ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИТиЕН

Кафедра Общей математики

Специальные задачи линейного и нелинейного программирования

Выпускная квалификационная работа

студента очной формы обучения
направления подготовки 01.03.02 прикладная математика и информатика
(код, наименование направления подготовки)
4 курса группы 07001206
Золотько Илья Владимирович

Научный руководитель:
кандидат физико-математических наук, доцент кафедры,
Полунин В.А.
(ученая степень, звание,
фамилия, инициалы)

БЕЛГОРОД 2016

Оглавление

Введение

В настоящее время кластерный анализ, позволяющий выделять однородные группы объектов, находит все более широкое применение в анализе данных экологического мониторинга. Применение методов кластеризации позволяет понять структуру данных; упростить дальнейшую обработку, используя различные методы анализа для каждого кластера; сократить исходную выборку, оставив по одному наиболее типичному представителю каждой группы; выявить новизну, нетипичные объекты, которые не удаётся присоединить ни к одному из классов, сформулировать или проверить гипотезы на основании полученных результатов.

Результаты, полученные различными методами кластерного анализа, могут значительно отличаться друг от друга. В большинстве задач возникает проблема выбора оптимального числа кластеров, соответствующего природе изучаемых объектов. Поэтому одним из актуальных вопросов кластерного анализа является оценка качества полученных результатов и поиск разбиения, что наиболее соответствует структуре исследуемых данных.

Целью данной работы является разработка информационной технологии кластерного анализа, которая позволит автоматизировать процесс принятия решений в условиях невозможности привлечения экспертов предметной области либо отсутствия информации об ожидаемых результатах.

В настоящий момент в литературе существует множество функционалов и индексов качества, позволяющие в количественном виде оценивать соответствие исходного разбиения естественной структуре данных, а также сравнивать результаты, полученные разными методами или при различных значениях параметров [1, 3]. Определение функционалов качества главным образом основывается на таких критериях как компактность и обособленность кластеров. Однако в силу того, что различные понятия кластера и однородности заложены в каждый из

функционалов, они довольно часто демонстрируют совершенно разные несогласованные результаты.

В данной работе предлагается информационная технология, которая позволяет учитывать результаты различных функционалов качества одновременно с помощью методов теории принятия решений, что обеспечивает более точную оценку результатов. Технология состоит из следующих этапов:

1. Проводим предварительную обработку данных: отбор информативных признаков и стандартизацию.
2. Получаем разбиение объектов на кластеры разными методами или при различных значениях параметров, и рассматриваем их в качестве альтернатив.
3. Для каждой альтернативы вычисляем значение функционалов качества, которые считаем экспертами: сумма внутрикластерных дисперсий, сумма квадратов расстояний до центров кластеров, отношение среднего внутрикластерного и среднего межкластерного расстояний, сумма внутрикластерных расстояний

1 Описание предметной области

1.1 Понятие кластерного анализа

Исследователь часто стоит перед лицом огромной массы индивидуальных наблюдений. Возникает задача сведения множества характеристик к небольшому ряду обобщающих итогов, выражающему действительно существенное для явления. Но пока каждый вовлеченный в анализ признак остается отдельным самостоятельным элементом со своими характеристиками, число параметров, выражающих результаты обработки, не поддается уменьшению. Единственный путь к нему - либо в отсечении большинства признаков и возвращении к малоразмерным классическим задачам, либо в объединении признаков, в замене целых «гроздей» их одним, искусственно построенным на их основе. Так и появилось направление - «многомерный анализ».

В многомерном статистическом анализе образовались разделы, которые не изолированы, а проникают, переходят один в другой. Это кластерный анализ, метод главных компонент, факторный анализ. Наиболее ярко отражают черты многомерного анализа в классификации объектов кластерный анализ, а в исследовании связей - факторный анализ.

Кластерный анализ - это способ группировки многомерных объектов, основанный на представлении результатов отдельных наблюдений точками подходящего геометрического пространства с последующим выделением групп как «сгустков» этих точек (кластеров, таксонов). «Кластер» (cluster) в английском языке означает «сгусток», «гроздь винограда», «скопление звезд» и т.д. Данный метод исследования получил развитие в последние годы в связи с возможностью компьютерной обработки больших баз данных.

Кластерный анализ предполагает выделение компактных, удаленных друг от друга групп объектов, отыскивает «естественное» разбиение совокупности на области скопления объектов. Он используется, когда исходные данные представлены в виде матриц близости или расстояний

между объектами либо в виде точек в многомерном пространстве. Наиболее распространены данные второго вида, для которых кластерный анализ ориентирован на выделение некоторых геометрически удаленных групп, внутри которых объекты близки.

Выбор расстояния между объектами является узловым моментом исследования, от него во многом зависит окончательный вариант разбиения объектов на классы при данном алгоритме разбиения.

Существует большое количество алгоритмов кластерного анализа, их можно разделить по способу построения кластеров на 2 типа: эталонные и неэталонные. В процедурах эталонного типа на множестве объектов задается несколько исходных зон, с которых начинает работу алгоритм. Эталоны могут представлять собой первоначальное разбиение на классы, центр тяжести класса и др. После задания эталонов алгоритм производит классификацию, иногда меняя определенным способом эталоны.

К алгоритмам кластеризации, работающим по иному принципу, относятся иерархические алгоритмы кластерного анализа, процедура разрезания и др.

Задача кластерного анализа

Пусть множество I={Ii,l2,...,In} обозначает n объектов. Результат измерения i-й характеристики Ij объекта обозначают символом Xy, а вектор Xj=[xy] отвечает каждому ряду измерений (для j-го объекта). Таким образом, для множества I объектов исследователь располагает множеством векторов измерений X={X1, X2,...,Xn}, которые описывают множество I. Множество X может быть представлено как n точек в p-мерном евклидовом пространстве Ep.

Пусть m - целое число, меньшее чем n. Задача кластерного анализа заключается в том, чтобы на основании данных, содержащихся во множестве X, разбить множество объектов I на m кластеров (подмножеств) льл2,..., nm так, чтобы каждый объект Ij принадлежал одному и только одному

подмножеству разбиения и чтобы объекты, принадлежащие разным кластерам, были разнородными (несходными).

Решением задачи кластерного анализа является разбиение, удовлетворяющее некоторому условию оптимальности. Этот критерий может представлять собой некоторый функционал, выражающий уровни желательности различных разбиений и группировок. Этот функционал часто называют целевой функцией. Задачей кластерного анализа является задача оптимизации, т.е. нахождение минимума целевой функции при некотором заданном наборе ограничений. Примером целевой функции может служить, в частности, сумма квадратов внутригрупповых отклонений по всем кластерам.

Кластерный анализ включает в себя набор различных алгоритмов классификации. Общий вопрос, задаваемый исследователями во многих областях, состоит в том, как организовать наблюдаемые данные в наглядные структуры. Например, биологи ставят цель разбить животных на различные виды, чтобы содержательно описать различия между ними.

Задача кластерного анализа состоит в разбиении исходной совокупности объектов на группы схожих, близких между собой объектов. Эти группы называют кластерами. Другими словами, кластерный анализ – это один из способов классификации объектов по их признакам. Желательно, чтобы результаты классификации имели содержательную интерпретацию.

Результаты, полученные методами кластерного анализа, применяют в самых различных областях. В маркетинге – это сегментация конкурентов и потребителей. В психиатрии для успешной терапии является решающей правильная диагностика симптомов, таких как паранойя, шизофрения и т.д. В менеджменте важна классификация поставщиков, выявление схожих производственных ситуаций, при которых возникает брак. В социологии – разбиение респондентов на однородные группы. В портфельном инвестировании важно сгруппировать ценные бумаги по сходству в тенденции доходности, чтобы составить на основе полученных сведений о

фондовом рынке оптимального инвестиционного портфеля, позволяющего максимизировать прибыль от вложений при заданной степени риска. По сути, кластерный анализ хорошо зарекомендовал себя во всех сферах жизнедеятельности человека. В общем, всякий раз, когда необходимо классифицировать большое количество информации такого рода и представлять её в виде, пригодном для дальнейшей обработки, кластерный анализ оказывается весьма полезным и эффективным.

Кластерный анализ позволяет рассматривать достаточно большой объём информации и сильно сжимать большие массивы социально-экономической информации, делать их компактными и наглядными.

Большое значение кластерный анализ имеет применительно к совокупностям временных рядов, характеризующих экономическое развитие (например, общехозяйственной и товарной конъюнктуры). Здесь можно выделять периоды, когда значения соответствующих показателей были достаточно близкими, а также определять группы временных рядов, динамика которых наиболее схожа.

В задачах социально-экономического прогнозирования весьма перспективно сочетание кластерного анализа с другими количественными методами (например, с регрессионным анализом).

Кластерный анализ позволяет провести объективную классификацию любых объектов, которые охарактеризованы рядом признаков. Из этого можно извлечь ряд преимуществ.

1. Полученные кластеры можно интерпретировать, то есть описывать, какие же собственно группы существуют.

2. Отдельные кластеры можно выбраковывать. Это полезно в тех случаях, когда при наборе данных допущены определённые ошибки, в результате которых значения показателей у отдельных объектов резко отклоняются. При применении кластерного анализа такие объекты попадают в отдельный кластер.

3. Для дальнейшего анализа могут быть выбраны только те кластеры, которые обладают интересующими характеристиками.

Как и любой другой метод, кластерный анализ имеет определенные недостатки и ограничения. В частности, состав и количество кластеров зависит от выбираемых критериев разбиения. При сведении исходного массива данных к более компактному виду могут возникать определённые искажения, а также могут теряться индивидуальные черты отдельных объектов за счёт замены их характеристиками обобщённых значений параметров кластера.

Методы кластеризации

В настоящее время известно более сотни разных алгоритмов кластеризации. Их разнообразие объясняется не только разными вычислительными методами, но и различными концепциями, лежащими в основе кластеризации. Дать рекомендации для выбора того или иного метода кластеризации можно только в общих чертах, а основной критерий выбора – практическая полезность результата.

В пакете Statistica реализуются следующие методы кластеризации.

1. Иерархические алгоритмы – древовидная кластеризация. В основе иерархических алгоритмов лежит идея последовательной кластеризации. На начальном шаге каждый объект рассматривается как отдельный кластер. На следующем шаге некоторые из ближайших друг к другу кластеров будут объединяться в отдельный кластер.

2. Метод К-средних. Этот метод используется наиболее часто. Он относится к группе так называемых эталонных методов кластерного анализа. Число кластеров К задаётся пользователем.

3. Двухвходовое объединение. При использовании этого метода кластеризация проводится одновременно как по переменным (столбцам), так и по результатам наблюдений (строкам). Процедура двухвходового объединения производится в тех случаях, когда можно ожидать, что одновременная кластеризация по переменным и наблюдениям даст

возможность получить осмысленные результаты. Результатами процедуры являются описательные статистики по переменным и наблюдениям, а также двумерная цветная диаграмма, на которой цветом отмечаются значения данных. По распределению цвета можно составить представление об однородных группах.

Двувходовое объединение используется относительно редко. Но некоторые исследователи полагают, что этот метод – мощное средство разведочного анализа данных

Нормирование переменных для кластеризации

Разбиение исходной совокупности объектов на кластеры связано с вычислением расстояний между объектами и выбора объектов, расстояние между которыми наименьшее из всех возможных.

Выбор расстояния между объектами неоднозначен и в этом состоит основная сложность кластерного анализа

Наиболее часто используется привычное всем нам евклидово (геометрическое) расстояние. Эта метрика отвечает интуитивным представлениям о близости объектов в пространстве (как будто расстояния между объектами измерены рулеткой). Но для данной метрики на расстояние между объектами могут сильно влиять изменения масштабов (единиц измерения). Например, если один из признаков измерен в миллиметрах, а затем его значение переведены в сантиметры, евклидово расстояние между объектами сильно изменится. Это приведет к тому, что результаты кластерного анализа могут значительно отличаться от предыдущих.

Метод К-средних в программе Statistica

Метод K-средних (K-means) разбивает множество объектов на заданное число K различных кластеров, расположенных на возможно больших расстояниях друг от друга. Обычно, когда результаты кластерного анализа методом K-средних получены, можно рассчитать средние для каждого кластера по каждому измерению, чтобы оценить, насколько кластеры различаются друг от друга. В идеале вы должны получить сильно

различающиеся средние для большинства измерений, используемых в анализе. Значения F-статистики, полученные для каждого измерения, являются другим индикатором того, насколько хорошо соответствующее измерение дискриминирует кластеры.

При анализе и прогнозировании социально-экономических явлений исследователь довольно часто сталкивается с многомерностью их описания. Это происходит при решении задачи сегментирования рынка, построении типологии стран по достаточно большому числу показателей, прогнозирования конъюнктуры рынка отдельных товаров, изучении и прогнозировании экономической депрессии и многих других проблем.

Методы многомерного анализа - наиболее действенный количественный инструмент исследования социально-экономических процессов, описываемых большим числом характеристик. К ним относятся кластерный анализ, таксономия, распознавание образов, факторный анализ.

Кластерный анализ наиболее ярко отражает черты многомерного анализа в классификации, факторный анализ – в исследовании связи.

Иногда подход кластерного анализа называют в литературе численной таксономией, численной классификацией, распознаванием с самообучением и т.д.

Первое применение кластерный анализ нашел в социологии. Название кластерный анализ происходит от английского слова cluster – гроздь, скопление. Впервые в 1939 был определен предмет кластерного анализа и сделано его описание исследователем Трионом. Главное назначение кластерного анализа – разбиение множества исследуемых объектов и признаков на однородные в соответствующем понимании группы или кластеры. Это означает, что решается задача классификации данных и выявления соответствующей структуры в ней. Методы кластерного анализа можно применять в самых различных случаях, даже в тех случаях, когда речь идет о простой группировке, в которой все сводится к образованию групп по количественному сходству.

Большое достоинство кластерного анализа в том, что он позволяет производить разбиение объектов не по одному параметру, а по целому набору признаков. Кроме того, кластерный анализ в отличие от большинства математико-статистических методов не накладывает никаких ограничений на вид рассматриваемых объектов, и позволяет рассматривать множество исходных данных практически произвольной природы. Это имеет большое значение, например, для прогнозирования конъюнктуры, когда показатели имеют разнообразный вид, затрудняющий применение традиционных эконометрических подходов.

Кластерный анализ позволяет рассматривать достаточно большой объем информации и резко сокращать, сжимать большие массивы социально-экономической информации, делать их компактными и наглядными.

Важное значение кластерный анализ имеет применительно к совокупностям временных рядов, характеризующих экономическое развитие (например, общехозяйственной и товарной конъюнктуры). Здесь можно выделять периоды, когда значения соответствующих показателей были достаточно близкими, а также определять группы временных рядов, динамика которых наиболее схожа.

Кластерный анализ можно использовать циклически. В этом случае исследование производится до тех пор, пока не будут достигнуты необходимые результаты. При этом каждый цикл здесь может давать информацию, которая способна сильно изменить направленность и подходы дальнейшего применения кластерного анализа. Этот процесс можно представить системой с обратной связью.

В задачах социально-экономического прогнозирования весьма перспективно сочетание кластерного анализа с другими количественными методами (например, с регрессионным анализом).

Как и любой другой метод, кластерный анализ имеет определенные недостатки и ограничения: В частности, состав и количество кластеров зависит от выбираемых критериев разбиения. При сведении исходного

массива данных к более компактному виду могут возникать определенные искажения, а также могут теряться индивидуальные черты отдельных объектов за счет замены их характеристиками обобщенных значений параметров кластера. При проведении классификации объектов игнорируется очень часто возможность отсутствия в рассматриваемой совокупности каких-либо значений кластеров.

В кластерном анализе считается, что:

а) выбранные характеристики допускают в принципе желательное разбиение на кластеры;

б) единицы измерения (масштаб) выбраны правильно.

Выбор масштаба играет большую роль. Как правило, данные нормализуют вычитанием среднего и делением на стандартное отклонение, так что дисперсия оказывается равной единице.

Задача кластерного анализа заключается в том, чтобы на основании данных, содержащихся во множестве X, разбить множество объектов G на m (m – целое) кластеров (подмножеств) Q1, Q2, …, Qm, так, чтобы каждый объект Gj принадлежал одному и только одному подмножеству разбиения и чтобы объекты, принадлежащие одному и тому же кластеру, были сходными, в то время, как объекты, принадлежащие разным кластерам были разнородными.

Например, пусть G включает n стран, любая из которых характеризуется ВНП на душу населения (F1), числом M автомашин на 1 тысячу человек (F2), душевым потреблением электроэнергии (F3), душевым потреблением стали (F4) и т.д. Тогда X1 (вектор измерений) представляет собой набор указанных характеристик для первой страны, X2 - для второй, X3 для третьей, и т.д. Задача заключается в том, чтобы разбить страны по уровню развития.

Решением задачи кластерного анализа являются разбиения, удовлетворяющие некоторому критерию оптимальности. Этот критерий может представлять собой некоторый функционал, выражающий уровни

желательности различных разбиений и группировок, который называют целевой функцией. Например, в качестве целевой функции может быть взята внутригрупповая сумма квадратов отклонения:

$$W = \sum_{j=1}^{n} (x_j - \bar{x})^2 = \sum_{j=1}^{n} x_j^2 - \frac{1}{n}\left(\sum_{j=1}^{n} x_j\right)^2$$

где xj - представляет собой измерения j-го объекта.

Для решения задачи кластерного анализа необходимо определить понятие сходства и разнородности.

Понятно то, что объекты i-ый и j-ый попадали бы в один кластер, когда расстояние (отдаленность) между точками Xi и Xj было бы достаточно маленьким и попадали бы в разные кластеры, когда это расстояние было бы достаточно большим. Таким образом, попадание в один или разные кластеры объектов определяется понятием расстояния между Xi и Xj из Ep, где Ep - p-мерное евклидово пространство. Неотрицательная функция d(Xi , Xj) называется функцией расстояния (метрикой), если:

а) d(Xi , Xj) i 0, для всех Xi и Xj из Ep

б) d(Xi, Xj) = 0, тогда и только тогда, когда Xi = Xj

в) d(Xi, Xj) = d(Xj, Xi)

г) d(Xi, Xj) J d(Xi, Xk) + d(Xk, Xj), где Xj; Xi и Xk - любые три вектора из Ep.

Значение d(Xi, Xj) для Xi и Xj называется расстоянием между Xi и Xj и эквивалентно расстоянию между Gi и Gj соответственно выбранным характеристикам (F1, F2, F3, ..., Fp).

Наиболее часто употребляются следующие функции расстояний:

1. Евклидово расстояние

$$d2(Xi , Xj) = \left[\sum_{k=1}^{p}\left(x_{ki} - x_{kj}\right)^2\right]^{\frac{1}{2}}$$

2. l1 - норма

$$d1(Xi , Xj) = \left[\sum_{k=1}^{p}\left|x_{ki} - x_{kj}\right|\right]$$

1. Сюпремум - норма

$$d\Gamma(Xi , Xj) = \sup\left\{\left|x_{ki} - x_{kj}\right|\right\}$$

k = 1, 2, ..., p

2. lp - норма

$$dp(Xi , Xj) = \left[\sum_{k=1}^{p}\left|x_{ki} - x_{kj}\right|^p\right]^{1/p}$$

Евклидова метрика является наиболее популярной. Метрика l1 наиболее легкая для вычислений. Сюпремум-норма легко считается и включает в себя процедуру упорядочения, а lp - норма охватывает функции расстояний 1, 2, 3,.

Пусть n измерений X1, X2,..., Xn представлены в виде матрицы данных размером p г n:

$$x = \begin{pmatrix} x_{11} & x_{12} & \dots x_{1n} \\ x_{21} & x_{22} & \dots x_{2n} \\ x_{p1} & x_{p2} & \dots x_{pn} \end{pmatrix} = \left(X_1, X_2, \dots, X_n\right)$$

Тогда расстояние между парами векторов d(Xi , Xj) могут быть представлены в виде симметричной матрицы расстояний:

$$D = \begin{pmatrix} 0 & d_{12} & \dots d_{1n} \\ d_{21} & 0 & \dots d_{n} \\ d_{n1} & d_{n2} & \dots 0 \end{pmatrix}$$

Понятием, противоположным расстоянию, является понятие сходства между объектами Gi. и Gj. Неотрицательная вещественная функция S(Xi ; Xj) = Sij называется мерой сходства, если :

1) 0J S(Xi , Xj)<1 для Xi № Xj

2) S(Xi , Xi) = 1

3) S(Xi , Xj) = S(Xj , Xi)

Пары значений мер сходства можно объединить в матрицу сходства:

$$S = \begin{pmatrix} 1 & s_{12} & \dots & s_{1n} \\ s_{21} & 1 & \dots & s_{2n} \\ s_{n1} & s_{n2} & \dots & 1 \end{pmatrix}$$

Величину Sij называют коэффициентом сходства.

1.2 Известные методы применения кластерного анализа

Методы кластерного анализа можно разделить на две группы:

• иерархические;

• неиерархические.

Каждая из групп включает множество подходов и алгоритмов.

Используя различные методы кластерного анализа, аналитик может получить различные решения для одних и тех же данных. Это считается нормальным явлением. Рассмотрим иерархические и неиерархические методы подробно.

Суть иерархической кластеризации состоит в последовательном объединении меньших кластеров в большие или разделении больших кластеров на меньшие.

Иерархические агломеративные методы (Agglomerative Nesting, AGNES)Эта группа методов характеризуется последовательным объединением исходных элементов и соответствующим уменьшением числа кластеров.

В начале работы алгоритма все объекты являются отдельными кластерами. На первом шаге наиболее похожие объекты объединяются в кластер. На последующих шагах объединение продолжается до тех пор, пока все объекты не будут составлять один кластер. Иерархические дивизимные (делимые) методы (DIvisive ANAlysis, DIANA)Эти методы являются логической противоположностью агломеративным методам. В начале работы алгоритма все объекты принадлежат одному кластеру, который на последующих шагах делится на меньшие кластеры, в результате образуется последовательность расщепляющих групп.

Неиерархические методы выявляют более высокую устойчивость по отношению к шумам и выбросам, некорректному выбору метрики, включению незначимых переменных в набор, участвующий в кластеризации. Ценой, которую приходится платить за эти достоинства метода, является слово "априори". Аналитик должен заранее определить количество кластеров, количество итераций или правило остановки, а также некоторые другие параметры кластеризации. Это особенно сложно начинающим специалистам.

Если нет предположений относительно числа кластеров, рекомендуют использовать иерархические алгоритмы. Однако если объем выборки не

позволяет это сделать, возможный путь - проведение ряда экспериментов с различным количеством кластеров, например, начать разбиение совокупности данных с двух групп и, постепенно увеличивая их количество, сравнивать результаты. За счет такого "варьирования" результатов достигается достаточно большая гибкость кластеризации.

Иерархические методы, в отличие от неиерархических, отказываются от определения числа кластеров, а строят полное дерево вложенных кластеров.

Сложности иерархических методов кластеризации: ограничение объема набора данных; выбор меры близости; негибкость полученных классификаций.

Преимущество этой группы методов в сравнении с неиерархическими методами - их наглядность и возможность получить детальное представление о структуре данных.

При использовании иерархических методов существует возможность достаточно легко идентифицировать выбросы в наборе данных и, в результате, повысить качество данных. Эта процедура лежит в основе двухшагового алгоритма кластеризации. Такой набор данных в дальнейшем может быть использован для проведения неиерархической кластеризации.

Существует еще одни аспект, о котором уже упоминалось в этой лекции. Это вопрос кластеризации всей совокупности данных или же ее выборки. Названный аспект существенен для обеих рассматриваемых групп методов, однако он более критичен для иерархических методов. Иерархические методы не могут работать с большими наборами данных, а использование некоторой выборки, т.е. части данных, могло бы позволить применять эти методы.

Результаты кластеризации могут не иметь достаточного статистического обоснования. С другой стороны, при решении задач кластеризации допустима нестатистическая интерпретация полученных результатов, а также достаточно большое разнообразие вариантов понятия

кластера. Такая нестатистическая интерпретация дает возможность аналитику получить удовлетворяющие его результаты кластеризации, что при использовании других методов часто бывает затруднительным.

1) Метод полных связей.

Суть данного метода в том, что два объекта, принадлежащих одной и той же группе (кластеру), имеют коэффициент сходства, который меньше некоторого порогового значения S. В терминах евклидова расстояния d это означает, что расстояние между двумя точками (объектами) кластера не должно превышать некоторого порогового значения h. Таким образом, h определяет максимально допустимый диаметр подмножества, образующего кластер.

2) Метод максимального локального расстояния.

Каждый объект рассматривается как одноточечный кластер. Объекты группируются по следующему правилу: два кластера объединяются, если максимальное расстояние между точками одного кластера и точками другого минимально. Процедура состоит из n - 1 шагов и результатом являются разбиения, которые совпадают со всевозможными разбиениями в предыдущем методе для любых пороговых значений.

3) Метод Ворда.

В этом методе в качестве целевой функции применяют внутригрупповую сумму квадратов отклонений, которая есть ни что иное, как сумма квадратов расстояний между каждой точкой (объектом) и средней по кластеру, содержащему этот объект. На каждом шаге объединяются такие два кластера, которые приводят к минимальному увеличению целевой функции, т.е. внутригрупповой суммы квадратов. Этот метод направлен на объединение близко расположенных кластеров.

4) Центроидный метод.

Расстояние между двумя кластерами определяется как евклидово расстояние между центрами (средними) этих кластеров:

d2 ij = (`X –`Y)T(`X –`Y) Кластеризация идет поэтапно на каждом из n–1 шагов объединяют два кластера G и p, имеющие минимальное значение d2ij Если n1 много больше n2, то центры объединения двух кластеров близки друг к другу и характеристики второго кластера при объединении кластеров практически игнорируются. Иногда этот метод иногда называют еще методом взвешенных групп.

Наиболее известный метод представления матрицы расстояний или сходства основан на идее дендограммы или диаграммы дерева. Дендограмму можно определить как графическое изображение результатов процесса последовательной кластеризации, которая осуществляется в терминах матрицы расстояний. С помощью дендограммы можно графически или геометрически изобразить процедуру кластеризации при условии, что эта процедура оперирует только с элементами матрицы расстояний или сходства.

Существует много способов построения дендограмм. В дендограмме объекты располагаются вертикально слева, результаты кластеризации – справа. Значения расстояний или сходства, отвечающие строению новых кластеров, изображаются по горизонтальной прямой поверх дендограмм.

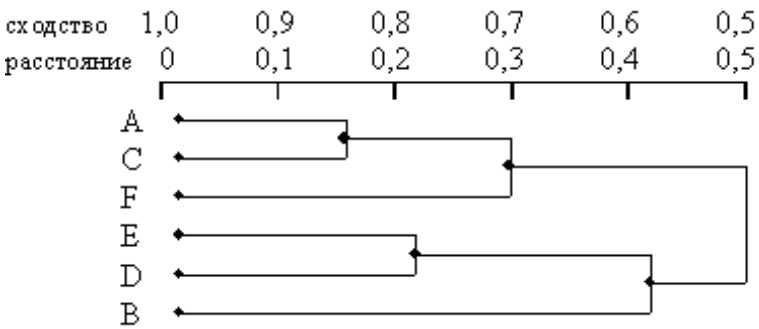

Рис. 1

На рисунке 1 показан один из примеров дендограммы. Рис 1 соответствует случаю шести объектов (n=6) и k характеристик (признаков). Объекты A и C наиболее близки и поэтому объединяются в один кластер на

уровне близости, равном 0,9. Объекты D и E объединяются при уровне 0,8. Теперь имеем 4 кластера:

(A, C), (F), (D, E), (B).

Далее образуются кластеры (A, C, F) и (E, D, B), соответствующие уровню близости, равному 0,7 и 0,6. Окончательно все объекты группируются в один кластер при уровне 0,5.

Вид дендограммы зависит от выбора меры сходства или расстояния между объектом и кластером и метода кластеризации. Наиболее важным моментом является выбор меры сходства или меры расстояния между объектом и кластером.

Число алгоритмов кластерного анализа слишком велико. Все их можно подразделить на иерархические и неиерархические.

Иерархические алгоритмы связаны с построением дендограмм и делятся на:

а) агломеративные, характеризуемые последовательным объединением исходных элементов и соответствующим уменьшением числа кластеров;

б) дивизимные (делимые), в которых число кластеров возрастает, начиная с одного, в результате чего образуется последовательность расщепляющих групп.

Алгоритмы кластерного анализа имеют сегодня хорошую программную реализацию, которая позволяет решить задачи самой большой размерности.

1.3 Постановка задачи на проектирование

Различают две основные группы методов кластеризации: иерархические и итеративные методы. Общим недостатком иерархических методов является трудоемкость алгоритмов при большом объеме данных анализа, а также, в зависимости от принятой меры расстояния, чувствительность к шуму [6]. В итеративных методах данные перемещаются между кластерами, чтобы минимизировать некоторый функционал качества.

Основные недостатки таких методов - чувствительность к начальной точке поиска и к шуму в данных, отыскание лишь локального минимума. Поскольку обе группы методов чувствительны к шуму данных, и в связи с тем, что кластеризация часто проводится по малым, зашумленным наборам данных и отсутствует априорная информация о форме кластеров, целесообразно при статистическом анализе ТП применить метод кластеризации с использованием ВП [14], который может позволить повысить качество кластеризации.

Задача кластеризации состоит в разделении данных измерений (образов) на группы (кластеры) с учетом их сходства, которое в метрическом пространстве обычно определяют через расстояние. Расстояние в выбранном подходе к кластеризации рассчитывается от образов к центрам кластеров, координаты которых заранее не известны и находятся одновременно с разделением данных на кластеры.

Важный путь обеспечения моделирования физико-химических процессов - управление процессами с использованием статистического анализа. Результаты считают статистически управляемым при влиянии на него только случайных воздействий. Но при неслучайных воздействиях, например, при износе инструментов, параметры ТП могут существенно отклоняться от целевых значений. Потому одной из основных задач управления ТП является распознавание его перехода в статисти-чески неуправляемое состояние.

Анализ последних достижений и публикаций. Одно из актуальных решений этой проблемы - использование контрольных карт (КК) по количественному признаку [1 - 5]. Эти карты формируют еще на стадии постановки изделия на производство для графического отображения состояния ТП - на них отмечены значения измеряемых параметров изделий во временной последовательности. После определения количества и способа получения выборок, типа КК при обработке результатов измерений выборку делят на однородные подгруппы (кластеры), по параметрам которых

определяют значения центральных линий (ЦЛ) и контрольных пределов (КП) КК. На основе такого анализа корректируют ТП, если значения параметров изделий по неслучайным причинам окажутся вне КП. При разделении выборки на кластеры используется процедура кластеризации [6], позволяющая автоматизировать этот процесс [1, 7, 8]. Одно из преимуществ такого подхода - анализ малых наборов данных измерений, позволяющий понизить стоимость и повысить оперативность отладки ТП, обусловливает существенный недостаток. Он связан с тем, что ошибки оператора, сбои оборудования при измерениях могут обусловливать неопределенность результатов измерений при малом объеме выборок, что эквивалентно их зашумленности. Влияние этих недостатков можно уменьшить, применяя кластеризацию с высоким качеством.

Однако у существующих методов кластериза-ции в указанных условиях низкое качество [6]. В работе [14] предложен метод кластеризации с ис-пользованием вейвлет-преобразования (ВП), позво-ляющий проводить кластеризацию при высоком уровне помех и малых объемах наборов данных с достаточно высоким качеством.

Цель работы - повысить качество кластеризации с использованием ВП для повышения помехоустойчивости статистического анализа моделирования физико-химических процессов. Для достижения этой цели решены задачи анализа методов кластеризации; разработки процедуры обработки данных при построении КК на примере X - R карты; оценки качества кластеризации.

2 Выбор средств разработки и проектирование программного средства

2.1 Выбор языка и среды программирования

Программное обеспечение (ПО) представляет собой совокупность компьютерных программ, описаний и инструкций по их применению на ЭВМ. ПО делится на два класса:

- Общее ПО (операционные системы, операционные оболочки, компиляторы, интерпретаторы, программные среды для разработки прикладных программ, СУБД, сетевые программы и так далее);
- Специальное ПО (Совокупность прикладных программ, разработанных для конкретных задач в рамках функциональных подсистем).

Программное обеспечение (ПО) — совокупность программ для реализации целей и задач автоматизированной системы. [2]

ПО делится на два вида: общее (операционные системы, операционные оболочки, компиляторы, интерпретаторы, программные среды для разработки прикладных программ, СУБД, сетевые программы и т.д.) и специальное (совокупность прикладных программ, разработанных для конкретных задач в рамках функциональных подсистем, и контрольные примеры). [2]

Для функционирования и использования программы необходима операционная система. Операционные системы осуществляют управление работой персональных компьютеров, их ресурсами, запускают на выполнение различные прикладные программы, выполняют всевозможные вспомогательные действия по запросу пользователя. ОС подразделяются на однопользовательские, многопользовательские, и сетевые. К факторам, влияющим на выбор конкретной ОС, относятся:

- необходимое число поддерживаемых программных продуктов,
- требования к аппаратным средствам,
- требование поддержки сетевой технологии,

- наличие справочной службы для пользователя,
- быстродействие,
- наличие дружественного интерфейса и простота использования.

Для создания программ под Windows существует огромное количество интегрированных сред разработки. К таковым можно отнести: Visual Basic, Visual C+ +, Delphi, C+ + Builder.

С появлением средств быстрой разработки приложений (RAD – rapid application development) появилась возможность программировать с помощью готовых компонентов и шаблонов. Сравним Delphi и C++Builder, выберем среду программирования для автоматизированной системы.

Система визуального программирования Delphi, разработана компанией Borland International на базе языка Object Pascal. Объектно-ориентированный подход к созданию компонент был серьезным шагом вперед. Дополнительный выигрыш обеспечивался за счет нормальной компиляции, обеспечивающей получение более производительных программ. Первые две версии Delphi довольно быстро завоевали симпатии не только у вузовских аудиторий, где Pascal пользовался особым уважением, но и среди профессионалов. Это подтолкнуло одно из подразделений фирмы Borland International на перенос визуальной технологии в среду C++. Так, почти одновременно с появлением Delphi 2.0 на рынке появилась первая версия Borland C++ Builder (BCB). Основу первой версии BCB составила библиотека визуальных компонент VCL (Visual Component Library), перенесенная без изменений из Delphi 2. Интерфейсы сред Delphi и BCB похожи друг на друга как близнецы, да и большая часть BCB была разработана на языке Object Pascal в среде Delphi. Благодаря своему происхождению система BCB оказалась двуязычной. Кроме своего основного языка программирования она позволяет практически без каких-либо доработок использовать формы, объекты и модули, разработанные в среде Delphi. Чтобы еще больше расширить сферу влияния среды BCB, ее авторы в последующих версиях обеспечили возможность использования

библиотеки классов MFC (Microsoft Foundation Classes), разработанной фирмой Microsoft.

Delphi 7 2010 г – это прекрасный инструмент, но в то же время и сложная программная среда, состоящая из многих элементов. Включает в себя новый интерфейс Galileo, а так же interbase server и desktop, remote debugger server, Model Maker, Install Shield

Особенно привлекательными в Delphi являются такие изначальные возможности, как объектно-ориентированный подход к программированию, основанный на формах, ее высоко производительный (32-разрядный оптимизирующий компилятор), замечательная поддержка баз данных, тесная интеграция с программированием под Windows и технология компонентов. Но самой важной частью является язык Object Pascal, на фундаменте которого строится все остальное.

Delphi 7 2010 г обладает открытой архитектурой, полностью поддерживает технологии Microsoft OLE Automation, ActiveX, ODBC. Компилятор позволяет иметь доступ ко всем ресурсам операционных систем, реализующих интерфейс Win32 (Windows XP и Windows 7).

Программы Delphi используют объектно-ориентированную структуру под названием VCL – Visual Component Library (Библиотека Визуальных Компонентов). Именно VCL поднимает быструю разработку приложений на новый уровень. Можно расширить свои возможности за счет создания своих собственных компонентов. К тому же независимые поставщики уже создали множество компонентов такого рода.

Delphi 7 2010г имеет много других улучшений IDE, расширенную поддержку баз данных (по специальным наборам данных ADO и InterBase), улучшенную версию MIDAS с поддержкой Интернета, инструмент управления версиями TeamSours, возможности перевода, концепцию фреймов и большое количество новых компонентов.

В основе идеологии Delphi лежат технологии визуального проектирования и программирование процедур обработки событий,

применение которых позволяет существенно сократить время разработки и облегчить процесс создания приложений.

C++Builder 6 2010 – очередная версия системы объектно-ориентированного программирования для операционных систем Windows 2000, Windows XP, Windows Vista и Windows 7. Интегрированная среда системы (Integrated Development Environment, IDE) обеспечивает ускорение визуального проектирования, а также продуктивность многократно используемых компонентов в сочетании с усовершенствованными инструментами и разномасштабными средствами доступа к базам данных.

Стандарты пользовательских интерфейсов меняются и развиваются так же быстро, как и операционные системы. Открытость среды IDE позволяет настраивать ее с учетом наиболее модных тенденций в области графических интерфейсов. Визуальный интерфейс сочетает в себе простоту использования для новичка и богатство возможностей для профессионала.

Система C++Builder 6 2010 может быть использована везде, где требуется дополнить существующие приложения (как прикладные, так и системные) расширенным стандартом языка C++, повысить быстродействие и надежность программ, придать пользовательскому интерфейсу качество профессионального уровня, позволяет быстро создавать использующие сенсорный ввод данных графические интерфейсы и приложения для КПК, сенсорных панелей и автономных общедоступных систем и модернизировать существующие приложения с минимальным добавлением кода или без него.

В контексте C++Builder RAD подразумевает не только реальное ускорение типичного цикла «редактирование – компиляция – компоновка – прогон – отладка», но и придает создаваемым проектам изящество компонентной модели.

Уникальная среда разработки C++Builder 6 2010 IDE Insight позволяет обращаться ко всем возможностям, параметрам и компонентам интегрированной среды разработки, не тратя время на их поиск в меню и диалоговых окнах; обозреватель классов, обеспечивающий управление

классами в проекте и быстрый переход между ними; объединяет Дизайнер форм, Инспектор объектов, Палитру компонентов, Менеджер проектов и полностью интегрированные Редактор кода и Отладчик – основные инструменты RAD.

В Builder C++ 6 2010 добавлены поддерживаемые отладчиком средства визуализации данных, упрощающие отладку, позволяя настраивать отображение типов данных в отладчике; поддерживаемые отладчиком средства управления потоками, обеспечивающие заморозку, разморозку и изоляцию потоков, а также установку контрольных точек для выбранных потоков, что упрощает разрешение проблем; Builder C++ 6 2010 содержит новые параметры отладчика: Scroll new events into view («Прокрутка новых событий в представлении») и Ignore non-user breakpoints («Игнорирование не пользовательских контрольных точек»), как на уровне исходных инструкций, так и на уровне ассемблерных команд - в расчете удовлетворить высокие требования программистов-профессионалов.

Оптимизирующий 32-разрядный компилятор построен по оригинальной и проверенной адаптивной технологии, обеспечивающей исключительно надежную и быструю оптимизацию, как объема выходного исполняемого кода, так и требуемой памяти.

Визуальная разработка методом «перетаскивания» (drag-and-drop) многократно упрощает и ускоряет трудоемкий процесс программирования приложений СУБД. В основе объектно-ориентированного взаимодействия клиент – сервер лежит понятие наборов данных (dataset) – таблиц, запросов, хранимых процедур – основных сущностей БД, которыми оперируют компоненты доступа. Широкий выбор компонентов визуализации и редактирования позволяет легко изменять вид представления наборов данных. C++Builder использует проводник баз данных и масштабируемый словарь данных для того, чтобы автоматически настроить средства отображения и редактирования применительно к специфике вашей информации.

Наряду с дизайнерами и художниками-оформителями к прикладным разработкам в ключевых областях сети все чаще привлекаются профессионалы-программисты. Качественное приложение способно динамически выбирать информацию с сервера и предоставлять ее в формате, удобном для потребителей разного уровня, так, чтобы они смогли составить свое заключение и принять адекватное решение в кратчайший срок. C++Builder 6 2010 полностью удовлетворяет этим требованиям, обеспечивая:

- высокую надежность, степень интеграции и качество управления;
- быстрое визуальное проектирование эффективных приложений для переработки больших объемов информации;
- поддержку механизмов принятия решения и доступа к удаленным базам данных.

C++Builder предоставляет свою мощь и широкие возможности языка C++ всему семейству систем объектно-ориентированного программирования. Система C++Builder может быть использована везде, где требуется дополнить существующие приложения расширенным промышленным стандартом языка C++, повысить быстродействие и придать пользовательскому интерфейсу профессиональный облик.

Все компоненты, формы и модули данных, которые накопили программисты, работающие в Delphi, могут быть многократно использованы в приложениях C++Builder без каких бы то ни было изменений. C++Builder идеально подойдет тем разработчикам, которые предпочитают выразительную мощность языка C++, однако хотят сохранить продуктивность Delphi. Уникальная взаимосвязь этих систем программирования позволяет при создании приложения без труда переходить из одной среды разработки в другую.

Какую систему выбрать? Delphi использует язык Объектный Паскаль, который преподается во многих специализированных школах и учебных институтах. Система C++Builder, как следует из названия, построена на языке C++, который наиболее распространен в крупных фирмах,

занимающихся разработкой математического обеспечения профессионального уровня. C++Builder является более мощной системой, которая идеально подходит разработчикам, которые предпочитают выразительную мощь языка C++, однако хотят сохранить продуктивность Delphi. Сопровождение программных продуктов, написанных в системе Delphi, проще чем написанных в C++Builder.

Проведем выбора среды программирования методом экспертного оценивания. Выделим критерии оценки среды программирования. Важность каждого из представленных критериев была оценена экспертами по 100 бальной шкале. Исходя из полученных данных, находится средний балл и коэффициент относительной важности критерия. Результаты экспертизы представлены в таблицах 2.1 –2.2.

Таблица 2.1 Результаты экспертизы сред разработки, первый этап

Функция	Эксперт 1	Эксперт 2	Эксперт 3	Средний балл по 100 бальной шкале	Коэффициент относительной важности
Стоимость	75	90	85	83	13,7
Простота сопровождения	80	75	86	80	13,2
Временные затраты на разработку	90	85	95	90	14,8
Быстродействие	89	95	90	91	15
Удобный дизайн	85	81	90	85	14
Мощность пакета	75	92	84	84	13,8
Возможности языка	100	89	94	94	15,5
Сумма				**606**	**100,0%**

Таблица 2.2 Результаты экспертизы сред разработки, второй этап

Функция	Коэффициент относительной важности	Среда программирования	
		Delphi	C++ Builder
Стоимость	13,7	+	+
Простота сопровождения	13,2	+	-
Временные затраты на	14,8	+	-

разработку			
Быстродействие	15	+	+
Удобный дизайн	14	+	+
Мощность пакета	13,8	+	+
Возможности языка	15,5	+	+
Сумма	**100,0%**	**100**	**72**

Учитывая все вышесказанное и результаты анализа экспертным оцениванием можно сделать выбор среды программной разработки в пользу Delphi, который обеспечивает чрезвычайно высокую производительность и удобство использования.

Для разработки будем использовать среду разработки RAD Studio Professional.

RAD Studio Professional включает высокопроизводительные интегрированные среды разработки собственных приложений Windows и .NET. Интегрированная среда разработки Delphi и C++Builder с поддержкой Unicode для разработки собственных приложений включает сотни готовых компонентов и функций, к числу которых относятся рефакторинг, дополнение кода, выделение синтаксиса, интерактивные шаблоны, полнофункциональная отладка и тестирование модулей. Интегрированная среда разработки Delphi Prism поддерживает разработку приложений для платформы .NET, включая поддержку новейших технологий .NET. — Подключение к локальным базам данных InterBase®, Blackfish™ SQL и MySQL в Delphi и C++Builder. — Подключение к базам данных в Visual Studio с помощью ADO.NET, включая подключение к локальным базам данных InterBase и Blackfish в Delphi Prism. — Развертывание Blackfish SQL в системах с одним пользователем и размером базы данных 512 МБ. — Библиотека VCL для Интернета с ограничением числа подключений (не более пяти).

2.2 Описание основных алгоритмов

Базовая последовательность действий программы представлена алгоритмом на рисунке 2.1.

Рисунок 2.1 Базовая последовательность действий программы

Метод идентификации (определения) кластеров представлен алгоритмом на рисунке 2.2.

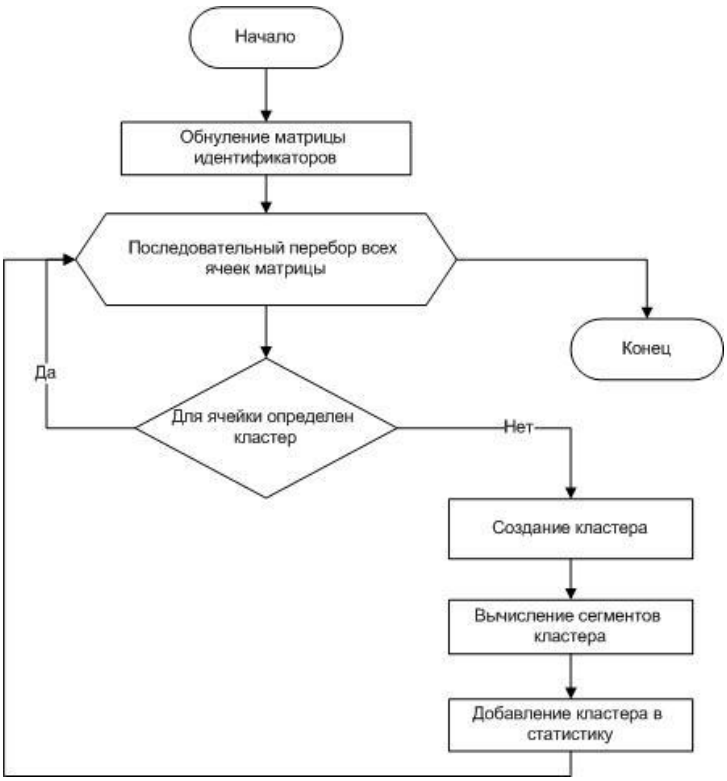

Рисунок 2.2 Метод идентификации (определения) кластеров (алгоритм)

Программный код, выполняющий данную последовательность, приведен ниже.

```
/// <summary>
/// Метод определения кластеров (данный метод заполняет матрицу "FClusterID" и список "FClusterList")
/// </summary>
procedure TClusterManager.GetClusters;
var
  Row, Col, ID: Cardinal;
  Cluster: TCluster;
begin
  // Обнуляем счетчик кластеров
  ID := 0;
  // Очищаем статистику по кластерам
  FClusterStat.Clear;
  // Очищаем список кластеров
  FClusterList.Clear;
  // Удаляем матрицу идентификаторов для кластеров
  if (Assigned(FClusterID)) then
    FClusterID.Free;
  // Делаем копию исходной матрицы (создаем матрицу идентификаторов)
  FClusterID := Matrix.Clone;
  // Заполняем матрицу идентификаторов нулями
  FClusterID.Fill(0);
  // Цикл обхода всех ячеек исходной матрицы с целью идентификации кластеров
  for Row := 1 to Matrix.RowCount do
  begin
    for Col := 1 to Matrix.ColCount do
    begin
      // Если для текущей ячейки исходной матриц не определен кластер, то запускаем рекурсивную процедуру вычисления кластеров
      if (ClusterID.Matrix[Row, Col] = 0) then
      begin
        // Выполняем приращение счетчика идентификаторов
        Inc(ID);
        // Создаем кластер
        Cluster := TCluster.Create;
        // Присваиваем кластеру идентификатор
        Cluster.ID := ID;
        // Присваиваем кластеру тип (A - 0, B - 1)
        Cluster.ClusterType := Matrix.Matrix[Row, Col];
        // Добавляем кластер в список кластеров
        FClusterList.AddObject('', Cluster);
        // Вызываем рекурсивную процедуру вычисления кластера
        FillCluster(Row, Col, Cluster);
        // После вычисления кластера запускаем процедуру расчета статистики
        StatAdd(Cluster);
      end;
    end;
  end;
end;
```

Рекурсивная процедура вычисления кластера (вызывается из GetClusters), рисунок 2.3.

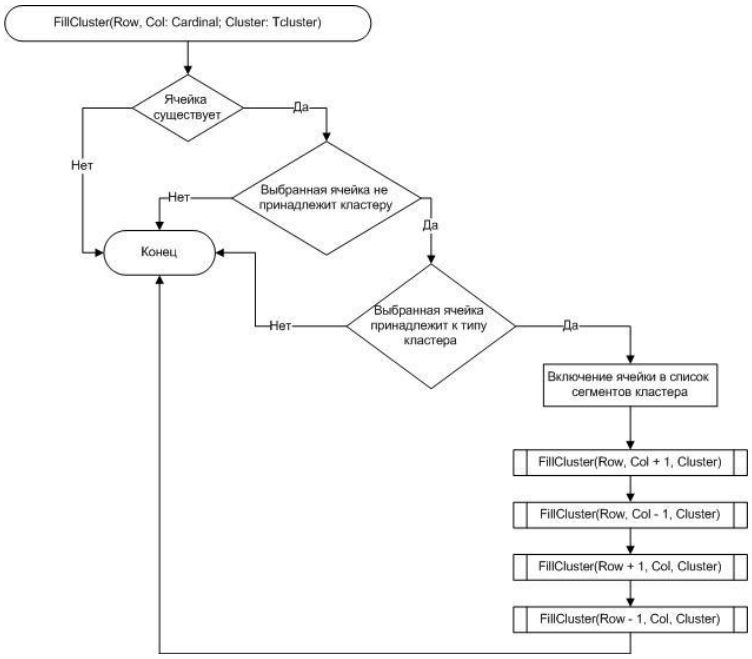

Рисунок 2.3 Алгоритм рекурсивной процедуры вычисления кластера

```
procedure TClusterManager.FillCluster(Row, Col: Cardinal; Cluster: TCluster);
var
   ClusterSegment: TClusterSegment;
begin
   // Если ячейка не существует, то выходим из процедуры
   if Matrix.TestCell(Row, Col, True) = False then
   begin
      Exit;
   end;
   // Если выбранная ячейка уже принадлежит кластеру, то выходим из процедуры
   if (ClusterID.Matrix[Row, Col] <> 0) then
   begin
      Exit;
   end;
   // Если выбранная ячейка не принадлежит к типу кластера, то выходим из процедуры
   if (Matrix.Matrix[Row, Col] <> Cluster.ClusterType) then
   begin
      Exit;
   end;
   // Если все проверки успешно пройдены, то создаем сегмент кластера
   ClusterSegment := TClusterSegment.Create;
   try
      // Определяем строку сегмента кластера
      ClusterSegment.Row := Row;
      // Определяем столбец сегмента кластера
      ClusterSegment.Col := Col;
      // Присваиваем данной ячейке идентификатор кластера в матрице идентификаторов
      ClusterID.Matrix[Row, Col] := Cluster.ID;
      // Добавляем сегмент в список сегментов кластера
      Cluster.Segments.AddObject('', ClusterSegment);

      // Запускаем рекурсивную процедуру идентификации кластера соседней ячейки
      FillCluster(Row, Col + 1, Cluster);
      // Запускаем рекурсивную процедуру идентификации кластера соседней ячейки
      FillCluster(Row - 1, Col, Cluster);
      // Запускаем рекурсивную процедуру идентификации кластера соседней ячейки
      FillCluster(Row + 1, Col, Cluster);
      // Запускаем рекурсивную процедуру идентификации кластера соседней ячейки
      FillCluster(Row, Col - 1, Cluster);

   except
      // в случае ошибки удаляем сегмент
      ClusterSegment.Free;
      // и отправляем исключение дальше
      raise;
   end;
end;
```

Процедура вычисления статистики (вызывается из GetClusters)

```
procedure TClusterManager.StatAdd(Cluster: TCluster);
var
   I, Size: Integer;
   CS: TClusterStat;

begin
   // Вычисляем размер кластера
   Size := Cluster.Segments.Count;

   // Выполняем обход списка статистики
   for I := FClusterStat.Count - 1 downto 0 do
   begin
      // Извлекаем элемент статистики из списка
      CS := FClusterStat.Objects[I] as TClusterStat;
      // Если размеры совпадают
      if CS.Size = Size then
      begin
         // в зависимости от типа кластера (A или B) плюсуем количество
         if Cluster.ClusterType = 0 then
            CS.AClusterCount := CS.AClusterCount + 1
         else
            CS.BClusterCount := CS.BClusterCount + 1;
         // выходим из процедуры
         Exit;
      end;
   end;

   // В случае, если такого размера нет в списке статистики
   // Создаем элемент статистики
   CS := TClusterStat.Create;
   // Определяем его размер
   CS.Size := Size;
   // в зависимости от типа кластера (A - 0 или B - 1) плюсуем количество
   if Cluster.ClusterType = 0 then
      CS.AClusterCount := CS.AClusterCount + 1
   else
      CS.BClusterCount := CS.BClusterCount + 1;
   // Добавлем размер в список статистики
   FClusterStat.AddObject('', CS);
end;
```

2.3 Разработка структуры программы

Главным классом в системе является TClusterManager. Данный класс реализует функции распознавания (идентификации) кластеров. Механизм идентификации кластеров основан на использовании матриц. Матричные вычисления реализует класс TMatrix. Исходная матрица (свойство «Matrix», класса «TClusterManager»), состоящая из нулей и единиц, реализуется экземпляром класса TMatrix, как и матрица принадлежности ячеек (свойство «ClusterID», класса «TClusterManager»).

Идентификацию кластеров выполняет метод «GetClusters» класса «TClusterManager». В цикле, начиная с первой ячейки, мы выполняем обход элементов исходной матрицы (свойство «Matrix», класса «TClusterManager»)

и для каждой ячейки (для которой ещё не определен кластер (в матрице «ClusterID», класса «TClusterManager» стоит значение «0»)) выполняем рекурсивную процедуру, которая определяет для текущей ячейки номер кластера и проверяет соседние ячейки на возможность включения в кластер.

Во время идентификации, для каждого найденного кластера создается объект (экземпляр класса «TCluster») который в полной мере описывает кластер. Объект содержит числовой идентификатор найденного кластера («ID»), тип кластера («ClusterType») (значение, на основе которого построен кластер, 0 или 1) и состав кластера (Список элементов, класс «TClusterSegment»).

Ниже приведены схема и подробные описания всех классов.

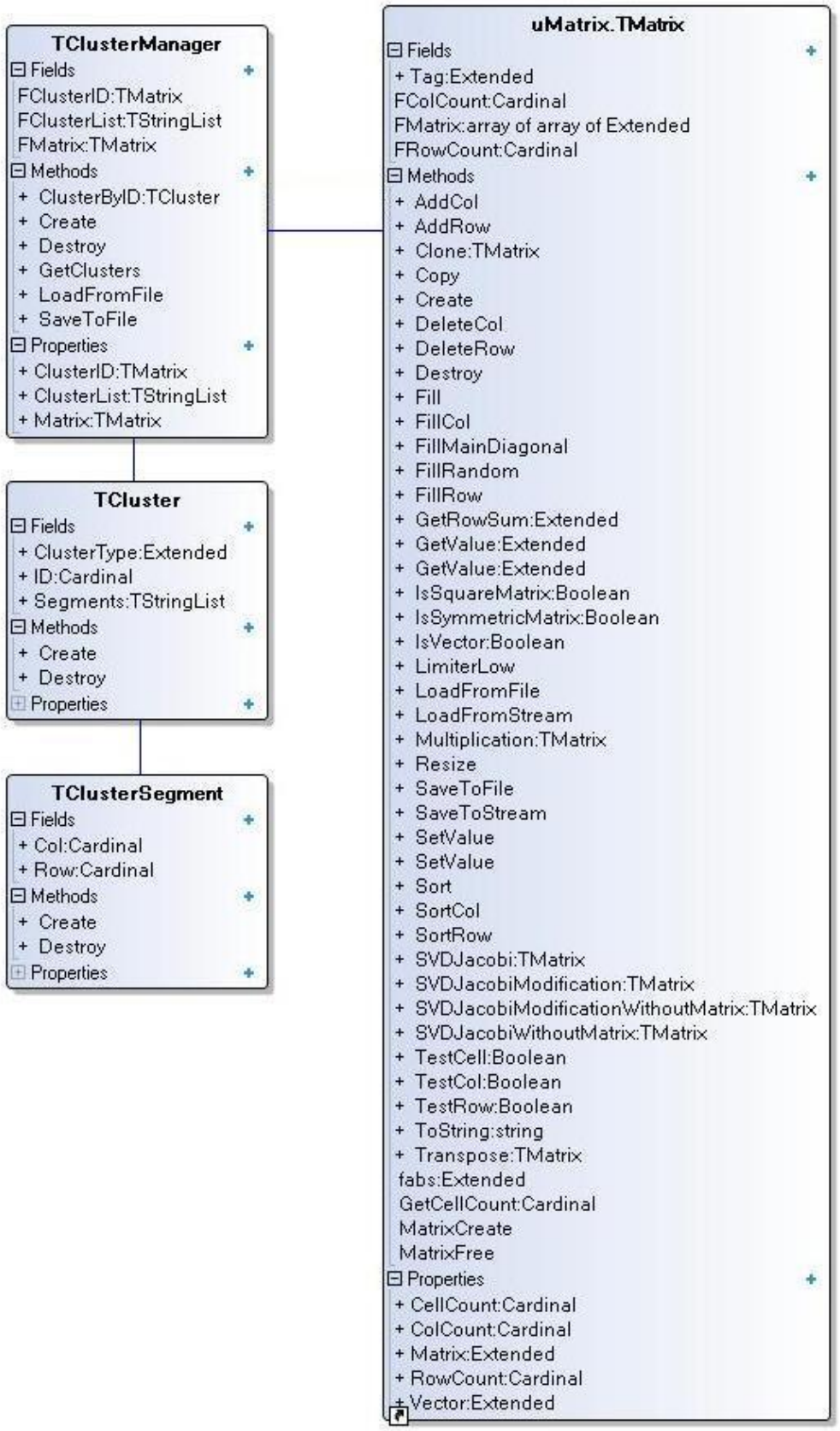

Рисунок 2.4 Диаграмма классов

TClusterSegment

Класс "Сегмент кластера". Служит для хранения точек одного кластера.

Field Summary

public **Cardinal**	**Col** Столбец матрицы, на которой находится точка
public **Cardinal**	**Row** Строка матрицы, на которой находится точка

Constructor Summary	
Create() Конструктор класса	

Method Summary	
public **Sub**	**Destroy**() Деструктор класса

TCluster

Класс "Кластер". Служит для хранения сведений одного кластера.

Field Summary	
public **Extended**	**ClusterType** Тип кластера (значение, по которому происходит группирование точек в кластер)
public **Cardinal**	**ID** Идентификатор кластера
public **TStringList**	**Segments** Список сегментов кластера. (список объектов класса TClusterSegment)

Constructor Summary	
Create() Конструктор класса	

Method Summary	
public **Sub**	**Destroy**() Деструктор класса

TClusterManager

Класс, осуществляющий идентификацию кластеров

Field Summary	
internal **uMatrix.TMatrix**	**FClusterID** Матрица идентификатором кластеров (данная матрица для каждой точки исходной матрицы определяет идентификатор кластера (или принадлежность точки к определенному кластеру))
internal **TStringList**	**FClusterList** Список кластеров (список объектов класса TCluster)
internal **uMatrix.TMatrix**	**FMatrix** Исходная матрица (содержит единицы и нули, которые будут сгруппированы по кластерам)

Property Summary	
public	**ClusterID**

uMatrix.TMatrix	Матрица идентификатором кластеров (данная матрица для каждой точки исходной матрицы определяет идентификатор кластера (или принадлежность точки к определенному кластеру))
public **TStringList**	**ClusterList** Список кластеров (список объектов класса TCluster)
public **uMatrix.TMatrix**	**Matrix** Исходная матрица (содержит единицы и нули, которые будут сгруппированы по кластерам)

Constructor Summary	
Create() Конструктор класса	

Method Summary	
public **function TCluster**	**ClusterByID**(ID: Cardinal) Метод поиска кластера по его идентификатору
public **Sub**	**Destroy**() Деструктор класса
public **Sub**	**GetClusters**() Метод определения кластеров (данный метод заполняет матрицу "FClusterID" и список "FClusterList")
public **Sub**	**LoadFromFile**(FileName: string) Метод загрузки исходной матрицы из файла
public **Sub**	**SaveToFile**(FileName: string) Метод сохранения исходной матрицы в файл

TMatrix

Класс для работы с матрицами и векторами

Field Summary	
internal **Cardinal**	**FColCount** Число столбцов матрицы
internal **array of array of Extended**	**FMatrix** Матрица
internal **Cardinal**	**FRowCount** Количество строк матрицы
public **Extended**	**Tag** Переменная для нужд прикладных программистов

Property Summary	
public **Cardinal**	**CellCount** Количество ячеек (только для векторов)
public **Cardinal**	**ColCount** Количество столбцов
public **Extended**	**Matrix** Значения матрицы
public **Cardinal**	**RowCount** Количество строк
public	**Vector**

Extended	Значение вектора

Constructor Summary	
Create(RowCount: Cardinal; ColCount: Cardinal) Конструктор класса	

Method Summary	
public **Sub**	**AddCol**() Метод добавления столбца матрицы
public **Sub**	**AddRow**() Метод добавления строки матрицы
public **function TMatrix**	**Clone**() Метод клонирования матрицы
public **Sub**	**Copy**(Source: TMatrix) Метод копирования значений из другой матрицы
public **Sub**	**DeleteCol**(Col: Cardinal) Метод удаления столбца матрицы
public **Sub**	**DeleteRow**(Row: Cardinal) Метод удаления строки матрицы
public **Sub**	**Destroy**() Деструктор класса
protected internal **function Extended**	**fabs**(X: Extended) Метод вычисления модуля
public **Sub**	**Fill**(Value: Extended) Метод заполнения матрицы
public **Sub**	**FillCol**(Col: Cardinal; Value: Extended) Метод заполнения столбца матрицы
public **Sub**	**FillMainDiagonal**(Value: Extended) Метод заполнения главной диагонали каким-либо значением (только для квадратных матриц)
public **Sub**	**FillRandom**(AFrom: Integer; ATo: Integer; Symmetric: Boolean) Метод заполнения матрицы случайными числами
public **Sub**	**FillRow**(Row: Cardinal; Value: Extended) Метод заполнения строки матрицы
internal **function Cardinal**	**GetCellCount**() Количество ячеек (только для векторов)
public **function Extended**	**GetRowSum**(Row: Cardinal) Метод вычисления суммы в строке
public **function Extended**	**GetValue**(Row: Cardinal; Col: Cardinal) Метод чтения значения ячейки матрицы
public **function Extended**	**GetValue**(Cell: Cardinal) Метод чтения значения элемента вектора
public **function Boolean**	**IsSquareMatrix**() Возвращает True, если матрица квадратная
public **function Boolean**	**IsSymmetricMatrix**() Метод проверки матрицы на симметрию
public **function Boolean**	**IsVector**() Возвращает True, если матрица - вектор

public **Sub**	**LimiterLow**(Limit: Extended; Value: Extended) Меторд заменяет все значения матрицы значением "Value" если они ниже "Limit".
public **Sub**	**LoadFromFile**(FileName: string) Метод загрузки матрицы из файла
public **Sub**	**LoadFromStream**(S: TStream) Метод загрузки матрицы из потока
internal **Sub**	**MatrixCreate**() Метод инициализации матрицы
internal **Sub**	**MatrixFree**() Метод разрушения матрицы
public **function TMatrix**	**Multiplication**(B: TMatrix) Умножение матриц R = A * B, где A - текущая матрица (Self)
public **Sub**	**Resize**(RowCount: Cardinal; ColCount: Cardinal) Метод изменения размера матрицы
public **Sub**	**SaveToStream**(S: TStream) Метод сохранения матрицы в поток
public **Sub**	**SetValue**(Row: Cardinal; Col: Cardinal; Value: Extended) Метод задающий значение ячейки матрицы
public **Sub**	**SetValue**(Cell: Cardinal; Value: Extended) Метод задающий значение элемента вектора
public **Sub**	**Sort**(SortType: TMatrixSortType) Сортировка (только для вектора)
public **Sub**	**SortCol**(SortType: TMatrixSortType ; Col: Cardinal) Метод сортировки столбцов матрицы
public **Sub**	**SortRow**(SortType: TMatrixSortType ; Row: Cardinal) Метод сортировки строк матрицы
public **function TMatrix**	**SVDJacobi**(Precision: Extended; IterationLimit: Cardinal) Расчет сингулярных чисел (SVD) методом Якоби (оригинальный алгоритм)
public **function TMatrix**	**SVDJacobiModification**(Precision: Extended; IterationLimit: Cardinal) Расчет сингулярных чисел (SVD) методом Якоби (Модификация выбора пары (P, Q))
public **function TMatrix**	**SVDJacobiModificationWithoutMatrix**(Precision: Extended; IterationLimit: Cardinal) Расчет сингулярных чисел (SVD) методом Якоби (Модификация выбора пары (P, Q) без активного использования матриц)
public **function TMatrix**	**SVDJacobiWithoutMatrix**(Precision: Extended; IterationLimit: Cardinal) Расчет сингулярных чисел (SVD) методом Якоби (оригинальный алгоритм) без активного использования матриц
public **function Boolean**	**TestCell**(Row: Cardinal; Col: Cardinal; WithoutException: Boolean) Метод проверки допустимости значения номера строки и столбца
public **function Boolean**	**TestCol**(Col: Cardinal; WithoutException: Boolean) Метод проверки допустимости значения номера столбца
public **function Boolean**	**TestRow**(Row: Cardinal; WithoutException: Boolean) Метод проверки допустимости значения номера строки
public **function string**	**ToString**() Перевод матрицы в текст
public **function TMatrix**	**Transpose**() Метод возвращает транспнированную матрицу

3 Анализ полученного решения

3.1 Описание работы программы

1. Пользователь запускает программу (файл «ClusterAnalyzer.exe»)

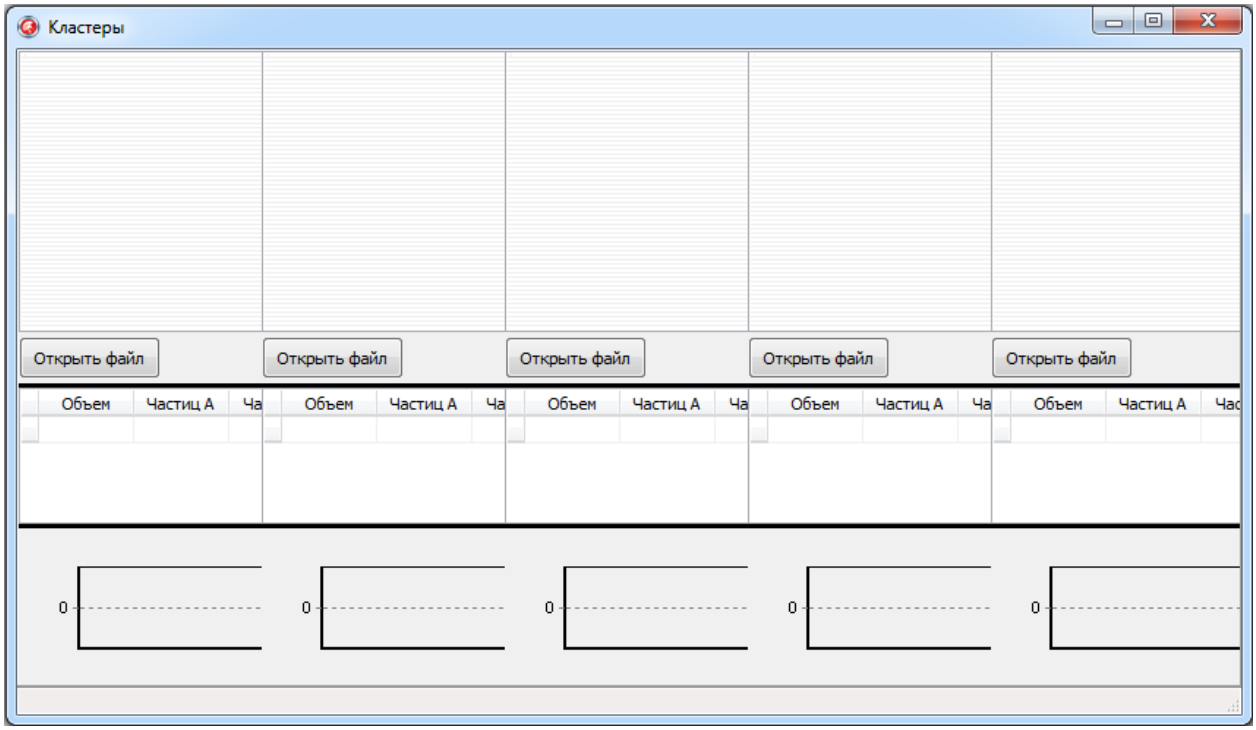

Рисунок 3.1 Главная форма программы

2. Пользователь открывает файлы исходных данных (кнопки «Открыть»)

3. Пользователь выбирает файлы (после нажатия на кнопку «Открыть» появляется диалоговое окно для выбора файлов исходных данных)

4. Пользователь получает результаты анализа файла исходных данных

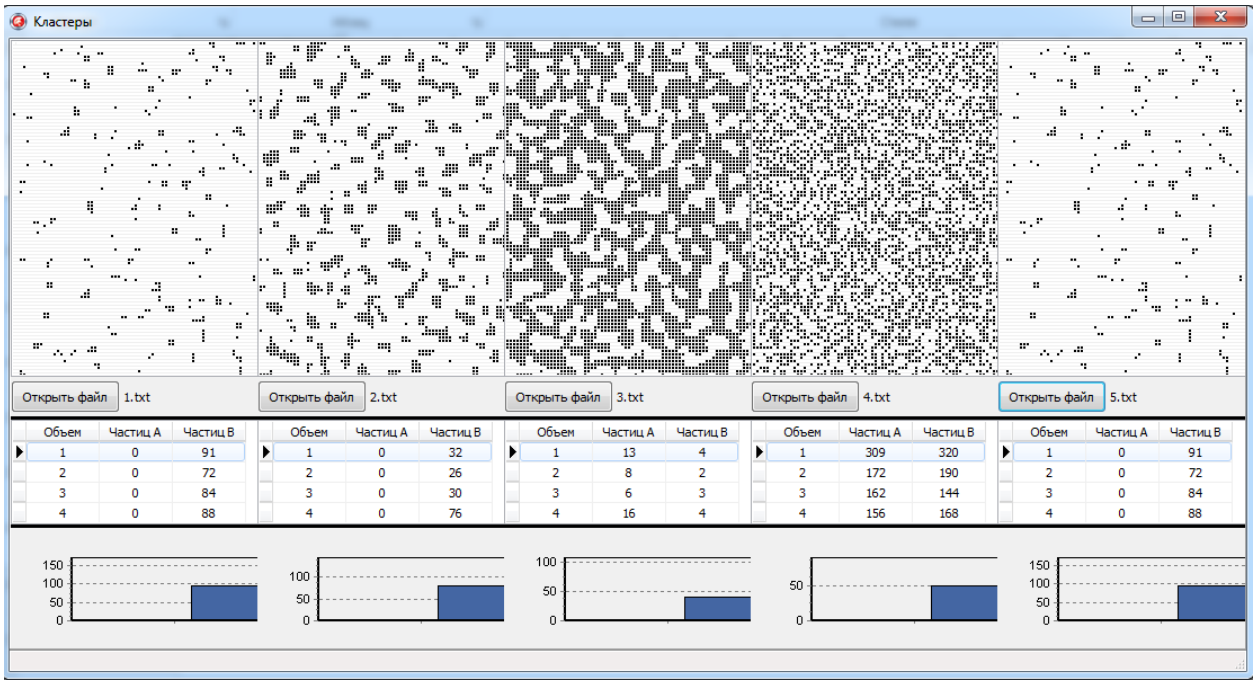

Рисунок 3.2 Результат анализа

3.2 Сравнительный анализ рассмотренных методов

Методы по способу обработки данных [1]:

Иерархические методы:

Агломеративные методы AGNES (Agglomerative Nesting):

- CURE;
- ROCK;
- CHAMELEON и т.д.

Дивизимные методы DIANA (Divisive Analysis):

- BIRCH;
- MST и т.д.

Неиерархические методы.

Итеративные

- К-средних (k-means)
- PAM (k-means + k-medoids)
- CLOPE
- LargeItem и т.д.

Методы по способу анализа данных:

- Четкие;
- Нечеткие.

Методы по количеству применений алгоритмов кластеризации:

- С одноэтапной кластеризацией;
- С многоэтапной кластеризацией.

Методы по возможности расширения объема обрабатываемых данных:

- Масштабируемые;
- Немасштабируемые.

Методы по времени выполнения кластеризации [1]:

- Потоковые (on-line);
- Не потоковые (off-line).

Описание алгоритмов кластеризации.

Иерархическая кластеризация.

При иерархической кластеризации выполняется последовательное объединение меньших кластеров в большие или разделение больших кластеров на меньшие [1].

Агломеративные методы AGNES (Agglomerative Nesting).

Эта группа методов характеризуется последовательным объединением исходных элементов и соответствующим уменьшением числа кластеров.

В начале работы алгоритма все объекты являются отдельными кластерами. На первом шаге наиболее похожие объекты объединяются в кластер. На последующих шагах объединение продолжается до тех пор, пока все объекты не будут составлять один кластер.

Алгоритм CURE (Clustering Using REpresentatives).

Выполняет иерархическую кластеризацию с использованием набора определяющих точек для определения объекта в кластер.

Назначение: кластеризация очень больших наборов числовых данных.

Ограничения: эффективен для данных низкой размерности, работает только на числовых данных. Достоинства: выполняет кластеризацию на

высоком уровне даже при наличии выбросов, выделяет кластеры сложной формы и различных размеров, обладает линейно зависимыми требованиями к месту хранения данных и временную сложность для данных высокой размерности.

Недостатки: есть необходимость в задании пороговых значений и количества кластеров.

Описание алгоритма [3]:

Шаг 1: Построение дерева кластеров, состоящего из каждой строки входного набора данных.

Шаг 2: Формирование «кучи» в оперативной памяти, расчет расстояния до ближайшего кластера (строки данных) для каждого кластера. При формировании кучи кластеры сортируются по возрастанию дистанции от кластера до ближайшего кластера. Расстояние между кластерами определяется по двум ближайшим элементам из соседних кластеров. Для определения расстояния между кластерами используются «манхеттенская», «евклидова» метрики или похожие на них функции.

Шаг 3: Слияние ближних кластеров в один кластер. Новый кластер получает все точки входящих в него входных данных. Расчет расстояния до остальных кластеров для новообразованного кластера. Для расчета расстояния кластеры делятся на две группы: первая группа - кластеры, у которых ближайшими кластерами считаются кластеры, входящие в новообразованный кластер, остальные кластеры - вторая группа. И при этом для кластеров из первой группы, если расстояние до новообразованного кластера меньше чем до предыдущего ближайшего кластера, то ближайший кластер меняется на новообразованный кластер. В противном случае ищется новый ближайший кластер, но при этом не берутся кластеры, расстояния до которых больше, чем до новообразованного кластера. Для кластеров второй группы выполняется следующее: если расстояние до новообразованного кластера ближе, чем предыдущий ближайший кластер, то ближайший кластер меняется. В противном случае ничего не происходит.

Шаг 4: Переход на шаг 3, если не получено требуемое количество кластеров.

Дивизимные методы DIANA (Divisive Analysis).

Эта группа методов характеризуется последовательным разделением исходного кластера, состоящего из всех объектов, и соответствующим увеличением числа кластеров.

В начале работы алгоритма все объекты принадлежат одному кластеру, который на последующих шагах делится на меньшие кластеры, в результате образуется последовательность расщепляющих групп.

Алгоритм BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies).

В этом алгоритме предусмотрен двухэтапный процесс кластеризации.

Назначение: кластеризация очень больших наборов числовых данных.

Ограничения: работа с только числовыми данными.

Достоинства: двухступенчатая кластеризация, кластеризация больших объемов данных, работает на ограниченном объеме памяти, является локальным алгоритмом, может работать при одном сканировании входного набора данных, использует тот факт, что данные неодинаково распределены по пространству, и обрабатывает области с большой плотностью как единый кластер.

Недостатки: работа с только числовыми данными, хорошо выделяет только кластеры сферической формы, есть необходимость в задании пороговых значений.

Описание алгоритма [4]:

Фаза 1: Загрузка данных в память.

Построение начального кластерного дерева (CF Tree) по данным (первое сканирование набора данных) в памяти;

Подфазы основной фазы происходят быстро, точно, практически нечувствительны к порядку.

Алгоритм построения кластерного дерева (CF Tree):

Кластерный элемент представляет из себя тройку чисел (N, LS, SS), где N - количество элементов входных данных, входящих в кластер, LS - сумма элементов входных данных, SS - сумма квадратов элементов входных данных.

Кластерное дерево - это взвешенно сбалансированное дерево с двумя параметрами: B - коэффициент разветвления, T - пороговая величина. Каждый нелистьевой узел дерева имеет не более чем B вхождений узлов следующей формы: [CFj, Childj], где i = 1, 2, ..., B; Childi - указатель на i-й дочерний узел.

Каждый листьевой узел имеет ссылку на два соседних узла. Кластер состоящий из элементов листьевого узла должен удовлетворять следующему условию: диаметр или радиус полученного кластера должен быть не более пороговой величины T.

Фаза 2 (необязательная): Сжатие (уплотнение) данных.

Сжатие данных до приемлемых размеров с помощью перестроения и уменьшения кластерного дерева с увеличением пороговой величины T.

Фаза 3: Глобальная кластеризация.

Применяется выбранный алгоритм кластеризации на листьевых компонентах кластерного дерева.

Фаза 4 (необязательная): Улучшение кластеров.

Использует центры тяжести кластеров, полученные в фазе 3, как основы.

Алгоритм MST (Algorithm based on Minimum Spanning Trees).

Назначение: кластеризация больших наборов произвольных данных.

Достоинства: выделяет кластеры произвольной формы, в т.ч. кластеры выпуклой и впуклой формы, выбирает из нескольких оптимальных решений самое оптимальное.

Описание алгоритма [5]:

Шаг 1: Построение минимального остовного дерева:

Связный, неориентированный граф с весами на ребрах G(V, E), в котором V — множество вершин (контактов), а E — множество их возможных попарных соединений (ребер), для каждого ребра (u,v) однозначно определено некоторое вещественное число w(u,v) — его вес (длина или стоимость соединения). Алгоритм Борувки:

1: Для каждой вершины графа находим ребро с минимальным весом.

2: Добавляем найденные ребра к остовному дереву, при условии их безопасности.

3: Находим и добавляем безопасные ребра для несвязанных вершин к остовному дереву.

Общее время работы алгоритма: O(ELogV).

Алгоритм Крускала:

Обход ребер по возрастанию весов. При условии безопасности ребра добавляем его к остновному дереву.

Общее время работы алгоритма: O(ELogE).

Алгоритм Прима:

1: Выбор корневой вершины.

2: Начиная с корня добавляем безопасные ребра к остовному дереву.

Общее время работы алгоритма: O(ELogV).

Шаг 2: Разделение на кластеры. Дуги с наибольшими весами разделяют кластеры.

Представление алгоритмов построения остовных деревьев на примерах:

Неиерархическая кластеризация.

Алгоритм k-средних (k-means).

Алгоритм k-средних строит k кластеров, расположенных на возможно больших расстояниях друг от друга. Основной тип задач, которые решает алгоритм k-средних, - наличие предположений (гипотез) относительно числа кластеров, при этом они должны быть различны настолько, насколько это

возможно. Выбор числа k может базироваться на результатах предшествующих исследований, теоретических соображениях или интуиции.

Общая идея алгоритма: заданное фиксированное число k кластеров наблюдения сопоставляются кластерам так, что средние в кластере (для всех переменных) максимально возможно отличаются друг от друга.

Ограничения: небольшой объем данных.

Достоинства: простота использования; быстрота использования; понятность и прозрачность алгоритма.

Недостатки: алгоритм слишком чувствителен к выбросам, которые могут искажать среднее; медленная работа на больших базах данных; необходимо задавать количество кластеров.

Описание алгоритма [5]:

Этап 1. Первоначальное распределение объектов по кластерам.

Выбирается число k, и на первом шаге эти точки считаются "центрами" кластеров. Каждому кластеру соответствует один центр.

Выбор начальных центроидов может осуществляться следующим образом: выбор k-наблюдений для максимизации начального расстояния; случайный выбор k-наблюдений; выбор первых k-наблюдений.

В результате каждый объект назначен определенному кластеру.

Этап 2. Вычисляются центры кластеров, которыми затем и далее считаются покоординатные средние кластеров. Объекты опять перераспределяются.

Процесс вычисления центров и перераспределения объектов продолжается до тех пор, пока не выполнено одно из условий:

кластерные центры стабилизировались, т.е. все наблюдения принадлежат кластеру, которому принадлежали до текущей итерации;

число итераций равно максимальному числу итераций.

Выбор числа кластеров является сложным вопросом. Если нет предположений относительно этого числа, рекомендуют создать 2 кластера, затем 3, 4, 5 и т.д., сравнивая полученные результаты.

Алгоритм PAM (partitioning around medoids).

Ограничения: небольшой объем данных.

Достоинства: простота использования; быстрота использования; понятность и прозрачность алгоритма, алгоритм менее чувствителен к выбросам в сравнении с k-means.

Недостатки: необходимо задавать количество кластеров; медленная работа на больших базах данных.

Описание алгоритма [5]:

Данный алгоритм берет на вход множество S и число кластеров k, на выходе алгоритм выдает разбиение множества S на k-кластеров: Si, S2, ..., Sk. Этот алгоритм аналогичен алгоритму k-средних, только при работе алгоритма перераспределяются объекты относительно медианы кластера, а не его центра.

Алгоритм CLOPE.

Назначение: кластеризация огромных наборов категорийных данных.

Достоинства: высокие масштабируемость и скорость работы, а так же качество кластеризации, что достигается использованием глобального критерия оптимизации на основе максимизации градиента высоты гистограммы кластера. Он легко рассчитывается и интерпретируется. Во время работы алгоритм хранит в RAM небольшое количество информации по каждому кластеру и требует минимальное число сканирований набора данных. CLOPE автоматически подбирает количество кластеров, причем это регулируется одним единственным параметром - коэффициентом отталкивания.

Самоорганизующиеся карты Кохонена.

Назначение: кластеризация многомерных векторов, разведочный анализ данных, обнаружение новых явлений.

Достоинства: используется универсальный апроксиматор - нейронная сеть, обучение сети без учителя, самоорганизация сети, простота реализации, гарантированное получение ответа после прохождения данных по слоям.

Недостатки: работа только с числовыми данными, минимизация размеров сети, необходимо задавать количество кластеров.

Описание алгоритма [6]:

Самоорганизующая карта Кохонена - нейронная однослойная сеть прямого распространения.

Этап i. Подготовка данных для обучения.

Обучающая выборка должна быть представительной, не должна быть противоречивой, преобразование и кодирование данных, нормализация данных.

Этап 2. Начальная инициализация карты.

На этом этапе выбирается количество кластеров и, соответственно, количество нейронов в выходном слое.

Перед обучением карты необходимо проинициализировать весовые коэффициенты нейронов сети. Существуют два способа инициализации весовых коэффициентов:

Инициализация случайными значениями, когда всем весам даются малые случайные величины. Инициализация примерами, когда в качестве начальных значений задаются значения случайно выбранных примеров из обучающей выборки.

Этап 3. Обучение сети.

Анализ рассмотренных методов приведен в таблице

Таблица 3.1 Анализ рассмотренных методов

Метод	Достоинства	Недостатки
CURE	выполняет кластеризацию на высоком уровне даже при наличии выбросов, выделяет кластеры сложной формы и различных размеров, обладает линейно зависимыми требованиями к месту хранения данных и временную сложность для данных высокой размерности	есть необходимость в задании пороговых значений и количества кластеров
BIRCH	двухступенчатая кластеризация, кластеризация больших объемов данных, работает на ограниченном объеме памяти, является локальным алгоритмом, может работать при одном сканировании входного	работа с только числовыми данными, хорошо выделяет только кластеры выпуклой или

	набора данных, использует тот факт, что данные неодинаково распределены по пространству, и обрабатывает области с большой плотностью как единый кластер	сферической формы, есть необходимость в задании пороговых значений
MST	выделяет кластеры произвольной формы, в т. ч. кластеры выпуклой и вогнутой форм, выбирает из нескольких оптимальных решений самое оптимальное	чувствителен к выбросам
k-средних	простота использования; быстрота использования; понятность и прозрачность алгоритма	алгоритм слишком чувствителен к выбросам, которые могут искажать среднее; медленная работа на больших базах данных; необходимо задавать количество кластеров; невозможность применения алгоритма на данных, где имеются пересекающиеся кластеры
PAM	простота использования; быстрота использования; понятность и прозрачность алгоритма, алгоритм менее чувствителен к выбросам в сравнении с k-means	необходимо задавать количество кластеров; медленная работа на больших базах данных
CLOPE	высокие масштабируемость и скорость работы, а так же качество кластеризации, что достигается использованием глобального критерия оптимизации на основе максимизации градиента высоты гистограммы кластера. Он легко рассчитывается и интерпретируется. Во время работы алгоритм хранит в RAM небольшое количество информации по каждому кластеру и требует минимальное число сканирований набора данных. CLOPE автоматически подбирает количество кластеров, причем это регулируется одним единственным параметром - коэффициентом отталкивания	
Самоорганизующиеся карты Кохонена	используется универсальный апроксиматор - нейронная сеть, обучение сети без учителя, самоорганизация сети, простота реализации, гарантированное получение ответа после прохождения данных по слоям	работа только с числовыми данными, минимизация размеров сети, необходимо задавать количество кластеров
Алгоритм НСМ	легкость реализации, вычислительная простота	задание количества кластеров, отсутствие гарантии в

		нахождении оптимального решения
Fuzzy C-means	нечеткость при определении объекта в кластер позволяет определять объекты, которые находятся на границе, в кластеры	вычислительная сложность, задание количества кластеров, возникает неопределенность с объектами, которые удалены от центров всех кластеров

Заключение

Теория клеточных автоматов имеет сравнительно небольшую, но достаточно продуктивную историю, основные этапы которой детально изложены в работе [11], в которой особое внимание уделено фундаментальным свойствам клеточных автоматов: параллельности и локальности.

Клеточный автомат состоит из конечной совокупности объектов (ячеек), как правило, образующих регулярную решетку. Другими словами, если не использовать строгие математические термины, это — совокупность ячеек, определенным образом соединенных между собой и образующих равномерную сетку (решетку). При этом состояние каждой i-й ячейки (клетки) в момент времени t характеризуется некоторым числом a(i, t) или набором чисел. Совокупность состояний всех клеток решетки называется состоянием решетки. В модели клеточного автомата каждое состояние решетки соответствует некоторому моменту времени, которое изменяется дискретно по шагам (итерациям).

Состояние решетки меняется в соответствии с некоторым законом, который называется правилом клеточного автомата. Правила определяют, какое состояние должно быть у клетки в следующий момент времени, в зависимости от состояний ее и некоторых других клеток в текущий момент времени.

Теоретически клеточные автоматы могут иметь любую размерность, однако чаще всего рассматривают одномерные и двумерные клеточные автоматы. В случае двумерного клеточного автомата решетка реализуется двумерным массивом и каждая клетка нумеруется упорядоченной парой чисел (a, b). В этом случае у каждой клетки ближайшими соседями считаются либо клетки, имеющие с исходной общую сторону (окрестность фон Неймана), таких клеток будет 4, либо имеющие с исходной общую вершину (окрестность Мура), таких клеток будет 8.

Классическая модель клеточного автомата имеет следующие свойства:

- сеть клеточного автомата является однородной, т. е. правила изменения состояний для всех клеток одинаковы;
- множество состояний каждой клетки конечно;
- на клетку могут повлиять лишь клетки из ее окрестности, ближайшие соседи;
- состояния всех клеток меняются единовременно, в конце итерации.

В модели клеточного автомата каждая клетка изменяет свое состояние, взаимодействуя с ограниченным числом клеток, как правило, ближайшего окружения. Однако имеется возможность одновременного (параллельного) изменения состояния всех клеток, всей решетки на основе общего правила клеточного автомата. Это свойство позволяет при моделировании связывать процессы, происходящие на микроуровне, с изменениями процессов, протекающими на макроуровне. Это важнейшее свойство клеточных автоматов, которое позволяет их успешно использовать для моделирования систем, в которых значительную роль играют пространственные взаимодействия между элементами. Они хорошо зарекомендовали себя при моделировании динамики жидкости и газа в различных средах, в пограничных зонах, при моделировании систем, состоящих из большого числа частиц, взаимодействующих друг с другом нелинейно, при описании возникновения коллективныхявлений — турбулентности, упорядочения, хаоса, нарушения симметрии, фрактальности и других.

В последние годы клеточные автоматы находят свое применение при мо-делировании социальных явлений как чисто теоретические модели для ка-чественного анализа процессов, так и для численного прогнозирования. Не-которые примеры клеточных автоматов, применяемых в задачах социологии, приведены в работе [9].

Результаты фундаментальных исследований в области клеточных автоматов представлены в работе [23], в которой сделан акцент на

возможности использования клеточных автоматов при моделировании физических, био-логических и вычислительных систем, в силу простоты реализации моделей, точности математических вычислений и способности моделей к моделированию сложных систем.

В целом у моделей клеточных автоматов достаточно много положительных качеств, которые, несомненно, влияют на активность их использования в различных областях науки. Однако есть и определенные негативные моменты, сдерживающие развитие этого направления математического моделирования, среди которых следует отметить слабую общую теоретическую базу клеточных автоматов, недостаточную разработку вопросов сходимости вычислительных экспериментов, вопросов устойчивости полученных численных решений.

В настоящее время существует достаточно много различных классификаций клеточных автоматов, из которых наиболее известна классификация Вольфрама, основанная на анализе возможной сложности поведения клеточного автомата. Классифицировать клеточные автоматы можно по разным признакам, в частности, по правилам изменения состояния клеток можно разделить автоматы на детерминированные и вероятностные. В детерминированных клеточных автоматах состояние каждой клетки в момент времени t + 1 определяется однозначно состоянием этой клетки и ее ближайших соседей в момент времени t.

Вероятностный клеточный автомат, описывающий диффузию инноваций, представлен в работе [6]. Каждой клетке соответствует агент, который может принять инновацию. Клетка может находиться в двух состояниях: S: — новинка принята, S0 — новинка не принята, состояние S: не может быть изменено. Автомат принимает или не принимает новинку, ориентируясь на мнение восьми или четырех ближайших соседей в окрестности данной клетки (окрестность Мура или Неймана). В зависимости от количества n приверженцев новинки существует пороговая функция P(n) — вероятность ее принятия. Для каждой клетки автомата на каждом шаге

(такте) генерируется случайное число x, определяется количество соседей n, принявших инновацию, проверяется неравенство $x < P(n)$, если оно выполняется, клетка принимает инновацию, то есть переходит в состояние Sr

Подобный автомат может быть использован для моделирования диффузии локальных инноваций в рамках одного города или района. Вероятностный клеточный автомат хорошо моделирует диффузию инноваций для сетевого маркетинга.

Список использованной литературы

1. Krylov V.N. Contour images segmentation in space of wavelet transform with the use of lifting / V.N. Krylov, M. V. Polyakova // Optical-electronic informatively-power technologies. - 2007. - № 2 (12). - P. 48 - 58.
2. Shcherbakova G. Electronic Apparatus Automation Inspection with Adaptive Clustering in Wavelet Domain. / G. Shcherbakova, V. Krylov, S. Antoshchuk // Досвід розроб- ки та застосування приладо-технологічних САПР в мт- роелектронщ: міжнар. наук.-техн. конф., 24—28 лют. 2009р., тези доп. - Л^в, 2009. - С. 153 - 154.
3. А. Васильев: Самоучитель C++ с примерами и задачами, Москва, Издательство: Наука и Техника, 2015 г., 480 стр.
4. Алексей Архангельский: Программирование в Delphi для Windows: Версии 2006, 2007, Turbo Delphi, Издательство: Бином, Москва, 2013 г., 1248 стр.
5. Алексей Архангельский: Программирование в Delphi. Учебник по классическим версиям Delphi, Издательство: Бином, Москва, 2013 г., 816 стр.
6. Алексей Сурядный: Microsoft Access 2010. Лучший самоучитель, Издательство: Астрель, Москва,, 2012 г., 448 стр.
7. Андрей Сеннов: С31 Access 2010. Учебный курс, Издательство: Питер, Москва, 2010 г., 288 стр.
8. Аникеев, Маркин: Разработка приложений баз данных в Delphi. Самоучитель, Издательство: Диалог-МИФИ, Москва, 2013 г., 160 стр.
9. Арнольд Виллемер: Программирование на C++, Москва, Издательство: Эксмо, 2013 г., 528 стр.
10. Бекаревич, Пушкина: Самоучитель Access 2010, Издательство: BHV, Москва, 2011 г., 432 стр.
11. Болтян А.В. Прогнозирование и оценка параметров продукции / А.В. Болтян, И.А. Горобец - Донецк: Норд-Пресс, 2004. - 192 с.

12. В. Тимофеев: Самоучитель C++ как он есть, Москва, Издательство: Бином, 2009 г., 336 стр.

13. Геннадий Гурвиц: Microsoft Access 2010. Разработка приложений на реальном примере, Издательство: BHV, Москва, 2010 г., 496 стр.

14. Герберт Шилдт: C++ для начинающих, Москва, Издательство: Эком, 2010 г., 640 стр.

15. Дмитрий Осипов: Delphi. Программирование для Windows, OS X, iOS, Издательство: BHV, Москва, 2014 г., 464 стр.

16. Дмитрий Осипов: Базы данных и Delphi. Теория и практика, Издательство: BHV, 2011 г., Издательство: BHV, Москва, 2010 г., 752 стр.

17. Дуда Р. Распознавание образов и анализ сцен / Р. Дуда, П. Харт. - М.: Мир, 1976. - 509 с.

18. Е. Санников: Курс практического программирования в Delphi. Объектно – ориентированное программирование. Практикум, Издательство: Солон-пресс, 2013 г., Москва, 188 стр.

19. Жулинский С.Ф. Статистические методы в со-временном менеджменте качества / С.Ф. Жулинский, Е.С. Новиков, В.Я. Поспелов - М.: Фонд «Новое тысячелетие», 2001.- 208 с.

20. Контроль качества с помощью персональных компьютеров / Т. Макино, М. Охаси, Х. Докэ, К. Макино; Пер. с яп. А.Б.Орфенова. Под ред. Ю.П.Адлера. - М.: Ма-шиностроение, 1991. - 224 с.

21. Крилов В.Н. Субградієнтний тративний метод оптимізаціl в просторі вейвлет-перетворення./ В.Н. Крилов, Г.Ю. Щербакова // Збірн. наук. праць Втського тституту Київського нац. ун-ту ім. Т. Шевченка. - К., 2008. - Вип. 12. - С. 56 - 60.

22. Крисилов В.А. Решение задачи таксономии на основе гипотезы компактности при анализе данных / В.А. Крисилов, Н.О. Блудян, С.А. Юдин // Искусственный интеллект. - 2005. - № 4. - С. 699 - 707.

23. М. Полубенцева: C/C++ Процедурное программирование, Москва, Издательство: BHV, 2014 г., 432 стр.

24. Никита Культин: Delphi в задачах и примерах, Издательство: BHV, Москва, 2012 г., 288 стр.

25. Никита Культин: Основы программирования в Delphi XE, Издательство: BHV, Москва, 2011 г., 416 стр.

26. Николай Мартынов: Программирование для Windows на C\C++. В 2-х томах, Москва, Издательство: Бином, 2013 г., 480 стр.

27. Полак Э. Численные методы оптимизации /Э. Полак.. - М.: Мир, 1976. — 509 с.

28. Роберт Лафоре: Объектно-ориентированное программирование в C++, Москва, Издательство: Питер, 2013 г., 928 стр.

29. Скотт Мейерс: Наиболее эффективное использование C++. 35 новых рекомендаций по улучшению ваших программ, Москва, Издательство: ДМК-Пресс, 2014 г., 294 стр.

30. Скотт Мэйерс: Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ, Москва, Издательство: ДМК-Пресс, 2014 г., 300 стр.

31. Статистические методы повышения качества: Пер. с англ. и доп. Ю.П.Адлера, Л.А.Конаревой / Под ред. Х. Кумэ. - М.: Финансы и статистика, 1990.- 304 с.

32. Статистические методы. Индексы возможностей процессов. ГОСТ Р50779.44-2001.- [Начало действия 2002-07-01]. М.: Госстандарт России 2001.

33. Статистические методы. Контрольные карты Шухарта. (ISO 8258-91) ГОСТ Р50779.42-99. - [Начало действия 1999-04-13]. М.: Госстандарт России 1999. - 36 с.

34. Статистические методы. Контрольные карты. Общее руководство и введение. (ISO 7870-93) ГОСТ Р50779.40-96. - [Начало действия 1996-07-01]. М.: Гос-стандарт России 1997.

35. Статистические методы. Статистическое управление качеством. Термины и определения (ISO 3534.2-93):ГОСТ Р50779.11 - 2000. - [Начало действия 2001-07-01]. М.: Госстандарт России 2000.

36. Стефан Дьюхэрст: Скользкие места C++. Как избежать проблем при проектировании и компиляции ваших программ, Москва, Издательство: ДМК-Пресс, 2014 г., 264 стр.

37. Ховард, Лебланк, Виега: Как написать безопасный код на C++, Java, Perl, PHP, ASP.NET, - Москва, Москва, Издательство: ДМК-Пресс, 2014 г., 288 стр.

38. Цыпкин Я.З. Адаптация и обучение в автомати-ческих системах. / Я.З. Цыпкин - М.: Наука, 1968. - 400 с.

39. Энтони Уильямс: Параллельное программирование на C++ в действии. Практика разработки многопоточных программ, Москва, Издательство: ДМК-Пресс, 2014 г., 672 стр.

40. Юрий Ревич: Нестандартные приемы программирования на Delphi, Издательство: BHV, Москва, 2008 г., 560 стр.

Приложение. Листинг программных модулей

```
unit uClusterManager;

interface

uses Classes, SysUtils, uMatrix;

type
  TClusterStat = class
    Size: Integer;
    AClusterCount: Integer;
    BClusterCount: Integer;
    constructor Create;
  end;

  /// <summary>
  /// Класс "Сегмент кластера". Служит для хранения точек одного кластера.
  /// </summary>
  TClusterSegment = class
  public
    /// <summary>
    /// Строка матрицы, на которой находится точка
    /// </summary>
    Row: Cardinal;
    /// <summary>
    /// Столбец матрицы, на которой находится точка
    /// </summary>
    Col: Cardinal;
    /// <summary>
    /// Конструктор класса
    /// </summary>
    constructor Create;
    /// <summary>
    /// Деструктор класса
    /// </summary>
    destructor Destroy; override;
  end;

  /// <summary>
  /// Класс "Кластер". Служит для хранения сведений одного кластера.
  /// </summary>
  TCluster = class
  public
    /// <summary>
    /// Идентификатор кластера
    /// </summary>
    ID: Cardinal;
    /// <summary>
    /// Тип кластера (значение, по которому происходит группирование точек в кластер)
    /// </summary>
    ClusterType: Extended;
    /// <summary>
    /// Список сегментов кластера. (список объектов класса TClusterSegment)
    /// </summary>
```

```
    Segments: TStringList;
    /// <summary>
    /// Конструктор класса
    /// </summary>
    constructor Create;
    /// <summary>
    /// Деструктор класса
    /// </summary>
    destructor Destroy; override;
  end;

  /// <summary>
  /// Класс, осуществляющий идентификацию кластеров
  /// </summary>
  TClusterManager = class
  private
    /// <summary>
    /// Исходная матрица (содержит единицы и нули, которые будут сгруппированы по кластерам)
    /// </summary>
    FMatrix: TMatrix;
    /// <summary>
    /// Матрица идентификатором кластеров (данная матрица для каждой точки исходной
матрицы определяет идентификатор кластера (или принадлежность точки к определенному
кластеру))
    /// </summary>
    FClusterID: TMatrix;
    /// <summary>
    /// Список кластеров (список объектов класса TCluster)
    /// </summary>
    FClusterList: TStringList;
    /// <summary>
    /// Статистика по количеству кластеров (список объектов класса TClusterStat)
    /// </summary>
    FClusterStat: TStringList;

    procedure StatAdd(Cluster: TCluster);



  protected
  public
    /// <summary>
    /// Конструктор класса
    /// </summary>
    constructor Create;
    /// <summary>
    /// Деструктор класса
    /// </summary>
    destructor Destroy; override;
    /// <summary>
    /// Метод загрузки исходной матрицы из файла
    /// </summary>
    procedure LoadFromFile(FileName: string);
    /// <summary>
    /// Метод сохранения исходной матрицы в файл
    /// </summary>
    procedure SaveToFile(FileName: string);
```

```
      /// <summary>
      /// Метод определения кластеров (данный метод заполняет матрицу "FClusterID" и список
"FClusterList")
      /// </summary>
      procedure GetClusters;
      /// <summary>
      /// Метод поиска кластера по его идентификатору
      /// </summary>
      function ClusterByID(ID: Cardinal): TCluster;
      /// <summary>
      /// Исходная матрица (содержит единицы и нули, которые будут сгруппированы по кластерам)
      /// </summary>
      property Matrix: TMatrix read FMatrix;
      /// <summary>
      /// Матрица идентификатором кластеров (данная матрица для каждой точки исходной
матрицы определяет идентификатор кластера (или принадлежность точки к определенному
кластеру))
      /// </summary>
      property ClusterID: TMatrix read FClusterID;
      /// <summary>
      /// Список кластеров (список объектов класса TCluster)
      /// </summary>
      property ClusterList: TStringList read FClusterList;
      /// <summary>
      /// Статистика по количеству кластеров (список объектов класса TClusterStat)
      /// </summary>
      property ClusterStat: TStringList read FClusterStat;
   end;

implementation



procedure TClusterManager.StatAdd(Cluster: TCluster);
var
   I, Size: Integer;
   CS: TClusterStat;

begin
   Size := Cluster.Segments.Count;
   for I := FClusterStat.Count - 1 downto 0 do
   begin
      CS := FClusterStat.Objects[I] as TClusterStat;
      if CS.Size = Size then
      begin
         if Cluster.ClusterType = 0 then
            CS.AClusterCount := CS.AClusterCount + 1
         else
            CS.BClusterCount := CS.BClusterCount + 1;
         Exit;
      end;
   end;

   CS := TClusterStat.Create;
   CS.Size := Size;
   if Cluster.ClusterType = 0 then
      CS.AClusterCount := CS.AClusterCount + 1
```

```
    else
      CS.BClusterCount := CS.BClusterCount + 1;
    FClusterStat.AddObject('', CS);
end;

constructor TClusterStat.Create;
begin
    Size := 0;
    AClusterCount := 0;
    BClusterCount := 0;
end;

function TClusterManager.ClusterByID(ID: Cardinal): TCluster;
var
    I: Integer;
    C: TCluster;
begin
    for I := 0 to FClusterList.Count - 1 do
    begin
      C := FClusterList.Objects[I] as TCluster;
      if C.ID = ID then
      begin
        Result := C;
        Exit;
      end;
    end;

    raise Exception.Create('Кластер не найден');
end;

constructor TClusterSegment.Create;
begin
    inherited;
    Row := 0;
    Col := 0;
end;

destructor TClusterSegment.Destroy;
begin
    inherited;
end;

constructor TCluster.Create;
begin
    inherited;
    ID := 0;
    Segments := TStringList.Create;
    Segments.OwnsObjects := True;
end;

destructor TCluster.Destroy;
begin
    Segments.Free;
    inherited;
end;

constructor TClusterManager.Create;
```

```
begin
   inherited;
   FClusterID := nil;
   FMatrix := TMatrix.Create(100, 100);
   FClusterList := TStringList.Create;
   FClusterList.OwnsObjects := True;
   FClusterStat := TStringList.Create;
   FClusterStat.OwnsObjects := True;
end;

destructor TClusterManager.Destroy;
begin
   FMatrix.Free;
   FClusterList.Free;
   FClusterStat.Free;
   inherited;
end;

procedure TClusterManager.GetClusters;
var
   Row, Col, ID: Cardinal;
   Cluster: TCluster;

   procedure FillCluster(Row, Col: Cardinal; Cluster: TCluster);
   var
      ClusterSegment: TClusterSegment;
   begin
      if (ClusterID.Matrix[Row, Col] <> 0) then
      begin
         Exit;
      end;

      if (Matrix.Matrix[Row, Col] <> Cluster.ClusterType) then
      begin
         Exit;
      end;

      ClusterSegment := TClusterSegment.Create;
      try
         ClusterSegment.Row := Row;
         ClusterSegment.Col := Col;
         ClusterID.Matrix[Row, Col] := Cluster.ID;
         Cluster.Segments.AddObject('', ClusterSegment);

         if Matrix.TestCell(Row, Col + 1, True) then
         begin
            FillCluster(Row, Col + 1, Cluster);
         end;

         if Matrix.TestCell(Row - 1, Col, True) then
         begin
            FillCluster(Row - 1, Col, Cluster);
         end;

         if Matrix.TestCell(Row + 1, Col, True) then
         begin
            FillCluster(Row + 1, Col, Cluster);
```

```
        end;

        if Matrix.TestCell(Row, Col - 1, True) then
        begin
          FillCluster(Row, Col - 1, Cluster);
        end;
      except
        ClusterSegment.Free;
        raise;
      end;
    end;

begin
    ID := 0;
    FClusterStat.Clear;
    FClusterList.Clear;
    if (Assigned(FClusterID)) then
      FClusterID.Free;
    FClusterID := Matrix.Clone;
    FClusterID.Fill(0);

    for Row := 1 to Matrix.RowCount do
    begin
      for Col := 1 to Matrix.ColCount do
      begin
        if (ClusterID.Matrix[Row, Col] = 0) then
        begin
          Inc(ID);
          Cluster := TCluster.Create;
          Cluster.ID := ID;
          Cluster.ClusterType := Matrix.Matrix[Row, Col];
          FClusterList.AddObject('', Cluster);
          FillCluster(Row, Col, Cluster);

          StatAdd(Cluster);

        end;
      end;
    end;

    {
     for Row := 1 to Matrix.RowCount do
     begin
     for Col := 1 to Matrix.ColCount do
     begin
     if (Matrix.Matrix[Row, Col] = 1) and (ClusterID.Matrix[Row, Col] = 0) then
     begin
     Inc(ID);
     Cluster := TCluster.Create;
     Cluster.ID := ID;
     Cluster.ClusterType := 1;
     FClusterList.AddObject('', Cluster);
     FillCluster(Row, Col, Cluster);
     end;
     end;
     end;
    }
```

```
end;

procedure TClusterManager.LoadFromFile(FileName: string);
var
  L: TStringList;
  I, J: Integer;

  procedure FileTest(L: TStringList);
  var
    I, J, ColCount, SLength: Integer;
    S: string;

  begin
    if (L.Count = 0) then
      raise Exception.Create('Файл не содержит данных');

    for I := 0 to L.Count - 1 do
    begin
      S := L.Strings[I];
      SLength := Length(S);
      if I = 0 then
      begin
        ColCount := SLength
      end
      else
      begin
        if ColCount <> SLength then
          raise Exception.Create('Файл содержит несимметричные строки');
      end;

      for J := 1 to SLength do
      begin
        if (S[J] <> '1') and (S[J] <> '0') then
          raise Exception.Create('Файл содержит недопустимый символ (строка: ' + IntToStr(I + 1)
+ '; столбец: ' + IntToStr(J) + ')');
      end;
    end;
  end;

begin
  L := TStringList.Create;
  try
    L.LoadFromFile(FileName);
    FileTest(L);
    FMatrix.Resize(L.Count, Length(L.Strings[0]));
    FMatrix.Fill(0);

    for I := 1 to FMatrix.RowCount do
    begin
      for J := 1 to FMatrix.ColCount do
      begin
        if (L.Strings[I - 1][J] = '1') then
          FMatrix.Matrix[I, J] := 1;
      end;
    end;
```

```
    finally
      L.Free;
    end;
  end;

  procedure TClusterManager.SaveToFile(FileName: string);
  var
    L: TStringList;
    I, J: Integer;
    S: string;

  begin
    L := TStringList.Create;
    try
      L.Clear;
      for I := 1 to FMatrix.RowCount do
      begin
        S := '';
        for J := 1 to FMatrix.ColCount do
        begin
          if (FMatrix.Matrix[I, J] <> 0) then
            S := S + '1'
          else
            S := S + '0';
        end;
        L.Add(S);
      end;
      L.SaveToFile(FileName);
    finally
      L.Free;
    end;
  end;

  end.

  unit uMatrix;

  interface

  uses Classes, SysUtils, Math, Dialogs;

  const
    c_matrix_data_identificator: Extended = 707808909.101202303;
    c_matrix_msg_multiplication_not_allowed_for_this_matrix = 'Перемножение данных матриц
невозможно';
    c_matrix_msg_method_for_square_matrix_only = 'Метод применяется только для квадратных
матриц';
    c_matrix_msg_method_for_symmetric_matrix_only = 'Метод применяется только для
симметричных матриц';
    c_matrix_msg_method_for_vector_only = 'Метод применяется только для векторов';
    c_matrix_msg_method_for_equal_size_matrix_only = 'Метод применяется только для матриц
одинаковых размеров';
    c_matrix_msg_col_count_must_be_greater_than_zero = 'Количество столбцов должно быть больше
нуля';
    c_matrix_msg_row_count_must_be_greater_than_zero = 'Количество строк должно быть больше
нуля';
    c_matrix_msg_bad_data_identificator = 'Данные имеют неверный формат';
```

```
  c_matrix_msg_col_index_out_of_range = 'Индекс столбца выходит за пределы допустимого
диапазона';
  c_matrix_msg_row_index_out_of_range = 'Индекс строки выходит за пределы допустимого
диапазона';
  c_matrix_msg_can_not_delete_last_col = 'Вы не можете удалить единственный столбец';
  c_matrix_msg_can_not_delete_last_row = 'Вы не можете удалить единственную строку';
  c_matrix_msg_precision_con_not_below_zero = 'Точность не может быть ниже нуля';
  c_matrix_msg_iteration_count_above_allowable_value = 'Число итераций превысило допустимое
значение';
  c_matrix_msg_bad_matrix_size = 'Размер матрицы не позволяет выполнять вычисления';

type
  TMatrixValue = class
    Row: Cardinal;
    Col: Cardinal;
    Value: Extended;
  end;

  /// <summary>
  /// Типы сортировки матрицы
  /// </summary>
  TMatrixSortType = (mstAsc, mstDesc);

  /// <summary>
  /// Класс для работы с матрицами и векторами
  /// </summary>
  TMatrix = class
  private
    /// <summary>
    /// Число столбцов матрицы
    /// </summary>
    FColCount: Cardinal;
    /// <summary>
    /// Количество строк матрицы
    /// </summary>
    FRowCount: Cardinal;
    /// <summary>
    /// Матрица
    /// </summary>
    FMatrix: array of array of Extended;
    /// <summary>
    /// Метод инициализации матрицы
    /// </summary>
    procedure MatrixCreate;
    /// <summary>
    /// Метод разрушения матрицы
    /// </summary>
    procedure MatrixFree;
    /// <summary>
    /// Количество ячеек (только для векторов)
    /// </summary>
    function GetCellCount: Cardinal;

  protected
    /// <summary>
    /// Метод вычисления модуля
    /// </summary>
```

```
    function fabs(X: Extended): Extended;
  public
    /// <summary>
    /// Переменная для нужд прикладных программистов
    /// </summary>
    Tag: Extended;
    /// <summary>
    /// Конструктор класса
    /// </summary>
    constructor Create(RowCount, ColCount: Cardinal);
    /// <summary>
    /// Деструктор класса
    /// </summary>
    destructor Destroy;
    /// <summary>
    /// Метод проверки допустимости значения номера строки и столбца
    /// </summary>
    function TestCell(Row, Col: Cardinal; WithoutException: Boolean = False): Boolean;
    /// <summary>
    /// Метод проверки допустимости значения номера строки
    /// </summary>
    function TestRow(Row: Cardinal; WithoutException: Boolean = False): Boolean;
    /// <summary>
    /// Метод проверки допустимости значения номера столбца
    /// </summary>
    function TestCol(Col: Cardinal; WithoutException: Boolean = False): Boolean;
    /// <summary>
    /// Возвращает True, если матрица - вектор
    /// </summary>
    function IsVector: Boolean;
    /// <summary>
    /// Возвращает True, если матрица квадратная
    /// </summary>
    function IsSquareMatrix: Boolean;
    /// <summary>
    /// Метод копирования значений из другой матрицы
    /// </summary>
    procedure Copy(Source: TMatrix);
    /// <summary>
    /// Метод заполнения матрицы
    /// </summary>
    procedure Fill(Value: Extended);
    /// <summary>
    /// Метод заполнения матрицы случайными числами
    /// </summary>
    procedure FillRandom(AFrom, ATo: Integer; Symmetric: Boolean = False);
    /// <summary>
    /// Метод заполнения строки матрицы
    /// </summary>
    procedure FillRow(Row: Cardinal; Value: Extended);
    /// <summary>
    /// Метод заполнения столбца матрицы
    /// </summary>
    procedure FillCol(Col: Cardinal; Value: Extended);
    /// <summary>
    /// Метод удаления строки матрицы
    /// </summary>
```

```
procedure DeleteRow(Row: Cardinal);
/// <summary>
/// Метод удаления столбца матрицы
/// </summary>
procedure DeleteCol(Col: Cardinal);
/// <summary>
/// Метод добавления столбца матрицы
/// </summary>
procedure AddCol;
/// <summary>
/// Метод добавления строки матрицы
/// </summary>
procedure AddRow;
/// <summary>
/// Метод задающий значение ячейки матрицы
/// </summary>
procedure SetValue(Row, Col: Cardinal; Value: Extended); overload;
/// <summary>
/// Метод задающий значение элемента вектора
/// </summary>
procedure SetValue(Cell: Cardinal; Value: Extended); overload;
/// <summary>
/// Метод чтения значения ячейки матрицы
/// </summary>
function GetValue(Row, Col: Cardinal): Extended; overload;
/// <summary>
/// Метод чтения значения элемента вектора
/// </summary>
function GetValue(Cell: Cardinal): Extended; overload;
/// <summary>
/// Перевод матрицы в текст
/// </summary>
function ToString: string;
/// <summary>
/// Метод клонирования матрицы
/// </summary>
function Clone: TMatrix;
/// <summary>
/// Метод вычисления суммы в строке
/// </summary>
function GetRowSum(Row: Cardinal): Extended;
/// <summary>
/// Метод изменения размера матрицы
/// </summary>
procedure Resize(RowCount, ColCount: Cardinal);
/// <summary>
/// Умножение матриц R = A * B, где A - текущая матрица (Self)
/// </summary>
function Multiplication(B: TMatrix): TMatrix;
/// <summary>
/// Метод сохранения матрицы в поток
/// </summary>
procedure SaveToStream(S: TStream);
/// <summary>
/// Метод загрузки матрицы из потока
/// </summary>
procedure LoadFromStream(S: TStream);
```

```
    /// <summary>
    /// Метод сохранения матрицы в файл
    /// </summary>
    procedure SaveToFile(FileName: string);
    /// <summary>
    /// Метод загрузки матрицы из файла
    /// </summary>
    procedure LoadFromFile(FileName: string);
    /// <summary>
    /// Метод заполнения главной диагонали каким-либо значением (только для квадратных
матриц)
    /// </summary>
    procedure FillMainDiagonal(Value: Extended);
    /// <summary>
    /// Меторд заменяет все значения матрицы значением "Value" если они ниже "Limit".
    /// </summary>
    procedure LimiterLow(Limit: Extended; Value: Extended);
    /// <summary>
    /// Метод сортировки столбцов матрицы
    /// </summary>
    procedure SortCol(SortType: TMatrixSortType; Col: Cardinal);
    /// <summary>
    /// Метод сортировки строк матрицы
    /// </summary>
    procedure SortRow(SortType: TMatrixSortType; Row: Cardinal);
    /// <summary>
    /// Сортировка (только для вектора)
    /// </summary>
    procedure Sort(SortType: TMatrixSortType);
    /// <summary>
    /// Расчет сингулярных чисел (SVD) методом Якоби (оригинальный алгоритм)
    /// </summary>
    /// <returns>
    /// Вектор сингулярных чисел матрицы
    /// </returns>
    function SVDJacobi(Precision: Extended = 0.00001; IterationLimit: Cardinal = 1000): TMatrix;
    /// <summary>
    /// Расчет сингулярных чисел (SVD) методом Якоби (оригинальный алгоритм) без активного
использования матриц
    /// </summary>
    /// <returns>
    /// Вектор сингулярных чисел матрицы
    /// </returns>
    function SVDJacobiWithoutMatrix(Precision: Extended; IterationLimit: Cardinal): TMatrix;
    /// <summary>
    /// Расчет сингулярных чисел (SVD) методом Якоби (Модификация выбора пары (P, Q))
    /// </summary>
    /// <returns>
    /// Вектор сингулярных чисел матрицы
    /// </returns>
    function SVDJacobiModification(Precision: Extended = 0.00001; IterationLimit: Cardinal = 1000):
TMatrix;
    /// <summary>
    /// Расчет сингулярных чисел (SVD) методом Якоби (Модификация выбора пары (P, Q) без
активного использования матриц)
    /// </summary>
    /// <returns>
```

```
      /// Вектор сингулярных чисел матрицы
      /// </returns>
      function SVDJacobiModificationWithoutMatrix(Precision: Extended = 0.00001; IterationLimit:
Cardinal = 1000): TMatrix;
      /// <summary>
      /// Метод проверки матрицы на симметрию
      /// </summary>
      function IsSymmetricMatrix: Boolean;
      /// <summary>
      /// Метод возвращает транспнированную матрицу
      /// </summary>
      function Transpose: TMatrix;
      /// <summary>
      /// Количество ячеек (только для векторов)
      /// </summary>
      property CellCount: Cardinal read GetCellCount;
      /// <summary>
      /// Количество строк
      /// </summary>
      property RowCount: Cardinal read FRowCount;
      /// <summary>
      /// Количество столбцов
      /// </summary>
      property ColCount: Cardinal read FColCount;
      /// <summary>
      /// Значение вектора
      /// </summary>
      property Vector[Cell: Cardinal]: Extended read GetValue write SetValue;
      /// <summary>
      /// Значения матрицы
      /// </summary>
      property Matrix[Row: Cardinal; Col: Cardinal]: Extended read GetValue write SetValue;

   end;

implementation

function TMatrix.IsSymmetricMatrix: Boolean;
var
   I, J: Cardinal;
begin
   if (IsSquareMatrix = False) then
      raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);
   Result := False;
   for I := 1 to RowCount do
   begin
      for J := I + 1 to RowCount do
      begin
         if Matrix[I, J] <> Matrix[J, I] then
            Exit;
      end;
   end;
   Result := True;
end;

procedure TMatrix.SortCol(SortType: TMatrixSortType; Col: Cardinal);
var
```

```
  I, J: Cardinal;
  B: Extended;
begin
  //
  case SortType of
    mstAsc:
      begin
        for I := 1 to RowCount do
        begin
          for J := I + 1 to RowCount do
          begin
            if Matrix[I, Col] > Matrix[J, Col] then
            begin
              B := Matrix[I, Col];
              Matrix[I, Col] := Matrix[J, Col];
              Matrix[J, Col] := B;
            end;
          end;
        end;
      end;
    mstDesc:
      begin
        for I := 1 to RowCount do
        begin
          for J := I + 1 to RowCount do
          begin
            if Matrix[I, Col] < Matrix[J, Col] then
            begin
              B := Matrix[I, Col];
              Matrix[I, Col] := Matrix[J, Col];
              Matrix[J, Col] := B;
            end;
          end;
        end;
      end;
  end;
end;

procedure TMatrix.SortRow(SortType: TMatrixSortType; Row: Cardinal);
var
  I, J: Cardinal;
  B: Extended;
begin
  //
  case SortType of
    mstAsc:
      begin
        for I := 1 to ColCount do
        begin
          for J := I + 1 to ColCount do
          begin
            if Matrix[Row, I] > Matrix[Row, J] then
            begin
              B := Matrix[Row, I];
              Matrix[Row, I] := Matrix[Row, J];
              Matrix[Row, J] := B;
            end;
```

```
            end;
          end;
        end;
      mstDesc:
        begin
          for I := 1 to ColCount do
          begin
            for J := I + 1 to ColCount do
            begin
              if Matrix[Row, I] < Matrix[Row, J] then
              begin
                B := Matrix[Row, I];
                Matrix[Row, I] := Matrix[Row, J];
                Matrix[Row, J] := B;
              end;
            end;
          end;
        end;
  end;
end;

procedure TMatrix.Sort(SortType: TMatrixSortType);
begin
  if IsVector then
  begin
    if (RowCount = 1) then
    begin
      SortRow(SortType, 1);
    end
    else
    begin
      SortCol(SortType, 1);
    end;
  end
  else
    raise Exception.Create(c_matrix_msg_method_for_vector_only);
end;

function TMatrix.GetCellCount: Cardinal;
begin
  if IsVector then
  begin
    if (RowCount = 1) then
    begin
      Result := ColCount;
    end
    else
    begin
      Result := RowCount;
    end;
  end
  else
    raise Exception.Create(c_matrix_msg_method_for_vector_only);
end;

procedure TMatrix.LimiterLow(Limit: Extended; Value: Extended);
var
```

```
  I, J: Cardinal;
begin
  //
  for I := 1 to RowCount do
  begin
    for J := 1 to ColCount do
    begin
      if Matrix[I, J] < Limit then
        Matrix[I, J] := Value;
    end;
  end;
end;

function TMatrix.Transpose: TMatrix;
var
  I, J: Cardinal;
begin
  //
  Result := TMatrix.Create(ColCount, RowCount);
  try
    for I := 1 to RowCount do
    begin
      for J := 1 to ColCount do
      begin
        Result.Matrix[J, I] := Matrix[I, J];
      end;
    end;
  except
    Result.Free;
    raise;
  end;
end;

function TMatrix.IsVector: Boolean;
begin
  Result := (RowCount = 1) or (ColCount = 1);
end;

function TMatrix.IsSquareMatrix: Boolean;
begin
  Result := (RowCount = ColCount);
end;

function TMatrix.Multiplication(B: TMatrix): TMatrix;
var
  A: TMatrix;
  I, J, K: Integer;
  S: Extended;
begin
  A := Self;
  if (A.ColCount <> B.RowCount) then
    raise Exception.Create(c_matrix_msg_multiplication_not_allowed_for_this_matrix);

  Result := TMatrix.Create(A.RowCount, B.ColCount);
  try
    for I := 1 to Result.RowCount do
    begin
```

```
        for J := 1 to Result.ColCount do
        begin
          S := 0;
          for K := 1 to B.RowCount do
          begin
            S := S + A.Matrix[I, K] * B.Matrix[K, J];
          end;
          Result.Matrix[I, J] := S;
        end;
      end;
    except
      Result.Free;
      raise;
    end;
end;

procedure TMatrix.SetValue(Cell: Cardinal; Value: Extended);
begin
    if IsVector then
    begin
      if (RowCount = 1) then
      begin
        Matrix[1, Cell] := Value;
      end
      else
      begin
        Matrix[Cell, 1] := Value;
      end;
    end
    else
      raise Exception.Create(c_matrix_msg_method_for_vector_only);
end;

function TMatrix.GetValue(Cell: Cardinal): Extended;
begin
    if IsVector then
    begin
      if (RowCount = 1) then
      begin
        Result := Matrix[1, Cell];
      end
      else
      begin
        Result := Matrix[Cell, 1];
      end;
    end
    else
      raise Exception.Create(c_matrix_msg_method_for_vector_only);
end;

procedure TMatrix.FillMainDiagonal(Value: Extended);
var
    I: Cardinal;
begin
    if (IsSquareMatrix = False) then
      raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);
    for I := 1 to FColCount do
```

```
      Matrix[I, I] := Value;
end;

procedure TMatrix.Resize(RowCount, ColCount: Cardinal);
begin
   if (RowCount < 1) then
      raise Exception.Create(c_matrix_msg_row_count_must_be_greater_than_zero);
   if (ColCount < 1) then
      raise Exception.Create(c_matrix_msg_col_count_must_be_greater_than_zero);
   FColCount := ColCount;
   FRowCount := RowCount;
   MatrixCreate;
end;

procedure TMatrix.SaveToStream(S: TStream);
var
   I, J: Cardinal;
begin
   S.WriteBuffer(c_matrix_data_identificator, SizeOf(Extended));
   S.WriteBuffer(FRowCount, SizeOf(Cardinal));
   S.WriteBuffer(FColCount, SizeOf(Cardinal));
   for I := 1 to FRowCount do
   begin
      for J := 1 to FColCount do
      begin
         S.WriteBuffer(FMatrix[I - 1, J - 1], SizeOf(Extended));
      end;
   end;
   S.WriteBuffer(c_matrix_data_identificator, SizeOf(Extended));
end;

procedure TMatrix.LoadFromStream(S: TStream);
var
   I, J: Cardinal;

   function ReadExtended: Extended;
   begin
      Result := 0;
      S.ReadBuffer(Result, SizeOf(Extended));
   end;

   function ReadCardinal: Cardinal;
   begin
      Result := 0;
      S.ReadBuffer(Result, SizeOf(Cardinal));
   end;

begin
   if ReadExtended <> c_matrix_data_identificator then
      raise Exception.Create(c_matrix_msg_bad_data_identificator);
   FRowCount := ReadCardinal;
   FColCount := ReadCardinal;
   Resize(FRowCount, FColCount);
   Fill(0);
   for I := 1 to FRowCount do
   begin
      for J := 1 to FColCount do
```

```
      begin
        SetValue(I, J, ReadExtended);
      end;
    end;
    if ReadExtended <> c_matrix_data_identificator then
      raise Exception.Create(c_matrix_msg_bad_data_identificator);
end;

procedure TMatrix.SaveToFile(FileName: string);
var
    S: TMemoryStream;
begin
    S := TMemoryStream.Create;
    try
      SaveToStream(S);
      S.SaveToFile(FileName);
    finally
      S.Free;
    end;
end;

procedure TMatrix.LoadFromFile(FileName: string);
var
    S: TMemoryStream;
begin
    S := TMemoryStream.Create;
    try
      S.LoadFromFile(FileName);
      LoadFromStream(S);
    finally
      S.Free;
    end;
end;

constructor TMatrix.Create(RowCount, ColCount: Cardinal);
begin
    Resize(RowCount, ColCount);
    Fill(0);
end;

destructor TMatrix.Destroy;
begin
    MatrixFree;
end;

function TMatrix.GetRowSum(Row: Cardinal): Extended;
var
    I: Cardinal;
begin
    Result := 0;
    for I := 1 to ColCount do
    begin
      Result := Result + GetValue(Row, I);
    end;
end;

procedure TMatrix.MatrixCreate;
```

```
var
   I: Cardinal;
begin
   SetLength(FMatrix, RowCount);
   for I := 0 to RowCount - 1 do
   begin
      SetLength(FMatrix[I], ColCount);
   end;
end;

procedure TMatrix.MatrixFree;
var
   I: Cardinal;
begin
   for I := 0 to RowCount - 1 do
   begin
      SetLength(FMatrix[I], 0);
   end;
   SetLength(FMatrix, 0);
end;

procedure TMatrix.SetValue(Row, Col: Cardinal; Value: Extended);
begin
   TestCell(Row, Col);
   FMatrix[Row - 1][Col - 1] := Value;
end;

function TMatrix.GetValue(Row, Col: Cardinal): Extended;
begin
   TestCell(Row, Col);
   Result := FMatrix[Row - 1][Col - 1];
end;

procedure TMatrix.Fill(Value: Extended);
var
   I, J: Cardinal;
begin
   for I := 1 to RowCount do
   begin
      for J := 1 to ColCount do
      begin
         Matrix[I, J] := Value;
      end;
   end;
end;

procedure TMatrix.FillRandom(AFrom, ATo: Integer; Symmetric: Boolean);
var
   I, J: Cardinal;
begin
   for I := 1 to RowCount do
   begin
      for J := 1 to ColCount do
      begin
         Matrix[I, J] := RandomRange(AFrom, ATo);
      end;
   end;
```

```
  if Symmetric then
  begin
    if (IsSquareMatrix = False) then
      raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);
    for I := 1 to RowCount do
    begin
      for J := I + 1 to ColCount do
      begin
        Matrix[J, I] := Matrix[I, J];
      end;
    end;
  end;
end;

procedure TMatrix.FillRow(Row: Cardinal; Value: Extended);
var
  J: Cardinal;
begin
  for J := 1 to ColCount do
  begin
    Matrix[Row, J] := Value;
  end;
end;

procedure TMatrix.FillCol(Col: Cardinal; Value: Extended);
var
  I: Cardinal;
begin
  for I := 1 to RowCount do
  begin
    Matrix[I, Col] := Value;
  end;
end;

function TMatrix.TestRow(Row: Cardinal; WithoutException: Boolean): Boolean;
begin
  Result := not((Row < 1) or (Row > RowCount));
  if (Result = False) and (WithoutException = False) then
    raise Exception.Create(c_matrix_msg_row_index_out_of_range);
end;

function TMatrix.TestCol(Col: Cardinal; WithoutException: Boolean): Boolean;
begin
  Result := not((Col < 1) or (Col > ColCount));
  if (Result = False) and (WithoutException = False) then
    raise Exception.Create(c_matrix_msg_col_index_out_of_range);
end;

function TMatrix.TestCell(Row, Col: Cardinal; WithoutException: Boolean): Boolean;
begin
  Result := TestRow(Row, WithoutException) and TestCol(Col, WithoutException);
end;

function TMatrix.ToString: string;
var
  I, J: Cardinal;
```

```
  List: TStringList;
  S: String;

  function CorrectValue(Value: Extended): Extended;
  const
    c_Pr = 100;
  begin
    Result := Round(Value * c_Pr) / c_Pr;
  end;

begin
  Result := '';
  List := TStringList.Create;
  try
    for I := 1 to RowCount do
    begin
      S := '';
      for J := 1 to ColCount do
      begin
        if (S <> '') then
          S := S + ', ';
        S := S + FloatToStr(CorrectValue(Matrix[I, J]));
      end;
      List.Add(S);
    end;
    Result := List.Text;
  finally
    List.Free;
  end;
end;

procedure TMatrix.Copy(Source: TMatrix);
var
  I, J: Cardinal;
begin
  if (ColCount <> Source.ColCount) or (RowCount <> Source.RowCount) then
    raise Exception.Create(c_matrix_msg_method_for_equal_size_matrix_only);
  for I := 1 to RowCount do
  begin
    for J := 1 to ColCount do
    begin
      Matrix[I, J] := Source.Matrix[I, J];
    end;
  end;
end;

function TMatrix.Clone: TMatrix;
begin
  Result := TMatrix.Create(RowCount, ColCount);
  Result.Copy(Self);
end;

procedure TMatrix.DeleteRow(Row: Cardinal);
var
  Z, W: Cardinal;
begin
  TestCol(Row);
```

```
  if (RowCount <= 1) then
  begin
    raise Exception.Create(c_matrix_msg_can_not_delete_last_row);
  end;
  for Z := Row to RowCount - 1 do
  begin
    for W := 1 to ColCount do
    begin
      Matrix[Z, W] := Matrix[Z + 1, W];
    end;
  end;
  Resize(RowCount - 1, ColCount);
end;

procedure TMatrix.DeleteCol(Col: Cardinal);
var
  Z, W: Cardinal;
begin
  TestCol(Col);
  if (ColCount <= 1) then
  begin
    raise Exception.Create(c_matrix_msg_can_not_delete_last_col);
  end;
  for W := Col to ColCount - 1 do
  begin
    for Z := 1 to RowCount do
    begin
      Matrix[Z, W] := Matrix[Z, W + 1];
    end;
  end;
  Resize(RowCount, ColCount - 1);
end;

procedure TMatrix.AddCol;
begin
  Resize(RowCount, ColCount + 1);
end;

procedure TMatrix.AddRow;
begin
  Resize(RowCount + 1, ColCount);
end;

function TMatrix.fabs(X: Extended): Extended;
begin
  //
  Result := X;
  if Result < 0 then
    Result := -Result;
end;

function TMatrix.SVDJacobi(Precision: Extended; IterationLimit: Cardinal): TMatrix;
var
  P, Q, IterationCount: Cardinal;
  A, mtxP, mtxPT, NewA: TMatrix;
  S, NewS: Extended;
```

```
    /// <summary>
    /// Вычисление матрицы поворота
    /// </summary>
    function GetMatrixP(P, Q: Cardinal; A: TMatrix): TMatrix;
    var
      //
      theta, t, c, S: Extended;
    begin
      //
      Result := TMatrix.Create(A.RowCount, A.RowCount);
      try
        theta := (A.Matrix[Q, Q] - A.Matrix[P, P]) / (2 * A.Matrix[P, Q]);
        t := Sign(theta) / (fabs(theta) + sqrt(theta * theta + 1));
        c := 1 / sqrt(t * t + 1);
        S := t * c;
        Result.FillMainDiagonal(1);
        Result.Matrix[P, P] := c;
        Result.Matrix[Q, Q] := c;
        if (P < Q) then
        begin
          Result.Matrix[P, Q] := S;
          Result.Matrix[Q, P] := -S;
        end
        else
        begin
          Result.Matrix[P, Q] := -S;
          Result.Matrix[Q, P] := S;
        end;
      except
        Result.Free;
        raise;
      end;
    end;

  /// <summary>
  /// Вычисление новой матрицы "A" (перемножение матриц PT, A, P)
  /// </summary>
    function GetNewA(mtxPT, A, mtxP: TMatrix): TMatrix;
    var
      M1: TMatrix;
    begin
      //
      M1 := mtxPT.Multiplication(A);
      try
        Result := M1.Multiplication(mtxP);
      finally
        M1.Free;
      end;
    end;

  /// <summary>
  /// Метод вычисления суммы недиагональных элементов матрицы
  /// </summary>
    function GetMatrixNonDiagonalSum(m: TMatrix): Extended;
    var
      I, J: Cardinal;
    begin
```

```
      //
      if (m.ColCount <> m.RowCount) then
        raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);

      Result := 0;
      for I := 1 to m.RowCount do
      begin
        for J := I + 1 to m.ColCount do
        begin
          if (I <> J) then
          begin
            Result := Result + m.Matrix[I, J] * m.Matrix[I, J];
          end;
        end;
      end;
    end;
/// <summary>
/// Функция вывода результатов вычислений
/// </summary>
    function GetResult(A: TMatrix): TMatrix;
    var
      I: Cardinal;
    begin
      // Создаем вектор результатов вычислений
      Result := TMatrix.Create(A.RowCount, 1);
      try
        // Записываем количество итераций
        Result.Tag := IterationCount;
        // Заносим в вектор результатов значения из главной диагонали матрицы
        for I := 1 to Result.CellCount do
          Result.Vector[I] := A.Matrix[I, I];
        // Сортируем значения вектора по убыванию
        // И получаем на выходе вектор сингулярных чисел
        Result.Sort(mstAsc);
      except
        Result.Free;
        raise;
      end;
    end;

/// <summary>
/// Метод выбора пары (P, Q) в цикле
/// </summary>
    procedure GetCycleSelectionPQ(A: TMatrix; var P: Cardinal; var Q: Cardinal);
    begin
      Inc(Q);

      if (P >= 1) and (P <= A.RowCount) and (Q > A.ColCount) then
      begin
        //
        P := P + 1;
        Q := P + 1;
      end;

      if (P < 1) or (P > A.RowCount - 1) then
      begin
        //
```

```
          P := 1;
          Q := P + 1;
          Exit;
        end;
    end;

begin
    // Проверяем, является ли матрица квадратной
    if (Self.ColCount <> Self.RowCount) then
        raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);
    // Проверяем исходную матрицу на симметричность
    if (Self.IsSymmetricMatrix = False) then
        raise Exception.Create(c_matrix_msg_method_for_symmetric_matrix_only);
    // Проверяем размер матрицы
    if Self.RowCount < 2 then
        raise Exception.Create(c_matrix_msg_bad_matrix_size);
    // Проверяем величину точности
    if Precision < 0 then
        raise Exception.Create(c_matrix_msg_precision_con_not_below_zero);
    // Обнуляем счетчик итераций
    IterationCount := 0;
    // Обнуляем P и Q
    P := 0;
    Q := 0;
    // Создаем клон матрицы
    A := Self.Clone;
    try
        // Вычисляем сумму недиагональных элементов матрицы
        S := GetMatrixNonDiagonalSum(A);
        // Запускаем цикл вычислений
        while True do
        begin
            // Проверяем лимит количества итераций
            if (IterationCount > IterationLimit) then
                raise Exception.Create(c_matrix_msg_iteration_count_above_allowable_value);
            // Выбираем пары P, Q
            GetCycleSelectionPQ(A, P, Q);
            // Вычисляем матрицу поворота
            mtxP := GetMatrixP(P, Q, A);
            try
                // Транспонируем матрицу поворота
                mtxPT := mtxP.Transpose;
                try
                    // Пересчитываем исходную матрицу
                    NewA := GetNewA(mtxPT, A, mtxP);
                    A.Free;
                    A := NewA;
                finally
                    mtxPT.Free;
                end;
            finally
                mtxP.Free;
            end;
            // Выполняем приращение счетчика итераций
            Inc(IterationCount);
            // Вычисляем сумму недиагональных элементов
            NewS := GetMatrixNonDiagonalSum(A);
```

```
        // Проверяем условие окончания цикла
        if (fabs(S - NewS) <= Precision) then
        begin
          // Выходим из цикла
          // Создаем вектор результатов вычислений
          Result := GetResult(A);
          Exit;
        end
        else
        begin
          S := NewS;
        end;
      end;
    finally
      A.Free;
    end;
end;

function TMatrix.SVDJacobiModification(Precision: Extended; IterationLimit: Cardinal): TMatrix;
var
    P, Q, IterationCount, Number: Cardinal;
    A, mtxP, mtxPT, NewA: TMatrix;
    S, NewS: Extended;
    MatrixValues: TStringList;

    /// <summary>
    /// Вычисление матрицы поворота
    /// </summary>
    function GetMatrixP(P, Q: Cardinal; A: TMatrix): TMatrix;
    var
      //
      theta, t, c, S: Extended;
    begin
      //
      Result := TMatrix.Create(A.RowCount, A.RowCount);
      try
        theta := (A.Matrix[Q, Q] - A.Matrix[P, P]) / (2 * A.Matrix[P, Q]);
        t := Sign(theta) / (fabs(theta) + sqrt(theta * theta + 1));
        c := 1 / sqrt(t * t + 1);
        S := t * c;
        Result.FillMainDiagonal(1);
        Result.Matrix[P, P] := c;
        Result.Matrix[Q, Q] := c;
        if (P < Q) then
        begin
          Result.Matrix[P, Q] := S;
          Result.Matrix[Q, P] := -S;
        end
        else
        begin
          Result.Matrix[P, Q] := -S;
          Result.Matrix[Q, P] := S;
        end;
      except
        Result.Free;
        raise;
      end;
```

```
    end;

/// <summary>
/// Вычисление новой матрицы "A" (перемножение матриц PT, A, P)
/// </summary>
    function GetNewA(mtxPT, A, mtxP: TMatrix): TMatrix;
    var
      M1: TMatrix;
    begin
      //
      M1 := mtxPT.Multiplication(A);
      try
        Result := M1.Multiplication(mtxP);
      finally
        M1.Free;
      end;
    end;

/// <summary>
/// Метод вычисления суммы недиагональных элементов матрицы
/// </summary>
    function GetMatrixNonDiagonalSum(m: TMatrix): Extended;
    var
      I, J: Cardinal;
    begin
      //
      if (m.ColCount <> m.RowCount) then
        raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);

      Result := 0;
      for I := 1 to m.RowCount do
      begin
        for J := I + 1 to m.ColCount do
        begin
          if (I <> J) then
          begin
            Result := Result + m.Matrix[I, J] * m.Matrix[I, J];
          end;
        end;
      end;
    end;
/// <summary>
/// Функция вывода результатов вычислений
/// </summary>
    function GetResult(A: TMatrix): TMatrix;
    var
      I: Cardinal;
    begin
      // Создаем вектор результатов вычислений
      Result := TMatrix.Create(A.RowCount, 1);
      try
        // Записываем количество итераций
        Result.Tag := IterationCount;
        // Заносим в вектор результатов значения из главной диагонали матрицы
        for I := 1 to Result.CellCount do
          Result.Vector[I] := A.Matrix[I, I];
        // Сортируем значения вектора по убыванию
```

```
        // И получаем на выходе вектор сингулярных чисел
        Result.Sort(mstAsc);
      except
        Result.Free;
        raise;
      end;
    end;

/// <summary>
/// Метод для получения следующей пары (P, Q)
/// </summary>
    procedure GetPQ(MatrixValueList: TStringList; var Number: Cardinal; var P: Cardinal; var Q:
Cardinal);
    var
      V: TMatrixValue;
    begin
      Inc(Number);
      if (Number > MatrixValueList.Count) or (Number < 1) then
        Number := 1;
      V := MatrixValueList.Objects[Number - 1] as TMatrixValue;
      P := V.Row;
      Q := V.Col;
    end;

/// <summary>
/// Метод расчета списка элементов (P, Q) который отсортирован по убыванию значения матрицы
соответствующего паре (P, Q)
/// </summary>
    function GetMatrixValueList(A: TMatrix): TStringList;
    var
      I, J: Cardinal;
      V, V1, V2: TMatrixValue;
    begin
      //
      Result := TStringList.Create;
      try
        Result.OwnsObjects := True;
        for I := 1 to A.RowCount do
        begin
          for J := I + 1 to A.ColCount do
          begin
            V := TMatrixValue.Create;
            V.Row := I;
            V.Col := J;
            V.Value := A.Matrix[I, J];
            Result.AddObject('', V);
          end;
        end;

        for I := 0 to Result.Count - 1 do
        begin
          for J := I + 1 to Result.Count - 1 do
          begin
            V1 := Result.Objects[I] as TMatrixValue;
            V2 := Result.Objects[J] as TMatrixValue;
            if (fabs(V1.Value) < fabs(V2.Value)) then
            begin
```

```
                //
                V := V1;
                Result.Objects[I] := V2;
                Result.Objects[J] := V;
              end;
            end;
          end;
        except
          Result.Free;
          raise;
        end;
      end;

begin
  // Проверяем, является ли матрица квадратной
  if (Self.ColCount <> Self.RowCount) then
    raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);
  // Проверяем исходную матрицу на симметричность
  if (Self.IsSymmetricMatrix = False) then
    raise Exception.Create(c_matrix_msg_method_for_symmetric_matrix_only);
  // Проверяем размер матрицы
  if Self.RowCount < 2 then
    raise Exception.Create(c_matrix_msg_bad_matrix_size);
  // Проверяем величину точности
  if Precision < 0 then
    raise Exception.Create(c_matrix_msg_precision_con_not_below_zero);
  // Обнуляем счетчик итераций
  IterationCount := 0;
  // Обнуляем P и Q
  P := 0;
  Q := 0;
  // Обнуляем номер пары
  Number := 0;
  // Создаем клон матрицы
  A := Self.Clone;
  try
    // Вычисляем убывающую последовательность
    MatrixValues := GetMatrixValueList(A);
    try
      // Вычисляем сумму недиагональных элементов матрицы
      S := GetMatrixNonDiagonalSum(A);
      // Запускаем цикл вычислений
      while True do
      begin
        // Проверяем лимит количества итераций
        if (IterationCount > IterationLimit) then
          raise Exception.Create(c_matrix_msg_iteration_count_above_allowable_value);
        // Выбираем пары P, Q
        GetPQ(MatrixValues, Number, P, Q);
        // Вычисляем матрицу поворота
        mtxP := GetMatrixP(P, Q, A);
        try
          // Транспонируем матрицу поворота
          mtxPT := mtxP.Transpose;
          try
            // Пересчитываем исходную матрицу
            NewA := GetNewA(mtxPT, A, mtxP);
```

```
                A.Free;
                A := NewA;
              finally
                mtxPT.Free;
              end;
            finally
              mtxP.Free;
            end;
            // Выполняем приращение счетчика итераций
            Inc(IterationCount);
            // Вычисляем сумму недиагональных элементов
            NewS := GetMatrixNonDiagonalSum(A);
            // Проверяем условие окончания цикла
            if (fabs(S - NewS) <= Precision) then
            begin
              // Выходим из цикла
              // Создаем вектор резултатов вычислений
              Result := GetResult(A);
              Exit;
            end
            else
            begin
              S := NewS;
            end;
          end;
        finally
          MatrixValues.Free;
        end;
      finally
        A.Free;
      end;
    end;

function TMatrix.SVDJacobiWithoutMatrix(Precision: Extended; IterationLimit: Cardinal): TMatrix;
var
  P, Q, IterationCount: Cardinal;
  A, mtxP, mtxPT, NewA: TMatrix;
  S, NewS: Extended;

  /// <summary>
  /// Вычисление новой матрицы "A"
  /// </summary>
  function GetNewA(P, Q: Cardinal; A: TMatrix): TMatrix;
  var
    //
    theta, t, c, S: Extended;
    R, I, J: Cardinal;
  begin
    // Клонируем матрицу
    Result := A.Clone;
    try
      // Вычисляем угол поворота
      theta := (A.Matrix[Q, Q] - A.Matrix[P, P]) / (2 * A.Matrix[P, Q]);
      // Расчитываем t
      t := Sign(theta) / (fabs(theta) + sqrt(theta * theta + 1));
      // Вычисляем числовые значения для модификации
      c := 1 / sqrt(t * t + 1);
```

```
        S := t * c;
        // Модифицируем строки и столбцы вне главной диагонали симметричной матрицы
        for R := 1 to A.RowCount do
        begin
          if (R <> P) and (R <> Q) then
          begin
            Result.Matrix[R, P] := c * A.Matrix[R, P] - S * A.Matrix[R, Q];
            Result.Matrix[R, Q] := c * A.Matrix[R, Q] + S * A.Matrix[R, P];
            Result.Matrix[P, R] := Result.Matrix[R, P];
            Result.Matrix[Q, R] := Result.Matrix[R, Q];
          end;
        end;
        // Модифицируем элементы расположенные на побочной и главной диагонали
        Result.Matrix[P, P] := c * c * A.Matrix[P, P] + S * S * A.Matrix[Q, Q] - 2 * c * S * A.Matrix[P,
Q];
        Result.Matrix[Q, Q] := S * S * A.Matrix[P, P] + c * c * A.Matrix[Q, Q] + 2 * c * S * A.Matrix[P,
Q];
        Result.Matrix[P, Q] := (c * c - S * S) * A.Matrix[P, Q] + c * S * (A.Matrix[P, P] - A.Matrix[Q,
Q]);
        Result.Matrix[Q, P] := Result.Matrix[P, Q];
      except
        Result.Free;
        raise;
      end;
    end;

/// <summary>
/// Метод вычисления суммы недиагональных элементов матрицы
/// </summary>
    function GetMatrixNonDiagonalSum(m: TMatrix): Extended;
    var
      I, J: Cardinal;
    begin
      //
      if (m.ColCount <> m.RowCount) then
        raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);

      Result := 0;
      for I := 1 to m.RowCount do
      begin
        for J := I + 1 to m.ColCount do
        begin
          if (I <> J) then
          begin
            Result := Result + m.Matrix[I, J] * m.Matrix[I, J];
          end;
        end;
      end;
    end;
/// <summary>
/// Функция вывода результатов вычислений
/// </summary>
    function GetResult(A: TMatrix): TMatrix;
    var
      I: Cardinal;
    begin
      // Создаем вектор результатов вычислений
```

```
      Result := TMatrix.Create(A.RowCount, 1);
      try
        // Записываем количество итераций
        Result.Tag := IterationCount;
        // Заносим в вектор результатов значения из главной диагонали матрицы
        for I := 1 to Result.CellCount do
          Result.Vector[I] := A.Matrix[I, I];
        // Сортируем значения вектора по убыванию
        // И получаем на выходе вектор сингулярных чисел
        Result.Sort(mstAsc);
      except
        Result.Free;
        raise;
      end;
    end;

/// <summary>
/// Метод выбора пары (P, Q) в цикле
/// </summary>
    procedure GetCycleSelectionPQ(A: TMatrix; var P: Cardinal; var Q: Cardinal);
    begin
      Inc(Q);

      if (P >= 1) and (P <= A.RowCount) and (Q > A.ColCount) then
      begin
        //
        P := P + 1;
        Q := P + 1;
      end;

      if (P < 1) or (P > A.RowCount - 1) then
      begin
        //
        P := 1;
        Q := P + 1;
        Exit;
      end;
    end;

begin
    // Проверяем, является ли матрица квадратной
    if (Self.ColCount <> Self.RowCount) then
      raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);
    // Проверяем исходную матрицу на симметричность
    if (Self.IsSymmetricMatrix = False) then
      raise Exception.Create(c_matrix_msg_method_for_symmetric_matrix_only);
    // Проверяем размер матрицы
    if Self.RowCount < 2 then
      raise Exception.Create(c_matrix_msg_bad_matrix_size);
    // Проверяем величину точности
    if Precision < 0 then
      raise Exception.Create(c_matrix_msg_precision_con_not_below_zero);
    // Обнуляем счетчик итераций
    IterationCount := 0;
    // Обнуляем P и Q
    P := 0;
    Q := 0;
```

```
  // Создаем клон матрицы
  A := Self.Clone;
  try
    // Вычисляем сумму недиагональных элементов матрицы
    S := GetMatrixNonDiagonalSum(A);
    // Запускаем цикл вычислений
    while True do
    begin
      // Проверяем лимит количества итераций
      if (IterationCount > IterationLimit) then
        raise Exception.Create(c_matrix_msg_iteration_count_above_allowable_value);
      // Выбираем пары P, Q
      GetCycleSelectionPQ(A, P, Q);
      // Пересчитываем исходную матрицу
      NewA := GetNewA(P, Q, A);
      A.Free;
      A := NewA;
      // Выполняем приращение счетчика итераций
      Inc(IterationCount);
      // Вычисляем сумму недиагональных элементов
      NewS := GetMatrixNonDiagonalSum(A);
      // Проверяем условие окончания цикла
      if (fabs(S - NewS) <= Precision) then
      begin
        // Выходим из цикла
        // Создаем вектор результатов вычислений
        Result := GetResult(A);
        Exit;
      end
      else
      begin
        S := NewS;
      end;
    end;
  finally
    A.Free;
  end;
end;

function TMatrix.SVDJacobiModificationWithoutMatrix(Precision: Extended; IterationLimit: Cardinal):
TMatrix;
var
  P, Q, IterationCount, Number: Cardinal;
  A, mtxP, mtxPT, NewA: TMatrix;
  S, NewS: Extended;
  MatrixValues: TStringList;

  /// <summary>
  /// Вычисление новой матрицы "A"
  /// </summary>
  function GetNewA(P, Q: Cardinal; A: TMatrix): TMatrix;
  var
    //
    theta, t, c, S: Extended;
    R, I, J: Cardinal;
  begin
    // Клонируем матрицу
```

```
      Result := A.Clone;
      try
        // Вычисляем угол поворота
        theta := (A.Matrix[Q, Q] - A.Matrix[P, P]) / (2 * A.Matrix[P, Q]);
        // Расчитываем t
        t := Sign(theta) / (fabs(theta) + sqrt(theta * theta + 1));
        // Вычисляем числовые значения для модификации
        c := 1 / sqrt(t * t + 1);
        S := t * c;
        // Модифицируем строки и столбцы вне главной диагонали симметричной матрицы
        for R := 1 to A.RowCount do
        begin
          if (R <> P) and (R <> Q) then
          begin
            Result.Matrix[R, P] := c * A.Matrix[R, P] - S * A.Matrix[R, Q];
            Result.Matrix[R, Q] := c * A.Matrix[R, Q] + S * A.Matrix[R, P];
            Result.Matrix[P, R] := Result.Matrix[R, P];
            Result.Matrix[Q, R] := Result.Matrix[R, Q];
          end;
        end;
        // Модифицируем элементы расположенные на побочной и главной диагонали
        Result.Matrix[P, P] := c * c * A.Matrix[P, P] + S * S * A.Matrix[Q, Q] - 2 * c * S * A.Matrix[P,
Q];
        Result.Matrix[Q, Q] := S * S * A.Matrix[P, P] + c * c * A.Matrix[Q, Q] + 2 * c * S * A.Matrix[P,
Q];
        Result.Matrix[P, Q] := (c * c - S * S) * A.Matrix[P, Q] + c * S * (A.Matrix[P, P] - A.Matrix[Q,
Q]);
        Result.Matrix[Q, P] := Result.Matrix[P, Q];
      except
        Result.Free;
        raise;
      end;
    end;

/// <summary>
/// Метод вычисления суммы недиагональных элементов матрицы
/// </summary>
    function GetMatrixNonDiagonalSum(m: TMatrix): Extended;
    var
      I, J: Cardinal;
    begin
      //
      if (m.ColCount <> m.RowCount) then
        raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);

      Result := 0;
      for I := 1 to m.RowCount do
      begin
        for J := I + 1 to m.ColCount do
        begin
          if (I <> J) then
          begin
            Result := Result + m.Matrix[I, J] * m.Matrix[I, J];
          end;
        end;
      end;
    end;
```

```
/// <summary>
/// Функция вывода результатов вычислений
/// </summary>
  function GetResult(A: TMatrix): TMatrix;
  var
    I: Cardinal;
  begin
    // Создаем вектор результатов вычислений
    Result := TMatrix.Create(A.RowCount, 1);
    try
      // Записываем количество итераций
      Result.Tag := IterationCount;
      // Заносим в вектор результатов значения из главной диагонали матрицы
      for I := 1 to Result.CellCount do
        Result.Vector[I] := A.Matrix[I, I];
      // Сортируем значения вектора по убыванию
      // И получаем на выходе вектор сингулярных чисел
      Result.Sort(mstAsc);
    except
      Result.Free;
      raise;
    end;
  end;

/// <summary>
/// Метод для получения следующей пары (P, Q)
/// </summary>
  procedure GetPQ(MatrixValueList: TStringList; var Number: Cardinal; var P: Cardinal; var Q:
Cardinal);
  var
    V: TMatrixValue;
  begin
    Inc(Number);
    if (Number > MatrixValueList.Count) or (Number < 1) then
      Number := 1;
    V := MatrixValueList.Objects[Number - 1] as TMatrixValue;
    P := V.Row;
    Q := V.Col;
  end;

/// <summary>
/// Метод расчета списка элементов (P, Q) который отсортирован по убыванию значения матрицы
соответствующего паре (P, Q)
/// </summary>
  function GetMatrixValueList(A: TMatrix): TStringList;
  var
    I, J: Cardinal;
    V, V1, V2: TMatrixValue;
  begin
    //
    Result := TStringList.Create;
    try
      Result.OwnsObjects := True;
      for I := 1 to A.RowCount do
      begin
        for J := I + 1 to A.ColCount do
        begin
```

```
            V := TMatrixValue.Create;
            V.Row := I;
            V.Col := J;
            V.Value := A.Matrix[I, J];
            Result.AddObject('', V);
          end;
        end;

        for I := 0 to Result.Count - 1 do
        begin
          for J := I + 1 to Result.Count - 1 do
          begin
            V1 := Result.Objects[I] as TMatrixValue;
            V2 := Result.Objects[J] as TMatrixValue;
            if (fabs(V1.Value) < fabs(V2.Value)) then
            begin
              //
              V := V1;
              Result.Objects[I] := V2;
              Result.Objects[J] := V;
            end;
          end;
        end;
      except
        Result.Free;
        raise;
      end;
    end;

begin
    // Проверяем, является ли матрица квадратной
    if (Self.ColCount <> Self.RowCount) then
      raise Exception.Create(c_matrix_msg_method_for_square_matrix_only);
    // Проверяем исходную матрицу на симметричность
    if (Self.IsSymmetricMatrix = False) then
      raise Exception.Create(c_matrix_msg_method_for_symmetric_matrix_only);
    // Проверяем размер матрицы
    if Self.RowCount < 2 then
      raise Exception.Create(c_matrix_msg_bad_matrix_size);
    // Проверяем величину точности
    if Precision < 0 then
      raise Exception.Create(c_matrix_msg_precision_con_not_below_zero);
    // Обнуляем счетчик итераций
    IterationCount := 0;
    // Обнуляем P и Q
    P := 0;
    Q := 0;
    // Обнуляем номер пары
    Number := 0;
    // Создаем клон матрицы
    A := Self.Clone;
    try
      // Вычисляем убывающую последовательность
      MatrixValues := GetMatrixValueList(A);
      try
        // Вычисляем сумму недиагональных элементов матрицы
        S := GetMatrixNonDiagonalSum(A);
```

```
        // Запускаем цикл вычислений
        while True do
        begin
          // Проверяем лимит количества итераций
          if (IterationCount > IterationLimit) then
            raise Exception.Create(c_matrix_msg_iteration_count_above_allowable_value);
          // Выбираем пары P, Q
          GetPQ(MatrixValues, Number, P, Q);
          // Пересчитываем исходную матрицу
          NewA := GetNewA(P, Q, A);
          A.Free;
          A := NewA;
          // Выполняем приращение счетчика итераций
          Inc(IterationCount);
          // Вычисляем сумму недиагональных элементов
          NewS := GetMatrixNonDiagonalSum(A);
          // Проверяем условие окончания цикла
          if (fabs(S - NewS) <= Precision) then
          begin
            // Выходим из цикла
            // Создаем вектор результатов вычислений
            Result := GetResult(A);
            Exit;
          end
          else
          begin
            S := NewS;
          end;
        end;
      finally
        MatrixValues.Free;
      end;
    finally
      A.Free;
    end;
end;

end.
```

```
<?xml version="1.0" encoding="utf-8"?>
<nodeSet version="1.0">
  <view uin="ulwsil8uk9l_v">
    <property name="$defaultDiagram" value="true" />
    <property name="$metaclass" value="Package Diagram" />
    <property name="$name" value="default" />
```

```
        <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uClusterManager.TClusterSeg
ment">
          <property name="$shortcutReference" value="true" />
          <property name="bounds" value="340,540,160,131" />
          <property name="location_set_by_user" value="True" />
          <property name="bounds_setted_by_user" value="True" />
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uClusterManager.TClusterSeg
ment.Row" />
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uClusterManager.TClusterSeg
ment.Col" />
          <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uClusterManager.TClusterS
egment.Create()" />
          <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uClusterManager.TClusterS
egment.Destroy()" />
        </reference>
        <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uClusterManager.TClusterMa
nager">
          <property name="$shortcutReference" value="true" />
          <property name="bounds" value="340,80,160,253" />
          <property name="location_set_by_user" value="True" />
          <property name="bounds_setted_by_user" value="True" />
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uClusterManager.TClusterMan
ager.FMatrix" />
```

```
		<reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uClusterManager.TClusterMan
ager.FClusterID" />
		<reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uClusterManager.TClusterMan
ager.FClusterList" />
		<reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uClusterManager.TCluster
Manager.Create()" />
		<reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uClusterManager.TCluster
Manager.Destroy()" />
		<reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uClusterManager.TCluster
Manager.LoadFromFile(System.string)" />
		<reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uClusterManager.TCluster
Manager.SaveToFile(System.string)" />
		<reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uClusterManager.TCluster
Manager.GetClusters()" />
		<reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uClusterManager.TCluster
Manager.ClusterByID(System.Cardinal)" />
		<reference
referencedUin="delphi:e_property:src:ClusterAnalyzer:uClusterManager.TCluster
Manager.Matrix" />
		<reference
referencedUin="delphi:e_property:src:ClusterAnalyzer:uClusterManager.TCluster
Manager.ClusterID" />
```

```
        <reference
referencedUin="delphi:e_property:src:ClusterAnalyzer:uClusterManager.TCluster
Manager.ClusterList" />
        <reference
referencedUin="delphi:l_ast:src:ClusterAnalyzer:uClusterManager.TClusterManag
er.Matrix">
          <property name="sourceAnchor" value="500,182" />
          <property name="targetAnchor" value="571,182" />
          <property name="bendpoints" value="" />
        </reference>
        <reference
referencedUin="delphi:l_ast:src:ClusterAnalyzer:uClusterManager.TClusterManag
er.FClusterID">
          <property name="sourceAnchor" value="500,190" />
          <property name="targetAnchor" value="571,190" />
          <property name="bendpoints" value="" />
        </reference>
        <reference
referencedUin="delphi:l_ast:src:ClusterAnalyzer:uClusterManager.TClusterManag
er.FMatrix">
          <property name="sourceAnchor" value="500,198" />
          <property name="targetAnchor" value="571,198" />
          <property name="bendpoints" value="" />
        </reference>
        <reference
referencedUin="delphi:l_ast:src:ClusterAnalyzer:uClusterManager.TClusterManag
er.ClusterID">
          <property name="sourceAnchor" value="500,206" />
          <property name="targetAnchor" value="571,206" />
          <property name="bendpoints" value="" />
```

```
            </reference>
          </reference>
          <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uClusterManager.TCluster">
            <property name="$shortcutReference" value="true" />
            <property name="bounds" value="340,360,160,146" />
            <property name="location_set_by_user" value="True" />
            <property name="bounds_setted_by_user" value="True" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uClusterManager.TCluster.ID"
/>
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uClusterManager.TCluster.Clu
sterType" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uClusterManager.TCluster.Seg
ments" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uClusterManager.TCluster.
Create()" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uClusterManager.TCluster.
Destroy()" />
          </reference>
          <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uClusterManager.@unitclass"
>
            <property name="$shortcutReference" value="true" />
          </reference>
```

```
        <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uMatrix.TMatrix">
            <property name="$shortcutReference" value="true" />
            <property name="bounds" value="571,68,274,868" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uMatrix.TMatrix.FColCount"
/>
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uMatrix.TMatrix.FRowCount"
/>
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uMatrix.TMatrix.FMatrix" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.MatrixCre
ate()" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.MatrixFre
e()" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.GetCellCo
unt()" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.fabs(Syste
m.Extended)" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:uMatrix.TMatrix.Tag" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.Create(Sys
tem.Cardinal,System.Cardinal)" />
```

<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.Destroy()" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.TestCell(System.Cardinal,System.Cardinal,System.Boolean)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.TestRow(System.Cardinal,System.Boolean)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.TestCol(System.Cardinal,System.Boolean)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.IsVector()" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.IsSquareMatrix()" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.Copy(uMatrix.TMatrix)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.Fill(System.Extended)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.FillRandom(System.Integer,System.Integer,System.Boolean)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.FillRow(System.Cardinal,System.Extended)" />

<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.FillCol(System.Cardinal,System.Extended)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.DeleteRow(System.Cardinal)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.DeleteCol(System.Cardinal)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.AddCol()" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.AddRow()" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.SetValue(System.Cardinal,System.Cardinal,System.Extended)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.SetValue(System.Cardinal,System.Extended)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.GetValue(System.Cardinal,System.Cardinal)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.GetValue(System.Cardinal)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.ToString()" />

```
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.Clone()"
/>
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.GetRowSu
m(System.Cardinal)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.Resize(Sy
stem.Cardinal,System.Cardinal)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.Multiplica
tion(uMatrix.TMatrix)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.SaveToStr
eam(System.Classes.TStream)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.LoadFrom
Stream(System.Classes.TStream)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.SaveToFil
e(System.string)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.LoadFrom
File(System.string)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.FillMainD
iagonal(System.Extended)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.LimiterLo
w(System.Extended,System.Extended)" />
```

```
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.SortCol(u
Matrix.TMatrixSortType,System.Cardinal)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.SortRow(u
Matrix.TMatrixSortType,System.Cardinal)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.Sort(uMat
rix.TMatrixSortType)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.SVDJacob
i(System.Extended,System.Cardinal)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.SVDJacob
iWithoutMatrix(System.Extended,System.Cardinal)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.SVDJacob
iModification(System.Extended,System.Cardinal)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.SVDJacob
iModificationWithoutMatrix(System.Extended,System.Cardinal)" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.IsSymmetr
icMatrix()" />
        <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:uMatrix.TMatrix.Transpose
()" />
        <reference
referencedUin="delphi:e_property:src:ClusterAnalyzer:uMatrix.TMatrix.CellCoun
t" />
```

```
            <reference
referencedUin="delphi:e_property:src:ClusterAnalyzer:uMatrix.TMatrix.RowCou
nt" />
            <reference
referencedUin="delphi:e_property:src:ClusterAnalyzer:uMatrix.TMatrix.ColCount
" />
            <reference
referencedUin="delphi:e_property:src:ClusterAnalyzer:uMatrix.TMatrix.Vector[S
ystem.Cardinal]" />
            <reference
referencedUin="delphi:e_property:src:ClusterAnalyzer:uMatrix.TMatrix.Matrix[S
ystem.Cardinal,System.Cardinal]" />
          </reference>
        </view>
      </nodeSet>
      <?xml version="1.0" encoding="utf-8"?>
      <nodeSet version="1.0">
        <view uin="800ec0m4y0l_v">
          <property name="$defaultDiagram" value="true" />
          <property name="$metaclass" value="Package Diagram" />
          <property name="$name" value="default" />
          <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufCluster.@unitclass">
            <property name="$shortcutReference" value="true" />
            <reference
referencedUin="delphi:l_ast:src:ClusterAnalyzer:ufCluster.@unitclass.FCluster"
/>
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.@unitclass.FCluster"
/>
```

```
        </reference>
        <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufCluster.TfCluster">
          <property name="$shortcutReference" value="true" />
          <property name="bounds" value="10,10,207,685" />
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.odMain"
/>
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.sdMain" />
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.amMain"
/>
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.aFileOpen
" />
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.aFileSave"
/>
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.aClear" />
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.sgCluster"
/>
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.mmMain"
/>
          <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.N1" />
```

```
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.N2" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.N3" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.N4" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.aCreateClu
ster" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.N8" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.aExit" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.N9" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.aAnalysis"
/>
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.aEditMode
" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.aEditMode
On1" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.N7" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.N10" />
        <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.N12" />
```

<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.N13" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.StatusBar1" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.FormCreate(System.TObject)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.FormDestroy(System.TObject)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.sgClusterDrawCell(System.TObject,System.Integer,System.Integer,System.Types.TRect,TGridDrawState)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.aFileSaveExecute(System.TObject)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.aFileOpenExecute(System.TObject)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.aClearExecute(System.TObject)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.aCreateClusterExecute(System.TObject)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.aExitExecute(System.TObject)" />

```
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.aAnalys
isExecute(System.TObject)" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.aEditM
odeExecute(System.TObject)" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.aEditM
odeOffExecute(System.TObject)" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.sgClust
erClick(System.TObject)" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.sgClust
erSelectCell(System.TObject,System.Integer,System.Integer,ref@System.Boolean)
" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.FClusterM
anager" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufCluster.TfCluster.FEditMode
" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:ufCluster.TfCluster.ShowCl
uster(uClusterManager.TClusterManager)" />
          </reference>
        </view>
      </nodeSet>

      <?xml version="1.0" encoding="utf-8"?>
```

```
<nodeSet version="1.0">
  <view uin="gelktauw7gi_v">
    <property name="$defaultDiagram" value="true" />
    <property name="$metaclass" value="Package Diagram" />
    <property name="$name" value="default" />
    <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo">
      <property name="$shortcutReference" value="true" />
      <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.PageControl1" />
      <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.StatusBar1" />
      <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.TabSheet1" />
      <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.TabSheet3" />
      <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.sgSource" />
      <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClusters" />
      <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersID" />
```

<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersSize" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.dsClusters" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.DBChart1" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersType" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersA" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersB" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersAID" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersASize" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersBID" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersBSize" />

<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersStat" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersStatSize" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersStatType" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cdsClustersStatPercent" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.Splitter1" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.Series1" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.Panel1" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.DBGrid1" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cbMoveToFirstPoint" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.

sgSourceDrawCell(System.TObject,System.Integer,System.Integer,System.Types.TRect,TGridDrawState)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.FormCreate(System.TObject)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.dsClustersDataChange(System.TObject,Data.DB.TField)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.cbMoveToFirstPointClick(System.TObject)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.sgSourceSelectCell(System.TObject,System.Integer,System.Integer,ref@System.Boolean)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.FormKeyPress(System.TObject,ref@System.Char)" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.FCurrentClusterID" />
<reference referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.FClusterManager" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.ShowClusterInfo(uClusterManager.TClusterManager)" />
<reference referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo.ShowMatrix(uMatrix.TMatrix,Vcl.Grids.TStringGrid)" />

```
            </reference>
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufClusterInfo.@unitclass">
              <property name="$shortcutReference" value="true" />
              <reference
referencedUin="delphi:l_ast:src:ClusterAnalyzer:ufClusterInfo.@unitclass.fCluste
rInfo" />
              <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterInfo.@unitclass.fClus
terInfo" />
            </reference>
          </view>
        </nodeSet>
        <?xml version="1.0" encoding="utf-8"?>
        <nodeSet version="1.0">
          <view uin="lg6ip94syzi_v">
            <property name="$defaultDiagram" value="true" />
            <property name="$metaclass" value="Package Diagram" />
            <property name="$name" value="default" />
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufClusterSettings.@unitclass"
>
              <property name="$shortcutReference" value="true" />
              <reference
referencedUin="delphi:l_ast:src:ClusterAnalyzer:ufClusterSettings.@unitclass.fCl
usterSettings" />
              <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.@unitclass.f
ClusterSettings" />
            </reference>
```

```
        <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufClusterSettings.TfClusterSet
tings">
            <property name="$shortcutReference" value="true" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.TfClusterSett
ings.eColCount" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.TfClusterSett
ings.Label1" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.TfClusterSett
ings.eRowCount" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.TfClusterSett
ings.Label2" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.TfClusterSett
ings.bOk" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.TfClusterSett
ings.bCancel" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.TfClusterSett
ings.cbFillRandom" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterSettings.TfCluster
Settings.bCancelClick(System.TObject)" />
```

```
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterSettings.TfCluster
Settings.bOkClick(System.TObject)" />
            <reference
referencedUin="delphi:e_method:src:ClusterAnalyzer:ufClusterSettings.TfCluster
Settings.FormKeyPress(System.TObject,ref@System.Char)" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.TfClusterSett
ings.RowCount" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.TfClusterSett
ings.ColCount" />
            <reference
referencedUin="delphi:e_field:src:ClusterAnalyzer:ufClusterSettings.TfClusterSett
ings.FillRandom" />
          </reference>
        </view>
      </nodeSet>
      <?xml version="1.0" encoding="utf-8"?>
      <nodeSet version="1.0">
        <view uin="lnzh0ode7s03s03sr_v">
          <property name="$defaultDiagram" value="true" />
          <property name="$metaclass" value="Package Diagram" />
          <property name="$name" value="default" />
          <reference referencedUin="design:view:::ulwsil8uk9l_v">
            <property name="$shortcutReference" value="true" />
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uClusterManager.TClusterSeg
ment" />
```

```
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uClusterManager.TClusterMa
nager" />
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uClusterManager.TCluster" />
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uClusterManager.@unitclass"
/>
          </reference>
          <reference referencedUin="design:view:::800ec0m4y0l_v">
            <property name="$shortcutReference" value="true" />
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufCluster.@unitclass" />
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufCluster.TfCluster" />
          </reference>
          <reference referencedUin="design:view:::gelktauw7gi_v">
            <property name="$shortcutReference" value="true" />
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufClusterInfo.TfClusterInfo"
/>
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufClusterInfo.@unitclass" />
          </reference>
          <reference referencedUin="design:view:::evh74yxj5uq_v">
            <property name="$shortcutReference" value="true" />
            <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ClusterAnalyzer.@unitclass"
/>
          </reference>
```

```
            <reference referencedUin="design:view:::uf6nviiwjzh_v">
              <property name="$shortcutReference" value="true" />
              <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uMatrix.TMatrixSortType" />
              <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uMatrix.@unitclass" />
              <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uMatrix.TMatrixValue" />
              <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:uMatrix.TMatrix" />
            </reference>
            <reference referencedUin="design:view:::lg6ip94syzi_v">
              <property name="$shortcutReference" value="true" />
              <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufClusterSettings.@unitclass"
/>
              <reference
referencedUin="delphi:e_class:src:ClusterAnalyzer:ufClusterSettings.TfClusterSet
tings" />
            </reference>
          </view>
        </nodeSet>
```