name == "nt" else "clear")  
  
def show_recommendations(self):  
    """Меню выбора стратегии рекомендаций"""  
    os.system("cls" if os.name == "nt" else "clear")  
    if self.rates.empty:  
        self.console.print(  
            Panel(  
                "[red]У вас нет оценок для генерации рекомендаций.[/red]",  
                title="Рекомендации",  
            )  
        )  
    input("Нажмите Enter, чтобы вернуться...")  
    return  
  
    strategy = Prompt.ask(  
        "Выберите стратегию рекомендаций",  
        choices=["user-based", "item-based", "markov-discrete", "markov-  
continuous"],  
        default="user-based",  
    )  
  
    if strategy == "user-based":  
        self.recommend_user_based()  
    elif strategy == "item-based":  
        self.recommend_item_based()  
    elif strategy == "markov-discrete":  
        self.recommend_markov_chain()  
    else:  
        self.recommend_continuous_markov_chain()  
  
def recommend_user_based(self):  
    """User-based рекомендации"""  
    R = self.get_rating_matrix()  
    user_ids = R.index.tolist()  
    movie_ids = R.columns.tolist()  
    current_user_id = -1  
  
    current_user_ratings = pd.Series(  

```

```

        [np.nan] * len(movie_ids), index=movie_ids, dtype=float
    )
    for movie_id in self.rates.index:
        if movie_id in movie_ids:
            current_user_ratings[movie_id] = self.rates.loc[movie_id,
"rating"]

    R = pd.concat(
        [R, pd.DataFrame([current_user_ratings], index=[current_user_id])]
    )
    current_user_idx = R.index.get_loc(current_user_id)

    metric = Prompt.ask(
        "Выберите метрику сходства [pearson/jaccard/lp/otiai]",
        choices=["pearson", "jaccard", "lp", "otiai"],
        default="pearson",
    )

    R_values = R.values.astype(np.float64)
    n_users, n_items = R_values.shape
    user_means = np.nanmean(R_values, axis=1)
    preds = np.full(n_items, np.nan)
    sims = np.zeros(n_users)
    current_ratings = R_values[current_user_idx, :]

    for u in range(n_users):
        if u == current_user_idx:
            continue
        other = R_values[u, :]
        mask = ~np.isnan(current_ratings) & ~np.isnan(other)
        if np.sum(mask) == 0:
            continue
        if metric == "pearson":
            sims[u] = np.corrcoef(current_ratings[mask], other[mask])[0, 1]
        elif metric == "lp":
            sims[u] = -np.linalg.norm(current_ratings[mask] - other[mask])
        elif metric == "jaccard":
            sims[u] = np.sum(
                (current_ratings[mask] > 0) & (other[mask] > 0)
            ) / np.sum((current_ratings[mask] > 0) | (other[mask] > 0))
        elif metric == "otiai":
            sims[u] = np.dot(current_ratings[mask], other[mask]) / (
                np.linalg.norm(current_ratings[mask]) *
np.linalg.norm(other[mask])
            )

    for i in range(n_items):
        if not np.isnan(R_values[current_user_idx, i]):
            continue
        mask = ~np.isnan(R_values[:, i])
        if np.sum(mask) == 0:
            continue
        numerator = np.sum(sims[mask] * (R_values[mask, i] -
user_means[mask]))
        denominator = np.sum(np.abs(sims[mask])) + 1e-8
        preds[i] = user_means[current_user_idx] + numerator / denominator

    recs_df = pd.DataFrame({"movieId": R.columns, "pred_rating": preds})
    recs_df = recs_df.dropna().sort_values("pred_rating", ascending=False)
    self.paginated_view(
        self.movies.loc[recs_df["movieId"]], title=f"User-based ({metric})"
    )

```

```

    )

    def recommend_item_based(self):
        """Item-based рекомендации"""
        R = self.get_rating_matrix()
        user_ids = R.index.tolist()
        movie_ids = R.columns.tolist()
        current_user_id = -1

        current_user_ratings = pd.Series(
            [np.nan] * len(movie_ids), index=movie_ids, dtype=float
        )
        for movie_id in self.rates.index:
            if movie_id in movie_ids:
                current_user_ratings[movie_id] = self.rates.loc[movie_id,
"rating"]

        R = pd.concat(
            [R, pd.DataFrame([current_user_ratings], index=[current_user_id])]
        )
        current_user_idx = R.index.get_loc(current_user_id)

        metric = Prompt.ask(
            "Выберите метрику сходства [pearson/jaccard/lp/otiai]",
            choices=["pearson", "jaccard", "lp", "otiai"],
            default="pearson",
        )

        R_values = R.values.astype(np.float64)
        n_items = R_values.shape[1]
        item_means = np.nanmean(R_values, axis=0)
        preds = np.full(n_items, np.nan)

        for i in range(n_items):
            if not np.isnan(R_values[current_user_idx, i]):
                continue
            rated_mask = ~np.isnan(R_values[current_user_idx, :])
            sims_i = []
            ratings_i = []
            for j in np.where(rated_mask)[0]:
                mask = ~np.isnan(R_values[:, i]) & ~np.isnan(R_values[:, j])
                if np.sum(mask) == 0:
                    sim = 0
                else:
                    if metric == "pearson":
                        sim = np.corrcoef(R_values[mask, i], R_values[mask,
j])[0, 1]

                    elif metric == "lp":
                        sim = -np.linalg.norm(R_values[mask, i] - R_values[mask,
j])

                    elif metric == "jaccard":
                        sim = np.sum(
                            (R_values[mask, i] > 0) & (R_values[mask, j] > 0)
                        ) / np.sum((R_values[mask, i] > 0) | (R_values[mask, j]
> 0))

                    elif metric == "otiai":
                        sim = np.dot(R_values[mask, i], R_values[mask, j]) / (
                            np.linalg.norm(R_values[mask, i])
                            * np.linalg.norm(R_values[mask, j])
                        )
                sims_i.append(sim)

```



### Продолжение Листинга А

```
        ratings_i.append(R_values[current_user_idx, j] - item_means[j])
    sims_i = np.array(sims_i)
    ratings_i = np.array(ratings_i)
    if np.sum(np.abs(sims_i)) > 0:
        preds[i] = item_means[i] + np.dot(sims_i, ratings_i) / np.sum(
            np.abs(sims_i)
        )

    recs_df = pd.DataFrame({"movieId": R.columns, "pred_rating": preds})
    recs_df = recs_df.dropna().sort_values("pred_rating", ascending=False)
    self.paginated_view(
        self.movies.loc[recs_df["movieId"]], title=f"Item-based ({metric})"
    )

def recommend_markov_chain(self):
    """Улучшенные рекомендации с помощью дискретной цепи Маркова"""
    if self.rates.empty:
        self.console.print(
            Panel("[red]Вы не оценили ни одного фильма![/red]",
title="Ошибка")
        )
        input("Нажмите Enter, чтобы вернуться...")
        return

    if len(self.rates) < 2:
        self.console.print(
            Panel(
                "[yellow]Оцените хотя бы 2 фильма для построения цепи
Маркова[/yellow]",
                title="Недостаточно данных",
            )
        )

        unrated_movies =
self.movies[~self.movies.index.isin(self.rates.index)]
        if not unrated_movies.empty:
            if Confirm.ask("Показать случайные рекомендации вместо цепи
Маркова?"):
                recommendations = unrated_movies.sample(
                    min(20, len(unrated_movies))
                )
                self.paginated_view(recommendations, title="Случайные
рекомендации")
            else:
                self.console.print("[red]Нет фильмов для рекомендаций[/red]")
                input("Нажмите Enter, чтобы вернуться...")
                return

        markov_rec = self.markov_recommender

        steps = Prompt.ask(
            "Количество шагов цепи Маркова", choices=["1", "2", "3"],
default="1"
        )
        steps = int(steps)

        top_k = Prompt.ask("Количество рекомендаций", default="20")
        top_k = int(top_k)

        recommendations = markov_rec.recommend_movies(steps=steps, top_k=top_k)
```

```

        if recommendations.empty:
            self.console.print(
                Panel(
                    "[yellow]Не найдено подходящих рекомендаций. Попробуйте
оценить больше фильмов разных жанров.[/yellow]",
                    title="Рекомендации",
                )
            )

            unrated_movies =
self.movies[~self.movies.index.isin(self.rates.index)]
            if not unrated_movies.empty:
                if Confirm.ask("Показать случайные рекомендации?"):
                    random_recs = unrated_movies.sample(min(20,
len(unrated_movies)))
                    self.paginated_view(random_recs, title="Случайные
рекомендации")
                else:
                    input("Нажмите Enter, чтобы вернуться...")
                    return

            if Confirm.ask("Показать матрицу переходов?"):
                os.system("cls" if os.name == "nt" else "clear")
                markov_rec.show_transition_matrix()
                input("Нажмите Enter, чтобы продолжить...")

            if Confirm.ask("Показать вероятности жанров?"):
                os.system("cls" if os.name == "nt" else "clear")
                self.show_markov_genre_probabilities(steps)
                input("Нажмите Enter, чтобы продолжить...")

        self.console.print(
            Panel(
                f"🎲 Рекомендации на основе цепи Маркова ({steps} шаг(a))",
                title="Цепь Маркова",
            )
        )

        self.paginated_view(
            recommendations, title=f"Рекомендации (Марковская цепь, {steps}
шаг(a))"
        )

    def show_markov_genre_probabilities(self, steps=1):
        """Показать вероятности жанров для цепи Маркова"""
        genre_probs = self.markov_recommender.get_genre_probabilities(steps)

        if not genre_probs:
            self.console.print(
                "[red]Не удалось вычислить вероятности жанров. Оцените больше
фильмов для построения цепи Маркова.[/red]"
            )
            return

        sorted_genres = sorted(genre_probs.items(), key=lambda x: x[1],
reverse=True)

        prob_table = Table(title=f"Вероятности жанров (через {steps} шаг(a))")
        prob_table.add_column("Жанр", style="cyan")
        prob_table.add_column("Вероятность", style="green")

```

### Продолжение Листинга А

```
prob_table.add_column("Процент", style="magenta")

for genre, prob in sorted_genres:
    if prob > 0.001:
        percentage = prob * 100
        prob_table.add_row(genre, f"{prob:.3f}", f"{percentage:.1f}%")

self.console.print(prob_table)

rated_movies = self.movies.loc[self.rates.index]
if not rated_movies.empty:
    last_movie = rated_movies.iloc[-1]
    self.console.print(
        f"\n[bold]Последний оцененный фильм:[/bold]
[cyan]{last_movie['title']}[/cyan]"
    )
    self.console.print(
        f"[bold]Жанры:[/bold] [green]{'',
'.join(last_movie['genres'])}[/green]"
    )

def get_ratings_with_timestamps(self):
    """Получение оценок с временными метками в хронологическом порядке"""
    if self.rates.empty:
        return []

    ratings_list = []
    for movie_id, row in self.rates.iterrows():
        if 'datetime' in row:
            timestamp = row['datetime']
        else:
            timestamp = datetime.now() - timedelta(days=len(ratings_list))

        ratings_list.append((movie_id, row['rating'], timestamp))

    ratings_list.sort(key=lambda x: x[2])
    return ratings_list

def recommend_continuous_markov_chain(self):
    """Рекомендации на основе непрерывной цепи Маркова с временными
метками"""
    if self.rates.empty:
        self.console.print(
            Panel("[red]Вы не оценили ни одного фильма![/red]",
title="Ошибка")
        )
        input("Нажмите Enter, чтобы вернуться...")
        return

    if len(self.rates) < 2:
        self.console.print(
            Panel(
                "[yellow]Оцените хотя бы 2 фильма для построения непрерывной
цепи Маркова[/yellow]",
                title="Недостаточно данных",
            )
        )
        return

    continuous_markov = self.continuous_markov_recommender
```

```

        time_t = Prompt.ask(
            "Введите время для прогноза (в днях)", default="1.0"
        )
    try:
        time_t = float(time_t)
    except ValueError:
        time_t = 1.0

    top_k = Prompt.ask("Количество рекомендаций", default="20")
    top_k = int(top_k)

    ratings_with_time =
self.continuous_markov_recommender.get_ratings_chronology()
    if ratings_with_time:
        self.console.print("\n[bold]Хронология ваших оценок:[/bold]")
        for i, (movie_id, rating, timestamp) in
enumerate(ratings_with_time):
            movie_title = self.movies.loc[movie_id, "title"]
            time_str = timestamp.strftime("%Y-%m-%d %H:%M") if
isinstance(timestamp, datetime) else "недавно"
            self.console.print(f" {i+1}. {movie_title} - оценка {rating} -
{time_str}")

    recommendations = continuous_markov.recommend_movies_continuous(
        time_t=time_t, top_k=top_k
    )

    if recommendations.empty:
        self.console.print(
            Panel(
                "[yellow]Не найдено подходящих рекомендаций[/yellow]",
                title="Рекомендации",
            )
        )
        input("Нажмите Enter, чтобы вернуться...")
        return

    if Confirm.ask("Показать матрицу интенсивностей?"):
        os.system("cls" if os.name == "nt" else "clear")
        continuous_markov.show_intensity_matrix()
        input("Нажмите Enter, чтобы продолжить...")

    if Confirm.ask("Показать вероятности жанров для непрерывной цепи?"):
        os.system("cls" if os.name == "nt" else "clear")
        self.show_continuous_markov_genre_probabilities(time_t)
        input("Нажмите Enter, чтобы продолжить...")

    self.console.print(
        Panel(
            f"🕒 Рекомендации на основе непрерывной цепи Маркова (t={time_t}
дней)",
            title="Непрерывная цепь Маркова",
        )
    )

    self.paginated_view(
        recommendations,
        title=f"Рекомендации (Непрерывная цепь Маркова, t={time_t} дней)"
    )

```

### Продолжение Листинга А

```
def show_continuous_markov_genre_probabilities(self, time_t=1.0):
    """Показать вероятности жанров для непрерывной цепи Маркова"""
    genre_probs =
self.continuous_markov_recommender.get_genre_probabilities_continuous(time_t)

    if not genre_probs:
        self.console.print(
            "[red]Не удалось вычислить вероятности жанров для непрерывной
цепи.[/red]"
        )
        return

    sorted_genres = sorted(genre_probs.items(), key=lambda x: x[1],
reverse=True)

    prob_table = Table(title=f"Вероятности жанров (непрерывная цепь,
t={time_t})")
    prob_table.add_column("Жанр", style="cyan")
    prob_table.add_column("Вероятность", style="green")
    prob_table.add_column("Процент", style="magenta")

    for genre, prob in sorted_genres:
        if prob > 0.001:
            percentage = prob * 100
            prob_table.add_row(genre, f"{prob:.3f}", f"{percentage:.1f}%")

    self.console.print(prob_table)

    rated_movies = self.movies.loc[self.rates.index]
    if not rated_movies.empty:
        last_movie = rated_movies.iloc[-1]
        self.console.print(
            f"\n[bold]Последний оцененный фильм:[/bold]
[cyan]{last_movie['title']}[/cyan]"
        )
        self.console.print(
            f"[bold]Жанры:[/bold] [green]{',
'.join(last_movie['genres'])}[/green]"
        )

    def plot_similarity_metrics(
        self, strategy: str = "user-based", test_ratio: float = 0.2
    ):
        metrics = ["pearson", "lp", "jaccard", "otiai"]
        results = []

        for metric in metrics:
            rmse, mae = self._compute_metrics_for_plot(metric, strategy,
test_ratio)
            results.append({"metric": metric, "RMSE": rmse, "MAE": mae})

        df = pd.DataFrame(results)

        sns.set_theme(style="whitegrid", font_scale=1.1)
        fig, axes = plt.subplots(1, 2, figsize=(12, 5))

        sns.barplot(x="metric", y="RMSE", data=df, ax=axes[0],
palette="Blues_d")
        axes[0].set_title(f"Сравнение RMSE ({strategy})")
        axes[0].set_xlabel("Метрика близости")
        axes[0].set_ylabel("RMSE")
```

```

sns.barplot(x="metric", y="MAE", data=df, ax=axes[1],
palette="Greens_d")
axes[1].set_title(f"Сравнение MAE ({strategy})")
axes[1].set_xlabel("Метрика близости")
axes[1].set_ylabel("MAE")

plt.tight_layout()
plt.show()

def _compute_metrics_for_plot(self, metric: str, strategy: str, test_ratio:
float):
    R = self.get_rating_matrix().copy().values.astype(float)
    n_users, n_items = R.shape

    rng = np.random.default_rng(42)
    train = R.copy()
    test_mask = ~np.isnan(R) & (rng.random(R.shape) < test_ratio)
    test_true = np.full_like(R, np.nan)
    test_true[test_mask] = R[test_mask]
    train[test_mask] = np.nan

    user_means = np.nanmean(train, axis=1)
    item_means = np.nanmean(train, axis=0)
    preds = np.full_like(R, np.nan)

    for u in range(n_users):
        if strategy == "user-based":
            sims = np.zeros(n_users)
            current = train[u, :]
            for v in range(n_users):
                if v == u:
                    continue
                other = train[v, :]
                mask = ~np.isnan(current) & ~np.isnan(other)
                if np.sum(mask) == 0:
                    sims[v] = 0
                    continue
                if metric == "pearson":
                    sims[v] = np.corrcoef(current[mask], other[mask])[0, 1]
                elif metric == "lp":
                    sims[v] = -np.linalg.norm(current[mask] - other[mask])
                elif metric == "jaccard":
                    sims[v] = np.sum(
                        (current[mask] > 0) & (other[mask] > 0)
                    ) / np.sum((current[mask] > 0) | (other[mask] > 0))
                elif metric == "otiai":
                    sims[v] = np.sum(current[mask] * other[mask]) / (
                        np.linalg.norm(current[mask]) *
np.linalg.norm(other[mask])
                    )
            else:
                sims[v] = 0
        for i in range(n_items):
            if not np.isnan(train[u, i]):
                continue
            mask = ~np.isnan(train[:, i])
            if np.sum(mask) == 0:
                continue
            numerator = np.sum(sims[mask] * (train[mask, i] -
user_means[mask]))

```

```

        denominator = np.sum(np.abs(sims[mask])) + 1e-8
        preds[u, i] = user_means[u] + numerator / denominator

    else:
        for i in range(n_items):
            if not np.isnan(train[u, i]):
                continue
            rated_mask = ~np.isnan(train[u, :])
            sims_i = []
            ratings_i = []
            for j in np.where(rated_mask)[0]:
                mask = ~np.isnan(train[:, i]) & ~np.isnan(train[:, j])
                if np.sum(mask) == 0:
                    sim = 0
                else:
                    if metric == "pearson":
                        sim = np.corrcoef(train[mask, i], train[mask,
j]))[0, 1]

                    elif metric == "lp":
                        sim = -np.linalg.norm(train[mask, i] -
train[mask, j])

                    elif metric == "jaccard":
                        sim = np.sum(
                            (train[mask, i] > 0) & (train[mask, j] > 0)
                        ) / np.sum((train[mask, i] > 0) | (train[mask,
j] > 0))

                    elif metric == "otiai":
                        sim = np.sum(train[mask, i] * train[mask, j]) /
(
                            np.linalg.norm(train[mask, i])
                            * np.linalg.norm(train[mask, j])
                        )
                    else:
                        sim = 0
                sims_i.append(sim)
                ratings_i.append(train[u, j] - item_means[j])
            sims_i = np.array(sims_i)
            ratings_i = np.array(ratings_i)
            if np.sum(np.abs(sims_i)) > 0:
                preds[u, i] = item_means[i] + np.dot(
                    sims_i, ratings_i
                ) / np.sum(np.abs(sims_i))

        mask_eval = ~np.isnan(test_true) & ~np.isnan(preds)
        if np.sum(mask_eval) == 0:
            return np.nan, np.nan

        diff = preds[mask_eval] - test_true[mask_eval]
        rmse = np.sqrt(np.mean(diff**2))
        mae = np.mean(np.abs(diff))
        return rmse, mae

if __name__ == "__main__":
    cinema = MovieLensCinema(r"../ml-latest-small",
enable_markov_visualization=True)
    cinema.menu()
    # cinema.plot_similarity_metrics()

```

## Приложение Б

### Код файла MarkovChain.py

#### *Листинг Б – Код файла MarkovChain.py*

```
import numpy as np
import os
import shutil
import networkx as nx
import matplotlib.pyplot as plt
from datetime import datetime
from rich.table import Table

class MarkovChainRecommender:
    """Внутренний класс для рекомендаций на основе цепи Маркова"""

    def __init__(self, cinema, auto_visualize=True):
        self.cinema = cinema
        self.transition_matrix = None
        self.genre_states = []
        self.auto_visualize = auto_visualize

        self._clean_markov_directory()

    def _clean_markov_directory(self):
        """Очистка папки markov при запуске программы"""
        try:
            if os.path.exists("markov"):
                shutil.rmtree("markov")
            os.makedirs("markov", exist_ok=True)
        except Exception as e:
            pass

    def reset_matrix(self):
        """Сброс матрицы переходов (вызывается при изменении оценок)"""
        self.transition_matrix = None
        self.genre_states = []

    def build_transition_matrix(self):
        """Построение матрицы переходов между жанрами с корректной нормализацией
        (вариант 1 – честная модель)"""
        if self.cinema.rates.empty:
            return False

        rated_movies = self.cinema.movies.loc[self.cinema.rates.index]

        all_genres = set()
        for genres in rated_movies["genres"]:
            all_genres.update(genres)
        self.genre_states = sorted(all_genres)

        if not self.genre_states:
            return False

        n_genres = len(self.genre_states)
        self.transition_matrix = np.zeros((n_genres, n_genres))
        state_to_idx = {genre: idx for idx, genre in
            enumerate(self.genre_states)}

        rated_indices = list(rated_movies.index)
```



```
    if len(rated_indices) < 2:
        return False

    for i in range(len(rated_indices) - 1):
        current_movie_id = rated_indices[i]
        next_movie_id = rated_indices[i + 1]

        current_genres = rated_movies.loc[current_movie_id, "genres"]
        next_genres = rated_movies.loc[next_movie_id, "genres"]

        if not current_genres or not next_genres:
            continue

        weight = 1.0 / len(next_genres)

        for curr_genre in current_genres:
            if curr_genre not in state_to_idx:
                continue
            curr_idx = state_to_idx[curr_genre]

            for next_genre in next_genres:
                if next_genre not in state_to_idx:
                    continue
                next_idx = state_to_idx[next_genre]
                self.transition_matrix[curr_idx, next_idx] += weight

    row_sums = self.transition_matrix.sum(axis=1)

    for i in range(n_genres):
        if row_sums[i] > 0:
            self.transition_matrix[i] /= row_sums[i]
        else:
            self.transition_matrix[i, i] = 1.0

    if self.auto_visualize:
        self._auto_visualize_markov_chain()
    return True

def get_genre_probabilities(self, steps=1):
    """Получение вероятностей жанров через steps шагов"""
    if self.transition_matrix is None:
        success = self.build_transition_matrix()
        if not success:
            return None

    if self.cinema.rates.empty:
        return None

    rated_movies = self.cinema.movies.loc[self.cinema.rates.index]
    last_movie_id = rated_movies.index[-1]
    last_genres = rated_movies.loc[last_movie_id, "genres"]

    initial_vector = np.zeros(len(self.genre_states))
    genre_count = len(last_genres)

    if genre_count == 0:
        return None

    for genre in last_genres:
        if genre in self.genre_states:
```

```

        idx = self.genre_states.index(genre)
        initial_vector[idx] = 1.0 / genre_count

    current_probs = initial_vector
    for _ in range(steps):
        current_probs = current_probs @ self.transition_matrix

    return dict(zip(self.genre_states, current_probs))

def recommend_movies(self, steps=1, top_k=20, min_probability=0.01):
    """Рекомендация фильмов на основе цепи Маркова"""
    genre_probs = self.get_genre_probabilities(steps)
    if not genre_probs:
        unrated_movies = self.cinema.movies[
            ~self.cinema.movies.index.isin(self.cinema.rates.index)
        ]
        return unrated_movies.sample(min(20, len(unrated_movies)))

    significant_genres = [
        genre for genre, prob in genre_probs.items() if prob >=
min_probability
    ]

    if not significant_genres:
        significant_genres = [max(genre_probs, key=genre_probs.get)]

    candidate_movies = self.cinema.movies[
        ~self.cinema.movies.index.isin(self.cinema.rates.index)
    ].copy()

    if candidate_movies.empty:
        return candidate_movies

    def calculate_score(genres):
        return sum(genre_probs.get(genre, 0) for genre in genres)

    candidate_movies["markov_score"] = candidate_movies["genres"].apply(
        calculate_score
    )

    recommendations = (
        candidate_movies[candidate_movies["markov_score"] > 0]
        .sort_values("markov_score", ascending=False)
        .head(top_k)
    )

    return recommendations.drop(columns=["markov_score"], errors="ignore")

def show_transition_matrix(self):
    """Отображение матрицы переходов"""
    if self.transition_matrix is None:
        success = self.build_transition_matrix()
        if not success:
            self.cinema.console.print(
                "[red]Не удалось построить матрицу переходов[/red]"
            )
        return

    table = Table(title="Матрица переходов между жанрами")
    table.add_column("From/To", style="cyan")

```

```
for genre in self.genre_states:
    table.add_column(genre, style="green", width=10)

for i, from_genre in enumerate(self.genre_states):
    row = [from_genre]
    for j, to_genre in enumerate(self.genre_states):
        prob = self.transition_matrix[i, j]
        if prob > 0:
            row.append(f"{prob:.3f}")
        else:
            row.append("0")
    table.add_row(*row)

self.cinema.console.print(table)

def _auto_visualize_markov_chain(self, steps=1):
    """Автоматическое построение и сохранение графа цепи Маркова с
    финальными настройками"""
    os.makedirs("markov", exist_ok=True)
    if self.transition_matrix is None or len(self.genre_states) == 0:
        return

    G = nx.DiGraph()

    for genre in self.genre_states:
        G.add_node(genre)

    edge_labels = {}

    for i, from_genre in enumerate(self.genre_states):
        for j, to_genre in enumerate(self.genre_states):
            prob = self.transition_matrix[i, j]
            if prob >= 0.001:
                G.add_edge(from_genre, to_genre, weight=prob)
                edge_labels[(from_genre, to_genre)] = f"{prob:.2f}"

    plt.figure(figsize=(18, 14), dpi=120)

    if len(self.genre_states) <= 6:
        pos = nx.circular_layout(G, scale=3.0)
    elif len(self.genre_states) <= 10:
        pos = nx.circular_layout(G, scale=3.5)
    else:
        pos = nx.spring_layout(G, k=4, iterations=150, scale=3)

    node_size = 8000
    nx.draw_networkx_nodes(
        G,
        pos,
        node_size=node_size,
        node_color="lightblue",
        alpha=0.95,
        edgecolors="navy",
        linewidths=3,
        node_shape="o",
    )

    nx.draw_networkx_labels(
        G,
        pos,
        font_size=16,
```

```

        font_weight="bold",
        font_family="DejaVu Sans",
        verticalalignment="center",
        horizontalalignment="center",
    )

    if G.edges():
        edge_weights = [G[u][v]["weight"] for u, v in G.edges()]
        max_weight = max(edge_weights) if edge_weights else 1

        edge_widths = [1.5 + (w / max_weight) * 2.5 for w in edge_weights]
        edge_alphas = [0.4 + (w / max_weight) * 0.6 for w in edge_weights]

        edges = nx.draw_networkx_edges(
            G,
            pos,
            edge_color="darkred",
            width=edge_widths,
            alpha=edge_alphas,
            arrows=True,
            arrowsize=30,
            arrowstyle="->",
            connectionstyle="arc3,rad=0.25",
            min_source_margin=20,
            min_target_margin=25,
            node_size=node_size,
            ax=plt.gca(),
        )

        nx.draw_networkx_edge_labels(
            G,
            pos,
            edge_labels=edge_labels,
            font_size=12,
            font_weight="bold",
            font_family="DejaVu Sans",
            label_pos=0.18,
            verticalalignment="center",
            horizontalalignment="center",
            bbox=dict(
                boxstyle="round,pad=0.3",
                facecolor="white",
                alpha=0.9,
                edgecolor="lightgray",
                linewidth=1.5,
            ),
        )

    title_info = f"Цепь Маркова: переходы между жанрами (k={steps})\n"
    title_info += f"Узлы: {len(G.nodes())} | Переходы: {len(G.edges())} | "
    title_info += f"Сгенерировано: {datetime.now().strftime('%d.%m.%Y %H:%M')}"

    plt.title(
        title_info,
        fontsize=18,
        fontweight="bold",
        pad=30,
        loc="center",
        fontfamily="DejaVu Sans",
    )

```

*Окончание Листинга Б*

```
plt.axis("off")
plt.tight_layout(pad=4.0)

timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
filename = f"markov/markov_chain_k{steps}_{timestamp}.png"
plt.savefig(
    filename, dpi=150, bbox_inches="tight", facecolor="white",
    edgecolor="none"
)
plt.close()
```

## Приложение В

### Код файла ContinuousMarkovChain.py

#### Листинг В – Код файла ContinuousMarkovChain.py

```
import numpy as np
import pandas as pd
from scipy.linalg import expm
from rich.table import Table
import os
import matplotlib.pyplot as plt
import networkx as nx
from datetime import datetime, timedelta

class ContinuousMarkovRecommender:
    """Исправленная реализация непрерывной цепи Маркова с реальными временными метками"""

    def __init__(self, cinema, auto_visualize=True):
        self.cinema = cinema
        self.intensity_matrix = None
        self.genre_states = []
        self.auto_visualize = auto_visualize
        self.time_unit = 1.0

    def reset_matrix(self):
        """Сброс матрицы интенсивностей"""
        self.intensity_matrix = None
        self.genre_states = []

    def build_intensity_matrix(self):
        """Построение матрицы интенсивностей с использованием реальных данных"""
        if self.cinema.rates.empty or len(self.cinema.rates) < 2:
            return False

        ratings_chronology = self.get_ratings_chronology()
        if len(ratings_chronology) < 2:
            return False

        all_genres = set()
        for movie_id, rating, time_info in ratings_chronology:
            movie_genres = self.cinema.movies.loc[movie_id, "genres"]
            all_genres.update(movie_genres)
        self.genre_states = sorted(all_genres)

        if not self.genre_states:
            return False

        n_genres = len(self.genre_states)
        state_to_idx = {genre: idx for idx, genre in
            enumerate(self.genre_states)}

        time_in_state = {genre: 0.0 for genre in self.genre_states}
        transition_counts = np.zeros((n_genres, n_genres))

        for i in range(len(ratings_chronology) - 1):
            current_movie_id, current_rating, current_time =
ratings_chronology[i]
            next_movie_id, next_rating, next_time = ratings_chronology[i + 1]

            current_genres = self.cinema.movies.loc[current_movie_id, "genres"]
```

```

        next_genres = self.cinema.movies.loc[next_movie_id, "genres"]

        if not current_genres or not next_genres:
            continue

        time_interval = self.calculate_time_interval(current_time,
next_time)

        for genre in current_genres:
            if genre in state_to_idx:
                time_in_state[genre] += time_interval / len(current_genres)

        for curr_genre in current_genres:
            if curr_genre not in state_to_idx:
                continue
            curr_idx = state_to_idx[curr_genre]

            for next_genre in next_genres:
                if next_genre not in state_to_idx:
                    continue
                next_idx = state_to_idx[next_genre]
                weight = 1.0 / (len(current_genres) * len(next_genres))
                transition_counts[curr_idx, next_idx] += weight

        self.intensity_matrix = np.zeros((n_genres, n_genres))

        for i, genre_i in enumerate(self.genre_states):
            total_time = time_in_state[genre_i]
            if total_time > 0:
                for j, genre_j in enumerate(self.genre_states):
                    if i != j and transition_counts[i, j] > 0:
                        self.intensity_matrix[i, j] = (
                            transition_counts[i, j] / total_time
                        )

                self.intensity_matrix[i, i] = -np.sum(self.intensity_matrix[i])

        symmetric_matrix = (self.intensity_matrix + self.intensity_matrix.T) / 2

        for i in range(n_genres):
            for j in range(n_genres):
                if i != j:
                    self.intensity_matrix[i, j] = max(symmetric_matrix[i, j], 0)

                self.intensity_matrix[i, i] = -np.sum(self.intensity_matrix[i])

        min_intensity = 0.01
        for i in range(n_genres):
            if abs(self.intensity_matrix[i, i]) < min_intensity:
                for j in range(n_genres):
                    if i != j:
                        self.intensity_matrix[i, j] += min_intensity / (n_genres
- 1)

                self.intensity_matrix[i, i] = -np.sum(self.intensity_matrix[i])

        if self.auto_visualize:
            self._auto_visualize_markov_chain()

        return True

    def get_ratings_chronology(self, random_intervals=True):

```

### Продолжение Листинга В

```
"""Создание хронологии оценок с реалистичными случайными интервалами"""
if self.cinema.rates.empty:
    return []

ratings_list = []
current_time = datetime.now()

prev_time = current_time - timedelta(days=len(self.cinema.rates) * 2)

for i, (movie_id, row) in enumerate(self.cinema.rates.iterrows()):
    if random_intervals:
        interval_days = np.random.exponential(scale=2.0)
    else:
        interval_days = 1.0

    timestamp = prev_time + timedelta(days=interval_days)
    ratings_list.append((movie_id, row["rating"], timestamp))
    prev_time = timestamp

return sorted(ratings_list, key=lambda x: x[2])

def calculate_time_interval(self, time1, time2):
    """Вычисление временного интервала между двумя оценками"""
    time_diff = abs((time2 - time1).total_seconds()) / (24 * 3600)
    return max(time_diff, 0.1)

def get_genre_probabilities_continuous(self, time_t):
    """Получение вероятностей жанров в момент времени t с нормализацией"""
    if self.intensity_matrix is None:
        success = self.build_intensity_matrix()
        if not success:
            return None

    if self.cinema.rates.empty:
        return None

    ratings_chronology = self.get_ratings_chronology()
    if not ratings_chronology:
        return None

    last_movie_id, last_rating, last_time = ratings_chronology[-1]
    last_genres = self.cinema.movies.loc[last_movie_id, "genres"]

    initial_vector = np.zeros(len(self.genre_states))
    genre_count = len(last_genres)

    if genre_count == 0:
        return None

    for genre in last_genres:
        if genre in self.genre_states:
            idx = self.genre_states.index(genre)
            initial_vector[idx] = 1.0 / genre_count

    try:
        transition_matrix = expm(self.intensity_matrix * time_t)
        current_probs = initial_vector @ transition_matrix

        prob_sum = np.sum(current_probs)
        if prob_sum > 0:
            current_probs = current_probs / prob_sum
```



```

        else:
            current_probs = np.ones_like(current_probs) / len(current_probs)

    except Exception as e:
        print(f"Ошибка вычисления матричной экспоненты: {e}")
        return None

    return dict(zip(self.genre_states, current_probs))

def recommend_movies_continuous(self, time_t=1.0, top_k=20):
    """Улучшенные рекомендации, чувствительные к времени"""
    genre_probs = self.get_genre_probabilities_continuous(time_t)
    if not genre_probs:
        unrated_movies = self.cinema.movies[
            ~self.cinema.movies.index.isin(self.cinema.rates.index)
        ]
        return unrated_movies.sample(min(top_k, len(unrated_movies)))

    candidate_movies = self.cinema.movies[
        ~self.cinema.movies.index.isin(self.cinema.rates.index)
    ].copy()

    if candidate_movies.empty:
        return candidate_movies

    def calculate_score(genres, time_factor=time_t):
        if not genres:
            return 0

        base_score = sum(genre_probs.get(genre, 0) for genre in genres)

        diversity_bonus = len(set(genres) & set(genre_probs.keys())) /
len(genres)

        time_weight = min(time_t / 5.0, 1.0)
        final_score = base_score * (1 - time_weight) + diversity_bonus *
time_weight

        return final_score

    candidate_movies["continuous_score"] = candidate_movies["genres"].apply(
        lambda genres: calculate_score(genres, time_t)
    )

    max_score = candidate_movies["continuous_score"].max()
    if max_score > 0:
        candidate_movies["continuous_score"] = (
            candidate_movies["continuous_score"] / max_score
        )

    recommendations = (
        candidate_movies[candidate_movies["continuous_score"] > 0.01]
        .sort_values("continuous_score", ascending=False)
        .head(top_k)
    )

    return recommendations.drop(columns=["continuous_score"],
errors="ignore")

def show_intensity_matrix(self):
    """Отображение матрицы интенсивностей с дополнительной информацией"""

```

### Продолжение Листинга В

```
        if self.intensity_matrix is None:
            success = self.build_intensity_matrix()
            if not success:
                self.cinema.console.print(
                    "[red]Не удалось построить матрицу интенсивностей[/red]"
                )
            return

        table = Table(
            title="Матрица интенсивностей переходов (непрерывная цепь Маркова)"
        )
        table.add_column("From/To", style="cyan")

        for genre in self.genre_states:
            table.add_column(genre, style="green", width=10)

        for i, from_genre in enumerate(self.genre_states):
            row = [from_genre]
            for j, to_genre in enumerate(self.genre_states):
                intensity = self.intensity_matrix[i, j]
                if abs(intensity) > 0.001:
                    if i == j:
                        row.append(f"[red]{intensity:+.3f}[/red]")
                    else:
                        row.append(f"{intensity:+.3f}")
                else:
                    row.append("0.000")
            table.add_row(*row)

        self.cinema.console.print(table)

        self.cinema.console.print(
            f"\n[bold]Размер матрицы:[/bold] {len(self.genre_states)}x{len(self.genre_states)}"
        )
        self.cinema.console.print(
            f"[bold]Количество жанров:[/bold] {len(self.genre_states)}"
        )

    def _auto_visualize_markov_chain(self, time_t=1.0):
        """Автоматическая визуализация и сохранение графа непрерывной цепи
        Маркова"""
        os.makedirs("markov", exist_ok=True)
        if self.intensity_matrix is None or len(self.genre_states) == 0:
            return

        G = nx.DiGraph()

        for genre in self.genre_states:
            G.add_node(genre)

        edge_labels = {}

        for i, from_genre in enumerate(self.genre_states):
            for j, to_genre in enumerate(self.genre_states):
                intensity = self.intensity_matrix[i, j]
                if intensity > 0.001:
                    G.add_edge(from_genre, to_genre, weight=intensity)
                    edge_labels[(from_genre, to_genre)] = f"{intensity:.2f}"

        plt.figure(figsize=(18, 14), dpi=120)
```

```
if len(self.genre_states) <= 6:
    pos = nx.circular_layout(G, scale=3.0)
elif len(self.genre_states) <= 10:
    pos = nx.circular_layout(G, scale=3.5)
else:
    pos = nx.spring_layout(G, k=4, iterations=150, scale=3)

node_size = 8000
nx.draw_networkx_nodes(
    G,
    pos,
    node_size=node_size,
    node_color="lightblue",
    alpha=0.95,
    edgecolors="navy",
    linewidths=3,
    node_shape="o",
)

nx.draw_networkx_labels(
    G,
    pos,
    font_size=16,
    font_weight="bold",
    font_family="DejaVu Sans",
    verticalalignment="center",
    horizontalalignment="center",
)

if G.edges():
    edge_weights = [G[u][v]["weight"] for u, v in G.edges()]
    max_weight = max(edge_weights) if edge_weights else 1

    edge_widths = [1.5 + (w / max_weight) * 2.5 for w in edge_weights]
    edge_alphas = [0.4 + (w / max_weight) * 0.6 for w in edge_weights]

    edges = nx.draw_networkx_edges(
        G,
        pos,
        edge_color="darkred",
        width=edge_widths,
        alpha=edge_alphas,
        arrows=True,
        arrowsize=30,
        arrowstyle="-|>",
        connectionstyle="arc3,rad=0.25",
        min_source_margin=20,
        min_target_margin=25,
        node_size=node_size,
        ax=plt.gca(),
    )

    nx.draw_networkx_edge_labels(
        G,
        pos,
        edge_labels=edge_labels,
        font_size=12,
        font_weight="bold",
        font_family="DejaVu Sans",
        label_pos=0.18,
```

### Окончание Листинга В

```
        verticalalignment="center",
        horizontalalignment="center",
        bbox=dict(
            boxstyle="round,pad=0.3",
            facecolor="white",
            alpha=0.9,
            edgecolor="lightgray",
            linewidth=1.5,
        ),
    ),

    title_info = f"Непрерывная цепь Маркова: интенсивности переходов
(t={time_t})\n"
    title_info += f"Узлы: {len(G.nodes())} | Переходы: {len(G.edges())} | "
    title_info += f"Сгенерировано: {datetime.now().strftime('%d.%m.%Y
%H:%M')}"

    plt.title(
        title_info,
        fontsize=18,
        fontweight="bold",
        pad=30,
        loc="center",
        fontfamily="DejaVu Sans",
    )

    plt.axis("off")
    plt.tight_layout(pad=4.0)

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"markov/continuous_markov_chain_t{time_t}_{timestamp}.png"
    plt.savefig(
        filename, dpi=150, bbox_inches="tight", facecolor="white",
    edgecolor="none"
    )
    plt.close()
```