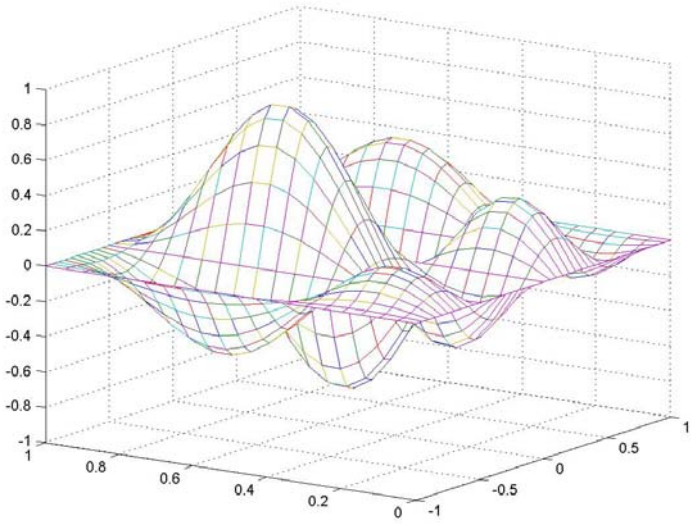
Д.С. Дворецкий, А.А. Ермаков, Е.В. Пешкова

РАСЧЕТ И ОПТИМИЗАЦИЯ ПРОЦЕССОВ И АППАРАТОВ ХИМИЧЕСКИХ И ПИЩЕВЫХ ПРОИЗВОДСТВ В СРЕДЕ MATLAB



ИЗДАТЕЛЬСТВО ТГТУ

Министерство образования и науки Российской Федерации

Государственное образовательное учреждение
высшего профессионального образования
«Тамбовский государственный технический университет»

Д.С. Дворецкий, А.А. Ермаков, Е.В. Пешкова

РАСЧЕТ И ОПТИМИЗАЦИЯ ПРОЦЕССОВ И АППАРАТОВ ХИМИЧЕСКИХ И ПИЩЕВЫХ ПРОИЗВОДСТВ В СРЕДЕ MATLAB

Учебное пособие

Под редакцией доктора технических наук, профессора
С.И. Дворецкого

Тамбов
Издательство ТГТУ
2005

УДК 66.011(07)
ББК Л11-1с116я73
Д24

Р е ц е н з е н т ы:

Доктор технических наук, профессор
Ю.В. Литовка

Доктор технических наук, профессор
А.А. Арзамасцев

Дворецкий Д.С., Ермаков А.А., Пешкова Е.В.

Д24 Расчет и оптимизация процессов и аппаратов химических и пищевых производств в среде MatLab: Учеб. пособие / Под ред. д-ра техн. наук, проф. С.И. Дворецкого. Тамбов: Изд-во Тамб. гос. техн. ун-та, 2005. 80 с.

Рассмотрены примеры расчета аппаратурного оформления и моделирования химических и пищевых процессов в среде MatLab, проведенного на основе единой стратегии компьютерного моделирования машин и аппаратов.

Рекомендовано для аспирантов и студентов специальностей 170500 – Машины и аппараты химических производств и 170600 – Машины и аппараты пищевых производств.

Работа выполнена при финансовой поддержке гранта «Студенты, аспиранты и молодые ученые – малому наукоемкому бизнесу "Ползуновские гранты"».

УДК 66.011(07)
ББК Л11-1с116я73

ISBN 5-8265-0213-4

Учебное издание

Дворецкий Дмитрий Станиславович,
Ермаков Александр Анатольевич,
Пешкова Евгения Владимировна

РАСЧЕТ И ОПТИМИЗАЦИЯ ПРОЦЕССОВ И АППАРАТОВ ХИМИЧЕСКИХ И ПИЩЕВЫХ ПРОИЗВОДСТВ В СРЕДЕ MATLAB

Учебное пособие

Редактор Т.М. Глинкина
Компьютерное макетирование Е.В. Кораблевой

Подписано в печать 15.02.2005.
Формат 60 × 84 / 16. Бумага офсетная. Печать офсетная.
Гарнитура Times New Roman. Объем: 4,65 усл. печ. л.; 4,5 уч.-изд. л.
Тираж 100 экз. С. 97^{M}

Издательско-полиграфический центр
Тамбовского государственного технического университета,
392000, Тамбов, Советская, 106, к. 14

ВВЕДЕНИЕ

Одним из наиболее мощных и универсальных пакетов прикладных программ, обеспечивающих решение типовых математических задач, возникающих в различных областях человеческой деятельности, является пакет MatLab фирмы MathWorks. Спектр численных методов математического анализа, которые реализованы в пакете MatLab, весьма широк и охватывает методы численного интегрирования, интерполяции и приближения функций, линейной алгебры, решения систем нелинейных уравнений и обыкновенных дифференциальных уравнений, уравнений математической физики, задач оптимизации, нечеткой логики и др. Пакет MatLab обладает хорошо развитыми возможностями интерпретации двумерных и трехмерных массивов данных, снабжен встроенным языком программирования, позволяющим сравнительно легко создавать собственные программы. Пользователь пакета MatLab может также в процессе работы совершенствовать свои знания как в области компьютерного моделирования и численных методов, так и программирования и визуализации результатов расчета.

1 МЕТОДЫ РАСЧЕТА И ОПТИМИЗАЦИИ ПРОЦЕССОВ И АППАРАТОВ ХИМИЧЕСКОЙ И ПИЩЕВОЙ ТЕХНОЛОГИЙ

Методы анализа и синтеза процессов и аппаратов химических и пищевых производств в отечественной науке и промышленности, как и их зарубежные аналоги в «Chemical Engineering», находятся в стадии интенсивного фундаментального, прикладного и методологического развития [1]. Научные успехи связаны, прежде всего, с существенным углублением физических представлений о механизме процессов, математизацией и компьютеризацией. Ускорение промышленного развития обусловлено как остро вставшими энергетическими, сырьевыми, экономическими проблемами, так и недавно появившимися возможностями интенсификации и совмещения технологических процессов, оптимизации и интегрированного проектирования аппаратурно-технологического оформления процессов и систем автоматического управления ими [2].

Методологические изменения отражают стоящие перед высшей школой задачи подготовки специалистов широкого профиля, способных не только представлять сложные физико-химические процессы в виде уравнений, но и уметь их рассчитывать и воплощать в конкретном аппаратурно-технологическом оформлении, обеспечивающем строгие требования к целевым продуктам, ресурсосбережению и экологической безопасности производства.

Особенностью современных химических и пищевых технологий является увеличение темпов и масштабов роста промышленности, повышение качества и конкурентоспособности выпускаемой продукции, резкое увеличение единичной мощности аппаратов и поточных линий, автоматизация и роботизация производства. Стали ведущими проблемы создания теории непрерывных технологических процессов, единых кинетических закономерностей, гибких автоматизированных производств и т.д., включая вопросы инженерной экологии и энергосбережения.

В настоящее время в большинстве технологических, машиностроительных и политехнических вузов студенты изучают теорию основных процессов, принципы устройства и методы расчета типовых машин и аппаратов, в которых осуществляются эти процессы, на основе фундаментальных законов физики, химии, математики, термодинамики и других наук. Особо следует отметить широкое применение методов математического моделирования, оптимизации и системного анализа.

Особенностью современных химических и пищевых технологий, протекающих с высокими скоростями при высоких температурах и давлениях в многофазных системах, является их большая сложность, обуславливаемая нелинейностью, большим числом переменных (параметров), определяющих течение процессов, внутренних связей между переменными и их взаимным влиянием. Кроме того, на процесс накладываются внешние случайные возмущения, которые не учитываются при расчете процессов и аппаратов химических и пищевых производств. В результате объем перерабатываемой информации при расчетах достаточно велик и для того, чтобы пропустить эту информацию по вполне ограниченным каналам нашего восприятия, мы вынуждены уменьшать (регулировать) этот объем и тем самым ограничивать количество возможностей, между которыми делается выбор. Это достигается познанием процесса (явления) через модели – упрощенные «эквиваленты», которые отражают в нужном направлении стороны явлений изучаемого процесса.

1.1 Методология математического моделирования

Невозможно себе представить современную науку о процессах и аппаратах без широкого применения математического (компьютерного) моделирования. Сущность этой методологии состоит в замене исходного объекта его «образом» – математической моделью и дальнейшем изучении модели с помощью реализуемых на компьютере вычислительно-логических алгоритмов [3]. Этот метод познания, конструирования, проектирования сочетает в себе многие достоинства как теории, так и эксперимента. Работа не с самим объектом, а с его моделью дает возможность относительно быстро и без существенных затрат исследовать его свойства и поведение в любых мыслимых ситуациях (преимущества теории). В то же время вычислительные (компьютерные, имитационные) эксперименты с моделями процессов позволяют, опираясь на мощь современных вычислительных методов и технических инструментов информатики, подробно и глубоко изучать процессы в достаточной полноте, недоступной чисто теоретическим методам (преимущества вычислительного эксперимента).

Технологические, технические, экологические, экономические и иные системы, изучаемые современной наукой, больше не поддаются исследованию (в нужной полноте и точности) обычными теоретическими методами. Прямой натурный эксперимент над ними длителен, дорог, часто опасен либо просто невозможен, так как многие из этих систем существуют в «единственном экземпляре». Цена ошибок и просчетов в обращении с ними недопустимо высока. Поэтому математическое моделирование является неизбежной составляющей научно-технического прогресса.

Сейчас математическое (компьютерное) моделирование вступает в третий принципиально важный этап своего развития, «встраиваясь» в структуру так называемого информационного общества [3]. Впечатляющий прогресс средств переработки, передачи и хранения информации отвечает мировым тенденциям к усложнению и взаимному проникновению различных сфер человеческой деятельности. Без овладения информационными «ресурсами» нельзя думать о решении все более усложняющихся задач химической и пищевой технологий. Однако информация как таковая зачастую мало что дает для анализа и синтеза, для принятия решений и контроля за их исполнением. Нужны надежные способы переработки информационного «сырья» в готовый «продукт», т.е. в точное знание. Математическое (компьютерное, информационное) моделирование и является интеллектуальным ядром информационных технологий, всего процесса информатизации общества.

Сама постановка задачи о математическом моделировании какого-либо объекта (явления, процесса) порождает четкий план действий. Его условно можно разбить на три этапа: модель – алгоритм – программа (рис. 1).

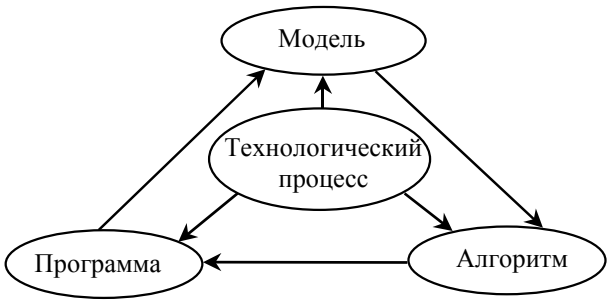


Рис. 1 Триада «модель – алгоритм – программа»

На первом этапе выбирается (или строится) «эквивалент» технологического процесса, отражающий в математической форме важнейшие его свойства – законы, которым он подчиняется, связи, присущие составляющим его частям, и т.д. Математическая модель (или ее фрагменты) исследуется теоретическими методами, что позволяет получить важные предварительные знания об объекте.

Второй этап – выбор (или разработка) алгоритма для реализации модели на компьютере. Модель представляется в форме, удобной для применения численных методов, определяется последовательность вычислительных и логических операций, которые нужно произвести, чтобы найти искомые величины с заданной точностью.

Вычислительные алгоритмы не должны искажать основные свойства модели и, следовательно, исходного технологического процесса, быть экономичными и адаптирующимися к особенностям решаемых задач и используемых компьютеров.

На третьем этапе создаются программы, «переводящие» модель и алгоритм на доступный компьютеру язык. К ним также предъявляются требования экономичности и адаптивности. Их можно

назвать «электронными» эквивалентами изучаемого технологического процесса, уже пригодными для непосредственного испытания на «экспериментальной установке» – компьютере.

Создав триаду «модель – алгоритм – программа», исследователь получает в руки универсальный, гибкий и недорогой инструмент, который вначале настраивается (отлаживается), тестируется в «пробных» вычислительных экспериментах. После того, как адекватность (соответствие с заданной точностью реальному процессу) триады удостоверена, с моделью (электронным эквивалентом) проводятся разнообразные и подробные «опыты», дающие все требуемые качественные и количественные свойства и характеристики технологического процесса. Процесс моделирования сопровождается улучшением и уточнением, по мере необходимости, всех звеньев триады.

Будучи методологией, математическое моделирование не подменяет собой математику, физику, биологию и другие научные дисциплины, не конкурирует с ними. Наоборот, трудно переоценить его синтезирующую роль. Создание и применение триады невозможно без опоры на самые разные методы и подходы – от качественного анализа нелинейных моделей до современных языков и систем программирования. Появляются новые дополнительные стимулы самым разным направлениям науки.

1.2 Общие принципы анализа и расчета процессов и аппаратов

К одним из важнейших принципов науки о процессах и аппаратах химической и пищевой технологий относятся теоретические и технологические обобщения и выявление физико-химических аналогий основных процессов.

При исследовании и расчете процессов и аппаратов важно знать кинетические закономерности основных процессов химической и пищевой технологий.

Кинетика – это учение о механизмах и скоростях процессов, в том числе гидродинамических, тепло- и массообменных. Кинетика является научной основой создания новых и совершенствования действующих процессов и аппаратов химической и пищевой технологий.

По общепринятой классификации, основанной на кинетических закономерностях процессов, различают [1]:

1 *Гидромеханические процессы* (рис. 1.2), скорость которых определяется законами гидродинамики:

$$j_{\text{г}} = \frac{dV}{Fd\tau} = \frac{\Delta p}{R_1} = k_1 \Delta p, \qquad (1.1)$$

где $j_{\text{г}}$ – скорость процесса; V – объем протекающей жидкости; F – площадь сечения аппарата; τ – время; k_1 – коэффициент скорости процесса (величина, обратная гидравлическому сопротивлению R_1); Δp – перепад давления (движущая сила процесса).

2 *Теплообменные процессы* (рис. 1.3), скорость которых определяется законами теплопередачи:

$$j_{\text{т}} = \frac{dQ}{Fd\tau} = \frac{\Delta t}{R_2} = k_2 \Delta t, \qquad (1.2)$$

где $j_{\text{т}}$ – скорость процесса; Q – количество переданного тепла; F – поверхность теплообмена; τ – время; k_2 – коэффициент теплопередачи (величина, обратная термическому сопротивлению R_2); Δt – средняя разность температур между обменивающимися теплом материалами (движущая сила процесса).

3 *Массообменные* (*диффузионные*) *процессы* (рис. 1.4), скорость которых определяется скоростью перехода вещества из одной фазы в другую:

$$j_{\text{м}} = \frac{dM}{Fd\tau} = \frac{\Delta c}{R_3} = k_3 \Delta c, \qquad (1.3)$$

где $j_м$ – скорость процесса; M – количество вещества, перенесенного из одной фазы в другую; F – поверхность контакта фаз; τ – время; k_3 – коэффициент массопередачи (величина, обратная диффузионному сопротивлению R_3); Δc – разность между равновесной и рабочей концентрациями вещества в фазах (движущая сила процесса).

4 *Механические процессы* (рис. 1.5), скорость которых определяется законами физики твердого тела.

5 *Химические процессы*, связанные с превращением веществ и изменением их химических свойств. Скорость этих процессов определяется закономерностями химической кинетики:

$$j_x = \frac{dM}{V_p d\tau} = k_4 f(c), \qquad (1.4)$$

где j_x – скорость процесса; M – количество прореагировавшего в химическом процессе вещества; V_p – объем реактора; τ – время; k_4 – коэффициент скорости химического процесса; $f(c)$ – движущая сила процесса, которая является функцией реагирующих веществ.

6 *Биохимические процессы,* связанные с синтезом веществ и осуществляемые под воздействием и при непосредственном участии живых микроорганизмов и выделенных из них ферментов (биологических катализаторов). Скорость биохимических процессов, как и в предыдущем случае, определяется скоростью роста культуры в зависимости от концентрации одного или нескольких наиболее важных компонентов среды, обеспечивающих основу метаболизма. Эти компоненты получили название *лимитирующих субстратов.*

Приведенная классификационная система основных процессов химической и пищевой технологий удобна тем, что позволяет устанавливать единую номенклатуру типовой аппаратуры, используемой для этих процессов. Так, например, при классификации химических реакторов можно руководствоваться несколькими классификационными признаками: 1) способом организации процесса; 2) фазовым составом смеси; 3) гидродинамическими условиями проведения процесса в реакторе; 4) теплообменными условиями процесса в реакторе; 5) временными изменениями процесса; 6) конструктивными особенностями реактора; 7) агрегатным состоянием фазы и др.

По первому признаку (по способу подвода сырья и отвода продукта) различают периодические, полупериодические (полунепрерывные) непрерывнодействующие аппараты-реакторы.

По второму для проведения гетерогенных процессов выделяются системы газ-жидкость, жидкость-твердое и газ-твердое; для проведения гомогенных процессов – газо- и жидкофазные; отдельно рассматриваются реакторы для гетеро-каталитических процессов.

По третьему признаку за основу классификации берется режим движения агентов в аппарате. В зависимости от гидродинамических условий аппараты для осуществления химических реакций разделяют на реакторы смешения (аппараты с мешалками), вытеснения (трубчатые) и промежуточного типа.

По четвертому признаку учитываются тепловые эффекты процессов и рассматриваются реакторы адиабатические (без теплообмена с окружающей средой), автотермические (необходимая для процесса температура поддерживается без внешних источников тепла), изотермические (постоянная температура в аппарате поддерживается за счет внешних источников теплоты) и с промежуточными тепловыми режимами.

По пятому признаку в реакторах могут быть реализованы стационарные (статические), нестационарные (динамические) режимы работы.

По шестому признаку – конструктивному – различают реакторы емкостные (аппараты с мешалкой, автоклавы, барботажные и пр.), колонные (с насадкой или тарелками); типа теплообменников (трубчатые, пленочные и пр.); со взвешенным, движущимся и неподвижным слоем катализатора; аппараты высокого давления и температуры, электролизеры и пр.

Классификация реакционных аппаратов по седьмому признаку – агрегатному состоянию основной фазы в реакторе – перекликается с классификацией по второму признаку: различают аппараты с газовой, жидкой и твердой фазой. Первые в свою очередь разделяют на контактные (с неподвижным и движущимся слоем катализатора) и высокотемпературные; вторые делят по конструктивным признакам на емкостные (вертикальные и горизонтальные), колонные (насадочные, тарельчатые и пустотелые) и змеевиковые; третьи – на камерные, барабанные, лопастные и с псевдоожиженным слоем.

Анализ технологического процесса начинается с определения условий равновесия системы в соответствии с законами гидродинамики и термодинамики [4]. Наибольшее число N переменных (парамет-

ров), которое можно изменять не нарушая равновесия, определяют с помощью правила фаз Гиббса для различных систем:

$$N = K + 2 - \Phi,$$

где Ф – число фаз; K – число компонентов системы; N – число степеней свободы, т.е. число независимых переменных, значения которых можно произвольно изменять без изменения числа или вида (состава) фаз в системе.

По характерным равновесным и рабочим параметрам определяют движущую силу процесса, используемую для расчета основных размеров технологического аппарата. По данным о равновесии составляют материальный баланс прихода и расхода веществ:

$$\sum M_{\text{н}} = \sum M_{\text{к}},$$

где $\sum M_{\text{н}}, \sum M_{\text{к}}$ – количество исходных и конечных веществ, соответственно.

Тепловой баланс системы можно описать уравнением вида:

$$\sum Q_{\text{н}} + \sum Q_p = \sum Q_{\text{к}} + \sum Q_{\text{пот}},$$

где $\sum Q_{\text{н}}, \sum Q_{\text{к}}$ – теплота, поступающая в аппарат с исходными материалами, и теплота, отводимая из аппарата с конечными продуктами, соответственно; $\sum Q_p$ – тепловой эффект процесса; $\sum Q_{\text{пот}}$ – потери теплоты в окружающую среду.

Используя уравнения материального и теплового балансов, определяют основной размер аппарата (площадь поперечного сечения, поверхность теплопередачи, диаметр и высоту массообменного аппарата, объем), например, из кинетических соотношений (1.1) – (1.4).

Наиболее полный и точный расчет процессов и аппаратов позволяют провести математические модели. С их помощью можно целенаправленно исследовать механизм процесса в целом, изучить его отдельные стороны и явления, а также влияние различных переменных (параметров), которое обеспечит оптимальные условия его осуществления. Результаты математического моделирования могут быть перенесены на промышленную установку лишь в том случае, если будет установлена адекватность гидродинамической модели структуре потоков в этой установке.

При решении уравнений модели применяют три класса методов [5]: точные, асимптотические и приближенные (численные).

Точные методы позволяют аналитически получать искомые величины, не допуская при этом каких-либо упрощений исходной задачи. Аналитическое решение допускают линейные задачи (модели), описываемые линейными дифференциальными уравнениями в обыкновенных и частных производных сравнительно невысокого порядка.

Подавляющее большинство задач (моделей) химической и пищевой технологии невозможно точно решить аналитическими методами из-за нелинейности уравнений или граничных условий, зависимости коэффициентов уравнений от потенциалов переноса, сложной формы границ и т.п.

Асимптотические методы позволяют решать сложные задачи путем составления упрощенных (модельных) уравнений, без которых невозможно выявить соответствующий физический механизм, адекватно интерпретировать и четко понять явление.

Кроме определения качественных закономерностей изучаемого процесса, такое упрощение описывающих его уравнений открывает путь и к количественному решению задач. На этом этапе исследования асимптотические разложения позволяют получить не только первое приближение, но и формализованные высшие. Однако для успешного применения их необходимо, чтобы исходное асимптотическое разложение имело достаточно большое число членов.

Приближенные (численные) методы основаны на замене дифференциальных соотношений дискретными моделями (например, конечно-разностными аппроксимациями), которые представляют собой системы нелинейных алгебраических уравнений [4]. Точность и скорость вычислительного процесса зависят от способа аппроксимации дифференциальных соотношений, выбора геометрии и плотности

покрывающих сеток, выбора и организации алгоритма решения полученных конечно-разностных уравнений.

Численные методы универсальны и позволяют эффективно решать различного рода задачи. Они остаются основным аппаратом исследования задач химической и пищевой технологии, связанных с расчетом, оптимизацией, управлением и проектированием технологических процессов.

Приближенные методы инженерного типа применяются для получения достаточно грубых оценок на предварительном этапе любого исследования, для сравнительно быстрого получения результата, а также для качественного понимания того или иного явления или процесса. При этом конкретные представления о механизме изучаемого процесса черпаются непосредственно из практической деятельности или эксперимента. Точность используемых приближенных методов обычно оценивается на примере частных случаев, для которых уже имеются необходимые точные численные или экспериментальные результаты.

1.3 ИНТЕГРИРОВАННОЕ ПРОЕКТИРОВАНИЕ ТЕХНОЛОГИЧЕСКИХ ПРОЦЕССОВ И АППАРАТОВ: СТРАТЕГИЯ, МЕТОДЫ, РЕАЛИЗАЦИЯ

1.3.1 Работоспособность технологических процессов (аппаратов)

Традиционно при проектировании технологических процессов и аппаратов решается следующая задача оптимизации: требуется определить тип аппаратурного оформления $a^* \in A$, векторы конструктивных $d^* \in D$ и режимных параметров $z^* \in Z$, при которых достигается минимум целевой функции $I(\cdot)$ проектирования, т.е.

$$\bar{I}\left(a^*, d^*, z^*, x, \xi^N\right) = \min_{a,d,z} I\left(a, d, z, \xi^N\right),$$

при связях и ограничениях

$$\begin{aligned} h_i\left(a, d, z, x, \xi^N\right) &= 0, \ i = 1, 2, ..., k; \\ g_j\left(a, d, z, x, \xi^N\right) &\le 0, \ j \in J, \end{aligned},$$

где $I(\cdot)$ – множество индексов ограничений работоспособности; x – вектор переменных состояния или выходных переменных технологического процесса; ξ^N – номинальное (среднее) значение вектора возмущающих воздействий (неопределенных параметров), имеющих место при проектировании.

Если вектор x выразить, может быть неявно, как функцию a, d, z, ξ^N из уравнений математической модели технологического процесса $h\left(a, d, z, x, \xi^N\right) = 0$ и представить в виде функции $\bar{I}\left(a, d, z, x, \xi^N\right)$ и $\bar{g}\left(a, d, z, x, \xi^N\right)$, то получим известную «приведенную» постановку задачи оптимизации:

$$\min_{a,d,z} \bar{I}\left(a, d, z, \xi^N\right), \tag{1.8}$$

при ограничениях

$$g_j\left(a, d, z, \xi^N\right) \le 0, \ j \in J. \tag{1.9}$$

Учет неопределенности вектора ξ при традиционном проектировании осуществлялся введением эмпирического коэффициента запаса $\gamma_{зап}$ (обычно $\gamma_{зап} = 1{,}25$) к размерам оборудования, полученным в результате решения задачи нелинейного программирования (1.8), (1.9). Понятно, что традиционная процедура не имеет рациональной основы для выбора коэффициента запаса $\gamma_{зап}$, что зачастую приводит к

неработоспособности спроектированного технологического процесса (аппарата) и необходимости его перепроектирования.

Новый подход к задаче проектирования заключается в учете имеющей место неопределенности в исходных данных. Предположим, нам известны номинальное значение вектора неопределенных параметров ξ^N и ожидаемые отклонения $\Delta\xi^+$, $\Delta\xi^-$ от него $\xi^L = \xi^N - \Delta\xi^-$, $\xi^U = \xi^N + \xi^+$. Тогда область Ξ, содержащую все возможные значения неопределенных параметров, можно представить в следующем виде:

$$\Xi = \left\{\xi \middle| \xi^L \le \xi \le \xi^U\right\}.$$

Задача анализа работоспособности проектируемого процесса (аппарата), определяемого векторами проектных параметров *a* и *d*, будет заключаться в определении режимных (управляющих) переменных *z* таких, чтобы выполнить ограничения работоспособности (требования по спецификации качества выпускаемой продукции, производительности, надежности, безопасности производства и др.):

$$g_j(a,\, d,\, z,\, \xi) \le 0,\ \ j \in J\,, \qquad (1.10)$$

для всех $\xi \in \Xi$. Математически эта задача может быть сформулирована следующим образом [6]:

$$\psi(a,\, d,\, \xi) = \min_z \max_{j \in J} g_j(a,\, d,\, z,\, \xi) \le 0\,, \qquad (1.11)$$

где $\psi(\cdot)$ – функция выполнимости ограничения (1.10).

Задачу (1.11) можно переформулировать в форме стандартной задачи математического программирования, определяя скалярную величину α такую, что

$$\psi_j(a, d, \xi) = \min_{z,\alpha} \alpha \qquad (1.12)$$

при ограничениях

$$g_j(a,\, d,\, z,\, \xi) \le \alpha,\ \ j \in J. \qquad (1.13)$$

Для установления работоспособности проектируемого процесса (аппарата) необходимо убедиться в том, что для всех $\xi \in \Xi$ выполняются ограничения (1.10). В этом случае задача анализа работоспособности проектируемого процесса (аппарата), описываемого векторами проектных параметров *a* и *d*, может быть сформулирована в следующем виде [6]:

$$\chi(a,\, d) = \max_{\xi \in \Xi} \psi(a,\, d,\, \xi), \qquad (1.14)$$

где $\psi(a,\, d)$ – соответствует функции работоспособности процесса (аппарата) с аппаратурным оформлением типа *a* и вектором конструктивных переменных *d*.

При $\chi(a,\, d) \le 0$ работоспособность процесса и аппарата может быть достигнута для всей области Ξ возможных изменений вектора ξ.

Функция работоспособности процесса (аппарата) может быть записана и в вероятностной форме:

$$\chi(a,\, d) = \text{Вер}_\xi \left\{ \min_z \max_{j \in J} g_j(a,\, d,\, z,\, \xi) \le 0 \right\}. \qquad (1.15)$$

1.3.2 ПОСТАНОВКА ЗАДАЧ И МЕТОДЫ РЕШЕНИЯ ЗАДАЧ ИНТЕГРИРОВАННОГО ПРОЕКТИРОВАНИЯ

Для формулировки задач оптимизации с учетом неопределенностей необходимо задать формулу целевой функции и определить ограничения. В основе этого определения лежит концепция двух этапов процесса и аппарата: этапа проектирования (на этом этапе неопределенность присутствует практически везде) и этапа эксплуатации. На втором этапе возможны следующие случаи:

Задача 1. На этапе эксплуатации процесса и аппарата область Ξ изменения неопределенных параметров та же, что и на этапе проектирования. В этом случае необходимо определить векторы a^*, d^*, z^*, при которых достигается минимум целевой функции

$$I_1(a^*, d^*, z^*) = \min_{a,d,z} E_\xi \{I(a, d, z, \xi)\}, \qquad (1.16)$$

при ограничениях:

в жесткой форме:

$$F_1(d) = \min_{z} \max_{j \in J} \max_{\xi \in \Xi} g_j(a, d, z, \xi) \le 0. \qquad (1.17)$$

или в мягкой «вероятностной» форме:

$$\text{Вер}_\xi \left[\max_{j \in J} g_j(a, d, z, \xi) \le 0 \right] \ge \rho_{\text{зад}}, \qquad (1.18)$$

где $\rho_{\text{зад}}$ – заданный уровень вероятности выполнения ограничений.

Сформулированная выше задача стохастической оптимизации относится к классу так называемых одноэтапных задач, которые находят широкое применение в практике проектирования технологических процессов и аппаратов.

В терминах теории А-задач оптимизации [2, 6] одноэтапную задачу с вероятностными ограничениями (1.16), (1.18) можно переформулировать следующим образом: требуется найти *m*-мерный вектор постоянных величин $\alpha^* = (\alpha_1^*, \alpha_2^*, ..., \alpha_m^*)$, тип аппаратурного оформления a_{α^*}, векторы конструктивных d_{α^*} и режимных z_{α^*} переменных, такие, что

$$I(a_{\alpha^*}, d_{\alpha^*}, z_{\alpha^*}) = \min_{\alpha \in \Lambda} \{ \min_{a,d,z} \sum_{k=1}^{K} \gamma_k I(a, d, z, \xi^k) \mid g_j(a, d, z, \xi^k) \le \alpha_j, \ j \in J \}, \qquad (1.19)$$

где $\Lambda = \{\alpha \mid \forall_j \ \text{Вер}_\xi [g_j(a_\alpha, d_\alpha, z_\alpha, \xi) \le] \ge \rho_{\text{зад}}\}$, γ_k – весовые коэффициенты, которые присвоены каждой точке ξ^k, $\sum_{k=1}^{K} \gamma_k = 1$.

Алгоритм решения задачи (1.19)

Шаг 1. Положить $\nu = 0$ и выбрать начальное приближение вектора $\alpha^\nu = (\alpha_1^\nu, \alpha_2^\nu, ..., \alpha_m^\nu)$.

Шаг 2. Методом последовательного квадратичного программирования решить задачу нелинейного программирования

$$I(a_\alpha, d_\alpha, z_\alpha) = \min_{a,d,z} \sum_{k=1}^{K} \gamma_k I(a, d, z, \xi)$$

при ограничениях $g_j = (a, d, z, \xi^k) \le \alpha_j^\nu, \ j \in J, \ k = \overline{1, K}$.

Пусть точка $(a_{\alpha^\nu}, d_{\alpha^\nu}, z_{\alpha^\nu})$ – решение этой задачи.

Шаг 3. В точке $(a_{\alpha^\nu}, d_{\alpha^\nu}, z_{\alpha^\nu})$ вычисляются вероятности выполнения ограничений с использованием имитационной модели технологического процесса и проверяется выполнение условий

$$\text{Вер}_\xi \{g_j(a_{\alpha^\nu}, d_{\alpha^\nu}, z_{\alpha^\nu}, \xi) \le 0\} \ge \rho_{\text{зад}}, \ j \in J.$$

Шаг 4. Если вероятностные ограничения не выполняются для каких-то номеров $j \in J$, т.е. $\alpha^\nu \notin \Lambda$, то включается в работу алгоритм входа в допустимую область Λ. Далее принять $\nu := \nu + 1$ и перейти к шагу 2. В противном случае перейти к следующему шагу 5.

Шаг 5. Определить вектор α^* из решения внешней А-задачи оптимизации

$$I\left(a_{\alpha^*}, d_{\alpha^*}, z_{\alpha^*}\right) = \min_{\alpha\in\Lambda} I\left(a_\alpha, d_\alpha, z_\alpha\right).$$

Последняя задача может быть решена подходящим методом нелинейного программирования.

Для решения одноэтапной задачи интегрированного проектирования с жесткими ограничениями целесообразно воспользоваться алгоритмом, описанным в работе [7].

Задача 2. На этапе эксплуатации неопределенные параметры могут быть определены в каждый момент времени и управляющие переменные могут быть использованы для обеспечения выполнения ограничений. В этом случае требуется определить векторы a^*, d^*, при которых достигается минимум целевой функции

$$I_2\left(a^*, d^*\right) = \min_{a,d} E_\xi\left\{\min_z I(a, d, z, \xi) \mid g_j(a, d, z, \xi) \le 0, \quad j \in J\right\}, \qquad (1.20)$$

при условии

$$F_2(a, d) = \max_{\xi\in\Xi} \min_z \max_{j\in J} g_j(a, d, z, \xi) \le 0. \qquad (1.21)$$

Это так называемая двухэтапная задача стохастической оптимизации с жесткими ограничениями.

Пусть $S_2^{(\nu)} = \left\{\xi^i \mid i \in II^{(\nu)}\right\}$ – некоторое множество критических точек; ν – номер итерации; $II^{(\nu)}$ – множество номеров точек в $S_2^{(\nu)}$; S_1 – совокупность аппроксимационных точек.

Алгоритм решения задачи (1.20), (1.21)

Шаг 1. Принять $\nu = 0$. Выбрать совокупность аппроксимационных точек S_1 и начальное множество критических точек $S_2^{(\nu)}$.

Шаг 2. Решить задачу

$$I_2^{(\nu)} = \min_{a,d,z^k} \sum_{k=1}^{K} \gamma_k I\left(a, d, z^k, \xi^k\right)$$

при ограничениях

$$g_j\left(a, d, z^k, \xi^k\right) \le 0, \quad k = 1, 2, ..., K;$$
$$g_j\left(a, d, z^i, \xi^i\right) \le 0, \quad i \in II^{(\nu)}.$$

Пусть $z^{k,(\nu)}$, $z^{i,(\nu)}$, $d^{(\nu)}$ – решение этой задачи.

Шаг 3. С использованием специального алгоритма [7] вычислить значение $F_2(a, d)$. При этом будет получено некоторое значение $\xi^{(\nu)*}$. Проверить выполнение условия $F_2(a, d) \le 0$. Если оно выполняется, то решение задачи получено. В противном случае перейти к следующему шагу 4.

Шаг 4. Образовать новое множество ξ – критических точек $S_2^{(\nu+1)} = S_2^{\nu} \cup \left\{\xi^{(\nu)*}\right\}$, принять $\nu := \nu + 1$ и перейти к шагу 2.

Таким образом, оптимизационные задачи при интегрированном проектировании процессов и аппаратов могут быть сформулированы с учетом различных уровней информации, доступных на этапе эксплуатации производства. Причем каждое решение дает оптимальный вариант процесса и аппарата для данного уровня информации.

Следовательно, при интегрированном проектировании нужно учитывать, что само получение информации связано с определенными затратами. Разработка более точных моделей, установка новых измерительных приборов и систем автоматизированного управления для стабилизации режимных переменных процесса повышают уровень доступной информации о процессе, но требуют дополнительных

затрат. При этом возникает важная проблема выбора оптимального (или разумного) уровня экспериментальной информации в качестве исходных данных для проектирования процессов и аппаратов.

2 ОСНОВЫ РАБОТЫ В MATLAB

Все данные MatLab представляет в виде массивов, даже число представляет собой массив размерностью 1 на 1. Поэтому важно понять, как работать с массивами, поскольку без них невозможны эффективная работа в MatLab и, в частности построение графиков, статистическая обработка данных, решение задач линейной алгебры и т.д. *Массив* – упорядоченная, пронумерованная совокупность однородных данных. У массива должно быть *имя*. Массивы различаются по числу *размерностей* или *измерений*: одномерные, двумерные, многомерные. Доступ к элементам осуществляется при помощи *индекса*. В MatLab нумерация элементов массивов начинается с единицы, т.е. индексы должны быть больше или равны единице.

2.1 Простейшие вычисления и элементарные функции

Для вычисления различных математических выражений и выполнения последовательных действий в MatLab можно работать с командной строкой. Символ >> в рабочем пространстве обозначает вызов командной строки и специально его набирать не нужно. Набор любой команды или выражения должен заканчиваться нажатием клавиши <Enter>, обозначающим выполнение этой команды или вычисление математического выражения. Точка с запятой в конце выражения используется для подавления вывода результата выражения на экран.

При вычислениях возможны некоторые исключительные ситуации, например, деление на ноль, которые в большинстве языков программирования приводят к ошибке. При делении положительного числа на ноль в MatLab получается Inf (бесконечность), а при делении отрицательного числа получается –Inf (минус бесконечность).

```
>> 1/0
Warning: Divide by zero.

ans =

  Inf
```

При делении нуля на ноль получается NaN (не число).

```
>> 0/0
Warning: Divide by zero.

ans =

  NaN
```

При вычислении, например, $\sqrt{-1}$ MatLab автоматически переходит в область комплексных чисел.

```
>> sqrt(-1.0)

ans =

   0 + 1.0000i
```

Комплексные числа можно использовать в качестве аргументов встроенных элементарных функций.

MatLab использует стандартные арифметические операции, выполняя их в обычном порядке, свойственном большинству языков программирования:

- Возведение в степень «^»;
- Умножение и деление «*», «/»;

– Сложение и вычитание «+», «-».

Для изменения порядка выполнения арифметических операторов следует использовать круглые скобки.

MatLab оперирует с большим числом встроенных элементарных (тригонометрические, гиперболические, экспоненциальные, логарифмические) и комплекснозначных функций. Часто используемые функции приведены в табл. 1.

1 Встроенные элементарные функции MatLab

Тригонометрические, гиперболические и обратные им функции		Экспоненциальные, степенные функции, логарифмы	
sin, cos, tan, cot	синус, косинус, тангенс, котангенс	exp	Экспоненциальная функция
sec, csc	секанс, косеканс	log, log10	Натуральный и десятичный логарифм
asin, acos, atan, acot	арксинус, арккосинус, арктангенс, арккотангенс	pow2	Возведение числа 2 в степень
asec, acsc	арксеканс, арккосеканс	sqrt	Квадратный корень
sinh, cosh, tanh, coth	Гиперболические синус, косинус, тангенс, котангенс		
sech, csch	Гиперболические секанс, косеканс		
asinh, acosh, atanh, acoth	Гиперболические арксинус, арккосинус, арктангенс, арккотангенс		
asech, acsch	Гиперболические арксеканс, арккосеканс		
Функции для работы с комплексными числами		Округление и остаток от деления	
abs, angle	Модуль **r** и фаза **φ** комплексного числа (от $-\pi$ до π) $a+ib=r\left(\cos\varphi+i\sin\varphi\right)$	fix, floor, ceil	Округление до ближайшего целого по направлению к нулю, минус бесконечности и плюс бесконечности
complex	Конструирует комплексное число по его действительной и мнимой части	round	Округление до ближайшего целого
conj	Возвращает комплексно-сопряженное число	mod, rem	Остаток целочисленного деления со знаком и без знака
imag, real	Возвращает мнимую и действительную	sign	Возвращает знак числа

	часть числа		

Подробный список встроенных элементарных функций MatLab и их использование можно узнать, набрав в командной строке help elfun.

2.2 Работа с массивами

1 Ввод элементов массивов. В MatLab наиболее часто используются массивы, представляющие вектор-столбцы, вектор-строки и матрицы. Ввод массивов осуществляется в квадратных скобках. Для ввода вектор-столбца каждый элемент отделяется точкой с запятой.

```
>> a = [1.3; 5.4; 6.9]

a =
   1.3000
   5.4000
   6.9000
```

Для ввода вектор-строки каждый элемент отделяется либо запятой, либо пробелом

```
>> b = [7.1 3.5 8.2]
b =
   7.1000    3.5000    8.2000
```

Существует несколько способов ввода двумерных массивов (матриц). Первый из них – рассматривать матрицу как вектор-столбец, элементами которого являются вектор-строки или наоборот, вектор-строку, элементами которой являются вектор-столбцы.

```
>> A = [3 1 -1; 2 4 3]

A =
    3    1   -1
    2    4    3
```

Другой вариант выглядит так. Наберите в командной строке

```
>> B = [4 3 -1
```

Нажмите клавишу <Enter>. Курсор мигает на следующей строке без символа >>. Продолжайте ввод матрицы построчно, нажимая в конце каждой строки <Enter>. Последнюю строку завершите закрывающей квадратной скобкой. В результате получается:

```
2 7 0
-5 1 2]

B =

    4    3   -1
    2    7    0
   -5    1    2
```

2 Обращение к элементам массивов. Доступ к элементу вектора осуществляется при помощи индекса, заключенного в круглые скобки после имени массива, в котором хранится вектор. Таким образом, обращаясь к элементу, можно присвоить ему новое значение или использовать элементы в выражениях.

```
>> a(2)

ans =
   5.4000
```

MatLab предоставляет удобный способ обращения к блокам последовательно расположенных элементов. Для этого служит индексация при помощи знака двоеточия. Предположим, надо заменить часть

элементов массива нулями, для этого просто и наглядно использовать индексацию при помощи двоеточия.

```
>> g = [0.1 2.9 3.3 5.1 2.6 7.1 9.8];
>> g(2:6) = 0;
>> g

g =
   0.1000     0     0     0     0     0    9.8000
```

Доступ к элементам матриц осуществляется при помощи двух индексов – номера строки и номера столбца, заключенного в круглые скобки, например

```
>> A(2, 3)

ans =
    3
```

3 Функции обработки данных и математические операции с массивами. На практике часто приходится обрабатывать данные массивов – перемножение и суммирование элементов, нахождение длины вектора или размера матрицы, сортировка, нахождение максимального и минимального элементов и т.д. Кроме того, полезным бывает использование матриц специального вида. В табл. 2 приведены основные команды, позволяющие производить обработку данных и создавать различные специальные матрицы.

Более подробно про обработку матричных данных можно узнать в справочной системе MatLab, набрав в командной строке help datafun, а затем посмотреть информацию о нужной функции, например help max.

2 Функции обработки данных и математические операции

Команды	Выполняемые функции
Применение функций обработки данных к массивам	
P = prod(a)	Перемножение элементов массива a и присвоение значения переменной P
L = length(a)	Вычисление наибольшего размера массива a и присвоение значения переменной L (для векторов – длина)
S = sum(a)	Суммирование элементов массива a и присвоение значения переменной S
M = max(a)	Нахождение максимального элемента по столбцам массива a и присвоение значения вектор-строке M
m = min(a)	Нахождение максимального элемента по столбцам массива a и присвоение значения вектор-строке m
R = sort(a)	Сортировка столбцов массива a по возрастанию и присвоение результата новому массиву R
[m, n] = size(a)	Определение количества строк и столбцов массива a.
Создание матриц специального вида	
A = zeros(m, n)	Создание и заполнение прямоугольной матрицы A нулями. Если в скобках стоит одно число n, то создается квадратная матрица A размером $n \times n$

B = eye(*m*, *n*)	Создание единичной матрицы B (единицы на главной диагонали). Если в скобках стоит одно число *n*, то создается квадратная матрица размером $n \times n$ с единицами на главной диагонали
C = ones(*m*, *n*)	Создание матрицы C, состоящей из единиц. Если в скобках стоит одно число *n*, то создается квадратная матрица размером $n \times n$
R = rand(*m*, *n*)	Создание матрицы R, заполненной случайными числами, распределенными по равномерному закону между нулем и единицей. Если в скобках стоит одно число *n*, то создается квадратная матрица размером $n \times n$
RN = randn(*m*, *n*)	Создание матрицы R, заполненной случайными числами, распределенными по нормальному закону. Если в скобках стоит одно число *n*, то создается квадратная матрица размером $n \times n$
D = diag(*a*, *k*)	Создание матрицы, у которой внедиагональные числа равны нулю. Если *k* не указано, то на главной диагонали расположится вектор *d*. Второй аргумент означает, насколько побочная диагональ отстоит от главной, а его знак указывает на направление, плюс – вверх, минус – вниз от главной диагонали.

Математическими операциями с массивами являются сложение, вычитание, умножение, транспонирование и возведение в степень. При использовании матричных операций следует помнить, что для сложения или вычитания матрицы должны быть одного размера, а при умножении число столбцов первой матрицы должно равняться числу строк второй матрицы. Сложение и вычитание матриц производится с помощью знаков плюс и минус, а при умножении используется звездочка.

```
>> A = [3 1 -1; 2 4 3];
>> C = [[3; 4] [-1; 2] [7; 0]];
>> S = A + C

S =
    6    0    6
    6    6    3

>> R = C-A

R =
    0   -2    8
    2   -2   -3
>> P = C*B
P =
  -25    9   11
   20   26   -4
```

Умножать матрицу на число можно как справа, так и слева.

```
>> P1 = A*3

P1 =
    9    3   -3
```

```
    6    12     9
```

Транспонирование массивов производится при помощи знака «.’», а символ «’» означает комплексное сопряжение (для массивов комплексных чисел).

```
>> B.'

ans =
    4     2    -5
    3     7     1
   -1     0     2
```

Возведение *квадратной* матрицы в целую степень производится по правилу перемножения матриц с использованием оператора «^». Если матрица не является квадратной, MatLab выдаст сообщение об ошибке.

```
>> B2= B^2

B2 =
   27    32    -6
   22    55    -2
  -28    -6     9
```

При расчете матричных выражений учитывается приоритет операций: сначала производится транспонирование, потом возведение в степень, затем умножение, а сложение и вычитание производятся в последнюю очередь.

4 Поэлементные операции с матрицами. Массивы могут использоваться в качестве аргументов математических функций, результатом которых станет новый массив, равный значениям функций от соответствующих элементов исходного вектора.

```
>> a = [1.3; 5.4; 6.9];
>> cos(a)

ans =
   0.2675
   0.6347
   0.8157
```

Таким образом происходит *поэлементное* вычисление вызываемой функции. Поэлементные математические операции над массивами производятся следующим образом.

Поэлементное умножение массива на массив производится при помощи знака «.*»:

```
>> b = [7.1 3.5 8.2];
>> a.*b'

ans =
   9.2300
  18.9000
  56.5800
```

Поэлементное возведение в степень (степенью может быть число или массив) – «.^»:

```
>> A.^C

ans =
   27     1    -1
   16    16     1
```

Поэлементное деление соответствующих элементов массива – «./»:

```
>> a./b'
```

```
ans =
   0.1831
   1.5429
   0.8415
```

При помощи такого же символа производится поэлементное деление числа на массив. Перемножать, возводить в степень и делить можно только массивы одинаковой размерности!

2.3 Считывание и запись данных

Иногда полезным бывает использование данных (вектора и матрицы), хранящихся в текстовых файлах. Для считывания из файла используется команда load, для записи – save. При работе с файлами данных используется следующий формат вызова этих команд.

```
>> save 'Gc.txt' Gtek -ascii;
```

Эта запись означает, что записывается величина Gtek в текстовом формате – ascii в файл 'Gc.txt'.

```
>> Gc = load('Gc.txt')
```

Эта запись означает, что данные, содержащиеся в текстовом файле 'Gc.txt' (число, вектор или матрица), считываются и присваиваются переменной Gc.

3 М-ФАЙЛЫ

Работа из командной строки MatLab затрудняется, если требуется вводить много команд и часто их изменять. Самым удобным способом выполнения команд MatLab является использование *М-файлов*, в которых можно набирать команды, выполнять их все сразу или частями, сохранять и использовать в дальнейшем. Для работы с М-файлами предназначен редактор М-файлов.

3.1 Работа в редакторе М-файлов

В меню **File** основного окна MatLab и в пункте **New** выберете подпункт **M-file**. Новый файл открывается в окне редактора М-файлов, которое для версий *5.2*, *5.3* и *6.x* имеет похожий вид. Набрав необходимый перечень команд, сохраните файл с желаемым именем с расширением *.m* в подкаталоге work основного каталога MatLab , выбрав пункт **Save as** меню **File** редактора.

Открытие существующего М-файла производится с помощью пункта **Open** меню **File** либо редактора М-файлов. В редакторе М-файлов может быть одновременно открыто несколько файлов. Переход между ними осуществляется при помощи закладок с именами файлов.

Отдельные блоки можно снабжать *комментариями*, которые пропускаются при выполнении. Комментарии начинаются со знака процента (%) и автоматически выделяются зеленым цветом.

3.2 Установка путей

Для корректного выполнения созданных М-файлов необходимо в MatLab установить к ним пути поиска. Для этого можно использовать навигатор путей, выбрав пункт **Set Path** в меню **File** (для версий *5.3* и выше).

Другой способ – это использование команды в командной строке. Текущий каталог устанавливается командой cd, например
cd c:\user\myfoulder. Для установки путей служит команда path, вызываемая с двумя аргументами:

- path(path, ‘c:\user\myfoulder’) – добавляет каталог c:\user\myfoulder с низшим приоритетом поиска;
- path(‘c:\user\myfoulder’, path) – добавляет каталог c:\user\myfoulder с высшим приоритетом поиска.

Использование команды path без аргументов приводит к отображению на экране списка путей поиска. Удалить путь из списка можно при помощи команды rmpath.

rmpath(‘c:\user\myfoulder’) удаляет путь к каталогу c:\user\myfoulder из списка путей.

3.3 Типы М-файлов

М-файлы в MatLab бывают двух типов: *файл-программы*, содержащие последовательность команд, и *файл-функции*, в которых описываются функции, определяемые пользователем. Если имеется проект, реализованный в нескольких М-файлах, необходимо сохранить все файлы в одной папке и при запуске – установить пути к этой папке.

Файл-программы не имеют входных и выходных переменных. Все переменные, объявленные в файл-программе, становятся доступны в рабочей среде сразу после ее выполнения. Выполнение команд, содержащихся в файл-программе, осуществляется из командной строки или другой файл-программы, при этом в качестве команды используется имя М-файла.

Пример: преобразование матрицы. Последовательность команд оформляется в виде файл-программы и сохраняется под именем preobr.m:

```
A = ones(400);
row = (1:400);
M = row'*row;
A = A./M;
```

Запуск файл-программ осуществляется вызовом ее имени в командной строке:

```
>> preobr
```

Файл-функции используются для применения численных методов, а также для написания собственных приложений MatLab. Они производят необходимые действия с входными аргументами и возвращают результат в выходных аргументах.

Слово function в первой строке определяет, что данный файл содержит файл-функцию. Первая строка является *заголовком функции*, в которой размещаются имя функции и списки входных и выходных аргументов. После заголовка идет *тело функции*, где вычисляется ее значение. Важно, что вычисленное значение присваивается выходной переменной. Для сохранения файла используется пункт **Save as** или **Save** меню **File**. В появившемся диалоговом окне поле **File name** уже содержит заданное название функции, которое не нужно менять. Таким образом, созданную функцию можно использовать так же, как и встроенные функции.

Файл-функция с несколькими входными аргументами. Все входные аргументы размещаются в круглых скобках, в списке входных переменных через запятую.

Пример: Функция расчета выражения $f(x, y) = \sin \pi x \sin \pi y$ является примером функции с несколькими входными переменными.

```
function f = prim1(x, y)
f = sin(pi*x).*sin(pi*y);
```

Сохраняем данную функцию под именем prim1. Для получения результата необходимо вызвать эту функцию, например из командной строки или в другом файле. Вызовем сохраненную функцию с входными аргументами $x = 0.5$, $y = 1.0$. Результат, рассчитанный в функции, присвоим переменной Rez.

```
>> Rez = prim1(0.5, 1.0)

Rez =

  1.2246e-016
```

Написание **файл-функции с одним входным аргументом** не отличается от случая с несколькими аргументами: в круглых скобках указывается входная переменная.

Если входные переменные являются массивами, то необходимо применять поэлементные операции с массивами!

Файл-функция с несколькими выходными аргументами. Удобно использовать при вычислении функций, возвращающих несколько значений. Выходные аргументы добавляют через запятую в список выходных аргументов, а сам список заключается в квадратные скобки.

П р и м е р: Перевести время, заданное в секундах в часы, минуты и секунды.

```
function [hour, minute, second] = hms(sec)

hour = floor(sec/3600);
minute = floor((sec-hour*3600)/60);
second = sec-hour*3600-minute*60;
```

При вызове файл-функции с несколькими выходными аргументами результат следует записывать в

вектор соответствующей длины:

```
>> [H, M, S] = hms(10000)
H =
     2
M =
    46
S =
    40
```

4 ОСНОВЫ ПРОГРАММИРОВАНИЯ В MATLAB

Для решения многих серьезных задач требуется писать программы, в которых действия повторяются циклически, в зависимости от некоторых условий выполняются различные части программы. Операторы цикла и ветвления можно использовать как в файл-программах, так и в файл-функциях, что позволяет создавать функции со сложной разветвленной структурой.

4.1 Операторы цикла

Выполнение схожих повторяющихся действий в MatLab осуществляется при помощи операторов циклов for и while. Цикл for предназначен для выполнения заданного числа повторяющихся действий, а while – для действий, число которых заранее неизвестно, но известно условие продолжения цикла. Самое простое использование цикла for осуществляется следующим образом:

```
for count = start : step : final
     команды MatLab
end
```

Здесь count – переменная цикла, start – ее начало, step – шаг, на который увеличивается count при каждом заходе в цикл. Цикл заканчивается, только когда count становится больше final. Переменная цикла может принимать не только целые, но и вещественные значения любого знака. Если шаг цикла равен 1, то его можно не указывать.

Если заранее неизвестно число повторяемых действий, циклом for не обойтись. Выход состоит в применении цикла while, который работает, пока выполняется *условие цикла*:

```
while условие цикла
                команды MatLab
end
```

Условие цикла while может содержать *операции отношения*, приведенные в табл. 3. Задание более сложных условий производится с применением *логических операторов.* Их возможная запись приведена в табл. 4

3 Операции отношения

Обозначение	Оператор

= =	Равенство
<	Меньше
<=	Меньше или равно
>=	Больше или равно
~=	Не равно

4 Логические операторы

Оператор	Условие	Запись в MatLab	**Эквивалентная запись**
Логическое «и»	$x < 3$ и $k = 4$	and (x <3, k==4)	(x<) & (k == 4)
Логическое «или»	$x = 1.2$	or (x==1, x==2)	(x==1) | (x==2)
Отрицание «не»	$a \neq 1.9$	not (a==1.9)	~(a == 1.9)

4.2 Операторы ветвления

Условный оператор if и оператор ветвления switch позволяют создать гибкий разветвляющийся алгоритм выполнения команд, в котором при выполнении определенных условий работает соответствующий блок операторов или команд MatLab. Практически во всех языках программирования имеются аналогичные операторы.

Применение оператора switch поясняет следующий пример, в котором осуществляется вывод на экран значения переменной *a*:

```
switch a
    case –1
        disp(‘a = -1’)
    case 0
        disp(‘a = 0’)
    case 1
        disp(‘a = 1’)
    case (2, 3, 4)
        disp(‘a равно 2 или 3 или 4’)
    otherwise
        disp(‘a не равно –1, 0, 1, 2, 3, 4’)
end
```

Каждая ветвь определяется оператором case, переход из нее выполняется тогда, когда переменная оператора switch принимает значение, указанное после case или одно из значений из списка case. После выполнения какой-либо из ветвей происходит выход из switch, при этом значения, заданные в других case, уже *не проверяются*. Если подходящих значений для *a* не нашлось, то выполняется ветвь программы, соответствующая otherwise.

Оператор if может применяться как в простом виде, для выполнения блока команд при удовлетворении некоторого условия, или в конструкции if-elseif-else. Применение этого оператора в самом простом случае выглядит так:

```
if условие
    команды MatLab
end
```

Если условие верно, то выполняются команды MatLab, размещенные между if и end, а если условие не верно, то происходит переход к командам, расположенным после end. При записи условия используются операции отношения (см. табл. 5).

Если необходимо организовать ветвление, то используется конструкция if-elseif-else.

Рассмотреть детально особенности работы данной конструкции в MatLab поможет следующий пример.

Пример: Кинетика процесса азосочетания зависит от уровня pH реакционной среды. Кинетические коэффициенты определяются по следующим аппроксимационным зависимостям:

$$\lg K_p = \begin{cases} -5{,}987 + 0{,}428\text{pH}, & \text{pH} < 7; \\ -19{,}86 + 4{,}428\text{pH} - 0{,}286(\text{pH})^2 - 4{,}75 * 10^{-4}(\text{pH})^3, & 7 \le \text{pH} \le 9{,}5; \\ 5{,}504 - \text{pH}, & \text{pH} > 9{,}5; \end{cases}$$

$$N_p = \begin{cases} 1, & \text{pH} < 7; \\ 0{,}4 - 3{,}32 * 10^{-3}(\text{pH})^2 + 2{,}21 * 10^{-3}(\text{pH})^3, & 7 \le \text{pH} \le 9{,}5; \\ 2, & \text{pH} > 9{,}5. \end{cases}$$

Для расчета данных коэффициентов в зависимости от текущего уровня pH используем следующую функцию.

```
function [k_p, n_p]=coeff(pH)
% рассчитываем кинетические коэффициенты k_p, n_p
% в зависимости от pH

if pH<7
   k_p=-6.04+0.428*pH;
   n_p=1;
elseif pH<=9.5
   k_p=-19.86+4.428*pH-0.286*pH^2-4.75e-4*pH^3;
   n_p=0.4-3.32e-3*pH^2+2.21e-3*pH^3;
else k_p=5.504-pH;
   n_p=2;
end
k_p=10^k_p;
```

Обратите внимание на следующие моменты:

- Число ветвей if-elseif-else равно трем.
- Во второй ветви достаточно проверить, что $\text{pH} \le 9{,}5$, а условие $\text{pH} < 7$ *уже выполнено,* иначе бы работала первая ветвь ifelseif-else и оператор if закончил работу.
- В последней ветви нет смысла проверять никакие условия, поскольку она работает, если все предыдущие неверны.

5 ГРАФИКА В MATLAB

MatLab обладает хорошо развитыми графическими возможностями для визуализации данных. Удобство заключается в том, что работу, связанную с масштабированием осей и подбором цветов MatLab берет на себя.

5.1 Построение графиков функций одной переменной

MatLab позволяет строить графики функций в линейном, логарифмическом и полулогарифмическом масштабах. Причем в одном и том же окне можно построить несколько функций. При построении нескольких графиков в разных окнах используется команда figure, которая служит для создания пустого графического окна и отображения его на экране. Окно становится *текущим*, т.е. все последующие графические функции будут осуществлять построение графиков в этом окне. Для получения нового графического окна следует снова использовать figure.

Построение графика функции одной переменной в линейном масштабе производится при помощи функции plot. В зависимости от входных аргументов эта функция позволяет строить один или несколько графиков, изменять цвет и стиль линий, добавлять маркеры на каждый график. Построение графика функции одной переменной (рис. 5.1) производится таким образом:

```
>> x = [0:0.05:1];
>> y = exp(-x).*sin(10*x);
>> plot(x, y)
```

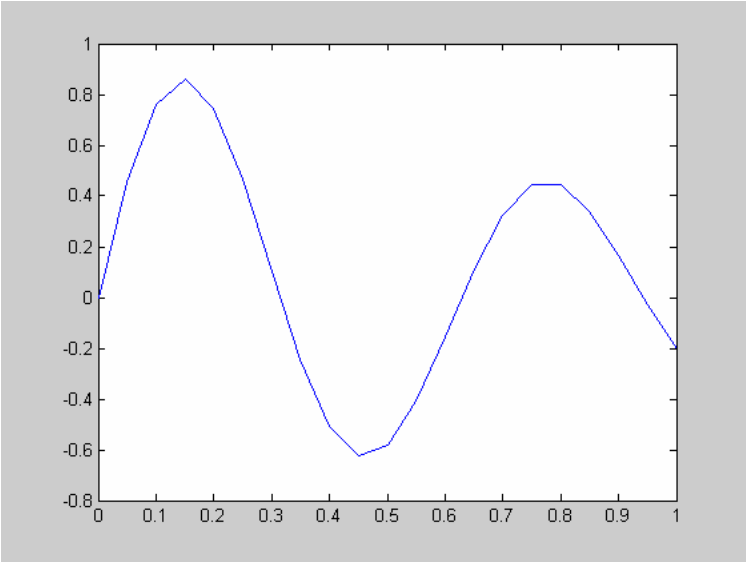


Рис. 5.1 График функции одной переменной

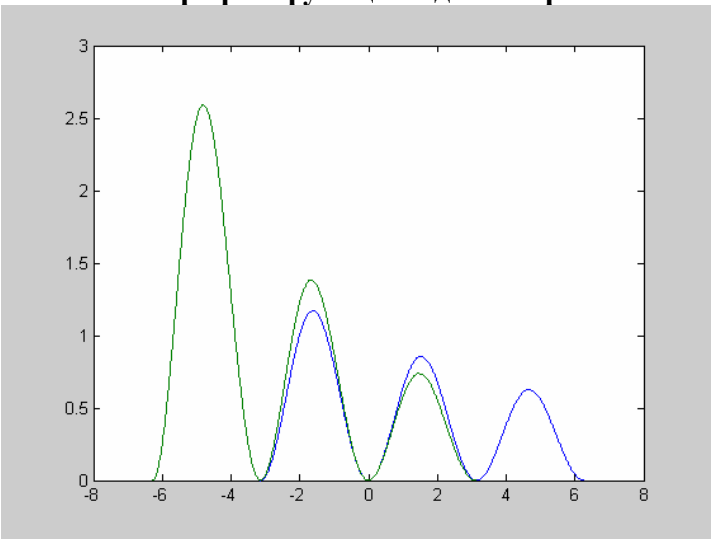


Рис. 5.2 Графики двух функций

Сравнение функций легко производить, построив их на одних координатных осях. Функции необязательно должны быть определены на одном и том же отрезке, главное, чтобы в каждой паре векторов абсцисс и ординат указать соответствующие вектора:

```
>> x1 = [-pi:0.01:2*pi];
>> f = exp(-0.1*x1).*sin(x1).^2;
>> x2 = [-2*pi:0.01:pi];
>> g = exp(-0.2*x2).*sin(x2).^2;
>> plot(x1, f, x2, g)
```

Таким образом, можно строить графики произвольного числа функций (рис. 5.2).

Для построения графиков функций в логарифмическом и полулогарифмическом масштабах служат следующие функции:

- loglog – логарифмический масштаб по обеим осям;
- semilogx – логарифмический масштаб только по оси абсцисс;
- semilogy – логарифмический масштаб только по оси ординат.

Использование этих функций аналогично функции plot.

Для построения графиков функций, записанных в файл-функцию, служит специальная команда fplot (для версий 6.x). При вызове fplot имя функции, график которой требуется построить, заключается в апострофы, а пределы построения указываются в вектор-строке из двух элементов. График, построенный при помощи fplot, более точно отражает поведение функции, так как fplot сама подбирает шаг аргумента, уменьшая его на участках быстрого изменения отображаемой функции.

Для более подробного описания возможностей функции plot наберите в командной строке help plot.

5.2 Оформление графиков

Построенные графики функций должны быть максимально удобными для восприятия. Часто требуется нанести маркеры, изменить цвет линий, а при подготовке к монохромной печати – задать тип линии. Для этого к функциям, приведенным выше, добавляется дополнительный аргумент, помещаемый за каждой парой векторов, который заключается в *апострофы* и состоит из трех символов: цвет, тип маркера и тип линии. Для построения графика функции $F(x)$ красными точечными маркерами без линии следует использовать команду

plot(x, F, ‘r.’)

Удобство использования графиков во многом зависит от дополнительных элементов оформления: координатной сетки, подписей к осям, заголовка и легенды.

Основные функции, применяемые при оформлении графиков, приведены в табл. 5.

5 Функции оформления графиков

Свойства линии					
Цвет		Тип маркера		Тип линии	
y	желтый	•	точка	-	сплошная
m	розовый	o	кружок	:	пунктирная
c	голубой	x	крестик	-•	штрих-пунктирная
r	красный	+	плюс	--	штриховая
g	зеленый	*	звездочка		
b	синий	s	квадрат		
k	черный	d	ромб		
w	белый	v	треугольник вершиной вниз		
		^	треугольник вершиной вверх		
		<	треугольник вершиной вправо		
		>	треугольник вершиной влево		
		p	пятиконечная звезда		
		h	шестиконечная звезда		

Продолжение табл. 5

Оформление графиков		
Функция	Назначение	Применение
grid on (grid off)	Нанесение сетки (отключение сетки)	

title	Заголовок графика	title (‘Текстовая строка’)
xlabel	Подпись к оси абсцисс	xlabel(‘x’)
ylabel	Подпись к оси ординат	ylabel(‘y’)
zlabel	Подпись к оси (для функций трех переменных)	zlabel(‘z’)
legend	Легенда	legend (‘График 1’, ‘График 2’)
hold on (hold off)	Объединение графиков разных функций в одном графическом окне (отключение данной функции – обязательно)	

ДОПОЛНИТЕЛЬНУЮ ИНФОРМАЦИЮ ОБ ОФОРМЛЕНИИ ГРАФИКОВ ФУНКЦИЙ МОЖНО ПОЛУЧИТЬ ИЗ СПРАВОЧНОЙ СИСТЕМЫ MATLAB.

5.3 Построение графиков функций двух переменных

MatLab предоставляет различные способы визуализации функций двух переменных – построение трехмерных графиков, поверхностей и линий уровня. Для отображения функции двух переменных следует:

1 Сгенерировать матрицы с координатами узлов сетки на прямоугольные области определения функций.

2 Вычислить функцию в узлах сетки и записать полученные значения в матрицу.

3 Использовать одну из графических функций MatLab.

4 Нанести на график дополнительную информацию.

Сетка генерируется при помощи команды meshgrid, вызываемой двумя аргументами. Аргументами являются векторы, элементы которых соответствуют сетке на прямоугольной области построения функции. Для построения *каркасной поверхности* используется функция mesh, с тремя входными аргументами (рис. 5.3):

```
>> [X, Y] = meshgrid(-1:0.05:1, 0:0.05:1);
>> Z = 4*sin(2*pi*X).*cos(1.5*pi*Y).*(1-X.^2).*Y.*(1-Y);
>> mesh(X, Y, Z)
```

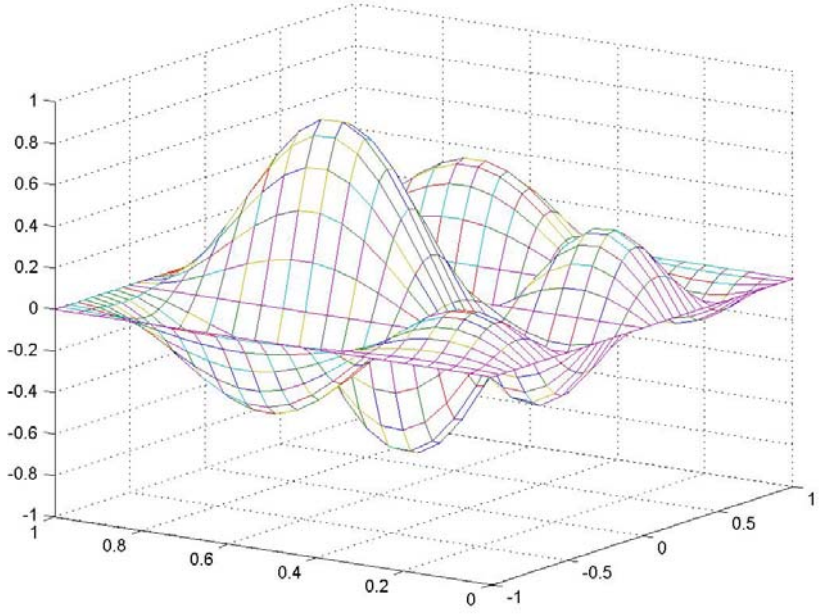


Рис. 5.3 Каркасная поверхность (mesh)

Цвет линий поверхности соответствует значениям функции. MatLab рисует только видимую часть поверхности. При помощи команды hidden off можно сделать каркасную поверхность прозрачной, добавив скрытую часть.

Основные функции для построения графиков функций двух переменных приведены в табл. 6.

6 Построения графиков функций двух переменных

Функция	Ее назначение
mesh (X, Y, Z)	Построение каркасной поверхности функции *Z*(*X*, *Y*)
hidden off (hidden on)	Добавить скрытую часть на каркасную поверхность (скрыть невидимую часть каркасной поверхности)
surf(X, Y, Z)	Каркасная поверхность функции *Z*(*X*, *Y*) с заливкой каждой клетки определенным цветом
shading interp (shading faseted)	Плавная заливка цветом поверхности, построенной с помощью функции surf (*X*, *Y*, *Z*) (отмена данной функции)

Продолжение табл. 6

Функция	Ее назначение
colorbar	Добавление к графику функции *Z*(*X*, *Y*) столбика с цветовой гаммой, соответствующей значениям функции
meshc(X, Y, Z)	Построение каркасной поверхности функции *Z*(*X*, *Y*) с нанесением линий равного уровня на плоскость *xy*
surfc(X, Y, Z)	Построение каркасной поверхности с заливкой цветом функции *Z*(*X*, *Y*) с нанесением линий равного уровня на плоскость *xy*
contour3(X, Y, Z, levels)	Построение поверхности функции *Z*(*X*, *Y*) из линий равного уровня, levels – число линий равного уровня либо вектор со значениями функции на линиях равного уровня
contour(X, Y, Z)	Построение линий равного уровня функции *Z*(*X*, *Y*)
contourf(X, Y, Z)	Построение линий равного уровня функции *Z*(*X*, *Y*) с заливкой цвета промежутков между линиями равного уровня
clabel(Cmatr, h)	Цифровые значения для линий равного уровня. Входные аргументы этой функции должны быть

	предварительно вызваны как выходные аргументы функций contour (*X*, *Y*, *Z*) или contourf (*X*, *Y*, *Z*).

ДОПОЛНИТЕЛЬНУЮ ИНФОРМАЦИЮ О ПОСТРОЕНИИ ГРАФИКОВ ФУНКЦИЙ ДВУХ ПЕРЕМЕННЫХ МОЖНО ПОЛУЧИТЬ В СПРАВОЧНОЙ СИСТЕМА MATLAB.

6 ЧИСЛЕННЫЕ МЕТОДЫ В MATLAB

MatLab обладает большим набором встроенных функций, реализующих различные численные методы. Нахождение корней уравнения, интегрирование, интерполирование, решение обыкновенных дифференциальных уравнений – вот далеко не полный перечень возможностей, предоставляемых MatLab.

6.1 Решение дифференциальных уравнений

MatLab предоставляет возможности для решения обыкновенных дифференциальных уравнений произвольного порядка и систем с начальными условиями, т.е. *задачи Коши*. Схема решения задачи такого вида в MatLab состоит из следующих этапов:

1 Приведение дифференциального уравнения к системе дифференциальных уравнений первого порядка. Для этого вводится столько дополнительных функций, каков порядок уравнения.

2 Написание специальной файл-функции для системы уравнений. Файл-функция содержит два входных аргумента: переменную *t*, по которой производится дифференцирование, даже если она входит в уравнение неявно, и вектор, размер которого равен числу неизвестных функций системы.

3 Вызов подходящего солвера (встроенной функции). Входными аргументами солвера, в простом случае, являются имя файл-функции в апострофах, вектор с начальным и конечным значениями переменной *t* и вектор начальных условий.

4 Визуализация результата.

В табл. 7 представлены основные функции (солверы) для решения систем обыкновенных дифференциальных уравнений, применимые во всех версиях MatLab.

7 Солверы для решения систем обыкновенных дифференциальных уравнений

Функция	Применение	Примечание
ode45(‘fun’, [$t_{нач}$ $t_{кон}$] x0)	Для нежестких систем	Метод Рунге-Кутта 4-го порядка
ode23(‘fun’, [$t_{нач}$ $t_{кон}$] x0)	Для нежестких систем и систем с небольшой жесткостью	Метод Рунге-Кутта более низкого порядка
ode23s(‘fun’, [$t_{нач}$ $t_{кон}$] x0)	Для жестких систем с невысокой точностью решения	Одношаговый метод Розенброка 2-го порядка
ode113(‘fun’, [$t_{нач}$ $t_{кон}$] x0)	Для нежестких систем, правые части которых вычисляются по сложным формулам	Метод Адамса-Бэшфорта-Милтона переменного порядка
ode15s(‘fun’, [$t_{нач}$	Для жестких сис-	Метод Гира

$t_{кон}$] x0)	тем	
ode23t('fun', [$t_{нач}$ $t_{кон}$] x0)	Для умеренно жестких систем дифференциальных и дифференциально-алгебраических уравнений	
ode23tb('fun', [$t_{нач}$ $t_{кон}$] x0)	Для жестких систем	

Дополнительную информацию об использовании встроенных методов решения систем дифференциальных уравнений можно узнать из справочной системы MatLab.

Пример: В трубчатом реакторе в изотермических стационарных условиях происходит реакция по двухстадийному кинетическому механизму: $A + 2B \xrightarrow{k_1} D,\ 2A + D \xrightarrow{k_2} E$. Известны концентрации исходных веществ и константы скорости отдельных стадий реакции при заданной температуре процесса: $A_0 = 2, B_0 = 1{,}6$ и $k_1 = 0{,}3,\ k_2 = 0{,}9$. Математическая модель кинетики химической реакции может быть представлена системой дифференциальных уравнений:

$$\begin{cases} \frac{dA}{dt} = -k_1 AB^2 - 2k_2 A^2 D; \\ \frac{dB}{dt} = -2k_1 AB^2; \\ \frac{dD}{dt} = k_1 AB^2 - k_2 A^2 D; \\ \frac{dE}{dt} = k_2 A^2 D, \end{cases},$$

где A, B, D, E – концентрации соответствующих веществ.

Решение данной системы с соответствующими начальными условиями с графическим представлением результатов, а также результаты проверки материального баланса в точке расчета с номером 10 представлены ниже.

Функция, содержащая уравнения модели, сохранена под именем plugreact.

```
function F = plugreact(l, y)
% функция расчета уравнений модели
k1 = 0.3;
k2 = 0.9;
% вектор-столбец, содержащий уравнения модели
F = [-k1*y(1)*y(2)^2-2*k2*y(2)^2*y(3); -2*k1*y(1)*y(2)^2;...
k1*y(1)*y(2)^2-k2*y(1)^2*y(3); k2*y(1)^2*y(3)];
```

Решение системы дифференциальных уравнений осуществляется в файл-программе solvdem методом Рунге-Кутта. В программе происходит вызов соответствующего солвера от файл-функции plugreact, а также визуализация полученных результатов.

```
% файл-программа solvdem
% расчет реактора идеального вытеснения
% формирование вектора начальных условий
Y0 = [2 1.6 0 0];
```

```
% вызов солвера от файл-функции plugreact,
% начальной и конечной точке реактора
% и вектора начальных условий
[L, Y] = ode45('plugreact', [0 10], Y0);
% вывод графика решен. исходных дифф. уравнений
plot (L, Y(:, 1), 'ko')
hold on
plot (L, Y(:, 2), 'k>')
plot (L, Y(:, 3), 'k')
plot (L, Y(:, 4), 'k--')
hold off
% оформление графика
title('Изменение концентраций веществ по длине реактора') xlabel('Длина')
ylabel('Концентрации')
legend('A', 'B', 'D', 'E')
grid on
% проверка материального баланса
Ca = Y0(1)-Y0(2)/2+Y(10, 2)/2-Y(10, 4)*2
Y10 = Y(10, 1)
```

Результаты при вызове программы solvdem:

```
>> solvdem

Ca =
   1.9975
Y10 =
   1.9975
```

Визуализация решения приведена на рис. 6.1.

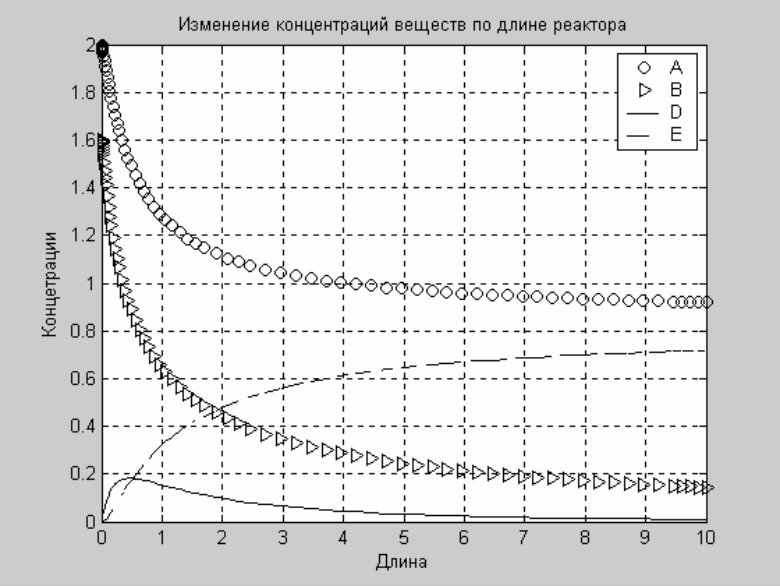


Рис. 6.1 Решение системы дифференциальных уравнений

6.2 Решение уравнений

Корни произвольного уравнения по заданному приближению позволяет найти встроенная функция fzero, которая вызывается следующим образом:

>> x = fzero(‘myfunction’, x0)

где ‘myfunction’ – имя файл-функции, вычисляющей левую часть уравнения; x0 – начальное приближение к корню либо вектор, содержащий концы отрезка, на котором ищутся корни; x – найденное приближенное значение корня. Данная функция находит решение с точностью до шестнадцатого знака после запятой.

Пример: Известна концентрация недиссоциированной кислоты $[HA]_0$ – 0,1 моль$^{-1}$; константа диссоциации K_a – 1,85 · 10^{-5}; ионное произведение воды K_w – 1 · 10^{-14}. Требуется определить кислотность раствора.

Зависимость pH от перечисленных функций представлена следующим уравнением: $[HA]_0 = \frac{[H^+] - K_w}{K_a} + [H^+] - \frac{K_w}{[H^+]}$, где $[H^+]$ – концентрация ионов водорода. Значение pH – это отрицательный десятичный логарифм концентрации ионов водорода: $pH = -\log(H^+)$.

Функция, содержащая выражение для расчета pH, сохранена под именем concentr, а входным аргументом ее является значение концентрации ионов водорода.

```
function f = concentr(x)
% определение pH растворов слабых кислот

kw = 1.85e-5;
ka = 1.0e-14;
ha = 0.1;
f = (x^2-kw)/ka+x-(kw/x)-ha;
```

В программе aim производится построение графика функции concentr для определения начального приближения, а затем решение нелинейного уравнения и определение pH.

```
% программа aim определения pH раствора
% построение графика функции для определения
% начального приближения
fplot ('concentr', [0 0.02])
grid on
% вызов fzero от файл-функции concentr, и точки приближения
% с определением значения функции в точке полученного решения
[H, f] = fzero('concentr', 0.004)
% расчет pH
pH = -log10(H)
```

Результаты поиска начального приближения приведены на рис. 6.2.

Результаты реализации программы aim:

```
>> aim

H =
   0.0043

f =
 -3.0669e-007
pH =
   2.3664
```

По графику легко определяется значение аргумента, близкого к нулю, которое является начальным приближением. Значение функции от полученного решения достаточно близко к нулю.

Решение систем линейных уравнений осуществляется при помощи символа «\», при этом система линейных уравнений представляется в матричном виде.

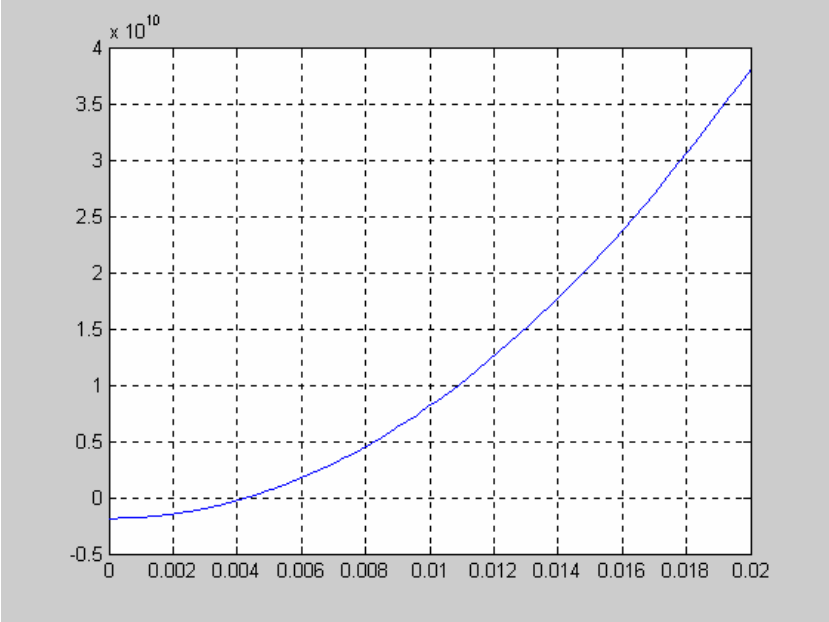


Рис. 6.2 Определение начального приближение для поиска корней уравнения

Пример: Требуется согласовать производство трех групп химических заводов, как по линии взаимных связей, так и по линии выполнения заданной им программы на производство конечной продукции. Исходные данные: программа на конечную продукцию каждой группы заводов и прогрессивные величины норм расходов этих продуктов как сырья для взаимного и собственного воспроизводства – представлены в табл. 8.

Математически согласованное производство по данным выпусков данных групп заводов с учетом прогрессивных норм расхода записывается в виде прогрессивных норм расхода или в матричной форме:

$$\begin{cases} (1-a_{1,1})x_1 - a_{1,2}x_2 - a_{1,3}x_3 = y_1; \\ x_2 - a_{2,1}x_1 - a_{2,2}x_2 - a_{2,3}x_3 = y_2; \\ x_3 - a_{3,1}x_1 - a_{3,2}x_2 - a_{3,3}x_3 = y_3; \end{cases}$$

$$\begin{vmatrix} 1-a_{1,1} & -a_{1,2} & -a_{1,3} \\ -a_{2,1} & 1-a_{2,2} & -a_{2,3} \\ -a_{1,3} & -a_{3,2} & 1-a_{3,3} \end{vmatrix} \cdot \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix} = \begin{vmatrix} y_1 \\ y_2 \\ y_3 \end{vmatrix}.$$

8 Исходные данные

	Норма расхода, т/т			Программа на продукцию
	Нефтехимические продукты	Химические продукты	Изделия из пластмассы	
Нефтехимические заводы, x_1	0,08	0,04	0,01	50 000
Заводы химической	0,07	0,06	0,02	30 000

промышленности, x_2				
Заводы переработки пластмасс, x_3	0,09	0,08	0,01	80 000

После подстановки исходных данных система имеет вид

$$\begin{vmatrix} 0,92 & -0,04 & -0,01 \\ -0,07 & 0,94 & -0,02 \\ -0,09 & -0,08 & 0,99 \end{vmatrix} \cdot \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix} = \begin{vmatrix} 50\ 000 \\ 30\ 000 \\ 80\ 000 \end{vmatrix}.$$

Решим данную задачу двумя возможными способами – с использованием матричных операций и оператора «\», и произведем проверку полученного решения. Расчет материального баланса приведен в листинге программы linesist.

```
% задание матрицы коэффициентов и столбца свободных членов линейной системы
A = [0.92 -0.04 -0.01; -0.07 0.94 -0.02; -0.09 -0.08 0.99];
B = [50000; 30000; 80000];
% решение системы с использованием матричных операций
X1 = A^-1*B
% решение системы с использованием оператора \
X2 = A\B
% проверка правильности результатов решений
REZ1 = A*X1  % решение, полученное с помощью матричных операций
REZ2 = A*X2   % решение, полученное с помощью оператора \
```

Результаты реализации программы linesist:

```
>> linesist

X1 =
  1.0e+004 *
    5.6970
    3.8052
    8.9062

X2 =
  1.0e+004 *
    5.6970
    3.8052
    8.9062

REZ1 =
  1.0e+004 *
    5.0000
    3.0000
    8.0000
```

```
REZ2 =
  1.0e+004 *
    5.0000
    3.0000
    8.0000
```

6.3 Вычисление корней полинома и производных от полинома

Полином задается вектором его коэффициентов, число элементов вектора, т.е. число коэффициентов полинома всегда на единицу больше его степени, нулевые коэффициенты должны содержаться в векторе.

Вычисление значения полинома от некоторого аргумента:

```
>> p = [1 0 3.2 -5.2 0 0.5 1 -3];
>> polyval(p, 3)

ans =

  2.5479e+003
```

Нахождение сразу всех корней полинома осуществляется при помощи функции roots, которая возвращает вектор корней полинома, в том числе и комплексные.

```
>> r = roots(p)

r =

  -0.5668 + 2.0698i
  -0.5668 – 2.0698i
  -0.6305 + 0.5534i
  -0.6305 – 0.5534i
   1.2149
   0.5898 + 0.6435i
   0.5898 – 0.6435i
```

Встроенная функция polyder предназначена для вычисления производных от полиномов. Вызов polyder с аргументом – вектором, соответствующим полиному, приводит к вычислению вектора коэффициентов производной полинома:

```
>> p = [1 0 1 0 0 1];
>> p1 = polyder(p)
p1 =
    5     0     3     0     0
```

6.4 Интегрирование функций

Для вычисления интеграла используется quad, задавая первым аргументом имя файл-функции, от которой вычисляется интеграл, а вторым и третьим – нижний и верхний предел интегрирования. Данная функция вычисляет приближенное значение интеграла с точностью 10e-3.

```
>> I = quad('myfunction', a, b)
```

Пример: Вычислить летучесть аммиака f как функцию давления при заданной температуре. Наиболее точные значения коэффициента летучести получают из экспериментальных значений коэффициента сжимаемости, определяемого из *P-V-T* измерений. Зависимость сжимаемости аммиака от давления определена экспериментально (табл. 9).

Для вычисления летучести как функции давления при заданной температуре используется зависимость:

$$\ln\frac{f}{P}=\int_0^P\frac{Z-1}{P}dP,$$

где Z – функция сжимаемости газа; P – давление; f – летучесть аммиака.

Вычислить летучесть и объем газа при заданных условиях.

9 Зависимость сжимаемости аммиака от давления

№ опыта	Давление	Функция сжимаемости газа Z	№ опыта	Давление	Функция сжимаемости газа Z
1	1	0,998	9	100	0,801
2	10	0,981	10	200	0,551
3	20	0,961	11	300	0,462
4	30	0,942	12	400	0,495
5	40	0,922	13	500	0,557
6	50	0,902	14	600	0,621
7	60	0,882	15	800	0,755
8	80	0,841			

В функции ammiak1 записывается подынтегральная функция

```
function f = ammiak1(p, z)
% расчет зависимости летучести аммиака
% z – независимый параметр (летучесть)
f = (z-1)./p;
```

Процедура ammiak рассчитывает летучесть и объем газа, результаты расчета выводятся на графики в одно окно.

```
% программа ammiak расчета летучести аммиака
% в зависимости от давления
% с использованием оператора for
R = 8.31514; % универсальная газовая постоянная
T = 473; % температура, К
% табличные данные
P = [0.001 1 10 20 30 40 50 60 80 100 200 300 ...
      400 500 600 800];
Z = [0 0.998 0.981 0.961 0.942 0.922 0.902 0.882...
      0.841 0.801 0.551 0.462 0.495 0.557 0.621 0.755];
% Расчет
for i=1:length(P)
   I = quad('ammiak1', 1.0e-10, P(i), [], [], Z(i));
   f(i) = P(i)*exp(I); % летучесть аммиака
   v(i) = (Z(i)*R*T)/P(i); % объем газа
end
% графический вывод результатов
subplot(3,1,1)
plot ( P, Z, 'k-')
```

```
xlabel('P')
ylabel('Z')
grid on
subplot(3,1,2)
plot ( P, v, 'k-')
xlabel('P')
ylabel('v')
grid on
subplot(3,1,3)
plot ( P, f, 'k-')
xlabel('P')
ylabel('f')
grid on
```

Результаты расчета на отдельных графиках представлены на рис. 6.3.

Реализованный алгоритм основан на квадратурной формуле Симпсона с автоматическим подбором шага интегрирования для достижения требуемой относительной погрешности. Интегрирование гладких функций лучше производить при помощи quad8, основанной на наиболее точных квадратурных формулах Ньютона-Котеса с автоматическим подбором шага.

При вычислении в MatLab 5.2, 5.3 интегралов от функций, имеющих интегрируемую особенность, например $1/\sqrt{x}$, в нуле выводится сообщение об ошибке. В версии 6.0 данный алгоритм улучшен для функций с особенностями, и, кроме того, появился quadl для нахождения определенного интеграла по квадратурным формулам Гаусса-Лобатто.

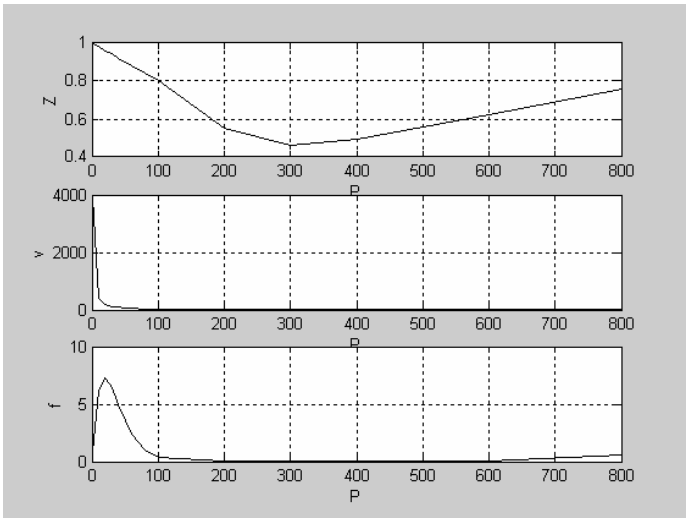


Рис. 6.3 Результаты реализации программы ammiak

Также в MatLab имеются функции для вычисления двойных интегралов: dblquad. Используется следующим образом:

```
>> dblquad('myfunction', a, b, c, d)
```

где a, b – нижний и верхний пределы внутреннего интеграла; c, d – нижний и верхний пределы наружного интеграла.

Также имеется возможность вычислять интегралы с переменным верхним пределом. Вычисление такого интеграла производится теми же функциями, что обычные определенные интегралы. Для нахождения такого интеграла пишутся две функции: для подынтегральной функции и находящую значения интеграла для каждого значения переменного верхнего предела (это число является входной переменной данной функции).

6.5 Интерполирование

Табличные данные очень часто удобно интерпретировать как некоторую функцию, в частности сплайн (полиномиальную) (рис. 6.4). Если возникает задача о построении полиномиальной или кусочно-полиномиальной функции для приближения некоторых исходных данных, то существуют в MatLab

встроенные функции для приближения сплайнами как одномерных, так и многомерных данных. Некоторые встроенные функции интерполирования в MatLab приведены в табл. 10.

10 Функции MatLab для интерполяции табличных данных

Функция	Назначение	Примечания
P = polyfit(x, y, a)	Приближение функции одной переменной по методу наименьших квадратов	x, y – вектора, содержащие таблично заданную функцию; a – число, равное степени приближающего полинома
P = interp1(x, y, xi, ‘text’) где text =	Приближение функции одной переменной сплайнами.	x, y – вектора, содержащие таблично заданную функцию; xi – вектор, содержащий промежуточные равноотстоящие точки; text – способ приближения функции
nearest	Интерполяция по соседним элементам.	
linear	Линейная интерполяция.	
Spline	Интерполяция кубическими сплайнами	

Продолжение табл. 10

Функция	Назначение	Примечания
P = interp2(X, Y, Z, Xi, Yi, ‘text’) где text =	Приближение функции двух переменной сплайнами.	X, Y, Z – вектора, содержащие таблично заданную функцию; Xi, Yi – вектора, содержащие промежуточные равноотстоящие точки; text – способ приближения функции
nearest	Интерполяция по соседним элементам.	
bilinear	Билинейная интерполяция.	
bicubic	Интерполяция бикубическими сплайнами	

Более подробно об интерполировании таблично заданных функций можно узнать из справочной системе MatLab.

Пример: Интерполяция таблично заданных функций.

```
% задание табличной функции
x = [0.1 0.2 0.4 0.5 0.6 0.8 1.2];
y = [-3.5 -4.8 -2.1 0.2 0.9 2.3 3.7];
% задание промежуточных точек для интерполирования
xi = [x(1):0.01:x(length(x))];
ynear = interp1(x, y, xi, 'nearest');
yline = interp1(x, y, xi, 'linear');
yspline = interp1(x, y, xi, 'spline');
```

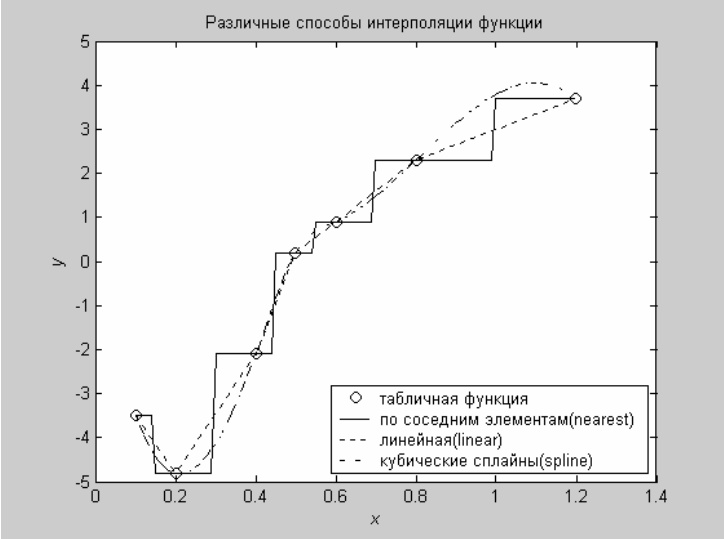


Рис. 6.4 Различные способы интерполяции функции одной переменной

6.6 Оптимизация

В состав MatLab входят функции, предназначенные для решения линейных и нелинейных, безусловных и условных оптимизационных задач. Основные функции оптимизации приведены в табл. 11.

11 Основные функции MatLab для решения задач оптимизации

Функция	Применение	Примечание
	Безусловная оптимизация	
fmin('F',x1,x2,… options,P1,P2)	Поиск локального минимума функции одной переменной. 'F' – имя файл-функции; x1,x2 – левая и правая границы интервала, на котором ищется минимум; options – параметры управления процессом решения; P1,P2 – независимые параметры	В поздних версиях перенесен в fminbnd
fminbnd('F', x1,x2,… options,P1,P2)	Поиск локального минимума функции одной переменной на заданном интервале. Применение аналогично fmin	Для версий 6.x
fmins('F',X0, … options,P1,P2)	Поиск локального минимума функции нескольких переменных. 'F' – имя файл-функции; X0 – вектор начальных приближений; options – параметры управления процессом решения; P1,P2 – независимые параметры	В поздних версиях перенесен в fminsearch
fminsearch('F',X0 ,… options,P1,P2)	Поиск локального минимума функции нескольких переменных. Применение аналогично fmins	Использует симплексные методы. Для версий 6.x
fmin-	Поиск минимума нелинейной	Использу-

unc('F',X0, … options,P1,P2)	функции нескольких переменных без ограничения на переменные. Применение аналогично fmins	ет квазиньютоновские методы. Для версий 6.x
	Условная оптимизация	
linprog(f,A,b,… Aeq,Beq,LB, … UB,X0,options)	Решение задач линейного программирования. f – вектор коэффициентов; A,b,Aeq,Beq – вектора коэффициентов нелинейных и линейных ограничений; LB,UB – вектора верхних и нижних границ изменения переменных; X0 – вектор начальных приближений; options – параметры управления процессом решения	Для версий 6.x

Продолжение табл. 11

Функция	Применение	Примечание
quadprog(H,f,A,b, … Aeq,Beq,LB, UB,… X0,options)	Решение задач квадратичного программирования. H– вектор соответствующих коэффициентов, остальные параметры – аналогично linprog	Для версий 6.x
fmincon('fun',X0, A,B,… Aeq,Beq,LB, UB,… 'nonlcon',options, … P1,P2)	Решение задач нелинейного программирования. 'fun' – имя файл-функции, содержащей целевую функцию, 'nonlcon' – имя файл-функции, содержащей нелинейные ограничения, остальные параметры – аналогично linprog	Для версий 6.x
fminimax('fun',X0 ,… A,B,Aeq,Beq ,LB,UB,… 'nonlcon',options, … P1,P2)	Решение минимаксной задачи. Применение аналогично fmincon	Для версий 6.x
fgoalattain('fun',X0, … g,w,A,B,Aeq, Beq,… LB,UB,'nonl con',… op-	Решение задачи о достижении границы. Решается задача вида $F(x) - w\gamma \le g$. Остальные параметры аналогичны fmincon.	Для версий 6.x

tions,P1,P2)		

1 Безусловная минимизация функций. Минимизация функции одной переменной производится с помощью команды fminbnd. Данная команда определяет локальный минимум в точке, близкой к точке приближения. Использование этой команды аналогично использованию команды fzero.

Пример: Найти локальный минимум уравнения $\sin x - x^2\cos(x) = 0$ на отрезке [–5, 0]. Исследуемая функция сохраняется в файл-функции 'myf', график исследуемой функции представлен на рис. 6.5.

```
>> x = fminbnd('myf', -5, 0)

x =

  -3.6052
```

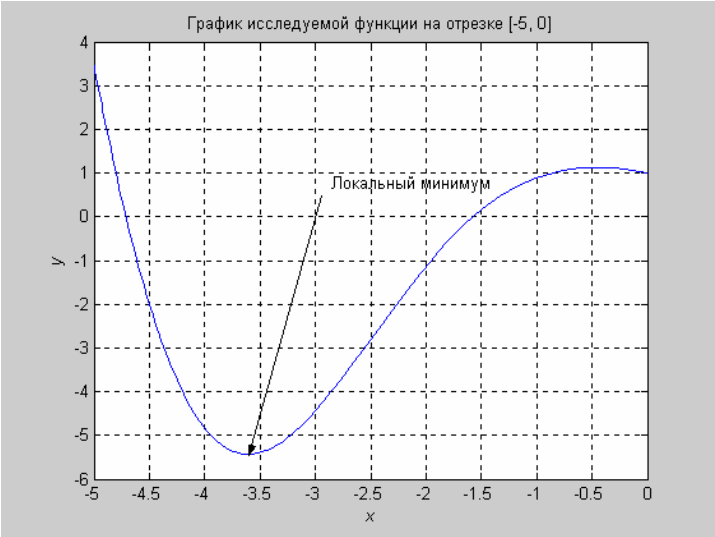


Рис. 6.5 Иллюстрация к примеру

Если необходимо найти безусловный минимум функции нескольких переменных, то используется команда fminsearch. перед применением fminsearch необходимо создать файл-функцию, вычисляющую значения искомой функции, причем аргументом данной функции должен быть вектор, элементы которого являются аргументами минимизируемой функции.

Пример. Определить локальный минимум функции $f(x,\ y) = \sin \pi x \sin \pi y$.

Сначала исследуем поведение данной функции для определения хорошего начального приближения. Для этого построим линии равного уровня исследуемой функции (рис. 6.6).

Файл-функция для оптимизации:

```
function f = ftest(arg)
x = arg(1);
y = arg(2);
f = sin(pi*x).*sin(pi*y);
```

Зная начальное приближение (определяется из графика на рис. 6.6), можно вызвать функцию fminsearch (указывая вектор начального приближения).

```
>> M = fminsearch('ftest', [1.4, 0.6])

M =
   1.5000    0.5000
```

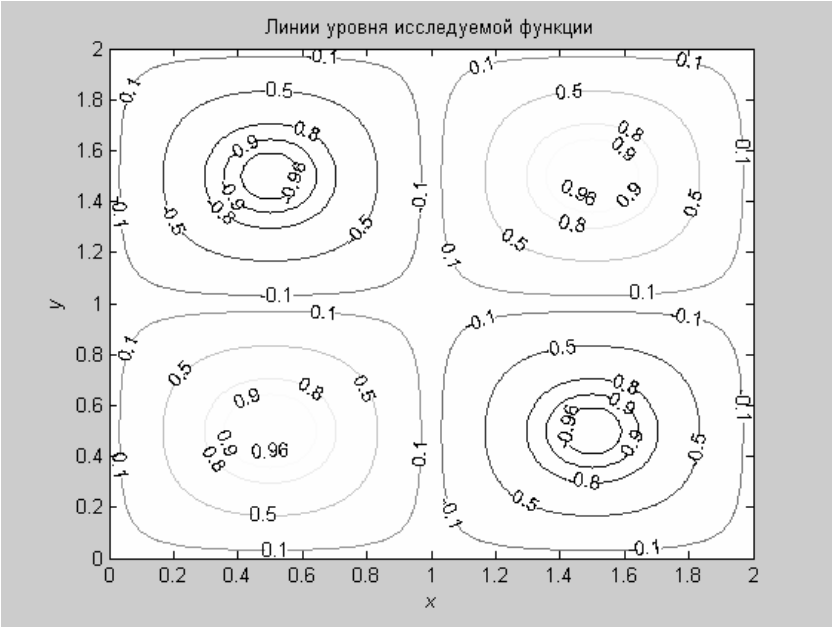


Рис. 6.6 Исследование поведения функции

2 Нелинейное программирование. Приложения MatLab позволяет решать ряд оптимизационных задач, в которых есть линейные и нелинейные ограничения. Общая постановка задачи нелинейного программирования такова: требуется разыскать $\min f(x)$ среди всех векторов x, удовлетворяющих системе неравенств и равенств

$$c(x) \le 0;\ c_{eq}(x) = 0;\ Ax \le b;\ A_{eq}x = b_{eq};\ lb \le x \le ub\,.$$

Решение поставленной задачи производится при помощи функции fmincon, обращение к которой в общем случае выглядит следующим образом:

x = fmincon(‘fun’, x0, A, b, Aeq, beq, lb, ub, ‘nonlcon’)

где x – приближенное решение; fun – имя файла, содержащего исследуемую функцию; x0 – вектор начальных приближений; A, b, Aeq, beq, lb, ub – вектора, соержащие линейные ограничения; nonlcon – имя файла, содержащего функцию с нелинейными ограничениями c, c_{eq}.

П р и м е р. Найти оптимальные конструктивные параметры эрлифтного аппарата, при которых капитальные затраты минимальны. Капитальные затраты эрлифтного аппарата могут быть определены по формуле

$$K = 248314 + 13378\cdot v + 2622\cdot d - 6727\cdot\varphi - 773\cdot d + 95\cdot v\cdot\varphi -$$
$$-1588\cdot d\cdot\varphi + 1226\cdot v^2 + 339\cdot d^2 - 1072\cdot\varphi^2$$

где v – объем аппарата, м^3; d – диаметр аппарата, м; φ – угол раскрытия конического днища, рад.

В соответствии со справочными данными для эрлифтных аппаратов конструктивные параметры могут находиться в диапазоне $v \in [4,\ 16]$, $d \in [0.3,\ 0.5]$, $\varphi \in [20,\ 50]$ град.

Задача не требует предварительной подготовки для задания функции цели. Начальными приближениями будут средние значения из диапазонов конструктивных параметров. Расчет целевой функции производится в файл-функции airlift.

```
function F = airlift(x)
% расчет целевой функции оптимизации
F = 248314+13378*x(1)+2622*x(2)-6727*x(3)-773*x(2)+...
+95*x(1)*x(3)-1588*x(2)*x(3)+1226*x(2)^2+339*x(2)^2-1072*x(3)^2;
```

Вызов функции fmincon для расчета минимальных капитальных затрат эрлифтного аппарата от функции airlift производится в программе boundopt:

```
% программа boundopt определения минимальных
```

```
% капитальных затрат на эрлифтный аппарат
% границы диапазонов конструктивных параметров
vmin = 4;
vmax = 16;
dmin = 0.3;
dmax = 0.5;
fimin = 20*pi/180;
fimax = 50*pi/180;
% формирование вектора начальных приближений
x0 = [(vmax+vmin)/2 (dmax+dmin)/2 (fimax+fimin)/2];
% вызов функции fmincon целевой функции airlift
% определение функции в точке решения
% и идентификатора успешности найденного решения
[x, f, flag] = fmincon('airlift', x0, [], [], [], [],...
[vmin;dmin;fimin], [vmax;dmax;fimax])
```

Результаты поиска минимума капитальных затрат:

```
>> boundopt
x =
   4.0000    0.3000    0.8727
f =
  2.9575e+005
flag =
    1
```

Значение параметра flag больше нуля свидетельствует об успешно найденном решении, значения вектора x – искомое решение, значение f – капитальные затраты при оптимальных значениях параметров.

Здесь приведен далеко не полный перечень встроенных функций, реализующих численные методы MatLab. Подробно ознакомиться с возможностями MatLab, интересующими функциями, а также примерами их применения можно в справочной системе MatLab, а также в специализированной литературе.

Все функции, реализующие численные методы, позволяют задать дополнительный параметр options, контролирующий вычислительный процесс. Значение options следует предварительно сформировать при помощи функции optimset в соответствии с характером требуемого контроля.

Пример. Задание точности 10^{-9} нахождения минимума функций одной переменной.

```
>> options = optimset(‘TolX’, 1.0e-9);
>> x = fminbnd('myf', -5, 0, options);
```

В общем случае аргументы optimset задаются попарно:
options = optimset(… вид контроля, значение, …)
Основные возможные сочетания параметров вид контроля и значение приведены в табл. 12.

12 Параметры optimset

Вид контроля	Значение	Результат
‘Display’	‘off’	Информация о вычислительном процессе не выводится
	‘iter’	Выводится информация о каждом шаге вычислительно-

		го процесса
	‘final’	Выводится информация только о завершении вычислительного процесса
‘MaxFunEvals’	Целое число	Максимальное количество вызовов исследуемой функции

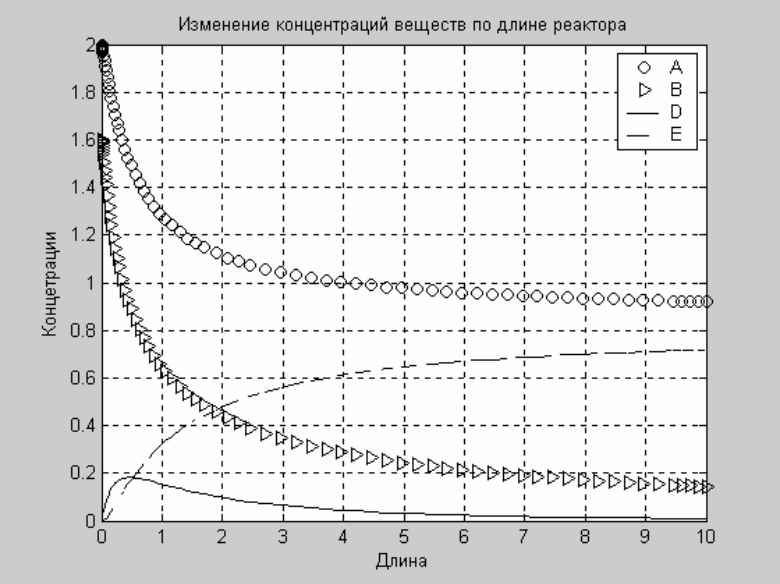


Рис. 6.1 Решение системы дифференциальных уравнений

6.2 Решение уравнений

Корни произвольного уравнения по заданному приближению позволяет найти встроенная функция fzero, которая вызывается следующим образом:

>> x = fzero(‘myfunction’, x0)

где ‘myfunction’ – имя файл-функции, вычисляющей левую часть уравнения; x0 – начальное приближение к корню либо вектор, содержащий концы отрезка, на котором ищутся корни; x – найденное приближенное значение корня. Данная функция находит решение с точностью до шестнадцатого знака после запятой.

Функция, содержащая выражение для расчета pH, сохранена под именем concentr, а входным аргументом ее является значение концентрации ионов водорода.

```
function f = concentr(x)
% определение pH растворов слабых кислот

kw = 1.85e-5;
ka = 1.0e-14;
ha = 0.1;
f = (x^2-kw)/ka+x-(kw/x)-ha;
```

В программе aim производится построение графика функции concentr для определения начального приближения, а затем решение нелинейного уравнения и определение pH.

```
% программа aim определения pH раствора
% построение графика функции для определения
% начального приближения
```

```
fplot ('concentr', [0 0.02])
grid on
% вызов fzero от файл-функции concentr, и точки приближения
% с определением значения функции в точке полученного решения
[H, f] = fzero('concentr', 0.004)
% расчет pH
pH = -log10(H)
```

Результаты поиска начального приближения приведены на рис. 6.2.

Результаты реализации программы aim:

```
>> aim

H =
   0.0043

f =
 -3.0669e-007
pH =
   2.3664
```

По графику легко определяется значение аргумента, близкого к нулю, которое является начальным приближением. Значение функции от полученного решения достаточно близко к нулю.

Решение систем линейных уравнений осуществляется при помощи символа «\», при этом система линейных уравнений представляется в матричном виде.

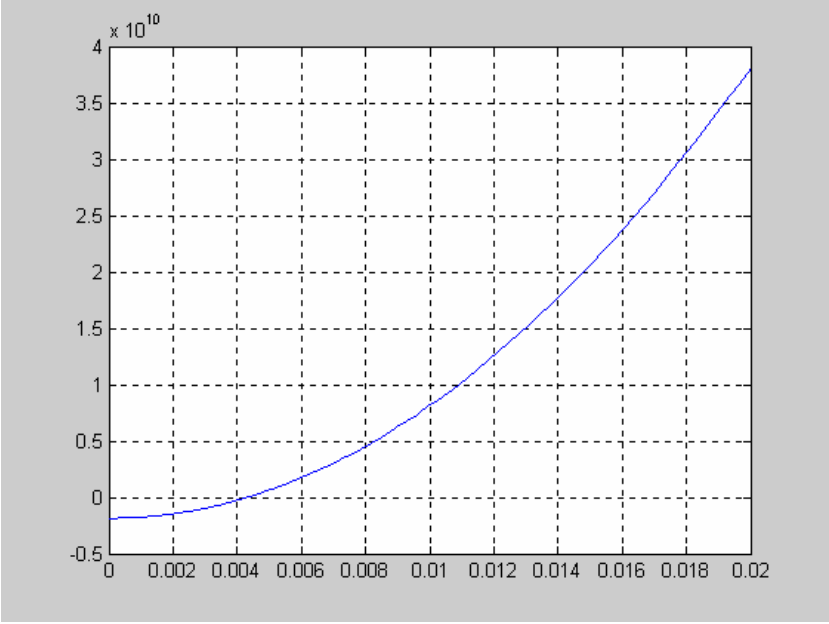


Рис. 6.2 Определение начального приближение для поиска корней уравнения

Пример: Требуется согласовать производство трех групп химических заводов как по линии взаимных связей, так и по линии выполнения заданной им программы на производство конечной продукции. Исходные данные: программа на конечную продукцию каждой группы заводов и прогрессивные величины норм расходов этих продуктов как сырья для взаимного и собственного воспроизводства – представлены в табл. 8.

Математически согласованное производство по данным выпусков данных групп заводов с учетом прогрессивных норм расхода записывается в виде прогрессивных норм расхода или в матричной форме:

$$\begin{cases} (1-a_{1,1})x_1 - a_{1,2}x_2 - a_{1,3}x_3 = y_1; \\ x_2 - a_{2,1}x_1 - a_{2,2}x_2 - a_{2,3}x_3 = y_2; \\ x_3 - a_{3,1}x_1 - a_{3,2}x_2 - a_{3,3}x_3 = y_3; \end{cases}$$

$$\begin{vmatrix} 1-a_{1,1} & -a_{1,2} & -a_{1,3} \\ -a_{2,1} & 1-a_{2,2} & -a_{2,3} \\ -a_{1,3} & -a_{3,2} & 1-a_{3,3} \end{vmatrix} \cdot \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix} = \begin{vmatrix} y_1 \\ y_2 \\ y_3 \end{vmatrix}.$$

8 Исходные данные

	Норма расхода, т/т			Программа на продукцию
	Нефтехимические продукты	Химические продукты	Изделия из пластмассы	
Нефтехимические заводы, x_1	0,08	0,04	0,01	50 000
Заводы химической промышленности, x_2	0,07	0,06	0,02	30 000
Заводы переработки пластмасс, x_3	0,09	0,08	0,01	80 000

После подстановки исходных данных система имеет вид

$$\begin{vmatrix} 0,92 & -0,04 & -0,01 \\ -0,07 & 0,94 & -0,02 \\ -0,09 & -0,08 & 0,99 \end{vmatrix} \cdot \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix} = \begin{vmatrix} 50\ 000 \\ 30\ 000 \\ 80\ 000 \end{vmatrix}.$$

Решим данную задачу двумя возможными способами – с использованием матричных операций и оператора «\», и произведем проверку полученного решения. Расчет материального баланса приведен в листинге программы linesist.

```
% задание матрицы коэффициентов и столбца свободных членов линейной системы
A = [0.92 -0.04 -0.01; -0.07 0.94 -0.02; -0.09 -0.08 0.99];
B = [50000; 30000; 80000];
% решение системы с использованием матричных операций
X1 = A^-1*B
% решение системы с использованием оператора \
X2 = A\B
% проверка правильности результатов решений
REZ1 = A*X1  % решение, полученное с помощью матричных операций
REZ2 = A*X2   % решение, полученное с помощью оператора \
```

Результаты реализации программы linesist:

```
>> linesist

X1 =
  1.0e+004 *
    5.6970
    3.8052
    8.9062

X2 =
  1.0e+004 *
    5.6970
    3.8052
    8.9062

REZ1 =
  1.0e+004 *
    5.0000
    3.0000
    8.0000

REZ2 =
  1.0e+004 *
    5.0000
    3.0000
    8.0000
```

6.3 Вычисление корней полинома и производных от полинома

Полином задается вектором его коэффициентов, число элементов вектора, т.е. число коэффициентов полинома всегда на единицу больше его степени, нулевые коэффициенты должны содержаться в векторе.

Вычисление значения полинома от некоторого аргумента:

```
>> p = [1 0 3.2 -5.2 0 0.5 1 -3];
>> polyval(p, 3)

ans =

  2.5479e+003
```

Нахождение сразу всех корней полинома осуществляется при помощи функции roots, которая возвращает вектор корней полинома, в том числе и комплексные.

```
>> r = roots(p)

r =

  -0.5668 + 2.0698i
  -0.5668 – 2.0698i
  -0.6305 + 0.5534i
  -0.6305 – 0.5534i
   1.2149
   0.5898 + 0.6435i
   0.5898 – 0.6435i
```

Встроенная функция polyder предназначена для вычисления производных от полиномов. Вызов polyder с аргументом – вектором, соответствующим полиному, приводит к вычислению вектора коэффициентов производной полинома:

```
>> p = [1 0 1 0 0 1];
>> p1 = polyder(p)
p1 =
    5     0     3     0     0
```

6.4 Интегрирование функций

Для вычисления интеграла используется quad, задавая первым аргументом имя файл-функции, от которой вычисляется интеграл, а вторым и третьим – нижний и верхний предел интегрирования. Данная функция вычисляет приближенное значение интеграла с точностью 10e-3.

```
>> I = quad('myfunction', a, b)
```

Пример: Вычислить летучесть аммиака f как функцию давления при заданной температуре. Наиболее точные значения коэффициента летучести получают из экспериментальных значений коэффициента сжимаемости, определяемого из *P-V-T* измерений. Зависимость сжимаемости аммиака от давления определена экспериментально (табл. 9).

Для вычисления летучести как функции давления при заданной температуре используется зависимость:

$$\ln\frac{f}{P} = \int_0^P \frac{Z-1}{P}\,dP,$$

где Z – функция сжимаемости газа; P – давление; f – летучесть аммиака.

Вычислить летучесть и объем газа при заданных условиях.

9 Зависимость сжимаемости аммиака от давления

№ опыта	Давление	Функция сжимаемости газа Z	№ опыта	Давление	Функция сжимаемости газа Z
1	1	0,998	9	100	0,801
2	10	0,981	10	200	0,551
3	20	0,961	11	300	0,462
4	30	0,942	12	400	0,495
5	40	0,922	13	500	0,557
6	50	0,902	14	600	0,621
7	60	0,882	15	800	0,755
8	80	0,841			

В функции ammiak1 записывается подынтегральная функция

```
function f = ammiak1(p, z)
% расчет зависимости летучести аммиака
% z – независимый параметр (летучесть)
f = (z-1)./p;
```

Процедура ammiak рассчитывает летучесть и объем газа, результаты расчета выводятся на графики в одно окно.

```
% программа ammiak расчета летучести аммиака
% в зависимости от давления
% с использованием оператора for
R = 8.31514; % универсальная газовая постоянная
T = 473; % температура, К
% табличные данные
P = [0.001 1 10 20 30 40 50 60 80 100 200 300 ...
      400 500 600 800];
Z = [0 0.998 0.981 0.961 0.942 0.922 0.902 0.882...
      0.841 0.801 0.551 0.462 0.495 0.557 0.621 0.755];
% Расчет
for i=1:length(P)
   I = quad('ammiak1', 1.0e-10, P(i), [], [], Z(i));
   f(i) = P(i)*exp(I); % летучесть аммиака
   v(i) = (Z(i)*R*T)/P(i); % объем газа
end
% графический вывод результатов
subplot(3,1,1)
plot ( P, Z, 'k-')
xlabel('P')
ylabel('Z')
grid on
subplot(3,1,2)
plot ( P, v, 'k-')
xlabel('P')
ylabel('v')
grid on
subplot(3,1,3)
plot ( P, f, 'k-')
xlabel('P')
ylabel('f')
grid on
```

Результаты расчета на отдельных графиках представлены на рис. 6.3.

Реализованный алгоритм основан на квадратурной формуле Симпсона с автоматическим подбором шага интегрирования для достижения требуемой относительной погрешности. Интегрирование гладких функций лучше производить при помощи quad8, основанной на наиболее точных квадратурных формулах Ньютона-Котеса с автоматическим подбором шага.

При вычислении в MatLab 5.2, 5.3 интегралов от функций, имеющих интегрируемую особенность, например $1/\sqrt{x}$, в нуле выводится сообщение об ошибке. В версии 6.0 данный алгоритм улучшен для функций с особенностями, и, кроме того, появился quadl для нахождения определенного интеграла по квадратурным формулам Гаусса-Лобатто.

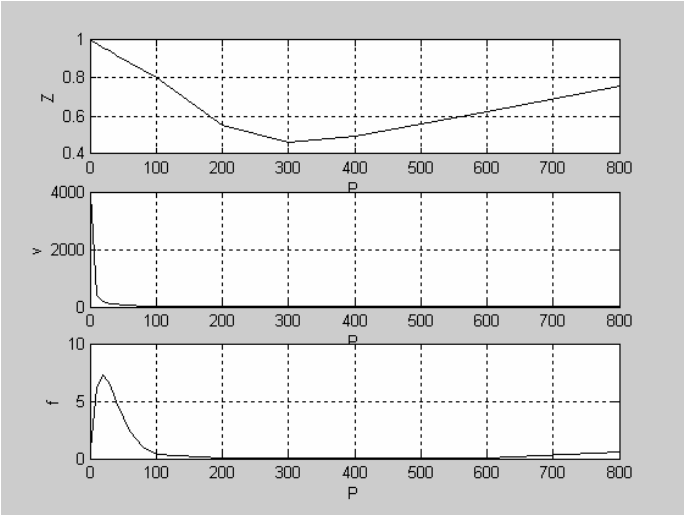


Рис. 6.3 Результаты реализации программы ammiak

Также в MatLab имеются функции для вычисления двойных интегралов: dblquad. Используется следующим образом:

>> dblquad('myfunction', a, b, c, d)

где a, b – нижний и верхний пределы внутреннего интеграла; c, d – нижний и верхний пределы наружного интеграла.

Также имеется возможность вычислять интегралы с переменным верхним пределом. Вычисление такого интеграла производится теми же функциями, что обычные определенные интегралы. Для нахождения такого интеграла пишутся две функции: для подынтегральной функции и находящую значения интеграла для каждого значения переменного верхнего предела (это число является входной переменной данной функции).

6.5 Интерполирование

Табличные данные очень часто удобно интерпретировать как некоторую функцию, в частности сплайн (полиномиальную) (рис. 6.4). Если возникает задача о построении полиномиальной или кусочно-полиномиальной функции для приближения некоторых исходных данных, то существуют в MatLab встроенные функции для приближения сплайнами как одномерных, так и многомерных данных. Некоторые встроенные функции интерполирования в MatLab приведены в табл. 10.

10 Функции MatLab для интерполяции табличных данных

Функция	Назначение	Примечания
P = polyfit(x, y, a)	Приближение функции одной переменной по методу наименьших квадратов	x, y – вектора, содержащие таблично заданную функцию; a – число, равное степени приближающего полинома
P = interp1(x, y, xi, 'text') где text = nearest	Приближение функции одной переменной сплайнами. Интерполяция по соседним элементам.	x, y – вектора, содержащие таблично заданную функцию; xi – вектор, содержащий промежуточные рав-

linear	Линейная интерполяция.	ноотстоящие точки; text – способ приближения функции
Spline	Интерполяция кубическими сплайнами	

Продолжение табл. 10

Функция	Назначение	Примечания
P = interp2(X, Y, Z, Xi, Yi, ‘text’) где text =	Приближение функции двух переменной сплайнами.	X, Y, Z – вектора, содержащие табличо заданную функцию; Xi, Yi – вектора, содержащие промежуточные равноотстоящие точки; text – способ приближения функции
nearest	Интерполяция по соседним элементам.	
bilinear	Билинейная интерполяция.	
bicubic	Интерполяция бикубическими сплайнами	

Более подробно об интерполировании таблично заданных функций можно узнать из справочной системе MatLab.

П р и м е р: Интерполяция таблично заданных функций.

```
% задание табличной функции
x = [0.1 0.2 0.4 0.5 0.6 0.8 1.2];
y = [-3.5 -4.8 -2.1 0.2 0.9 2.3 3.7];
% задание промежуточных точек для интерполирования
xi = [x(1):0.01:x(length(x))];
ynear = interp1(x, y, xi, 'nearest');
yline = interp1(x, y, xi, 'linear');
yspline = interp1(x, y, xi, 'spline');
```

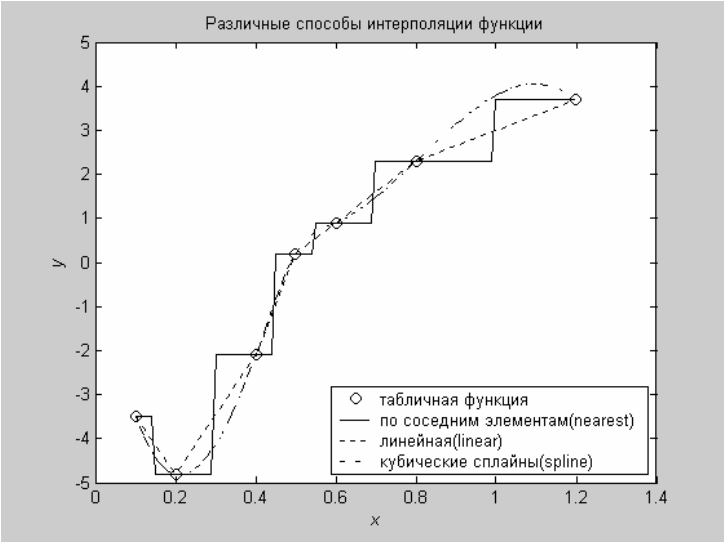


Рис. 6.4 Различные способы интерполяции функции одной переменной

6.6 Оптимизация

В состав MatLab входят функции, предназначенные для решения линейных и нелинейных, безусловных и условных оптимизационных задач. Основные функции оптимизации приведены в табл. 11.

11 Основные функции MatLab для решения задач оптимизации

Функция	Применение	Примечание
	Безусловная оптимизация	
fmin('F',x1,x2,… options,P1,P2)	Поиск локального минимума функции одной переменной. 'F' – имя файл-функции; x1,x2 – левая и правая границы интервала, на котором ищется минимум; options – параметры управления процессом решения; P1,P2 – независимые параметры	В поздних версиях перенесен в fminbnd
fminbnd('F',x1,x2,… options,P1,P2)	Поиск локального минимума функции одной переменной на заданном интервале. Применение аналогично fmin	Для версий 6.x
fmins('F',X0,… options,P1,P2)	Поиск локального минимума функции нескольких переменных. 'F' – имя файл-функции; X0 – вектор начальных приближений; options – параметры управления процессом решения; P1,P2 – независимые параметры	В поздних версиях перенесен в fminsearch
fminsearch('F',X0,… options,P1,P2)	Поиск локального минимума функции нескольких переменных. Применение аналогично fmins	Использует симплексные методы. Для версий 6.x
fminunc('F',X0,… options,P1,P2)	Поиск минимума нелинейной функции нескольких переменных без ограничения на переменные. Применение аналогично fmins	Использует квазиньютоновские методы. Для версий 6.x
	Условная оптимизация	
linprog(f,A,b,… Aeq,Beq,LB,… UB,X0,options)	Решение задач линейного программирования. f – вектор коэффициентов; A, b, Aeq, Beq – вектора коэффициентов нелинейных и линейных ограничений; LB, UB – вектора верхних и нижних границ изменения переменных; X0 – вектор начальных приближений; options – параметры управления процессом решения	Для версий 6.x

Продолжение табл. 11

Функция	Применение	Примечание
quadprog(H,f,A,b,… Aeq,Beq,LB, UB,… X0,options)	Решение задач квадратичного программирования. H – вектор соответствующих коэффициентов; остальные параметры – аналогично linprog	Для версий 6.x
Fmincon (‘fun’,X0,A, B,… Aeq,Beq,LB, UB,… ’nonlcon’,options,… P1,P2)	Решение задач нелинейного программирования. ‘fun’ – имя файл-функции, содержащей целевую функцию; ’nonlcon’ – имя файл-функции, содержащей нелинейные ограничения; остальные параметры – аналогично linprog	Для версий 6.x
fminimax(‘fun’,X0 ,… A,B,Aeq,Beq ,LB,UB,… ’nonlcon’,options,… P1,P2)	Решение минимаксной задачи. Применение аналогично fmincon	Для версий 6.x
fgoalattain(‘fun’,X0, … g,w,A,B,Aeq,Beq,… LB, UB,’nonlcon’ ,… options,P1,P2)	Решение задачи о достижении границы. Решается задача вида $F(x) - w\gamma \le g$. Остальные параметры аналогичны fmincon.	Для версий 6.x

1 Безусловная минимизация функций. Минимизация функции одной переменной производится с помощью команды fminbnd. Данная команда определяет локальный минимум в точке, близкой к точке приближения. Использование этой команды аналогично использованию команды fzero.

Пример: Найти локальный минимум уравнения $\sin x - x^2 \cos(x) = 0$ на отрезке [–5, 0]. Исследуемая функция сохраняется в файл-функции 'myf', график исследуемой функции представлен на рис. 6.5.

```
>> x = fminbnd('myf', -5, 0)

x =

  -3.6052
```

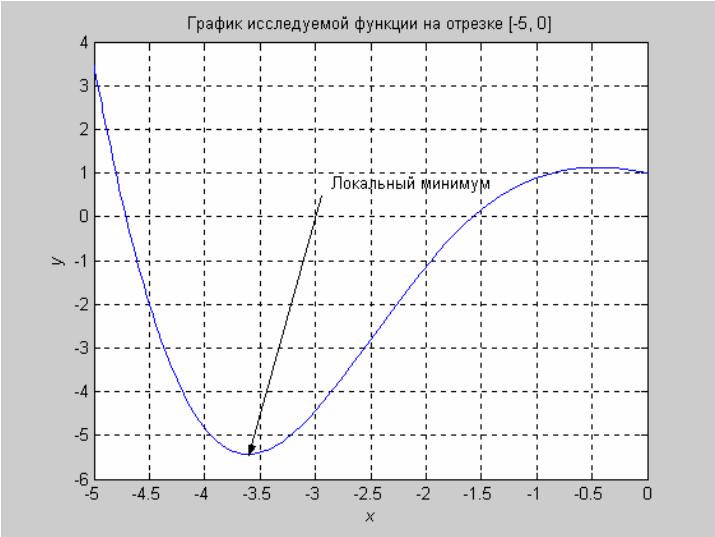


Рис. 6.5 Иллюстрация к примеру

Если необходимо найти безусловный минимум функции нескольких переменных, то используется команда fminsearch. Перед применением fminsearch необходимо создать файл-функцию, вычисляющую значения искомой функции, причем аргументом данной функции должен быть вектор, элементы которого являются аргументами минимизируемой функции.

Пример. Определить локальный минимум функции $f(x,\ y) = \sin \pi x \sin \pi y$.

Сначала исследуем поведение данной функции для определения хорошего начального приближения. Для этого построим линии равного уровня исследуемой функции (рис. 6.6).

Файл-функция для оптимизации:

```
function f = ftest(arg)
x = arg(1);
y = arg(2);
f = sin(pi*x).*sin(pi*y);
```

Зная начальное приближение (определяется из графика на рис. 6.6), можно вызвать функцию fminsearch (указывая вектор начального приближения).

```
>> M = fminsearch('ftest', [1.4, 0.6])

M =
   1.5000    0.5000
```

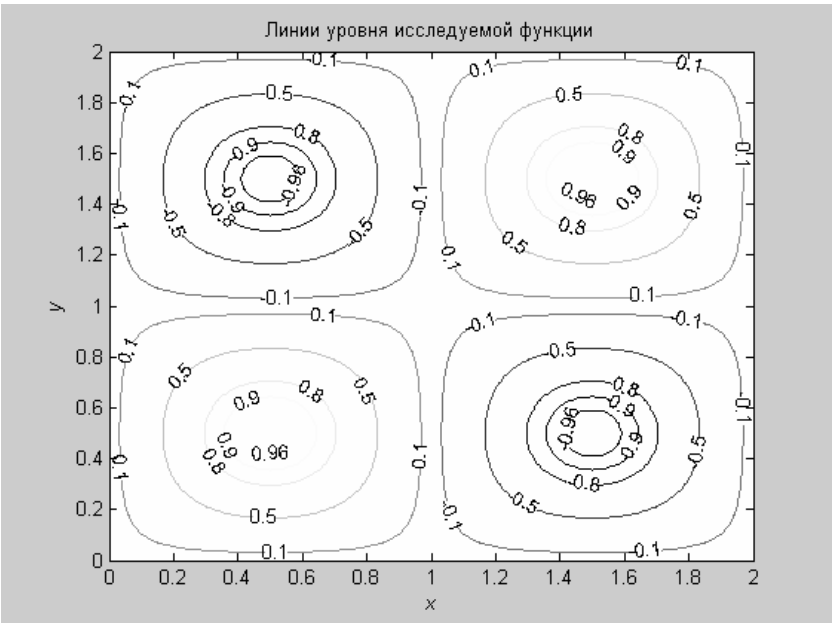


Рис. 6.6 Исследование поведения функции

2 Нелинейное программирование. Приложения MatLab позволяет решать ряд оптимизационных задач, в которых есть линейные и нелинейные ограничения. Общая постановка задачи нелинейного программирования такова: требуется разыскать $\min f(x)$ среди всех векторов x, удовлетворяющих системе неравенств и равенств

$$c(x) \le 0;\ c_{eq}(x) = 0;\ Ax \le b;\ A_{eq}x = b_{eq};\ lb \le x \le ub\,.$$

Решение поставленной задачи производится при помощи функции fmincon, обращение к которой в общем случае выглядит следующим образом:

x = fmincon('fun', x0, A, b, Aeq, beq, lb, ub, 'nonlcon')

где x – приближенное решение; fun – имя файла, содержащего исследуемую функцию; x0 – вектор начальных приближений; A, b, Aeq, beq, lb, ub – вектора, содержащие линейные ограничения; nonlcon – имя файла, содержащего функцию с нелинейными ограничениями c, c_{eq}.

Пример. Найти оптимальные конструктивные параметры эрлифтного аппарата, при которых капитальные затраты минимальны. Капитальные затраты эрлифтного аппарата могут быть определены по формуле

$$K = 248314 + 13378 \cdot v + 2622 \cdot d - 6727 \cdot \varphi - 773 \cdot d + 95 \cdot v \cdot \varphi -$$
$$-1588 \cdot d \cdot \varphi + 1;, \quad v^2 + 339 \cdot d^2 - 1072 \cdot \varphi^2$$

где v – объем аппарата, м^3; d – диаметр аппарата, м; φ – угол раскрытия конического днища, рад.

В соответствии со справочными данными для эрлифтных аппаратов конструктивные параметры могут находиться в диапазоне $v \in [4, 16],\ d \in [0.3, 0.5],\ \varphi \in [20,\ 50]$ град.

Задача не требует предварительной подготовки для задания функции цели. Начальными приближениями будут средние значения из диапазонов конструктивных параметров. Расчет целевой функции производится в файл-функции airlift.

```
function F = airlift(x)
% расчет целевой функции оптимизации
F = 248314+13378*x(1)+2622*x(2)-6727*x(3)-773*x(2)+...
+95*x(1)*x(3)-1588*x(2)*x(3)+1226*x(2)^2+339*x(2)^2-1072*x(3)^2;
```

Вызов функции fmincon для расчета минимальных капитальных затрат эрлифтного аппарата от функции airlift производится в программе boundopt:

```
% программа boundopt определения минимальных
% капитальных затрат на эрлифтный аппарат
% границы диапазонов конструктивных параметров
vmin = 4;
vmax = 16;
dmin = 0.3;
dmax = 0.5;
fimin = 20*pi/180;
fimax = 50*pi/180;
% формирование вектора начальных приближений
x0 = [(vmax+vmin)/2 (dmax+dmin)/2 (fimax+fimin)/2];
% вызов функции fmincon целевой функции airlift
% определение функции в точке решения
% и идентификатора успешности найденного решения
[x, f, flag] = fmincon('airlift', x0, [], [], [], [],...
[vmin;dmin;fimin], [vmax;dmax;fimax])
```

Результаты поиска минимума капитальных затрат:

```
>> boundopt
x =
   4.0000    0.3000    0.8727
f =
 2.9575e+005
flag =
    1
```

Значение параметра flag больше нуля свидетельствует об успешно найденном решении, значения вектора x – искомое решение, значение f – капитальные затраты при оптимальных значениях параметров.

Здесь приведен далеко не полный перечень встроенных функций, реализующих численные методы MatLab. Подробно ознакомиться с возможностями MatLab, интересующими функциями, а также примерами их применения можно в справочной системе MatLab, а также в специализированной литературе.

Все функции, реализующие численные методы, позволяют задать дополнительный параметр options, контролирующий вычислительный процесс. Значение options следует предварительно сформировать при помощи функции optimset в соответствии с характером требуемого контроля.

Пример. Задание точности 10^{-9} нахождения минимума функций одной переменной.

```
>> options = optimset('TolX', 1.0e-9);
>> x = fminbnd('myf', -5, 0, options);
```

В общем случае аргументы optimset задаются попарно:
options = optimset(… вид контроля, значение, …)
Основные возможные сочетания параметров вид контроля и значение приведены в табл. 12.

12 Параметры optimset

Вид контроля	Значение	Результат
'Display'	'off'	Информация о вычислительном процессе не выводится
	'iter'	Выводится информация о каждом шаге вычислительного процесса
	'final'	Выводится информация только о завершении вычислительного процесса
'MaxFunEvals'	Целое число	Максимальное количество вызовов исследуемой функции

Продолжение табл. 12

Вид контроля	Значение	Результат
'MaxIter'	Целое число	Максимальное количество итераций вычислительного процесса

‘TolFun’	Положительное вещественное число	Точность по функции для останова вычислений
‘TolX’	Положительное вещественное число	Точность по аргументу функции для останова вычислений

Подробнее об использовании параметров, контролирующих вычислительный процесс, можно узнать из справочной системы MatLab.

7 ПАКЕТ РАСШИРЕНИЯ MATLAB ПО НЕЙРОННЫМ СЕТЯМ NEURAL NETWORKS TOOLBOX

Пакет Neural Networks Toolbox (нейронные сети) содержит средства для проектирования, моделирования, обучения и использования множества известных парадигм аппарата *искусственных нейронных сетей* (ИНС): от базовых моделей *персептрона* (*нейрона*) до самых современных *ассоциативных* и *самоорганизующихся сетей*. Пакет может быть использован для решения множества разнообразных задач, таких как обработка сигналов, нелинейное управление, финансовое моделирование, аппроксимация, предсказание, прогноз и т.п.

Для каждого типа архитектуры и обучающего алгоритма ИНС в Toolbox имеются функции инициализации, обучения, адаптации, создания, моделирования, демонстрации (всего более 150 функций), а также примеры применения.

7.1 Структура и свойства искусственного нейрона

Нейрон – это основная составная часть нейронной сети (НС). На рис. 7.1 показана его структура.

Основные структурные функции нейрона следующие. Нейрон получает на входе вектор входных сигналов s_i, вычисляет сумму скалярных произведений на вектор весов w_i, прибавляет к ней вектор смещений b и вычисляет некоторую функцию одного переменного $f(s)$ (см. рис. 7.1). Таким образом, математическая модель нейрона описывается соотношениями вида

$$s = \sum_{i=1}^{n} w_i s_i + b \quad \text{и} \quad y = f(s).$$

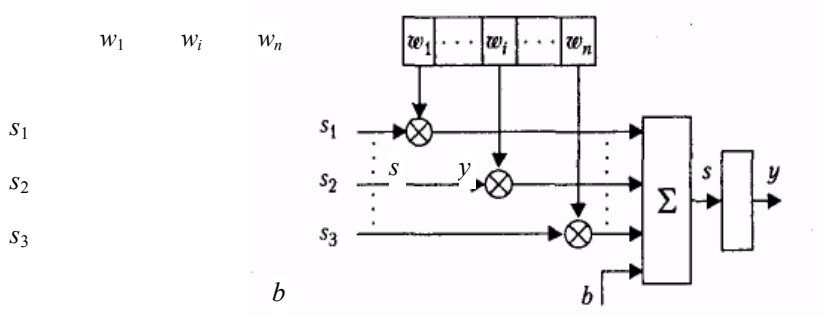


Рис. 7.1 Структура искусственного нейрона

Результат выполнения этих операций рассылается на входы других нейронов или передается на выход. Таким образом, нейронные сети вычисляют суперпозиции простых функций одного переменного и их линейных комбинаций.

В состав нейрона входят *умножители* (*синапсы*), *сумматор* и *нелинейный преобразователь*. *Синапсы* осуществляют связь между нейронами и умножают входной сигнал на число, характеризующее силу связи, – *вес синапса*. *Сумматор* выполняет сложение сигналов, поступающих по синаптическим связям от других нейронов и внешних входных сигналов. *Нелинейный преобразователь* реализует нелинейную функцию одного аргумента – выхода сумматора. Эта функция называется *функцией активации* или *передаточной функцией* нейрона.

В общем случае, нейрон полностью описывается своими весами w_i, смещением b и передаточной функцией $f(s)$. В качестве передаточных функций нейронов (нелинейного преобразователя нейронов) часто используют гладкие функции различного вида (табл. 13).

13 Перечень передаточных функций нейронов

Название	Формула	Область значений	Функция MatLab
Пороговая	$f(s)=\begin{cases}0, s<T\\1, s\ge T\end{cases}$	(0…1)	hardlim
Знаковая (сигнатурная)	$f(s)=\begin{cases}1, s>0\\-1, s\le 0\end{cases}$	(–1…1)	hardlims
Радиальная базисная (гауссова)	$f(s)=\exp(-s^2)$	(0…1)	radbas
Линейная	$f(s)=s$	$(-\infty\dots\infty)$	purelin

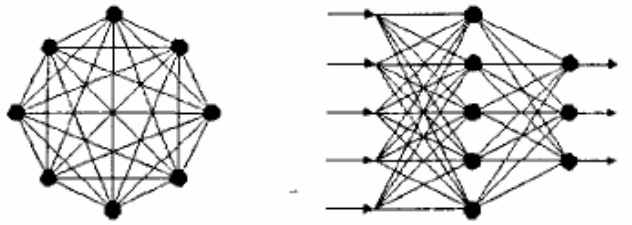
Продолжение табл. 13

Название	Формула	Область значений	Функция MatLab
Гиперболический тангенс (сигмоидальная)	$f(s)=\frac{e^s-e^{-s}}{e^s+e^{+s}}$	(–1…1)	tansig
Сигмоидальная (логистическая)	$f(s)=\frac{1}{1+e^{-s}}$	(0…1)	logsig
Треугольная	$f(s)=\begin{cases}1-\lvert s\rvert, \lvert s\rvert\le 1\\0, \quad \lvert s\rvert>1\end{cases}$	(0…1)	tribas

Среди нейронных сетей можно выделить две базовые архитектуры: *слоистые* и *полносвязные* сети.

Полносвязные сети. Каждый нейрон передает свой выходной сигнал остальным нейронам, включая самого себя (рис. 7.2, *а*). Выходными сигналами сети могут быть все или некоторые выходные сигналы нейронов после нескольких тактов функционирования сети. Все выходные сигналы подаются всем нейронам.

Слоистые сети. Нейроны расположены в несколько слоев (рис. 7.2, *б*). Нейроны первого слоя получают входные сигналы, преобразуют их и через точки ветвления передают нейронам второго слоя. Далее срабатывает второй слой и т.д. до *k*-го слоя, который выдает выходные сигналы для пользователя. Если не оговорено противное, то каждый выходной сигнал *i*-го слоя подается на вход всех нейронов (*i* + 1)-го слоя. Число нейронов в каждом слое может быть любым и никак заранее не связано с количеством нейронов в других слоях. Стандартный способ подачи входных сигналов: все нейроны первого слоя получают каждый входной сигнал. Особое распространение



а) *б*)

Рис. 7.2 Архитектуры нейронных сетей:
а – полносвязная сеть; *б* – слоистая сеть

ПОЛУЧИЛИ ТРЕХСЛОЙНЫЕ СЕТИ, В КОТОРЫХ КАЖДЫЙ СЛОЙ ИМЕЕТ СВОЕ НАИМЕНОВАНИЕ: ПЕРВЫЙ – ВХОДНОЙ, ВТОРОЙ – СКРЫТЫЙ, ТРЕТИЙ – ВЫХОДНОЙ.

Прогноз, аппроксимация и другие операции с функциями с помощью нейронных сетей сводятся к их обучению. При этом входные сигналы s_i подаются обучаемой сети на обработку, задаются значения весовых коэффициентов w_i, а сигналы на выходе из сети (при ее обучении) g сравниваются с экспериментальными данными y. Затем строится оценка работы сети, например, как критерий максимального правдоподобия

$$E(w) = \frac{1}{2}\sum_{\lambda=1}^{P}\sum_{i=1}^{N_K}(y_{i\lambda} - g_i(s^{(\lambda)}, w))^2 ,$$

где $g_i(s^{(\lambda)}, w)$ – i-й выход сети, соответствующий векторам входных сигналов $s^{(\lambda)}$ и весовых коэффициентов w; P – объем обучающей выборки $(s^{(\lambda)}, y_\lambda)$.

Поиск оптимальных значений весовых коэффициентов w, при которых критерий $E(w)$ минимален, производится с помощью известных методов решения экстремальных задач.

7.2 Создание и использование ИНС с помощью Neural Networks Toolbox

В качестве основных этапов реализации нейронно-сетевого подхода для решения множества разнообразных задач можно выделить:

1 Подготовку данных для тренировки сети.
2 Создание сети.
3 Обучение сети.
4 Тестирование сети.
5 Моделирование сети (использование сети для решения конкретной задачи).

В качестве примера рассмотрим следующую задачу. В реакторе периодического типа действия проводится синтез эфиров ортокремневой кислоты – алокси- (арокси) – силанов. Реакция проводится при температурах от 20 до 80 °С и протекает через последовательные ступени замещения в соответствии со следующей формальной кинетикой:

$$SiCl_4 + 4C_2H_5OH = Si(OC_2H_5)_4 + 4HCl .$$

В ходе проведенных экспериментальных исследований было изучено влияние изменения температуры проведения процесса в диапазоне температур от 20 до 60 °С и концентрации тетрахлорсилана (от 1 до 3 моль/л) на выход целевого продукта. На основании полученных экспериментальных данных необходимо создать, обучить и протестировать НС. Используя обученную НС, дать прогноз по выходу целевого продукта при изменении температуры процесса от 60 до 80 °С и концентрации тетрахлорсилана от 3 до 5 моль/л.

7.3 Подготовка данных для обучения НС

Для обучения сети необходимо сформировать массив входных векторов x_i (x_1 – температура проведения процесса; x_2 – концентрация раствора тетрахлорсилана) для различных значений параметра y (выход в процентах от теоретически возможного алокси- (арокси) – силанов). Каждое значение параметра y является *вектором-эталоном* для обучения ИНС.

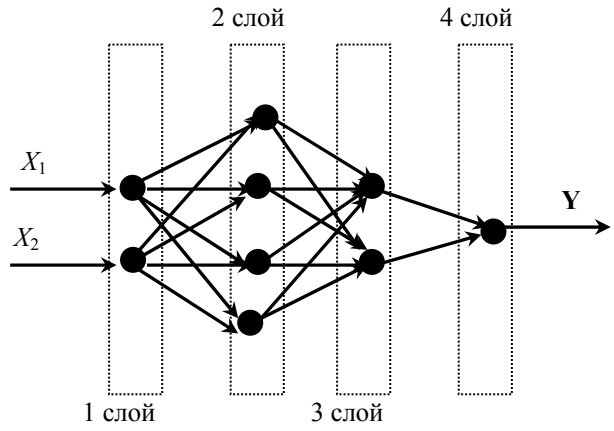
Чтобы создать, обучить, протестировать и использовать для прогнозирования нейронную сеть для нашего примера, создадим файл-программу. Для подготовки входного и эталонного массивов в данный М-файл с именем NeT2 запишем:

```
% формирование массива входных векторов xi (P(2,30))
P=[21 1.2; 28 1.9; 51 2.1; 59 2.7; 37 2.5; 55 2.8; 26 2.7; ... 26 1.3; 22 2; 35 2; 53 3; 27 2.1; 34 2.7; 42 3; 57 1.5;
...
22 1.5; 34 1.5; 28 2.8; 30 3; 32 2.1; 48 1.8; 51 1.1; ...
56 3; 48 1.1; 24 2.5; 48 2.4; 47 1.5; 38 1.6; 27 1.3; 22 1.8];
P=P';
% формирование вектора эталона y (выходной массив T(1,30))
```

22.6
26.03
30.54

T=[20.13 24.44 30.78 31.45 28.03 31.23 23.88 21.81... 27.14 30.88 24.2 27.02 29.09 30.24 21.18 24.7 25.35... 26.19 29.98 28.66 31.1 28.27 23 29.23 27.45 23.02 21.61];

7.4



Создание НС

Выбор архитектуры сети для решения конкретной задачи основывается на опыте разработчика. Поэтому предложенная ниже архитектура сети является одним вариантом из множества возможных конфигураций (рис. 7.3). Для решения поставленной задачи сформируем *четырехслойную полносвязную сеть*, включающую 2 нейрона во входном слое (по числу компонентов входного вектора) с передаточной функцией logsig, 4 нейрона во втором слое с передаточной функцией logsig, 2 нейрона в третьем слое с передаточной функцией purelin и одним нейроном в выходном слое (по числу компонентов выходного вектора) с передаточной функцией purelin. При этом в качестве обучающего алгоритма выбран алгоритм Levenberg-Marquardt (trainlm).

Рис. 7.3 Структура создаваемой нейронной сети:
X_1, X_2 – входные параметры сети;
Y – выходной (рассчитанный) параметр сети

Указанная сеть формируется с помощью следующей процедуры, которую записываем в программу NeT2:

```
net=newff(minmax(P),[2,4,2,1],{'logsig' 'logsig' 'purelin' …
'purelin'},'trainlm');
```

где первый аргумент – матрица минимальных и максимальных значений компонент входных векторов вычисляется с помощью процедуры minmax. Результатом выполнения процедуры newff является объект – нейронная сеть net заданной конфигурации.

Информацию о других функциях активации алгоритмов оптимизации можно получить, набрав команду help nnet.

7.5 Обучение сети

Следующий шаг – обучение созданной сети, т.е. определение весов и смещений всех синапсов во всех слоях НС в соответствии со следующей общей схемой (рис. 7.4).

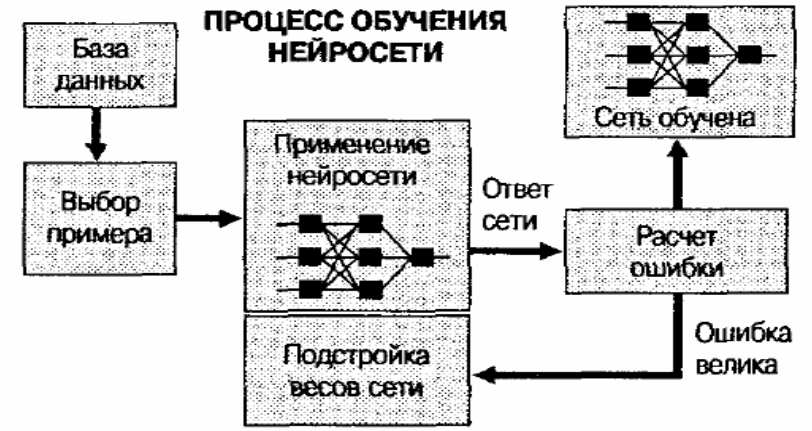


Рис. 7.4 Общая схема обучения НС

При этом перед обучением любой сети необходимо задать параметры обучения (в скобках приведены значения по умолчанию).

–net.trainParam.epochs (100) — заданное количество циклов обучения;

–net.trainParam.show (25) – количество циклов для показа промежуточных результатов;

–net.trainParam.goal (0) – целевая ошибка обучения;

–net.trainParam.time (co) – максимальное время обучения в секундах;

–net.trainParam.min_grad (10^{-6}) – целевое значение градиента;

–net.trainParam.max_fail (5) – максимально допустимая кратность превышения ошибки проверочной выборки по сравнению с достигнутым минимальным значением;

–net.trainParam.searchFcn ('srchcha') – имя используемого одномерного алгоритма оптимизации.

В нашем случае программный код обучения сети будет следующий.

```
% задаем функцию оценки функционирования
net.performFcn='sse';
% задаем критерий окончания обучения
net.trainParam.goal=0.01;
% задаем максимальное количество циклов обучения
net.trainParam.epochs=1000;
% обучаем сеть
[net,tr]=train(net,P,T);
```

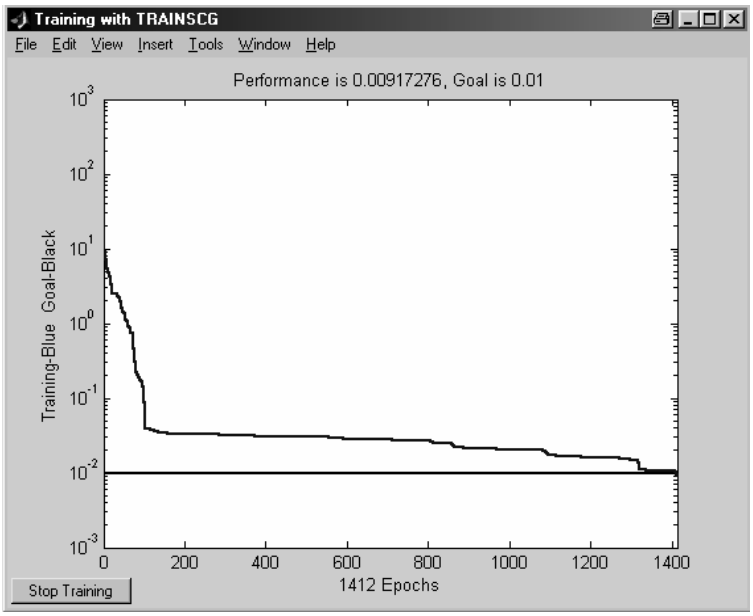


Рис. 7.5 Зависимость *E*(*w*) от цикла обучения

Данный программный код также содержится в программе NeT2.

Процесс обучения иллюстрируется графиком (рис. 7.5) зависимости оценки функционирования от номера цикла обучения.

7.6 Тестирование сети

Перед тем, как воспользоваться нейронной сетью, необходимо исследовать степень достоверности результатов вычислений сети на тестовом массиве входных векторов (на массиве экспериментальных данных). Для оценки достоверности результатов работы сети можно воспользоваться результатами регрессионного анализа, полученными при сравнении эталонных (экспериментальных) значений со значениями, полученными на выходе сети, когда на вход поданы входные векторы тестового массива. В среде MatLab для этого можно воспользоваться функцией postreg. Дополняем NeT2 следующим набором команд, иллюстрирующим описанную процедуру:

```
% создание тестового массива выходных данных
y=sim(net,P);
% регрессионный анализ результатов обработки
[m,b,r]=postreg(y(1,:),T(1,:));
```

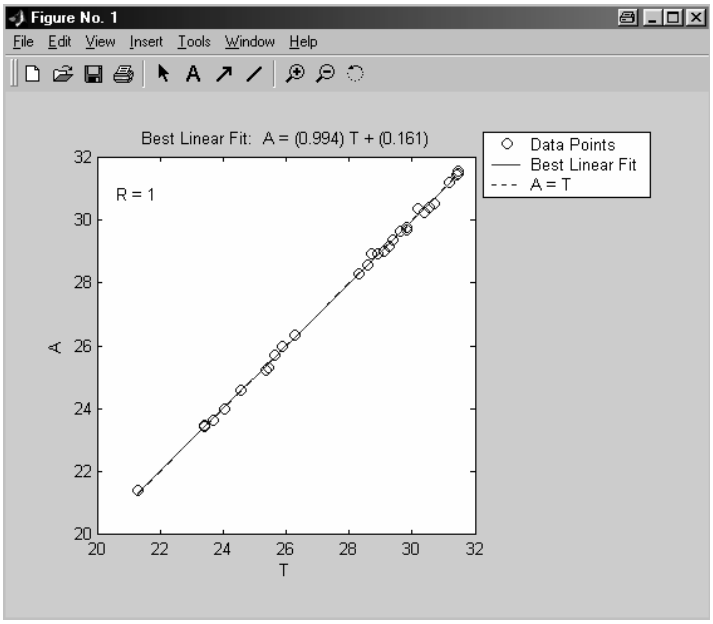


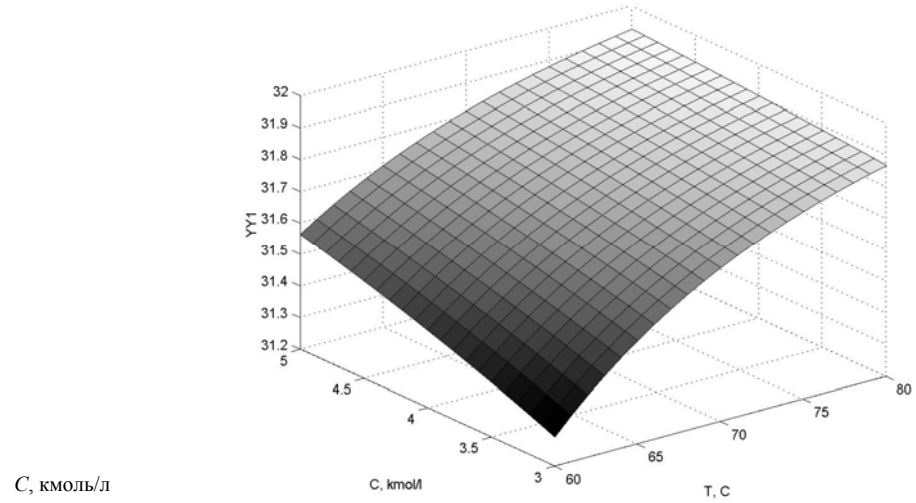
Рис. 7.6 Сравнение эталонных значений с результатами обучения сети

Из сравнения компонентов эталонных векторов с соответствующими компонентами выходных векторов сети видно, что все точки легли на прямую (рис. 7.6), что говорит о правильной работе сети на тестовом массиве.

7.7 Моделирование сети

В результате выполнения предыдущих этапов мы получили обученную сеть и теперь можем перейти собственно к использованию данной сети для прогнозирования выхода по целевому продукту. Для того, чтобы применить обученную сеть для обработки данных, необходимо воспользоваться функцией sim. Например, используя обученную нейронную сеть, дадим прогноз по выходу целевого продукта при изменении температуры процесса от 60 до 80 °С и изменении концентрации тетрахлорсилана от 3 до 5 моль/л.

Для этого произведём просчёт для всех интервалов изменения температуры и концентрации тетрахлорсилана. Дополним файл-программу NeT2 следующим кодом. Результаты просчета представлены на графике (рис. 7.7).



C, кмоль/л

Рис. 7.7 Влияние температуры на выход целевого продукта при концентрации тетрахлорсилана 4 М

```
% вводим значен. входного вектора x
T=[60:80];
C=[3:0.1:5];
[T1, C1]= meshgrid(T, C);
for i=1:length(T1)
    PP = cat(1, T1(i, :), C1(i, :));
    % получаем прогноз выходного вектора y
    YY=sim(net,PP);
    YY1(i,:) = YY;
end
% выводим результаты на экран
figure(1)
colormap(gray) %линии графика серого цвета
surf(T1, C1, YY1)
xlabel('T, C')
ylabel('C, kmol/l')
zlabel('YY1')
```

Имеющиеся в Neural Networks Toolbox функции позволяют, кроме того, получить полную информацию о структуре сети и значениях смещений для каждого слоя и удельные веса каждого синапса. Чтобы вывести информацию о смещениях, необходимо использовать функцию celldisp, дополнив этой функцией программу NeT2:

```
% выводим информацию о смещениях сети
celldisp(net.b)
ans{1} =
-0.0154    -2.4057
ans{2} =
-0.5744    0.6745    0.1748    0.6368
ans{3} =
0.3337    1.3917
ans{4} =
1.3384
```

Аналогично можно получить информацию о величинах весов синапсов

```
% для входного слоя
celldisp(net.IW)
```

% для прочих слоев
celldisp(net.LW)

Полная информация о структуре и свойствах созданной сети выводится с помощью функции disp, например, в нашем случае

disp(net)

8 Расчет вальцевой сушилки в MatLab

Технологический расчет вальцевой сушилки (рис. 8.1) заключается в определении скорости испарения влаги с поверхности материала, находящегося на вальце, нахождении необходимого количества тепла для проведения процесса и требуемой для этого поверхности обогрева.

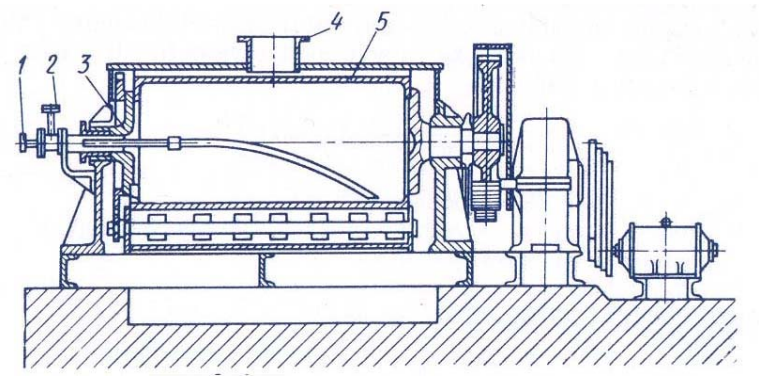


Рис. 8.1 Одновальцевая сушилка:
1 – сифонная трубка; *2* – патрубок для пара; *3* – цапфа;
4 – патрубок для отвода воздуха; *5* – чугунный валец

Порядок расчета [11]:

1 Скорость испарения влаги с поверхности материала, кг/(м^2·с)

$$M = \beta_h (p_н - p_в),$$

где $p_н$, $p_в$ – давление водяных паров у материала и в движущемся воздухе, Па; β_h – коэффициент массоотдачи, кг/(м^2 · с · Па), рассчитываемый по формуле

$$\beta_h = 2{,}083 \cdot 10^{-6} \cdot (0{,}0229 + 0{,}0174\omega_в).$$

Здесь $p_н$ выбирается по таблицам водяного пара при температуре насыщения, равной температуре материала на поверхности вальца, $p_в$ определяется по I – X диаграмме при $t_в$ и $\varphi_в$.

2 Коэффициент теплоотдачи от испаряющейся влаги к воздуху, Вт/(м^2 · К)

$$\alpha_2 = \frac{Mr}{(\nu_м - t_в)},$$

где r – теплота испарения воды при атмосферном давлении.

Коэффициент теплоотдачи от конденсирующегося пара к стенке вальца можно принять равным α_1 = 11 640 Вт/(м^2 · К).

3 Общий коэффициент теплопередачи от конденсирующегося пара к воздуху, Вт/(м^2 · К)

$$K = \frac{1}{\frac{1}{\alpha_1} + \frac{\delta_1}{\lambda_1} + \frac{\delta_2}{\lambda_2} + \frac{1}{\alpha_2}},$$

где δ_1 = 0,01 м – толщина стенки чугунного вальца; δ_2 = 0,001 м – толщина слоя пасты на вальце; λ_1 = 46,5 Вт/(м · К) – коэффициент теплопроводности чугуна; λ_2 – коэффициент теплопроводности пасты красителя, Вт/(м · К).

4 Температура материала на поверхности вальца, °С

$$\nu_м = t_в + \Delta t',$$

где $\Delta t'$ – разность температур материала и воздуха, вычисляемая по соотношению

$$\Delta t' = \frac{K(t_к - t_в)}{\alpha_2},$$

где $t_к$ – температура кипения воды при рабочем давлении в сушилке, так как давление атмосферное, то $t_к$ = 100 °С.

Если рассчитанное значение температуры материала отличается от ранее принятого более чем на 5 %, то принимают $\nu_м = \nu_{м.расч}$ и повторяют расчет.

5 Требуемый расход тепла на сушку материала, Вт

$$Q = Wr + G_1 c_1 (\nu_м - \nu_1),$$

где W – расход влаги, кг/с; c_1 – теплоемкость пасты красителя, Дж/(кг·К).

Количество влаги, которое необходимо удалить из материала в единицу времени (кг/с) для обеспечения снижения влажности продукта со значения ω_1, которое обычно обуславливается типом фильтрующего оборудования, применяемого для разделения суспензии красителя, до величины ω_2, предусматриваемой в стандартах на выпускаемую продукцию, определяют по уравнению материального баланса процесса сушки

$$W = G_1 \frac{\omega_1 - \omega_2}{100 - \omega_2}.$$

6 Расчетная поверхность теплопередачи сушильного вальца, м^2

$$F_р = \frac{Q}{K \Delta t \varphi},$$

где φ = 0,75 – коэффициент, учитывающий поверхность активного контакта между вальцом и материалом.

7 Ориентировочное значение расхода греющего пара (с учетом потери 10 % полезного тепла в окружающую среду) вычисляется по уравнению, кг/с

$$D = \frac{1{,}1 \cdot Q}{r},$$

где r – удельная теплота конденсации пара с давлением P.

Расчет сушилки ведется итерациями и на каждой итерации необходимо задавать давление водяного пара в зависимости от его температуры. Однако эти данные представлены в справочной литературе в виде таблиц [12], а для проведения автоматизированного расчёта необходимо иметь эти данные в виде функции. Используем возможности MatLab для приближения табличной функции в виде полинома. Результаты приближения визуализируются для выбора более точного приближения.

```
% приближение по методу наименьших квадратов
% зависимости давления водяных паров от температуры
% в интервале от 50 град. до 110 град.
% ЗАДАНИЕ ТАБЛИЧНЫХ ДАННЫХ
% температуры
x = [50 55 60 65 70 75 80 85 90 95 100 105 110];
% соответствующие давления
y = [0.1258 0.1605 0.2031 0.2550 0.3177 0.393 0.483 0.590 0.715 0.862 1.033 1.232 1.461];
% %вычисление коэффициентов полиномов разных степеней,
% % приближающих искомую функцию по методу наименьших квадратов
P1 = polyfit(x, y, 1);
P2 = polyfit(x, y, 2);
P5 =  polyfit(x, y, 5)
```

```
% вывод графика табличной функции маркерами
plot (x, y, 'ko')
% построение графиков полиномов
t = [50:5:110];
p1 = polyval(P1, t);
p2 = polyval(P2, t);
p5 = polyval(P5, t);
hold on
plot (t, p1, 'k:', t, p2, 'k-', t, p5, 'k-.')
legend('табличные данные', 'n=1', 'n=2', 'n=5', 0)
title('Приближение табличной функции полиномами степени n')
xlabel('t, град.C')
ylabel('P, кг/см2')
```

Результаты приближения методом наименьших квадратов представлены на рис. 8.2. Очевидно, что приближение полиномом 5-го порядка наилучшее из рассмотренных, поэтому можно воспользоваться полученным полиномом для расчета.

```
function P = vap(temp)
% функция зависимости давления паров воды
% от температуры (приближена полиномом 5-ой степени)

% использование P = vap(t),
% где P - давление, Па;
% temp - температура, град.C

p=[0.00000000007089 -0.00000001398190 0.00000284748046 ...
-0.00017754514603 0.00743024544080 -0.09255158370997];
P1 = polyval(p, temp);
P = 98100*P1;
```

Расчет вальцевой сушилки производится в программе dryer. Данная программа использует файл-функцию vap, которая должна быть записана в ту же папку, что и программа dryer. Результаты расчёта выводятся в командное окно.

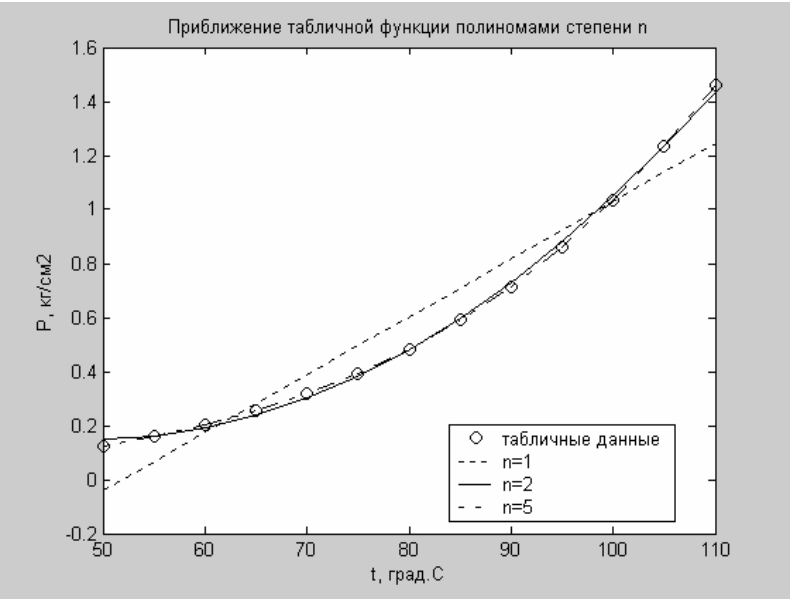


Рис. 8.2 Результаты приближения методом наименьших квадратов

```
% расчёт вальцевой сушилки

% исходные данные
```

```
G = 0.019; % кг/с, производительность сушилки
w_b = 60;% %, начальная влажность материала
w_f = 8;% %, конечная влажность материала
t1 = 20; % град.С, начальная температура материала
delta = 0.001; % м, толщина плёнки пасты на вальцах
delta1 = 0.01; % м, толщина стенки чугунного вальца
lambda = 0.310; % Вт/(м*К), коэфф. теплопроводности пасты lambda1 =  46.5; %Вт/(м*К) – коэфф. теплопроводности чугуна
vel = 1.2;% м/с, скорость воздуха над поверхностью материала
c1 = 1.2;% Дж/с, теплоёмкость
P_st = 0.12;% МПа, давление пара, обогревающего вальцы
T_st = 104.2; % град.С, температура пара, обогревающего вальцы
fi_air = 68; % %, влажность агента (воздух)
T_air = 40;%  град.С, температура агента
% давление воздуха при fi_air и T_air по I-x диаграмме
P_air = 37*133.3; % Па
r = 2390e3; % Дж/кг, удельная теплота парообразования

% РАСЧЕТ
% Задаёмся температурой материала на поверхности вальца
t_mat = 104.2; % град.С, равна температуре пара
epsil = 1;% принимаем максимальное значение точности решения
P_mat = 0.12e6;% давление пара при t_mat, Па

while epsil>0.05
   betta = 2.083e-6*(0.0229+0.0174*vel); % коэфф. массоотдачи
 % скорость испарения влаги с поверхности материала, кг/(м2с)
   M = betta*(P_mat-P_air);

 % Вт/м2*К, Коэфф. теплоотдачи от испаряющейся влаги к воздуху
  alfa2 = M*r/(t_mat-T_air);
 % Вт/м2*К, Коэфф. теплоотдачи от конденс. пара к стенке вальца
  alfa1 = 11640;
 % Общ. коэффициент теплопередачи
  K = (alfa1^-1+alfa2^-1+(delta1/lambda1)+(delta/lambda))^-1;

% Разность температур материала и воздуха
  dT = K*(100-T_air)/alfa2;
% Температура материала на поверхности вальца, град.С
  T_mat = T_air+dT;

  epsil = abs((t_mat-T_mat)/t_mat);
  t_mat = T_mat;
  P_mat = vap(t_mat);
end
% температура материала на поверхности вальца, град.С
% давление пара при t_mat, Па
% Кол-во удаляемой из материала влаги в единицу времени (кг/с)
W = G*(w_b-w_f)/(100-w_f)
% Требуемый расход тепла на сушку материала, Вт
Q = W*r+G*c1*(t_mat-t1)
% Расчетная поверхность теплопередачи сушильного вальца, м2
deltat=((T_st-t1)-(T_st-t_mat))/(log((T_st-t1)/(T_st-t_mat)));
F = Q/(K*0.75*deltat)
% Ориентировочное значение расхода греющего пара, кг/с
D = 1.1*Q/r
```

Список литературы

1 Процессы и аппараты химической технологии. Явления переноса, макрокинетика, подобие, моделирование, проектирование: В 5 т. Т. 1: Основы теории процессов химической технологии / Под. ред. А.М. Кутепова. М.: Логос, 2003. 480 с.

2 Дворецкий С.И., Егоров А.Ф., Дворецкий Д.С. Компьютерное моделирование и оптимизация технологических процессов и оборудования: Учебное пособие. Тамбов: Изд-во Тамб. гос. техн. ун-та, 2003. 224 с.

3 Самарский А.А., Михайлов А.П. Математическое моделирование: Идеи. Методы. Примеры. М.: Физматлиб, 2001. 320 с.

4 Дильман В.В., Полянин А.Д. Методы модельных уравнений и аналогий в химической технологии. М.: Химия, 1988. 304 с.

5 Бодров В.И., Дворецкий С.И., Дворецкий Д.С. Оптимальное проектирование энерго- и ресурсосберегающих процессов и аппаратов химической технологии // ТОХТ. 1997. Т. 31, № 5. С. 542 – 548.

6 Островский Г.М., Волин Ю.М., Гловашкин Д.В. Новые подходы к исследованию гибкости и оптимизации химико-технологических процессов в условиях неопределенности // ТОХТ. 1997. Т. 31, № 2. С. 202 – 207.

7 Ануфриев И.Е. Самоучитель Matlab 5.3/6.x. СПб.: БХВ – Петербург, 2002. 736 с.

8 Дьконов В.П. MatLab. СПб.: Питер, 2001. 553 с.

9 Потемкин В.Г. Система MatLab: Справ. пособие. М.: ДИАЛОГ-МИФИ, 1997. 350 с.

10 Потемкин В.Г. Система инженерных и научных расчетов Matlab 5.x: В 2 т. М.: ДИАЛОГ-МИФИ, 1997.

11 Чернобыльский И.И., Тананайко Ю.М. Сушильные установки химической промышленности. Киев: Техника, 1969. 279 с.

12 Павлов К.Ф., Романков П.Г., Носков А.А. Примеры и задачи по курсу процессов и аппаратов химической технологии: Учеб. пособие для вузов. Л.: Химия, 1981. 560 с.